1986

The Vector Processor

Ramachandra K. Manja

THE VECTOR PROCESSOR

by

Ramachandra K. Manja

A thesis paper
in partial fulfillment of the requirements of the
Degree, Master of Science, Department of
Electrical Engineering
South Dakota State University

1986

THE VECTOR PROCESSOR

by

Ramachandra K. Manja

This thesis paper is approved as a creditable and independent investigation by a candidate for the degree, Master of Science, and is acceptable for meeting the thesis paper requirements for this degree. Acceptance of this thesis paper does not imply that the conclusions reached by candidate are necessarily the conclusions of the major department.

Thesis Paper Advisor	Date
Head, Electrical Engineering Department	Date

ACKNOWLEDGEMENTS

It is with immense pleasure that I acknowledge the excellent guidance and suggestion of Dr. D. B. Miron, Associate Professor of Electrical Engineering. I express my sincere appreciation to Dr. V. G. Ellerbruch, Professor and Head of Department of Electrical Engineering, for his valuable review and encouragement. I must also thank Mr. W. Mostad, Electrical Technician, Department of Electrical Engineering, for his technical assistance. I am thankful to Mrs. Mary Lou Michalewicz, Secretary to the Department of Electrical Engineering, for her skillfull and excellent typing of the manuscript.

Manja

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The Vector Processor is a device designed for the purpose of economically achieving maximum speed in the computer execution of a group of common scientific and engineering calculations. In these operations, a variable name represents a group of data items which may be thought of as being arranged in a line. This is known as a vector. Each element of the vector, the individual data item, may be a number, a logical value, or a character. A vector's elements must be all of the same type. A vector may have meaning in its own right, or it may be part of an array, a row or a column of elements. Manipulations and operations with vectors have been used by mathematicians, scientists and engineers for centuries in procedures for analysis and design.

The general arrangement of the major elements of The Vector Processor are shown in Figure 1.1. The IBM PC is choosen as the General Purpose Processor and the Vector Controller is designed to coordinate the activities of a group of components called Math Coprocessors with that of the IBM PC. Figure 1.2 shows the conventional architecture containing the 8088 CPU and the 8087 Math Coprocessor. The

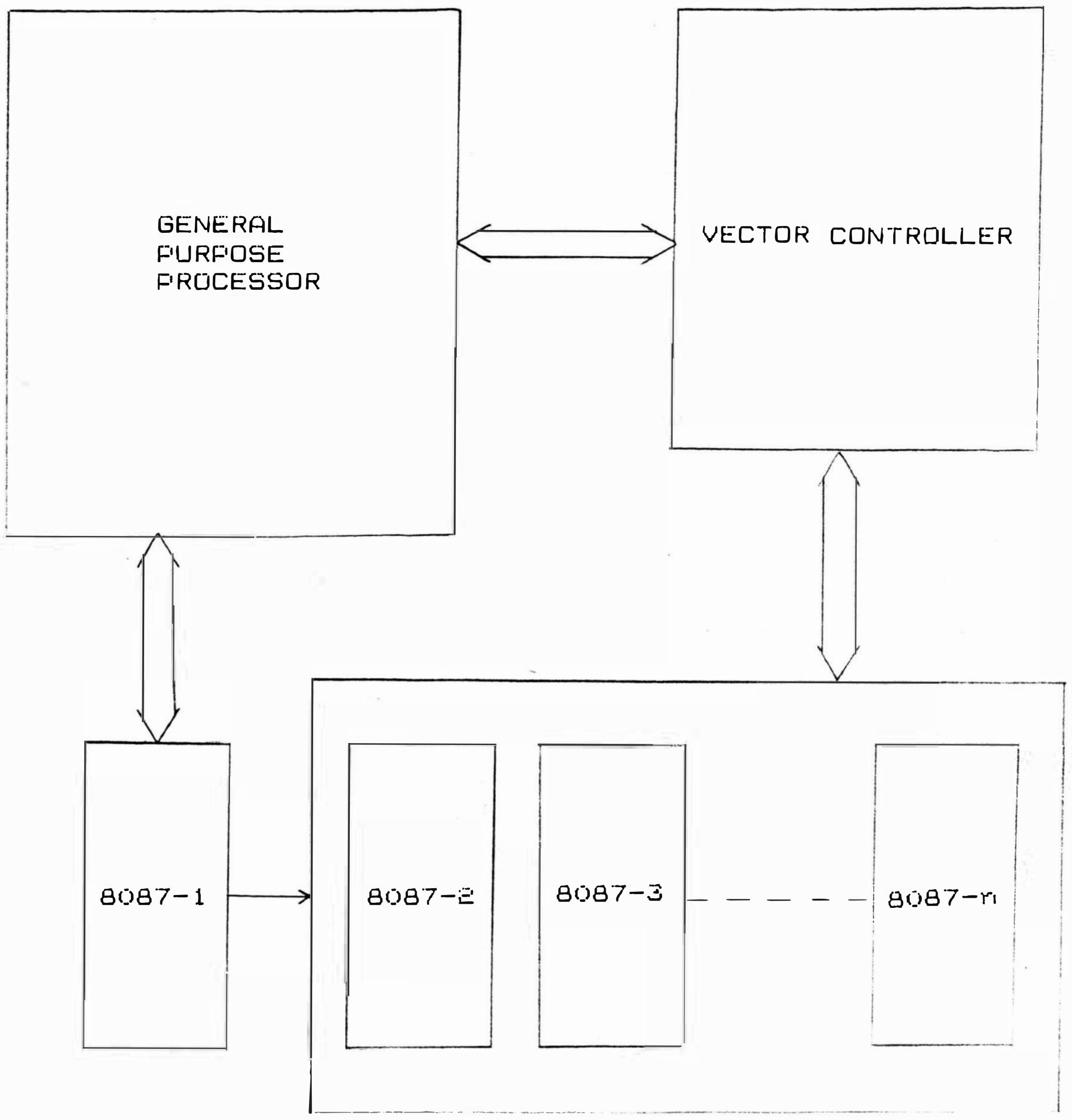


Figure 1.1 The Vector Processor Block Diagram

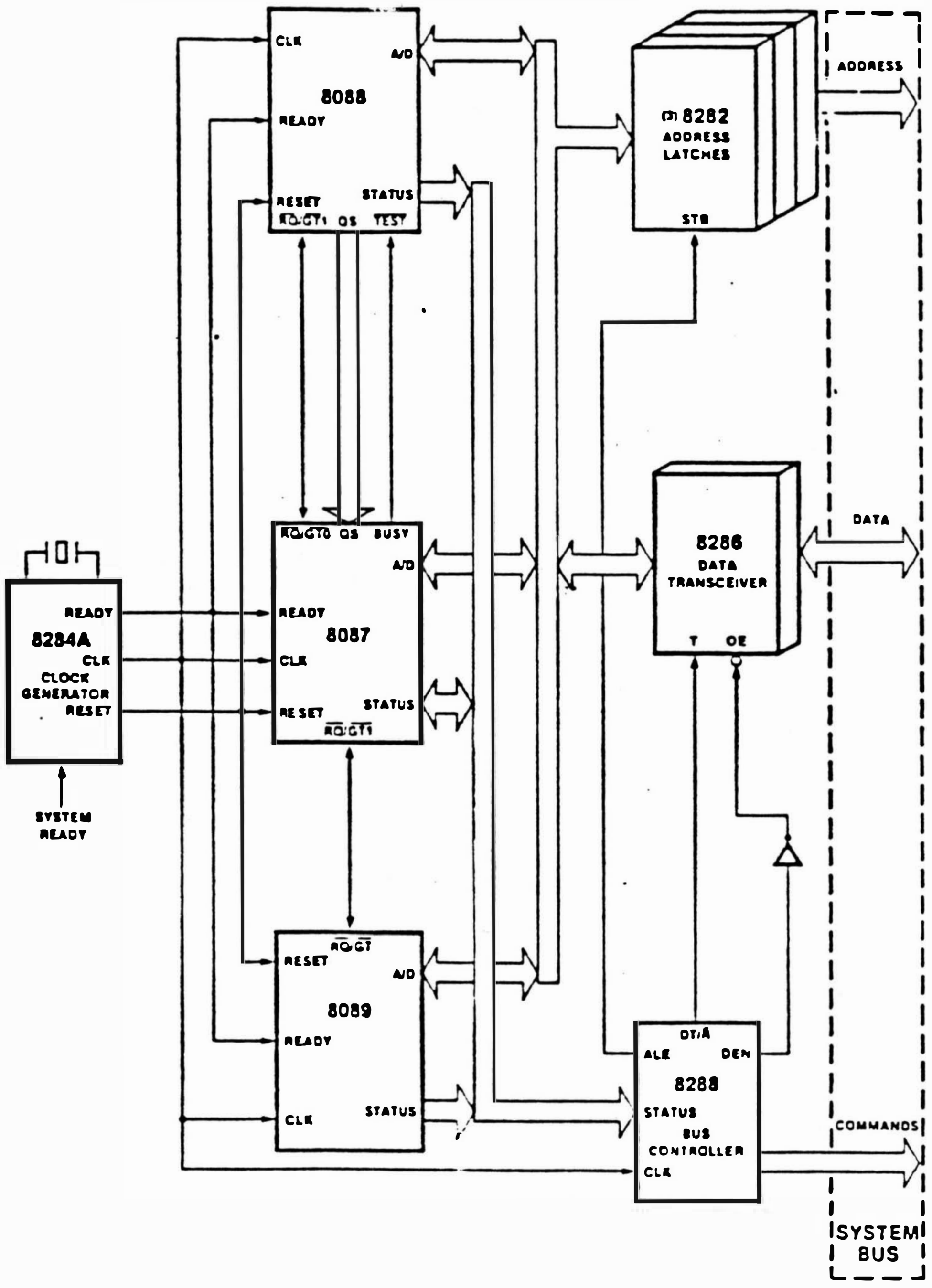


Figure 1.2 General System Configuration

two processors cooperate by sharing the buses and ignoring each others instructions. The bus sharing is managed by the interface signals.

The vector operations are done in the Math Coprocessors (Figure 1.1), by loading the data elements sequentially into the Math Coprocessors and executing an instruction in parallel. The sequential and the parallel operations are controlled by the Vector Controller which recognizes the vector instructions. In the absence of instructions involving vector quantities, the Vector Controller has one of the Math Coprocessors connected to the General Purpose Processor as in the conventional architecture of Figure 1.2. This allows for maximum speed in executing ordinary arithmetic operations.

When a vector instruction appears, it is decoded by the Vector Controller and appropriate action taken. The effect of these actions is to load the elements of the required vectors sequentially from Memory into the Math Coprocessors, do the required operations simultaneously, and then write the results out sequentially to Memory. This simultaneous operation on a number of data elements is the key to the speed of this processor.

The ensuing chapters describe the hardware and software requirements for the design. Understanding of the hardware design requires the knowledge of the architecture of the 8088 and the 8087 processors. The next two chapters, Chapter II and Chapter III, explain the architecture of the 8088 CPU and the 8087 Math Coprocessor, in brief. Chapter IV describes System Clock and Bus Cycles giving an idea about timing requirements of the system. Chapter V and the subsequent Chapters explain in detail the hardware implementation of the Vector Controller. Finally the software requirements are described in Chapter IX, with Conclusions in Chapter X.

CHAPTER II

THE 8088 MICROPROCESSOR

The heart of the IBM PC is its 8/16-bit 8088 microprocessor. A microprocessor is a general purpose processing unit built into a single integrated circuit. The 8088 was the first 8/16 -bit microprocessor introduced by Intel Corporation. The 8088 is enclosed in a 40-pin dual-in-line package as shown in Figure 2.1 and requires a +5V power supply.

The 8088 is called an 8/16-bit processor as it has an 8-bit external data path, whereas, its internal bit is 16-bits wide. The 20-bit wide address bus enables it to address up to one Mega byte of memory. It can also address up to 64K of byte-wide input/output ports. The pins AD0 through AD7 serve as time multiplexed address and data bus.

The 8088 has two system modes of operation and can be configured to operate in either of these two modes, viz, the minimum system mode and the maximum system mode. By applying logic 1 or 0 to the MN/MX input lead (pin 33), one of the two required modes can be selected. The minimum mode systems are smaller and contain a single processor, whereas the maximum system mode feature lets the 8088 coordinate the activities of other processors in the system

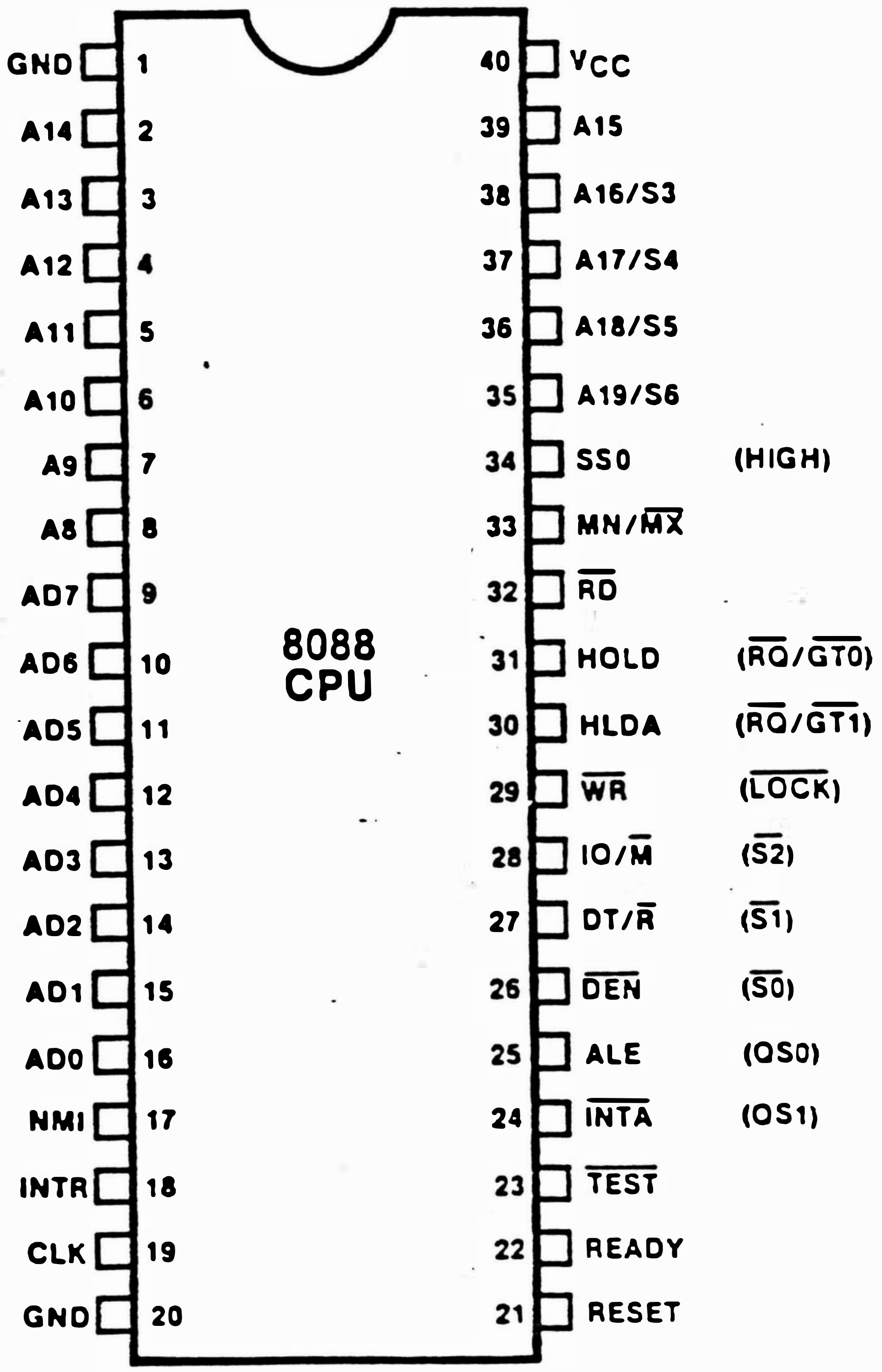


Figure 2.1 8088 CPU Pin Diagram

like the 8087 Math Coprocessor, the 8089 Input/Output (I/O) processor, etc. Figure 2.2 gives the pin descriptions of the 8088 in both minimum and maximum mode system configuration.

As indicated in Figure 2.2, in the minimum system mode of operation, the 8088 provides all the control signals needed to implement the memory and the I/O interfaces. For the Vector Processor interface, the 8088 must be configured in maximum system mode and hence requires that the MN/MX input lead (pin 33) be tied low. In this mode the 8088 produces signals for implementing a Coprocessor or a multiprocessor system environment. This mode also facilitates the passing of bus control to other Coprocessors through the RQ/GT lead. Looking at Fig 2.2, it is shown that the 8088 does not directly provide all the signals that are required to control the memory, I/O and interrupt interfaces. Instead, it generates three status signals S0, S1 and S2 prior to the beginning of each machine cycle or bus cycle. Each 3-bit status code represents the type of bus cycle that is to follow. The Vector Controller and the 8288 bus controller decode the status information to identify the type of microprocessor bus cycle. Figure 2.3 shows the decoded status signal and also the command signals generated by the 8288 bus controller.

(a)

Common Signals		
Name	**Function**	**Type**
AD15-AD0	Address/Data Bus	Bidirectional, 3-State
A19/S6-A16/S3	Address/Status	Output, 3-State
$\overline{BHE}$/S7	Bus High Enable/ Status	Output, 3-State
MN/$\overline{MX}$	Minimum/Maximum Mode Control	Input
$\overline{RD}$	Read Control	Output, 3-State
$\overline{TEST}$	Wait On Test Control	Input
READY	Wait State Control	Input
RESET	System Reset	Input
NMI	Non-Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
V_{CC}	+5V	Input
GND	Ground	

(b)

Minimum Mode Signals (MN/MX = V_{CC})		
Name	**Function**	**Type**
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
$\overline{WR}$	Write Control	Output, 3-State
M/$\overline{IO}$	Memory/IO Control	Output, 3-State
DT/$\overline{R}$	Data Transmit/ Receive	Output, 3-State
$\overline{DEN}$	Data Enable	Output, 3-State
ALE	Address Latch Enable	Output
$\overline{INTA}$	Interrupt Acknowledge	Output

(c)

Maximum Mode Signals (MN/MX = GND)		
Name	**Function**	**Type**
$\overline{RQ}$/$\overline{GT1, 0}$	Request/Grant Bus Access Control	Bidirectional
$\overline{LOCK}$	Bus Priority Lock Control	Output, 3-State
$\overline{S2}$-$\overline{S0}$	Bus Cycle Status	Output, 3-State
QS1, QS0	Instruction Queue Status	Output

Figure 2.2 (a) Common Signals (b) Minimum Mode Signals (c) Maximum Mode Signals

Status Inputs			CPU Cycle	8288 Command
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$		
0	0	0	Interrupt Acknowledge	$\overline{\text{INTA}}$
0	0	1	Read I/O Port	$\overline{\text{IORC}}$
0	1	0	Write I/O Port	$\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{\text{MRDC}}$
1	0	1	Read Memory	$\overline{\text{MRDC}}$
1	1	0	Write Memory	$\overline{\text{MWTC}}$, $\overline{\text{AMWC}}$
1	1	1	Passive	None

Figure 2.3 Bus Status Codes and Command Signals

The maximum system mode 8088 also produces two more signals known as Queue Status outputs QS0 and QS1. This 2-bit Queue Status code informs the external processor like the 8087 Math Coprocessor about the status of the instruction queue that is maintained inside the processor as shown in Figure 4.10. Thus QS0 and QS1 allow external tracking of the internal 8088 instruction queue.

RQ/GT0 and RQ/GT1 of the maximum system mode provide a prioritized bus access for the external 8087 Coprocessor, allowing it to become the bus master.

A. Internal Architecture of the 8088.

The processing unit of the 8088 microprocessor is divided into two separate units known as the Execution Unit (EU) and the Bus Interface Unit (BIU). Figure 2.4 shows the internal architecture of the 8088 microprocessor with all its internal registers.

1. 8088 Registers.

The 8088 contains in all a total of 14 16-bit registers which are user accessible (Figure 2.4). These registers are usually grouped as a Instruction pointer, four data registers, four pointer and index registers, four segment registers and a status register.

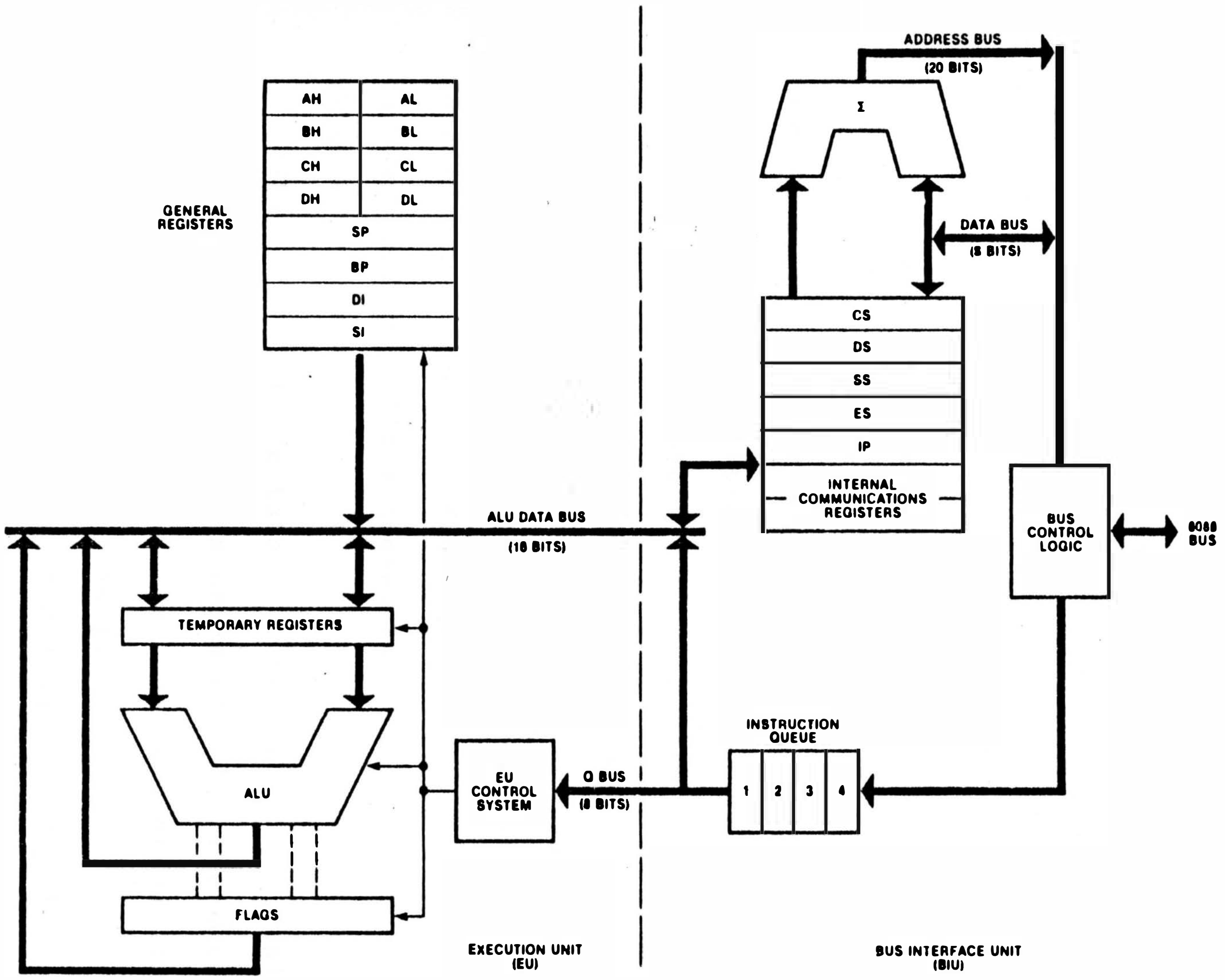


Figure 2.4 8088 Functional Block Diagram

The Instruction Pointer (IP) is a 16-bit register that locates the next instruction to be executed in the current code segment. The IP contains an offset value which must be combined with the value of the code segment register to form the 20-bit physical address of the memory containing the instruction.

The data registers are the general purpose registers and the four registers are referred to as: AX, BX, CX, and DX. Each of these data registers can be used as either 16-bit register or two 8-bit registers.

There are two Pointer registers and two Index registers that are used for storing offset addresses of memory locations relative to the segment registers. The two pointer registers are the stack pointer (SP) and the base pointer (BP). The stack pointer enables access to a location in the stack segment of memory and the base pointer allows access to data within the stack segment.

The two Index registers - Source index register (SI) and Destination Index Register (DI) are used to store an offset address for a source operand and a destination operand respectively.

The memory space of the 8088 is divided into logical segments of 64K bytes each and the four segment registers are used to access these memory segments. The four segment registers are:

CS - Code segment register

DS - Data segment register

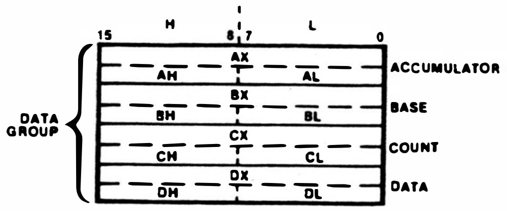
SS - Stack segment register

ES - Extra segment register.

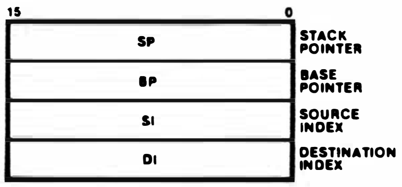
The code segment register is used to point to the current code segment and the instructions are fetched from the segment.

The data is stored in the memory space pointed to by the data segment register. The stack segment register identifies the current stack segment in memory on the locations of which all the stack operations are performed. Lastly, the current extra segment of memory space is pointed to by the Extra segment register which is also used for data storage.

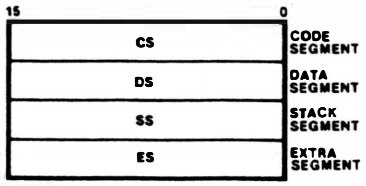
Figure 2.5 shows these various registers of the 8088 except the Instruction pointer register. The status register makes use of its 9 bits as 6 1-bit status flags and 3 1-bit control flags as shown in Figure 2.5. The status flags that indicate conditions on execution of different instructions are: CF, PF, AF, ZF, SF,and OF.



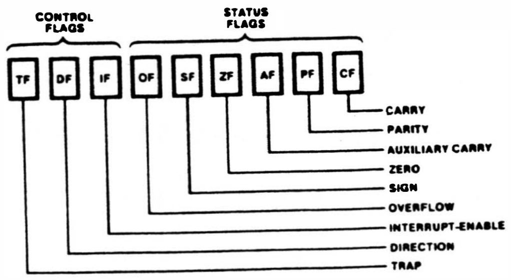
(a) General-Purpose Registers



(b) Printer and Index Registers



(c) Segment Registers



(d) Status and Control Flags

Figure 2.5 8088 Registers

The three control flags that control certain functions of the 8088 are: Trap flag (TF), Interrupt flag (IF), and Direction flag(DF).

2. Execution Unit and Bus Interface Unit.

The Execution Unit (EU) and the Bus Interface Unit (BIU) are the two divisions of the 8088 procesing unit (Figure 2.4). The Execution Unit obtains an instruction from the instruction queue maintained by the Bus Interface Unit. The instructions are then decoded and executed by the execution unit. While the EU is executing an instruction, the Bus Interface Unit will be fetching an instruction. The BIU forms a 4 byte instruction queue of these prefetched instructions as shown in Figure 2.4. The two units can operate independently of one another and hence operate in parallel under most circumstances. If the EU requires a data transfer, it requests the bus interface unit to perform the read or write cycles to memory or Input/Output. The BIU then suspends the instruction fetch and does the data transfer bus cycle for the EU and continues with its instruction fetch bus cycle. The request for the bus cycle from the EU depends on the instruction being executed and hence is an asynchronous operation.

CHAPTER III

THE 8087 MATH COPROCESSOR

The 8087 Math Coprocessor operates in parallel with the main 8088 CPU. The 8088 CPU acts as a host to this 8087 Numeric processor. The 8087 Numeric processor depends on the host CPU for instruction fetch, read, and write bus cycles. Hence, the name Coprocessor. The Coprocessor can decode and execute on its own.

The 8087 Numeric processor, from here on referred to as 8087 Math Coprocessor or simply Coprocessor, is enclosed in a standard 40-pin dual-in-line package as shown in Figure 3.1. It needs a single power supply of +5 volts. The Coprocessor enhances the computational capability of the CPU by being able to perform arithmetic and comparison operations on various data types. The internal data path of the Coprocessor is 64 bits wide and its registers can handle 80-bit long data. This is four to five times larger compared with the 16-bit length of the 8088 CPU itself. It also has built-in transcendental functions such as Log and Tangent functions. In effect, the Coprocessor increases the number of registers and the instruction sets of the

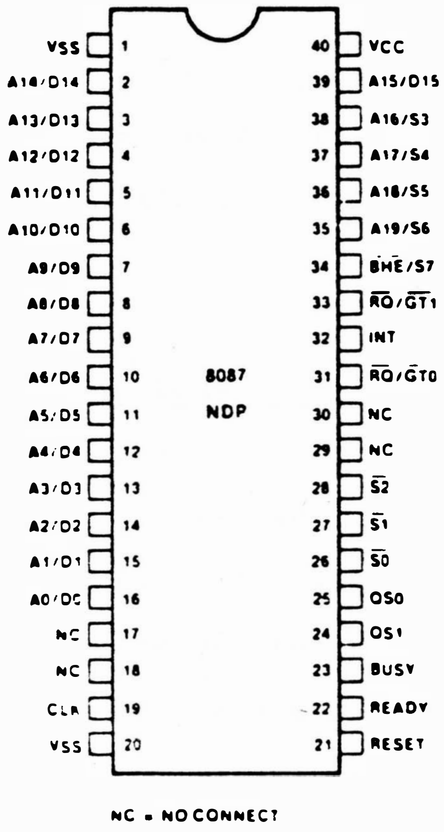


Figure 3.1 8087 Math Coprocessor Pin Diagram

host CPU and allows the computation of new and large data types.

The Math Coprocessor has its own instruction set and hence, can be invoked directly. In case of an error the Coprocessor can interrupt the CPU and thus trap to a user-defined procedure. The Coprocessor shares the Clock Generator and system bus interface components, such as bus controller, latches, transceiver, etc with the 8088 CPU, as shown on Figure 1.2.

A. Math Coprocessor Pin Description.

Table 3-1 shows a detailed pin description of the 8087 Coprocessor. Sixteen of the 40 pins are used for the multiplexed addresse/data bus, AD0 through AD15. Hence this can work as a Coprocessor to an 8-bit or a 16-bit CPU. BHE/S7 (pin 34) is used to determine this bus width of the CPU. There are four more pins for Address A16 to A19 which are time multiplexed with the status lines S3 to S6. The power, ground, clock, and Reset pins are all connected directly to the respective pins of the CPU and serve the same purpose as for the CPU. When in operation, the Busy pin is held high by the Coprocessor. Ready, Busy, Queue status pins QS0, QS1 and status pins S0, S1, and S2 are of special importance. These pin lines are the ones that are

Symbol	Type	Name and Function
AD15–AD0	I/O	**Address Data:** These lines constitute the time multiplexed memory address (T_1) and data (T_2, T_3, T_W, T_4) bus. A0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D7–D0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory operations. Eight-bit oriented devices tied to the lower half of the bus would normally use A0 to condition chip select functions. These lines are active HIGH. They are input/output lines for 8087 driven bus cycles and are inputs which the 8087 monitors when the 8086/8088 is in control of the bus. A15-A8 do not require an address latch in an iAPX 88/20. The 8087 will supply an address for the T_1-T_4 period.
A19/S6, A18/S5, A17/S4, A16/S3	I/O	**Address Memory:** During T_1 these are the four most significant address lines for memory operations. During memory operations, status information is available on these lines during T_2, T_3, T_W, and T_4. For 8087 controlled bus cycles, S6, S4, and S3 are reserved and currently one (HIGH), while S5 is always LOW. These lines are inputs which the 8087 monitors when the 8086/8088 is in control of the bus.
BHE/S7	I/O	**Bus High Enable:** During T_1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D15–D8. Eight-bit oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read and write cycles when a byte is to be transferred on the high portion of the bus. The S7 status information is available during T_2, T_3, T_W, and T_4. The signal is active LOW. S7 is an input which the 8087 monitors during 8086/8088 controlled bus cycles.
$\overline{S2}$, $\overline{S1}$, $\overline{S0}$	I/O	**Status:** For 8087 driven bus cycles, these status lines are encoded as follows: $\overline{S2}$ — $\overline{S1}$ — $\overline{S0}$ 0 (LOW) — X — X — Unused 1 (HIGH) — 0 — 0 — Unused 1 — 0 — 1 — Read Memory 1 — 1 — 0 — Write Memory 1 — 1 — 1 — Passive Status is driven active during T_4, remains valid during T_1 and T_2, and is returned to the passive state (1, 1, 1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory access control signals. Any change in $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These signals are monitored by the 8087 when the 8086/8088 is in control of the bus.
$\overline{RQ}/\overline{GT0}$	I/O	**Request/Grant:** This request/grant pin is used by the NPX to gain control of the local bus from the CPU for operand transfers or on behalf of another bus master. It must be connected to one of the two processor request/grant pins. The request grant sequence on this pin is as follows: 1. A pulse one clock wide is passed to the CPU to indicate a local bus request by either the 8087 or the master connected to the 8087 $\overline{RQ}/\overline{GT1}$ pin. 2. The 8087 waits for the grant pulse and when it is received will either initiate bus transfer activity in the clock cycle following the grant or pass the grant out on the $\overline{RQ}/\overline{GT1}$ pin in this clock if the initial request was for another bus master. 3. The 8087 will generate a release pulse to the CPU one clock cycle after the completion of the last 8087 bus cycle or on receipt of the release pulse from the bus master on $\overline{RQ}/\overline{GT1}$.

Table 3-1 8087 Math Coprocessor Pin Description

Symbol	Type	Name and Function
$\overline{RQ}/\overline{GT}1$	I/O	**Request/Grant:** This request/grant pin is used by another local bus master to force the 8087 to request the local bus. If the 8087 is not in control of the bus when the request is made the request/grant sequence is passed through the 8087 on the $\overline{RQ}/\overline{GT0}$ pin one cycle later. Subsequent grant and release pulses are also passed through the 8087 with a two and one clock delay, respectively, for resynchronization. $\overline{RQ}/\overline{GT}1$ has has an internal pullup resistor, and so may be left unconnected. If the 8087 has control of the bus the request/grant sequence is as follows: 1. A pulse 1 CLK wide from another local bus master indicates a local bus request to the 8087 (pulse 1). 2. During the 8087's next T_4 or T_1 a pulse 1 CLK wide from the 8087 to the requesting master (pulse 2) indicates that the 8087 has allowed the local bus to float and that it will enter the "RQ/GT acknowledge" state at the next CLK. The 8087's control unit is disconnected logically from the local bus during "RQ/GT acknowledge." 3. A pulse 1 CLK wide from the requesting master indicates to the 8087 (pulse 3) that the "RQ/GT" request is about to end and that the 8087 can reclaim the local bus at the next CLK. Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW.
QS1, QS0	I	**QS1, QS0:** QS1 and QS0 provide the 8087 with status to allow tracking of the CPU instruction queue. **QS1** **QS0** 0 (LOW) 0 No Operation 0 1 First Byte of Op Code from Queue 1 (HIGH) 0 Empty the Queue 1 1 Subsequent Byte from Queue
INT	O	**Interrupt:** This line is used to indicate that an unmasked exception has occurred during numeric instruction execution when 8087 interrupts are enabled. This signal is typically routed to an 8259A. INT is active HIGH.
BUSY	O	**Busy:** This signal indicates that the 8087 NEU is executing a numeric instruction. It is connected to the CPU's $\overline{TEST}$ pin to provide synchronization. In the case of an unmasked exception BUSY remains active until the exception is cleared. BUSY is active HIGH.
READY	I	**Ready:** READY is the acknowledgment from the addressed memory device that it will complete the data transfer. The RDY signal from memory is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH.
RESET	I	**Reset:** RESET causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. RESET is internally synchronized.
CLK	I	**Clock:** The clock provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}		**Power:** V_{CC} is the +5V power supply pin.
GND		**Ground:** GND are the ground pins.

Table 3-1 8087 Math Coprocessor Pin Description (continued) either monitored or controlled by the Vector Controller and are hence explained in detail in the next chapter.

The interrupt pin INT is an output pin used to interrupt the CPU in case of an exception. Usually this is connected through an interrupt controller to the CPU. The Request/Grant pin gives the Coprocessor the power to become the bus master when needed. Through this pin the Coprocessor requests the bus from the CPU and controls the bus during certain operations. The release of the bus to the CPU is signalled through the same pin. The spare Request/Grant pin is for the other processors on the system to request the bus from the CPU through this pin of the Coprocessor.

B. Internal Architecture of the 8087 Math Coprocessor.

The internal processing unit of the 8087 Math Coprocessor is divided into two units, viz, the Control Unit (CU) and the Numeric Execution Unit (NEU) as shown in Figure 3.2. The Control Unit synchronizes the Coprocessor activities with that of the CPU. It consists of exception pointer, operands queue, addressing and bus tracking circuitry and two 16-bit registers - Control Word and Status Word. The Numeric Execution Unit executes the numeric class of instructions. It is made up of eight

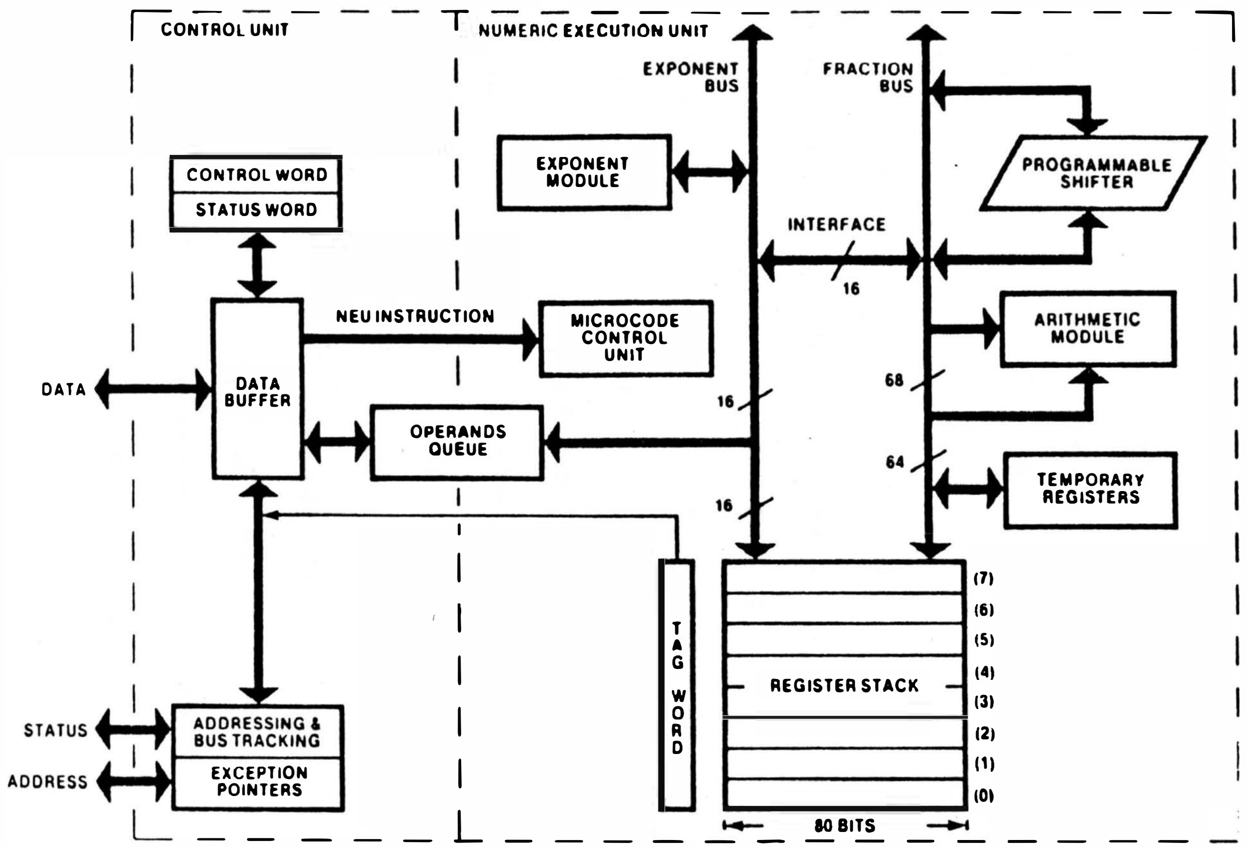


Figure 3.2 8087 Functional Block Diagram

80-bit registers, tagword, exponent modulus, programmable shifter, arithmetic module, and temporary registers.

1. Control Unit.

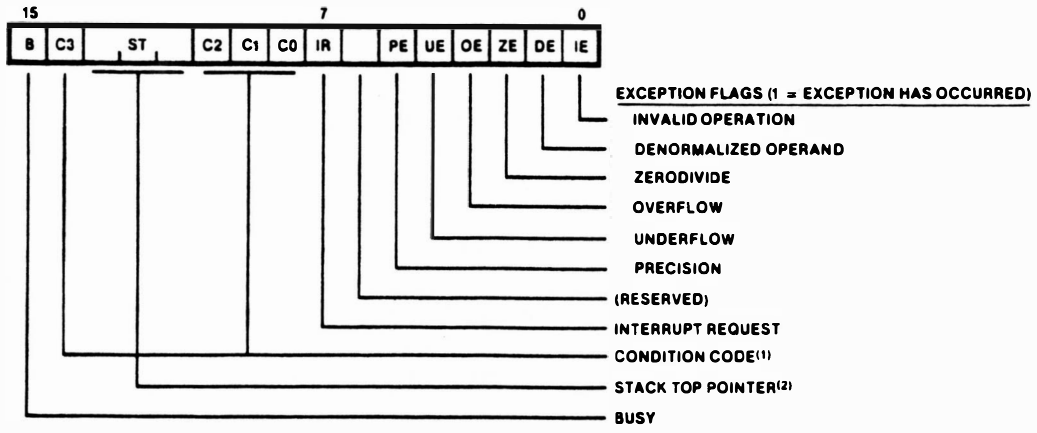
The Coprocessor's instructions are intermixed with CPU instructions in a single instruction stream fetched by the CPU. The Control Unit monitors the CPU status lines and latches on to the instructions as it is available on the data bus. Thus, the Control Unit of the Coprocessor is able to maintain an identical queue (operands queue of Figure 3.2) to that of the instruction queue of the CPU (Figure 2.2). Note that the Coprocessor is not capable of instruction fetching without the help of the CPU. By monitoring the CPU's queue status lines, the Control Unit is able to decode instructions from the operands queue in synchronism with the CPU. In effect, both the Coprocessor and the CPU fetch and decode an instruction from the instruction stream in parallel. The Control Unit ignores all the instructions pertaining to the CPU and executes the control class of Coprocessor instructions. The numeric instructions are passed on by the Control Unit to the Numeric Execution Unit.

2. Numeric Execution Unit.

Arithmetic, comparison, transcendental, constant and data transfer instructions are executed by the numeric execution unit. The 80-bit wide data path (64 fractional bits, 15 exponent bits and a sign bit) aids in very high speed internal operand transfers increasing the Coprocessor performance.

The eight registers residing in the Numeric Execution Unit are 80 bits wide and are arranged in the form of a stack. These registers can be used like the conventional register of a CPU, for holding constants, accumulation, etc. or in a stack mode with operands pushed on and results popped off. In conventional mode the registers are addressed explicitly. While implicit addressing is used in the stack mode, the register set can also be divided to use a few in both modes.

Explicit register addressing is top-relative while instruction using implicit addressing operate on the register at the top of the stack pointed to by ST (Stack Top) pointer. The subroutine parameter-passing during subroutine programming is simplified because of this stack type architecture and the top relative addressing. The NEU, when executing an arithmetic



(1) See descriptions of compare, test, examine and remainder instructions in section S.7 for condition code interpretation.

(2) ST values:
000 = register 0 is stack top
001 = register 1 is stack top
•
•
•
111 = register 7 is stack top

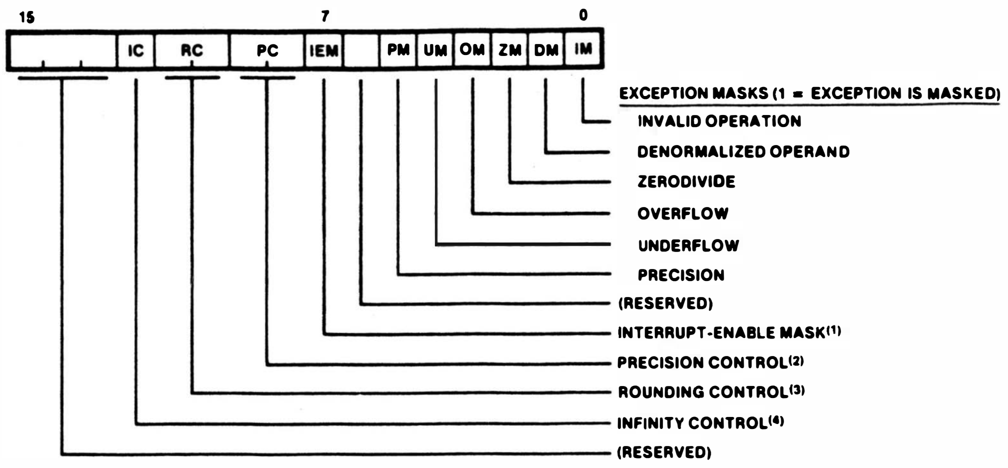
Figure 3.3 Status Word Format

instruction will indicate the zero divide, overflow or other exception through the Status Word register.

3. Status Word and Control Word Registers.

Status Word and Control Word are 16-bit registers present in the Control Unit of the Coprocessor. The Status Word indicates the overall conditions of the Coprocessor. It can be examined by storing it into memory using Coprocessor instruction. Figure 3.3 shows the Status Word format of the 16 different bits. The first 5-bits are used as exception flags, which are set on occurrence of exception during execution of numeric instructions. These exceptions can then be used to interrupt the CPU. The bits C0, C1, C2 and C3 are together known as condition code which is used for conditional branching usually after a comparison instruction. The rest of the fields are self explanatory and are used for functions indicated in Figure 3.3.

The Status Word is a read only register. To mask any of the interrupts arising due to the exception the Control Word register should be used. Control Word is again a 16-bit register that can be written to by loading a word from memory using a Coprocessor instruction. Figure 3.4 shows different fields of the



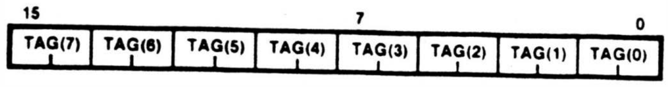
(1) Interrupt-Enable Mask:
- 0 = Interrupts Enabled
- 1 = Interrupts Disabled (Masked)

(2) Precisi n Control:
- 00 = 24 bits
- 01 = (reserved)
- 10 = 53 bits
- 11 = 64 bits

(3) Rounding Control:
- 00 = Round to Nearest or Even
- 01 = Round Down (toward $-\infty$)
- 10 = Round Up (toward $+\infty$)
- 11 = Chop (Truncate Toward Zero)

(4) Infinity Control:
- 0 = Projective
- 1 = Affine

Figure 3.4 Control Word Format



Tag values:
- 00 = Valid (Normal or Unnormal)
- 01 = Zero (True)
- 10 = Special (Not-A-Number, ∞, or Denormal)
- 11 = Empty

Figure 3.5 Tag Word Format

Control Word format wherein the first 5 bits can be used to mask an exception. The fields PC, RC and IC are used for precision control, rounding control, and infinity control respectively. Different modes of these fields are indicated in Figure 3.4.

The tagword register which is used under certain circumstances, as shown in Figure 3.5 is 16 bits in length. This register, residing in the Numeric Execution Unit, is divided into eight fields. The tag value indicates the conditions, shown in Figure 3.5, of the eight registers in the stack.

C. Number System and Data Types

Ideally, it is desired that a processor be able to operate on the entire real number system. But there is no upper or lower limit to the magnitude of the number or to the precision that these numbers can represent. Processors must have fixed-size registers and memories. This results in a limit to the system of numbers that can be represented resulting in a set of finite and discrete numbers.

The range of numbers that the Coprocessor can handle is approximately $\pm$ 4.19×10^{-307} to $\pm$ 1.67×10^{308}. This range is for data and final results of calculation representation. The capacity of the Coprocessor to handle this large range of numbers can be better appreciated when compared with the range of the IBM 370, which is about $\pm 0.54 \times 10^{-78}$ to $\pm 0.72 \times 10^{76}$. The capability of the 8087 Math Coprocessor to operate on large floating point numbers will be further enhanced in terms of speed, when ten of these Coprocessors operate in parallel.

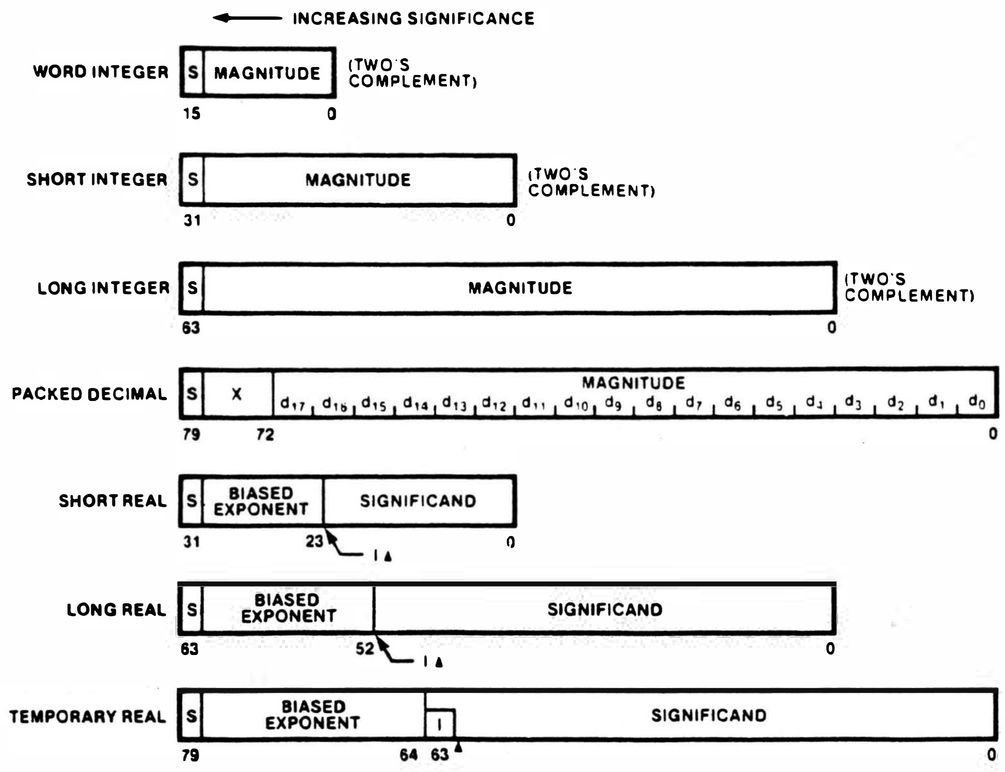
The internal format adopts a number system which extends the range to about $\pm 3.4 \times 10^{-4932}$ to $\pm 1.2 \times 10^{4932}$. This format is used only for constants and intermediate results internal to the Coprocessor. Thus, the Math Coprocessor can accommodate 18 digit numbers (decimal equivalent) for data and final results and 19 digit numbers for constants and intermediate results.

Figure 3.6 shows the several data formats of the Math Coprocessor. Seven numeric data types are recognized by the Coprocessor. These data types are divided into three classes.

1) Binary integers
2) Packed decimal integers and
3) Binary reals.

As indicated in Figure 3.6. the first three formats are binary integer types, which are:

i) Word integer - 16 bits
ii) Short integer - 32 bits
iii) Long integer - 64 bits.



NOTES:
S = Sign bit (0 = positive, 1 = negative)
d_n = Decimal digit (two per byte)
X = Bits have no significance; 8087 ignores when loading, zeros when storing.
▲ = Position of implicit binary point
I = Integer bit of significand; stored in temporary real, implicit in short and long real
Exponent Bias (normalized values):
Short Real: 127 (7FH)
Long Real: 1023 (3FFH)
Temporary Real: 16383 (3FFFH)

Figure 3.6 Data Formats

The most significant bit of all the types is a sign bit. Packed decimal format uses 73 bits to represent an 18 digit decimal number. There are three types of real formats:

i) Short real

ii) Long real

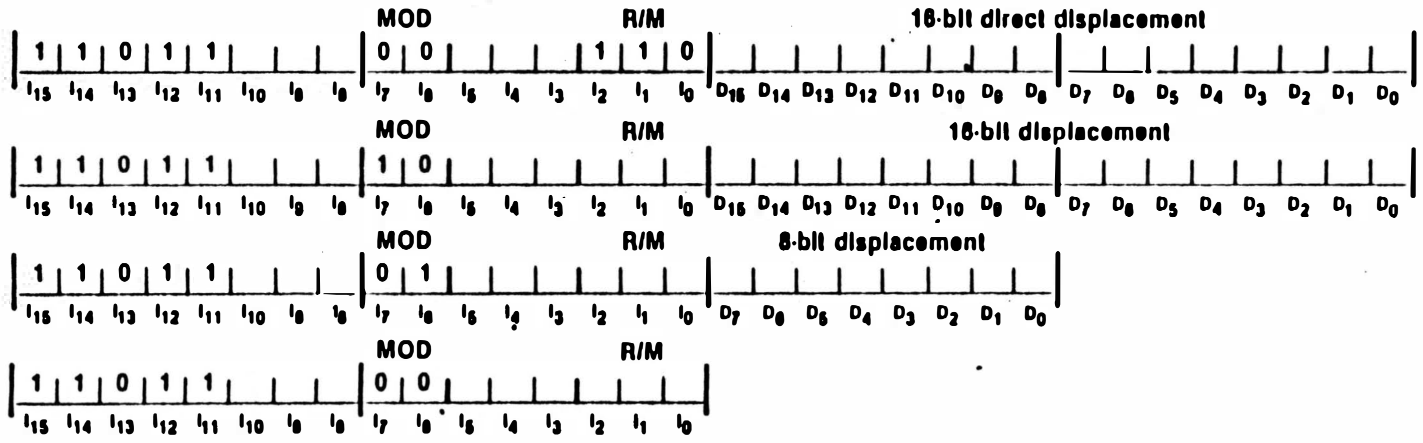
iii) Temporary real.

Only the first two are available for data and final results representation. The temporary real format is used in internal operations only, for increasing the accuracy of the intermediate results.

D. Instruction Decoding and Instruction Set.

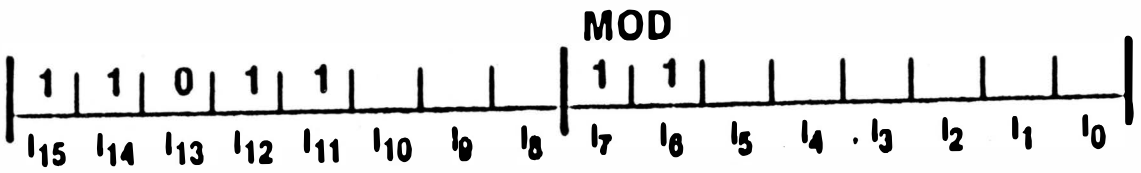
The instruction set of the 8087 Coprocessor is made up of 69 different instructions. These instructions are intermixed with the CPU instructions and appear as ESCAPE instructions to the CPU. Hence all the Coprocessor instructions will contain an Escape code that forms the five most significant bits. A few examples of the Coprocessor instructions starting with the Escape code, 11011, are shown on Figure 3.7.

As mentioned earlier, both the CPU and the Coprocessor fetch and decode an instruction simultaneously. However, there lies a difference in the method of execution. The



MOD
R/M
16-bit direct displacement
16-bit displacement
8-bit displacement
1 1 0 1 1
0 0 1 1 0
1 0
0 1
0 0
I_{15} I_{14} I_{13} I_{12} I_{11} I_{10} I_9 I_8 I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0
D_{15} D_{14} D_{13} D_{12} D_{11} D_{10} D_9 D_8 D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

(a)



MOD
1 1 0 1 1
1 1
I_{15} I_{14} I_{13} I_{12} I_{11} I_{10} I_9 I_8 I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0

(b)

Figure 3.7 (a) Non-memory Reference and
(b) Memory Reference Escape Instruction Forms

Control Unit of the Coprocessor will ignore all the CPU instructions or the instructions that do not match the Escape code. On the other hand, the instructions containing the Escape code will be neglected by the CPU.

If the Coprocessor requires loading or storing of an operand, i.e., a memory access, then the CPU aids the Coprocessor by initiating a dummy read cycle. The requirement of a dummy read cycle will be indicated to the CPU through the MOD bits (Figure 3.7) of the Coprocessor ESCAPE instruction. The Control Unit of the Coprocessor will capture and save the address placed on the bus by the CPU dummy read bus cycle. In addition, for a load instruction the Control Unit also captures the data word when it becomes available on the data bus. The CPU ignores the data obtained from the dummy Read cycle. The Read cycle might also result in the Coprocessor becoming the bus master. The Coprocessor can become the bus master by requesting the bus from the CPU, using the request/grant protocol during two occasions:

1) the data to be read is longer than one word

2) it is ready to perform store operation.

Moving a step above the machine language in Assembly language level, all the Coprocessor instructions are preceded by an alphabet \`F\` (Figure 3-2) indicating an ESCAPE instruction to the assembler. The Assembly Language

Addition	
FADD	Add real
FADDP	Add real and pop
FIADD	Integer add
Subtraction	
FSUB	Subtract real
FSUBP	Subtract real and pop
FISUB	Integer subtract
FSUBR	Subtract real reversed
FSUBRP	Subtract real reversed and pop
FISUBR	Integer subtract reversed
Multiplication	
FMUL	Multiply real
FMULP	Multiply real and pop
FIMUL	Integer multiply
Division	
FDIV	Divide real
FDIVP	Divide real and pop
FIDIV	Integer divide
FDIVR	Divide real reversed
FDIVRP	Divide real reversed and pop
FIDIVR	Integer divide reversed
Other Operations	
FSQRT	Square root
FSCALE	Scale
FPREM	Partial remainder
FRNDINT	Round to integer
FXTRACT	Extract exponent and significand
FABS	Absolute value
FCHS	Change sign

(a) Arithmetic Instructions

FINIT/FNINIT	Initialize processor
FDISI/FNDISI	Disable interrupts
FENI/FNENI	Enable interrupts
FLDCW	Load control word
FSTCW/FNSTCW	Store control word
FSTSW/FNSTSW	Store status word
FCLEX/FNCLEX	Clear exceptions
FSTENV/FNSTENV	Store environment
FLDENV	Load environment
FSAVE/FNSAVE	Save state
FRSTOR	Restore state
FINCSTP	Increment stack pointer
FDECSTP	Decrement stack pointer
FFREE	Free register
FNOP	No operation
FWAIT	CPU wait

(c) Processor Control Instructions

FLDZ	Load $+0.0$
FLD1	Load $+1.0$
FLDPI	Load π
FLDL2T	Load $\log_2 10$
FLDL2E	Load $\log_2 e$
FLDLG2	Load $\log_{10} 2$
FLDLN2	Load $\log_e 2$

(b) Constant Instructions

Figure 3.8 8087 Coprocessor Instructions instruction set of the 8087 Coprocessor is divided into six functional groups:

1) Data Transfer
2) Arithmetic
3) Comparison
4) Transcendental
5) Constants, and
6) Processor Control.

Instructions falling into some of these categories are shown on Figure 3.8. The figure does not indicate the operands required for these instructions. Typically a Coprocessor instruction contains one or two operands as inputs to operate on and produces a result as an output. Operands of some of the instructions are implied and hence need not be specified. The implicit operand is the top stack element. Lastly, some instruction allows the coding of their operand in more than one way. For example, a multiply instruction may be written in any of the following ways:

```
FMUL
FMUL source
FMUL destination, source.
```

CHAPTER IV

SYSTEM CLOCK AND BUS CYCLES

A. 8284 Clock Generator/Driver.

The Vector Controller derives its clock from the 8284 clock generator and driver. The same clock generator provides the clock signal for the 8088 microprocessor and 8087 Coprocessor that serves as the system clock. The 8284 Clock Generator/Driver block diagram and the clock waveform with the timing requirements for the 8088 microprocessor are shown in Figures 4.1 and 4.2, respectively.

The inputs like AEN. RES, RDY1, etc. are provided for hardware reset interface and for insertion of Wait states in the bus cycle. A clock crystal that can oscillate at three times the CPU frequency must be connected between X1 and X2 inputs. The clock signal with fast rise and fall times of 10ns maximum, is required for the 8088 CPU.

The IBM PC runs at 4.77 MHz and hence the Vector Controller, which becomes an integral part of the IBM-PC is run at the same clock speed. Figure 4.3 shows CPU and Coprocessor sharing a common 8284. The Coprocessors of the Vector Processor, also share the same 8284 generated clock and hence the clock output should be buffered for proper

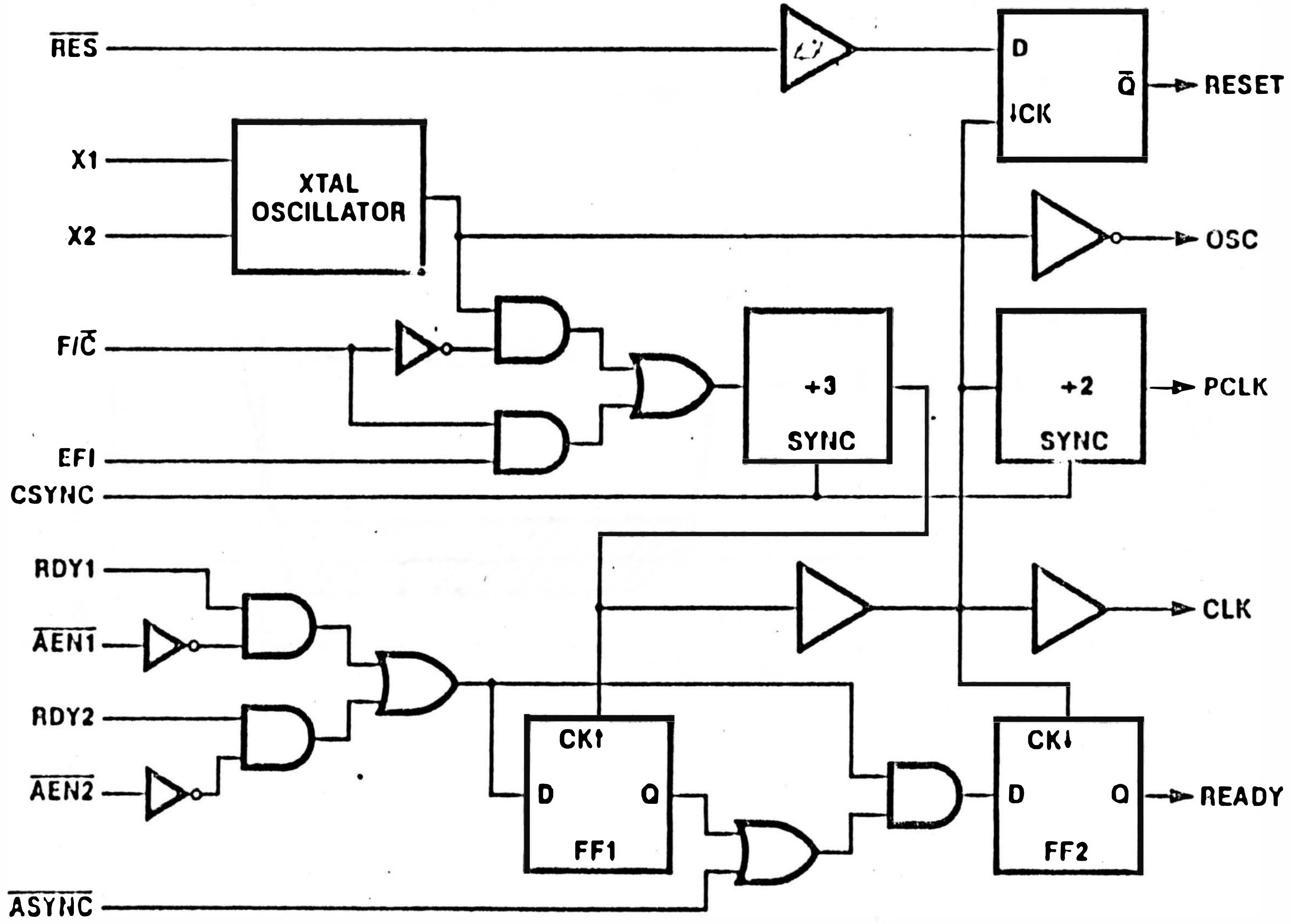


Figure 4.1 8284 Clock Generator/Driver Functional Diagram

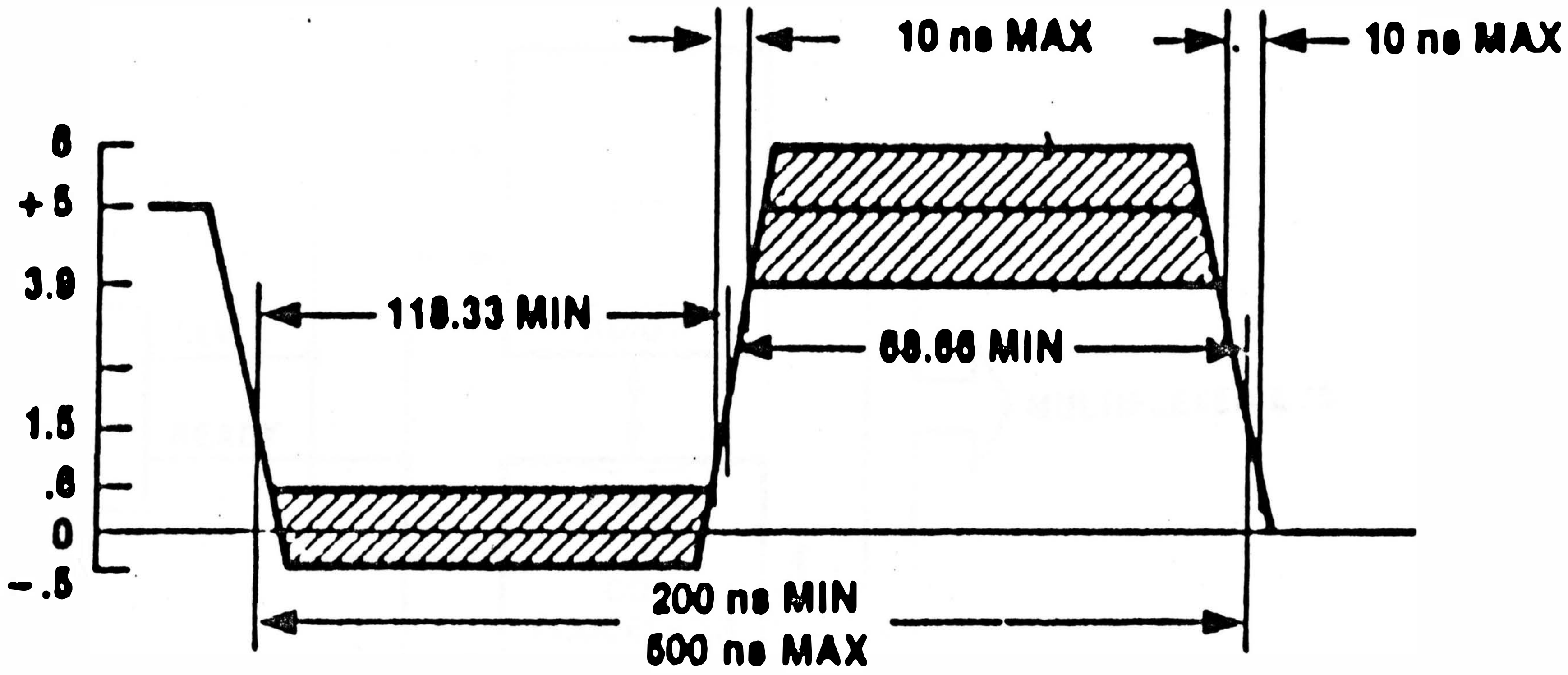


Figure 4.2 8088 Clock Waveform

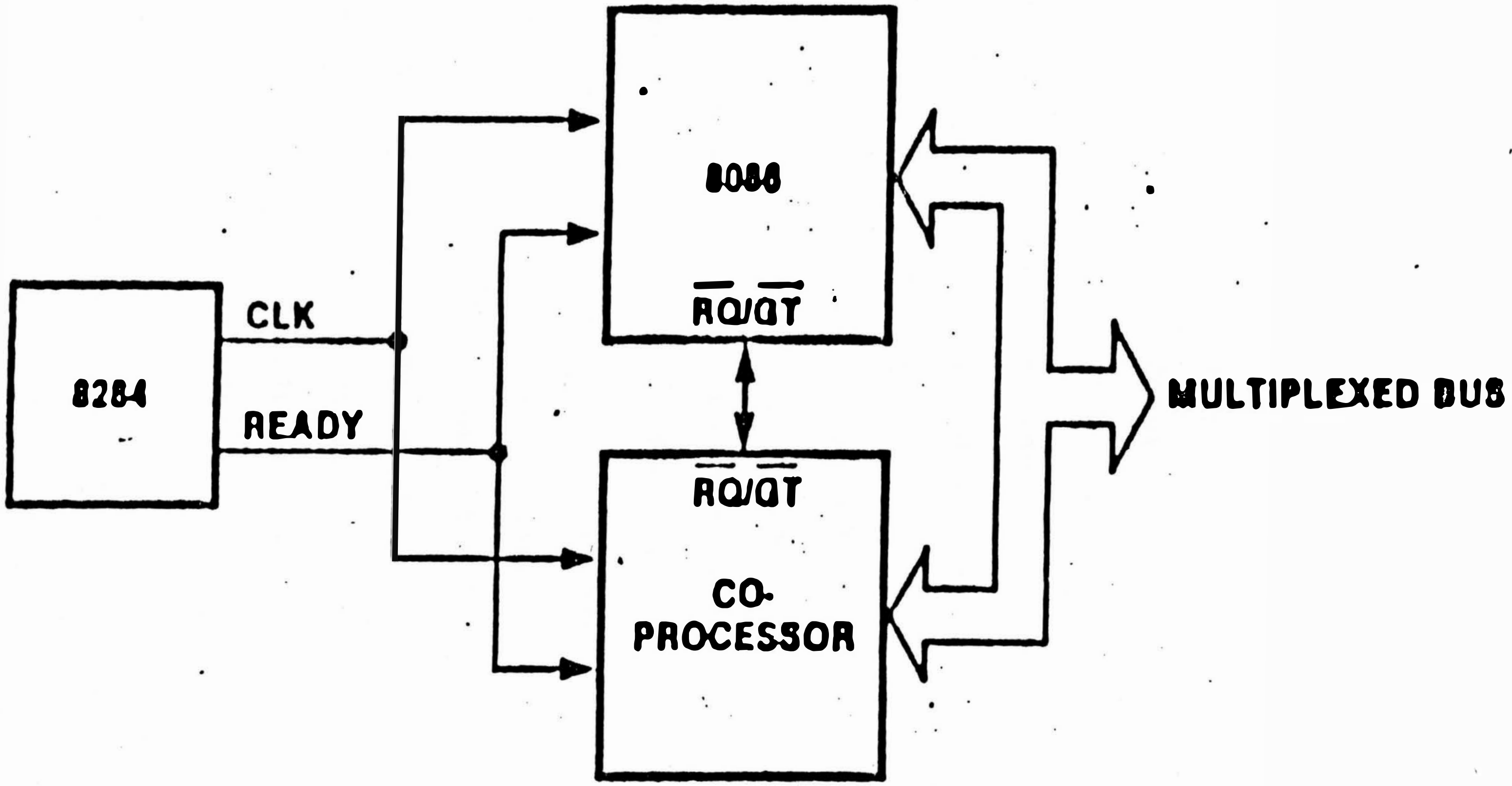


Figure 4.3 8088 and Coprocessor on the Local Bus share a Common 8284

drive current. Due to the fast transition and high drive of the 8284 clock output, it is necessary to put a 100 ohm resistor in series with the clock.

B. T-Clock States.

A detailed discussion of the bus cycle, Status lines, Queue Status lines and Ready line are presented in this section in order to enable the reader to understand the following chapters. A knowledge of these signals will be assumed in the succeeding chapters.

A bus cycle, consisting of 4 or more clocks, is a cycle initiated by the bus controller 8288, using the Status lines S0, S1, and S2 of the CPU. A bus cycle starts with clock designated T1 and ends with T4. From T1 to T4 there will be a minimum of four or more clocks. The four clock states are called T1, T2, T3 and T4. If there are more than four clocks, they are termed Wait states. Figure 4.4 shows two bus cycles; one with four clocks between T1 and T4 and a second one with five clocks. The clock between T3 and T4 of the second bus cycle is the Wait state clock. During T1, the address is put on the address/data bus and is indicated through the address latch enable (see ALE of Figure 4.4) pulse to the external circuitry. Remember that the 8088 CPU and the Coprocessor have a time-multiplexed address/data bus. Which means that both

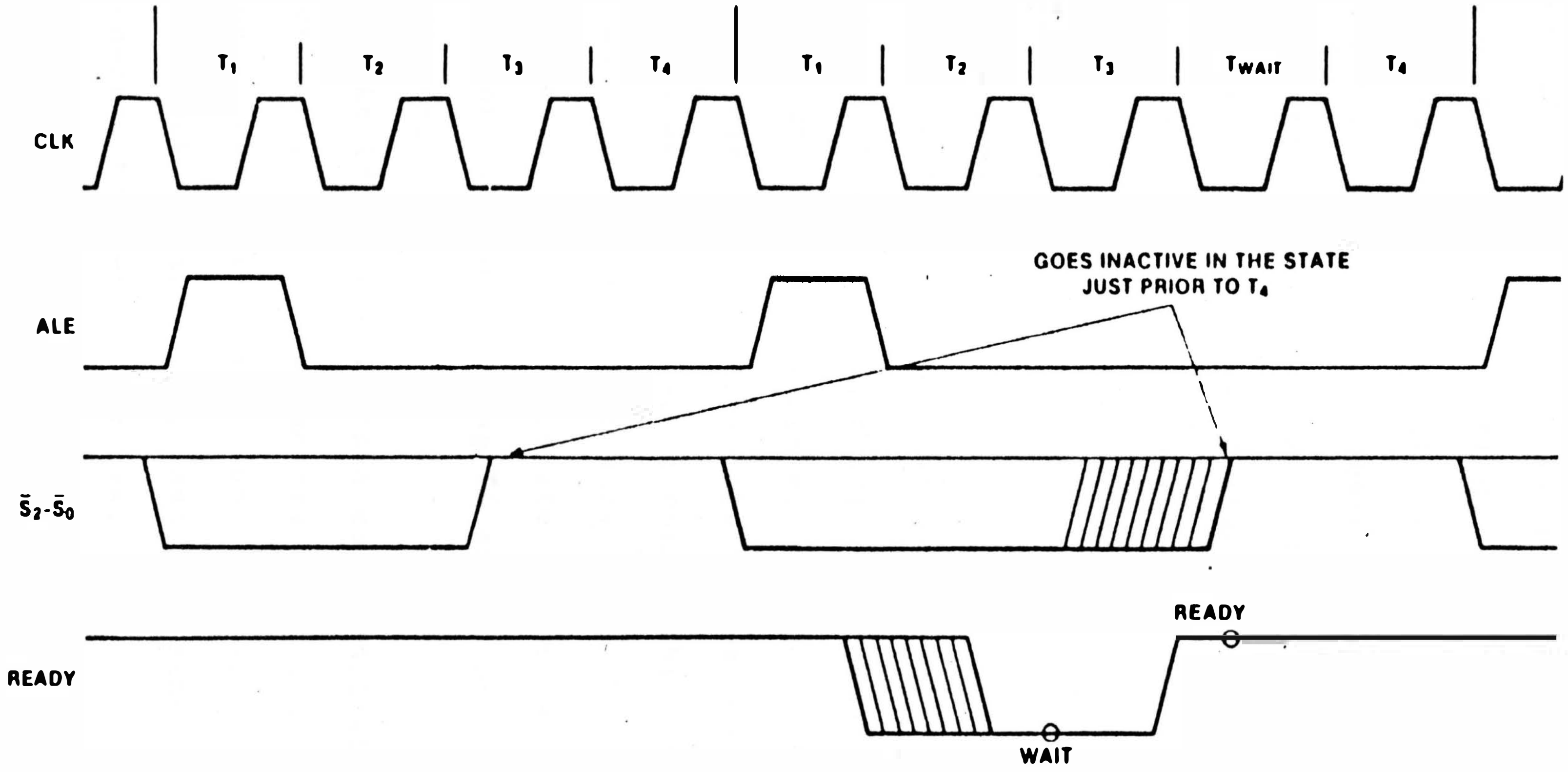


Figure 4.4 Status Line Activation and Termination

the address and the data are presented on the same bus separated by a time period.

During the T2 clock the address is removed and the bus floats. For a write command the data will be put on the bus during this T2 clock. During T3 or the clock before T4 (if there are any wait-states) the data is put on the address/data bus. Thus, the address and data are time-multiplexed by one or more clock cycles.

C. Status Lines.

The Status lines tells the 8288 controller when to start a bus cycle, what type of command to issue and when to terminate the bus cycle. To indicate the beginning of a bus cycle the CPU drives the Status lines from the passive state, S0, S1, S2 = 0, to one of the seven possible command codes as shown on Figure 4.5. The Status lines, as indicated in Figure 4.4 are driven active on the rising edge of the clock during T4 of the previous bus cycle or an idle cycle.

An idle cycle T1 is a clock indicating no current bus activity. Idle clock occurs if no bus cycles are required and if the instruction Queue is full. This idle state is one clock period long and any number of them can be inserted between bus cycles.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	TYPES OF BUS CYCLE
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	HALT
1	0	0	Instruction Fetch
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive; no bus cycle

Figure 4.5 Status Codes

The Status becomes inactive just before T4 (see Figure 4.4) calling for the termination of the bus cycle. During T4 the current bus cycle ends and the Status for the next bus cycle will be set.

The Status lines are used to decode the T-states, in the design of the Vector Controller. The Vector Controller decodes two of these T-clocks. They are T2 clock and the clock just prior to the T4 clock, which might be either T3 or a Wait state clock. The T2 is for the synchronization of the Control of Ready lines to the 8087 Coprocessors. The clock prior to the T4, tells the Vector Controller of the presence of data on the bus during vector instruction decoding.

The Status lines connected from the 8088 CPU to the 8087 Coprocessor are bidirectional signals. Under normal conditions, when the CPU is in control of the bus, the Status codes are established by the CPU and decoded by the 8288 bus controller and the 8087 Coprocessor. But, during a write bus cycle or a long read cycle, when the Coprocessor is the bus master, the Status codes are put out by the Coprocessor and decoded by the 8288 bus controller.

D. Read, Write and Instruction Fetch Cycles.

Read, write and instruction fetch bus cycles are three out of seven possible bus cycles. The seven bus cycles are:

1. Read memory
2. Read input/output
3. Instruction fetch
4. Write memory
5. Write input/output
6. Interrupt acknowledge, and
7. Halt.

The timing diagram of the first five bus cycles are pretty much the same except for one or two signal variations. The timing for memory read bus cycle is shown in Figure 4.6. M/IO signal differentiates between the memory read and read I/O bus cycles. With no other changes in the signal timing. The bus cycle for the instruction fetch is similar to that of the read memory bus cycle. The two cycles are differentiated through the Status codes shown on Figure 4.5.

The write bus cycle timing, shown in Figure 4.7 is similar to the read cycle in Figure 4.6. The difference lies in the data transmit/receive (DT/R) signal, which is switched to logic 1 for a write cycle. Logic 1 on the DT/R line signals the memory or I/O that the data is going to be

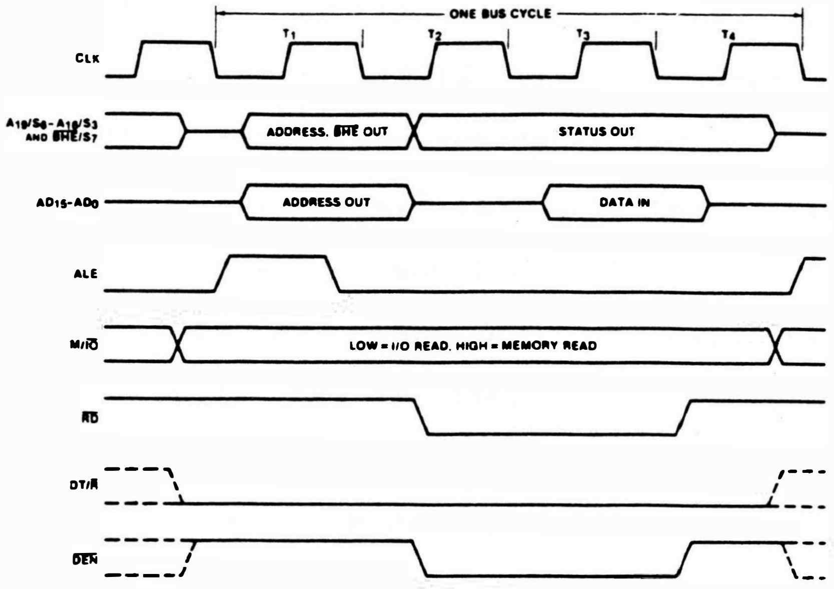


Figure 4.6 Memory Read Bus Cycle

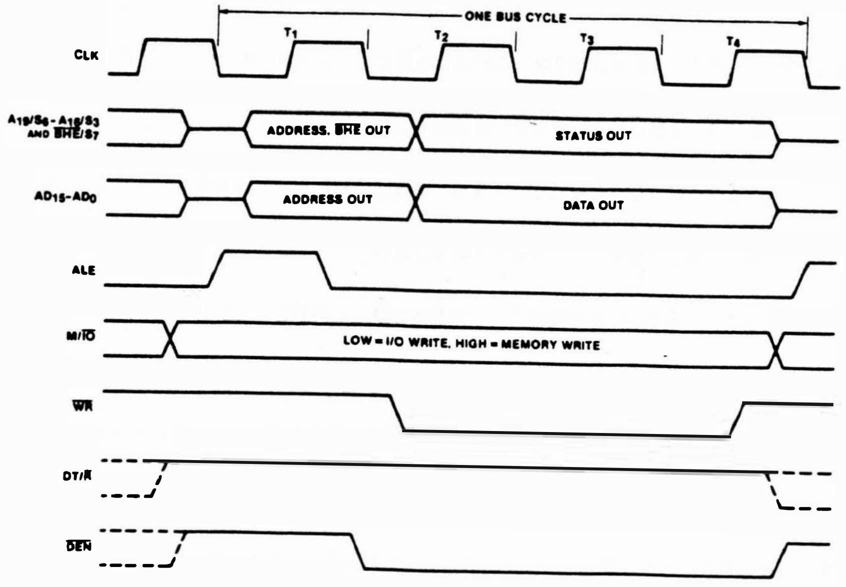


Figure 4.7 Memory Write Bus Cycle

transferred from the processor over the bus. On the other hand, a logic 0 on the DT/R line is a request by the processor for the data from memory or I/O.

The Figures 4.6 and 4.7 show only four cycles for one bus cycle. The length of the bus cycle can be increased by inserting Wait Status. Any number of Wait Status can be inserted between T3 and T4 clocks. The Wait states are required during several events, for instance slow memory. A bus cycle may be extended by switching READY input to logic 0, upon which Wait states are added between periods T3 and T4. The processor remains in the Wait state until READY is returned back to logic 1.

E. <u>Ready and Queue Status Lines.</u>

In the IBM Personal Computer system, the Ready inputs for the 8088 microprocessor and the 8087 Coprocessor are supplied by the Ready output of the 8284 clock generator circuit. The Vector Controller deviates from this connection, instead the Ready input of the 8087 Math Coprocessors are controlled by a combinatorial logic circuit synchronized with the system clock.

The Vector Controller uses the Ready input to force the Coprocessors into Wait state. For this the Ready signal must be inactive (low) by the end of T2. To activate the Coprocessor the Ready must switch to logic 1 within a specified setup time prior to the positive transition during T3. Hence, the Ready implementation takes two approaches in the design of Vector Controller:

1) normally Ready system (the CPU and the first 8087 Coprocessor),

2) normally not Ready system (the remaining nine 8087 Coprocessors)

Figure 4.8 (a) and (b), shows the setup and hold time requirements of both the above mentioned systems.

The Ready input must be disabled within 8 ns after the end of T2 to force the Coprocessor into Wait state (see Figure 4.8(a)). Also as indicated in Figure 4.8(b), to avoid Wait states, Ready must be active 119 ns prior to the positive lock transition during T3. Therefore, to guarantee the insertion of Wait states and the transition of the Coprocessor into active state, the Vector Controller synchronizes the logic required to control the Ready signal with the T2 clock of the bus cycle.

Other signals controlled by the Vector Controller are the Queue Status inputs of the Coprocessors. Recall that the Queue Status, QS0 and QS1 are the outputs of the maximum mode 8088 CPU. QS0 and QS1 together form a 2-bit Status code. This code, emitted by the CPU informs the Coprocessors about the conditions of the instruction Queue as shown in Figure 4.9.

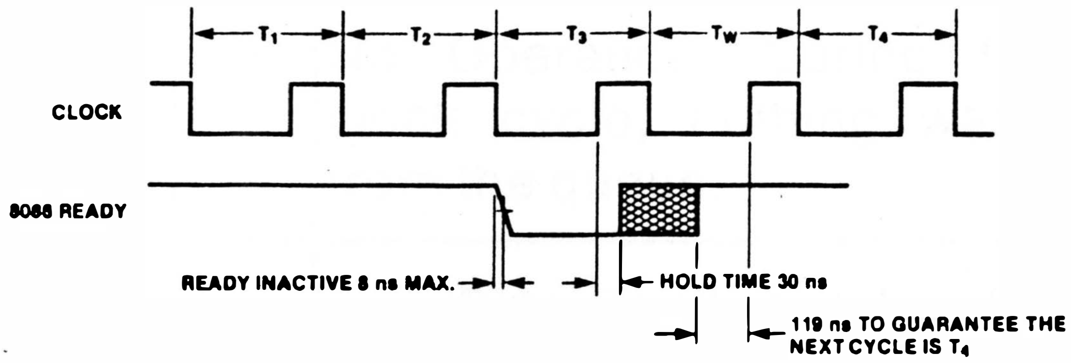


Figure 4.8 (a) Normally Ready System Inserting a Wait State

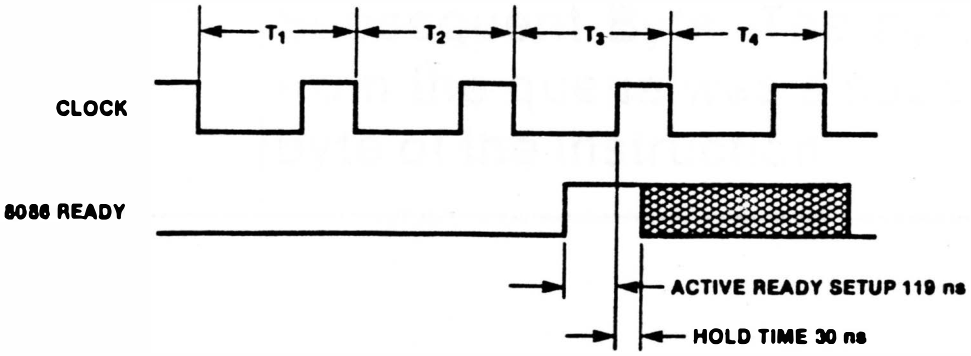


Figure 4.8 (b) Normally Not Ready System

QS1	QS0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

Figure 4.9 Queue Status Codes

An instruction may be one or more bytes long. The Queue Status lines indicate the first byte of an instruction, removed from the Queue, through the code QS0 QS1 = 10. The subsequent bytes of that instruction are coded as 11. The code QS0 QS1 = 01 indicates that the Queue is emptied. This code is output on decoding of an instruction resulting in a fresh instruction fetch starting from a new location: (example: a JUMP instruction). The reinitialization code (empty Queue) is used to recognize vector instructions. The design of the Vector Controller requires that a vector instruction be preceded by an emptying of the Queue instruction.

Finally, QS0 QS1 = 00 is a no operation code meaning, no current instruction Queue activity. The Vector Controller drives the Queue Status inputs low when the Coprocessors are in Wait state. The active state Coprocessors are allowed to receive the Queue Status information. At one stage, even the active state Coprocessors Queue Status lines are held low. This technique aids in advance storing of an instruction into the operands Queue of the Coprocessors and execution of it at a later time by activating the Queue Status lines.

The transition of the Queue Status lines to low state and back to active state are synchronized with the instruction Queue activities by the Vector Controller.

CHAPTER V

VECTOR CONTROLLER

The Vector Controller (VC) is a circrit designed to control the activities of the Vector Processor in vector mode. It provides the necessary interfacing signals for the IBM PC. The Vector Controller also coordinates the activities of the Coprocessors with that of the 8088 CPU. The block diagram of the Vector Controller is as shown in Figure 5.1. The three main functional units forming the Vector Controller are:

1) Vector Instructions Decoder

2) Sequential LOAD/STORE Control, and

3) Parallel Execution Control.

Hardware implementations of each of the above three functions are discussed in detail in the subsequent chapters.

The Data lines, Status lines and Queue status lines of the 8088 CPU serve as inputs to the VC. The output of the VC controls the Ready and the Queue status lines of all the 8087 Math Coprocessors.

The Vector Controller permits a programmer to enter a vector mode or return to scalar mode through vector instuctions. The vector instructions are intermixed with

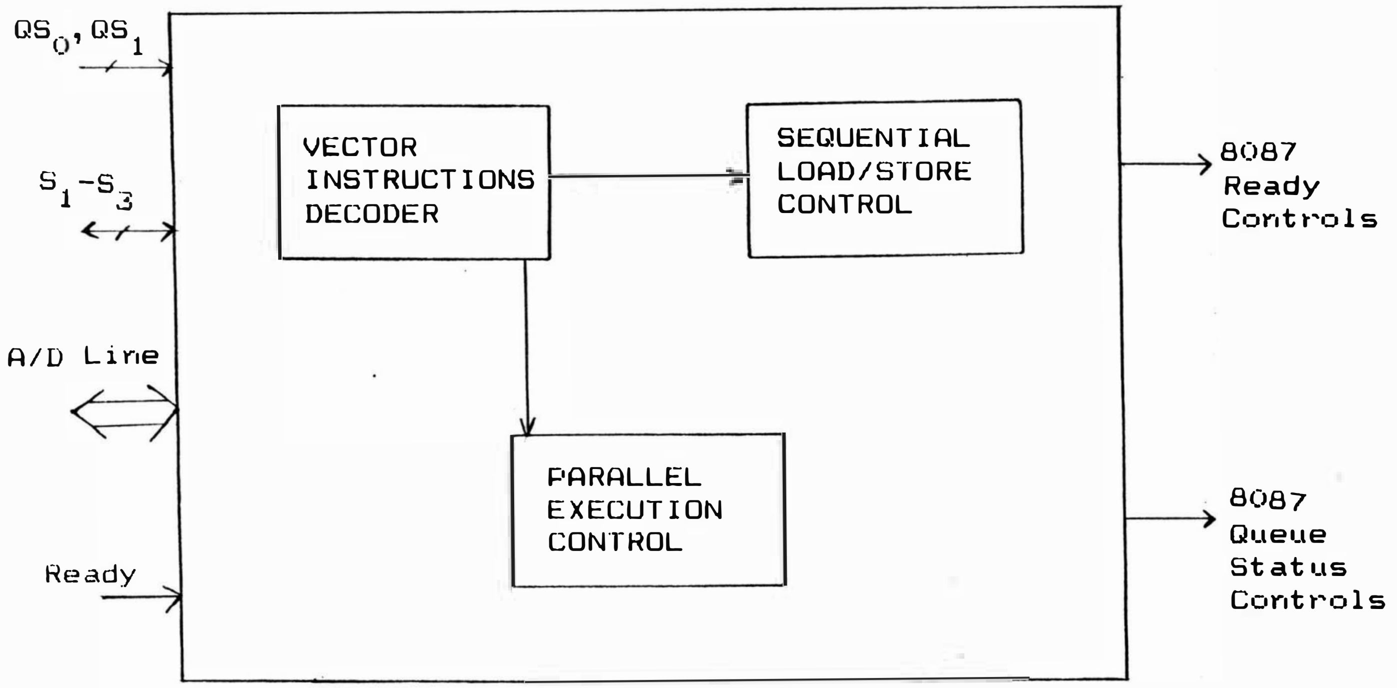

QS_0, QS_1
$S_1 - S_3$
A/D Line
Ready
VECTOR INSTRUCTIONS DECODER
SEQUENTIAL LOAD/STORE CONTROL
PARALLEL EXECUTION CONTROL
8087 Ready Controls
8087 Queue Status Controls

Figure 5.1 Vector Controller

8088 CPU and 8087 Coprocessor instructions. The VC ignores CPU and Coprocessor instructions, decodes vector instructions and takes necessary actions depending on the type of instruction. The Vector Processor has two basic modes of operation:

1) the scalar mode, and

2) the vector mode.

A. <u>The Scalar Mode.</u>

In the scalar mode of operation the Vector Controller deactivates all the 8087 Math Coprocessors, except the first one. The first 8087 Coprocessor operates in parallel with the 8088 CPU. This is also the power up condition. That is, when the power supply is turned on for the IBM PC the presence of the Vector Controller will not be felt. Hence, the scalar mode is also the normal mode of operation without the intervention of the Vector Controller.

In the scalar mode of operation, the Vector Controller pulls low the Ready lines of all the Coprocessors except the first one. This forces the Coprocessors to enter the Wait state. Queue status lines QS0 and QS1, are held at logic 0. Recall the QS0 QS1 = 00 is a no operation indication to the Coprocessors. In the Wait state, status codes are ignored by the Coprocessors and hence are not controlled.

The Vector Controller monitors the Queue Status code emitted by the CPU for a reinitialization code. Every time the queue is emptied and a fresh instruction fetched, the VC decodes this instruction. If the instruction is a vector instruction then the processor enters a vector mode, other wise the instruction is neglected. From the above discussion, one can see that a vector instruction should be preceded by an empty-the-queue instruction resulting in reinitialization of the instruction queue.

This approach is the result of asynchronous operation between the CU and the BIU of the 8088 CPU discussed earlier. This makes a designer unable to track the first byte of an instruction while being fetched.

B. The Vector Mode.

Depending on the type of vector instruction, one of the two vector modes can be entered: (1) Serial mode, or 2) Parallel mode.

1. Serial Mode.

In serial mode the Vector Controller activates the 8087 Coprocessors in series. The first 8087 instruction following the serial mode vector instruction is fed to all the Coprocessors in parallel. None of the 8087 Coprocessor instructions are greater than four bytes long, which eliminates the risk of instruction queue overflow.

After loading the instructions in parallel, all but the first Coprocessor are forced into Wait state. Note that, even though this is a serial mode by name, in reality the Coprocessors do enter parallel mode for a short duration during instruction loading. The first 8087 Coprocessor executes the instruction and indicates the end of operation through the Busy line. The Vector Controller then activates the next Coprocessor and forces the first Coprocessor into Wait state. At the end of execution by the second Coprocessor, the third one is activated and the second Coprocessor is pushed into the Wait state. This process continues, with the succeeding Coprocessors being activated and the preceding forced into Wait state until terminated by a vector instruction. On termination of the serial mode, the system returns to scalar mode.

Serial mode is used to LOAD or STORE different numbers into the Coprocessors in series, on which the operations are to be executed in parallel. All the activities in serial mode are controlled by the Sequential LOAD/STORE Control unit.

2. Parallel Mode.

After loading the data to be operated on into the required number of Coprocessors, the parallel mode is entered through a vector instruction. In parallel mode all the Coprocessors are activated simultaneously. Any 8087 Coprocessor instruction, in parallel mode, will be executed by all the Coprocessors at the same time. The time saved or the speed of the Vector Processor is directly proportional to the number of operations executed in parallel.

All the activities in parallel mode are controlled by the Parallel Execution Control Unit. The parallel mode is terminated with a vector instruction that returns the system back to scalar mode. The results of the operations from the Coprocessors are stored into memory using serial mode operation.

C. Vector Instructions.

A vector instruction must be an Escape instruction to the 8088 CPU and should be ignored by the 8087 Coprocessor. Fortunately, there are several of these Escape instructions that are left unused by the 8087 Coprocessor. Figure 5.2(a) shows the available Escape instructions that can be used as vector instructions.

I_{15}	I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
1	1	0	1	1				1	1						

I_{10}	I_9	I_8	I_5	I_4	I_3	I_2	I_1	I_0	Available codes
0	0	1	0	1	0	0	0	1	1
0	0	1	0	1	0	0	1	—	2
0	0	1	0	1	0	1	—	—	4
0	0	1	1	0	0	0	1	—	2
0	0	1	1	0	0	1	1	—	2
0	0	1	1	0	1	1	1	1	1
0	0	1	1	1	0	1	0	1	1
0	0	1	1	1	1	0	1	1	1
0	0	1	1	1	1	1	1	—	2
0	1	1	1	0	0	1	0	1	1
0	1	1	1	0	0	1	1	—	2
0	1	1	1	0	1	—	—	—	8
0	1	1	1	1	—	—	—	—	16
1	0	1	1	—	—	—	—	—	32
1	1	1	1	0	0	0	0	1	1
1	1	1	1	0	0	0	1	0	1
1	1	1	1	0	0	1	—	—	4
1	1	1	1	0	1	—	—	—	8
1	1	1	1	1	—	—	—	—	16
									105 total

Available Non-Memory Reference Escape Instructions

(a)

1	1	1	1	1	—	—	—	—	16

(b)

Figure 5.2 (a) Avilable Escape Instructions
(b) Selected Vector Instructions

Out of the 105 instructions shown on Figure 5.2(a), only 16 of these are chosen to operate as vector instructions and are shown on Figure 5.2(b). The MOD bits of these instructions are 11 indicating a non-memory reference Escape instruction. This means that the CPU does not initiate a dummy read cycle on execution of these instructions. Instead the CPU just neglects these non-memory reference Escape instuctions and proceeds with the following instructions. In effect, these 15 vector instuctions will be ignored by both the 8088 CPU and the 8087 Coprocessor, but must be recognized by the Vector Controller. So then, the first step in the design of the Vector Controller is a circuit to decode these sixteen vector instructions.

CHAPTER VI

VECTOR INSTRUCTIONS DECODER

The Vector Instructions Decoder (VID), as the name explains, is a decoding circuit that recognizes the vector instructions. This circuit monitors the data lines on which the instructions are available and decodes the 16 vector instructions of the Vector Processor.

The Vector Instructions Decoder circuit can be divided into five distinct blocks as shown in Figure 6.1. They are:

(1) Instruction Fetch Bus Cycle Monitor

(2) Data Enable Signal Generator

(3) Data Line Monitor for 'DF'

(4) Clearing and Control, and

(5) Subsequent Byte Decoder.

Whenever the instruction queue is emptied, the Instruction Fetch Bus Cycle Monitor circuit is activated. This allows the Data Enable Signal Generator block to decode the T3 clock or the clock just prior to the T4 clock. The Data Line Monitor for DF block checks the data line to see if the instruction begins with 'DF'- the hexadecimal code. If it does not, then the Clearing and Control block reinitializes the whole VID circuit to the

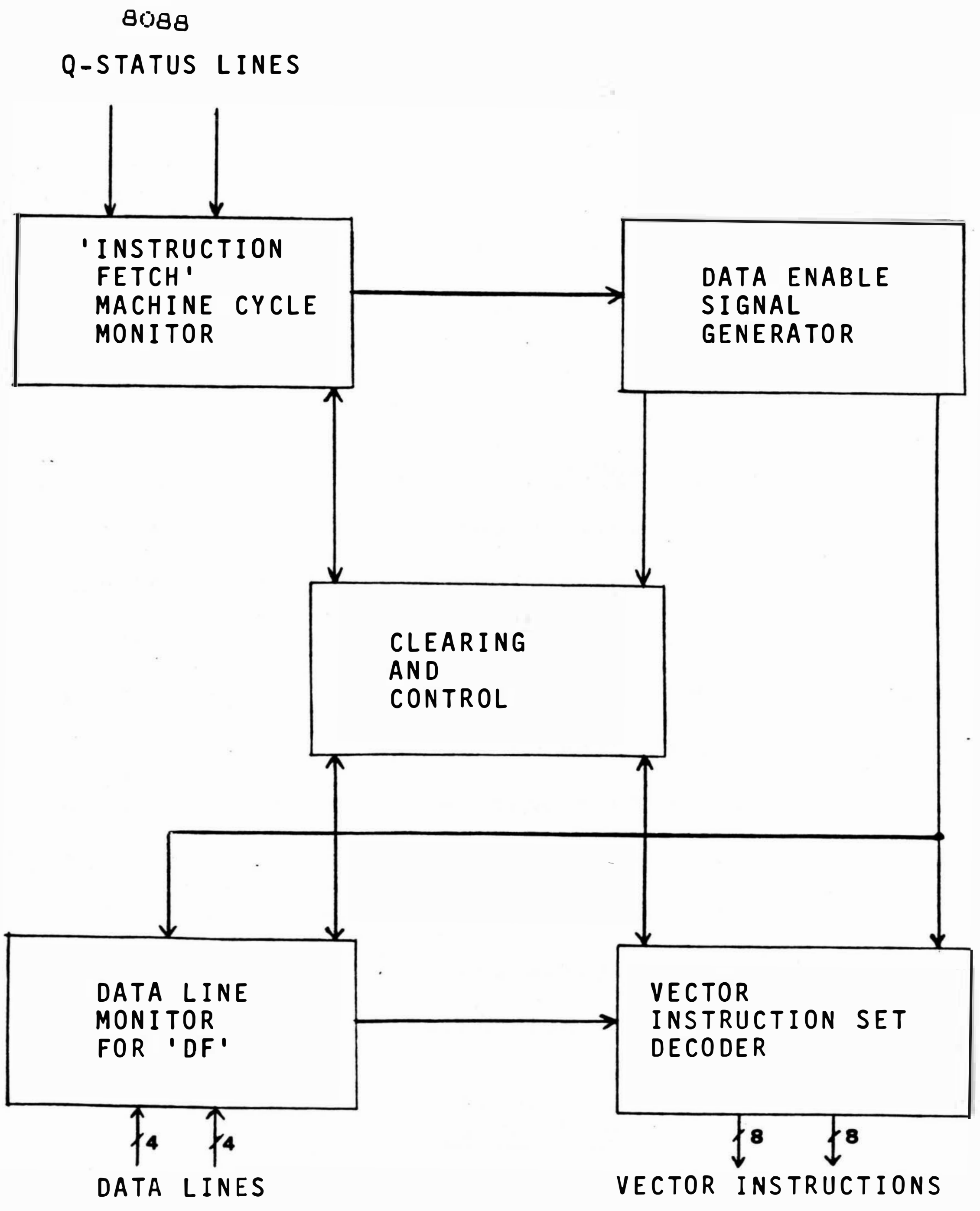

8088
Q-STATUS LINES
'INSTRUCTION FETCH' MACHINE CYCLE MONITOR
DATA ENABLE SIGNAL GENERATOR
CLEARING AND CONTROL
DATA LINE MONITOR FOR 'DF'
VECTOR INSTRUCTION SET DECODER
4
4
DATA LINES
8
8
VECTOR INSTRUCTIONS

Figure 6.1 Vector Instructions Monitor Block Diagram

original state. However, if the instruction begins with DF, the Subsequent Byte Decoder block checks for the following byte and decodes it if it is one of the 16 vector instructions. The original conditions are restored again by the Clearing and Control block, after decoding. The machine code of the 16 vector instructions are shown in Figure 6.2.

A. <u>Instruction Fetch Bus Cycle Monitor.</u>

As has been mentioned earlier, a vector instruction must be preceded by an instruction that reinitializes the instruction queue. For this reason a jump (JMP) instruction is used before any vector instructions. Jumping is done to the very next location containing the vector instruction. The result of a jump instruction is that the queue is emptied and a fresh instruction is fetched which may be the first byte of a vector instruction. The following example shows a sample format containing a vector instruction

```
ADDRESS          ASSEMBLY
LOCATION         INSTRUCTION
0200
0201             JMP NEXT    ; Reinitializes the queue
0202      NEXT   VECTOR-OP   ; a vector instruction
0204             WAIT
0205
0206
```

								MOD			
I15	I14	I13	I12	I11	I10	I9	I8	I7	I6	I5	I4
1	1	0	1	1	1	1	1	1	1	1	1

I3	I2	I1	I0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Figure 6.2 Machine Code for Vector Instructions

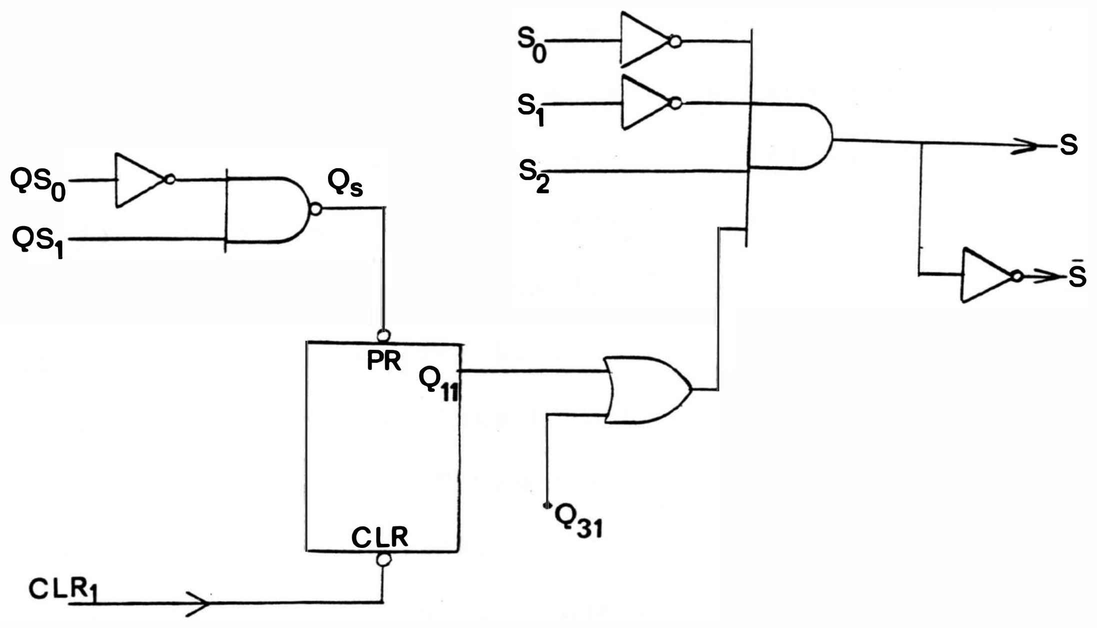


Figure 6.3 Instruction Fetch Bus cycle Monitor

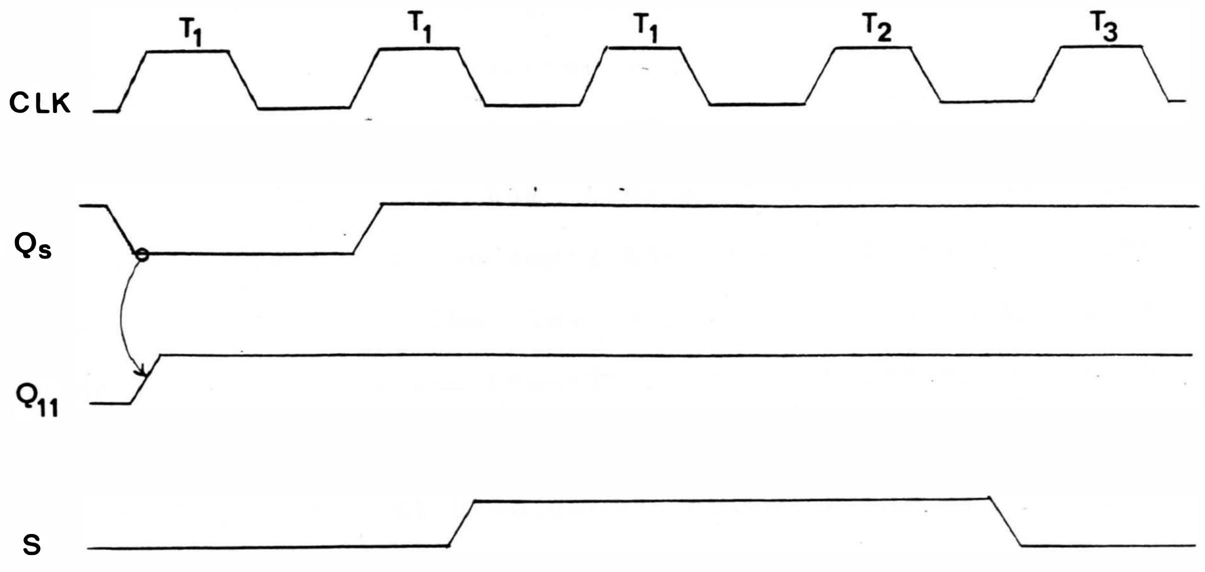


Figure 6.4 Timing Waveform

The address location is pointed to by the instruction pointer and is available on the address/data bus during T1. The instructions are available on the same bus during the clock just prior to T4 of an instruction fetch bus cycle.

The logic circuit for the Instruction Fetch Bus Cycle Monitor is shown in Figure 6.3. The inputs to the circuit shown are Queue Status lines QS0,QS1, status lines S0, S1, and S2 from the 8088 CPU and clear signal (CLR) from the Clearing and Control block.

On reinitialization of the instruction queue, the Queue Status lines trigger the flip flop shown to a high state (Q11 = 1). This enables the monitoring of the status lines for an instruction fetch bus cycle. The outputs S and S serve as inputs to the Clearing and Control, and Data enable generator blocks respectively. The Status signal S transits to a high state during a Read bus cycle.

The timing diagram of the circuit is shown in Figure 6.4. As indicated in the timing diagram, signal Qs, switches to low state on empty the queue signal from the Queue Status lines. The low state of Qs is one clock period long and this low transition of Qs triggers Q11 into high state. Q11 remains at logic 1 until cleared by CLR1.

The Status signal S becomes active at the end of the idle clock state T1 and transits to low state during the clock prior to T4. This activities of the Status signal S, helps in data enable clock generation. The input to OR gate from the Sequential LOAD/STORE Control circuit is for status lines monitoring for that circuit.

B. <u>Data Enable Generator.</u>

The Data enable generator decodes the T3 clock or the clock just prior to the T4 clock. During this clock the data (instruction) is on the address/data bus, enabling the Data monitoring circuit to check for vector instruction.

Figure 6.5 shows the circuit of the Data Enable Generator. Signal S is the input to this circuit from the Instruction Fetch Monitoring block. The output T31 is the decoded clock which serves as input to the Data Monitoring circuit. Another output CL1 is fed to the Clearing and Control block.

The timing waveforms at different points of the circuit are as shown in Figure 6.6. As indicated by the timing waveform, Q21 (D-flip-flop) is switched to logic 1 by a high to low transition of the status S during an instruction fetch bus cycle following the queue reinitialization. The S signal becomes high during the clock just prior to T4 enabling T31 as the output. The data enable clock, T31, triggers a D-flip-flop Q22 to a high state. During the low state of T4, logic 1 on Q22 allows the transition of CL1 into high state. Clearing of

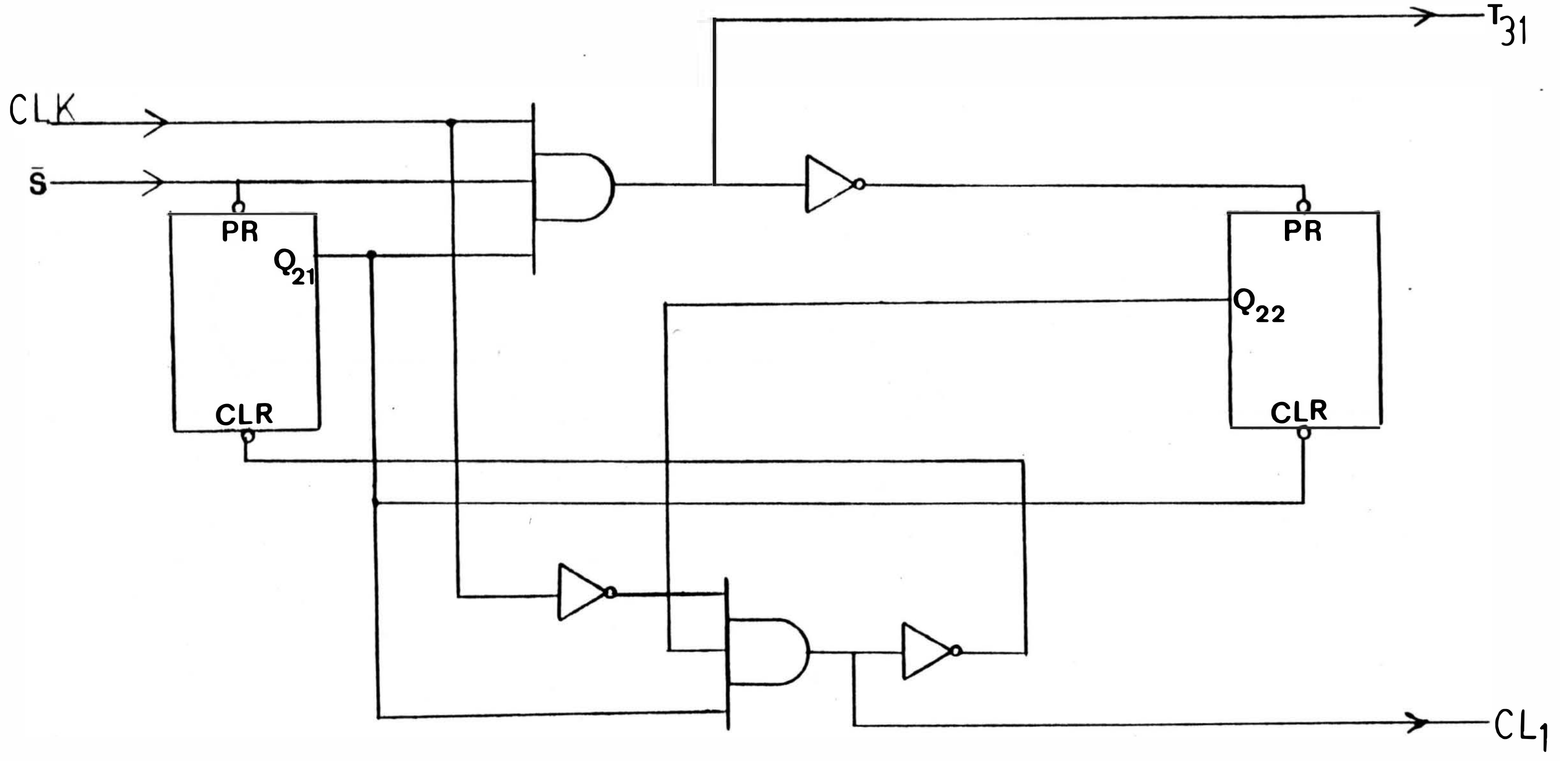


Figure 6.5 Data Enable Generator (DEG)

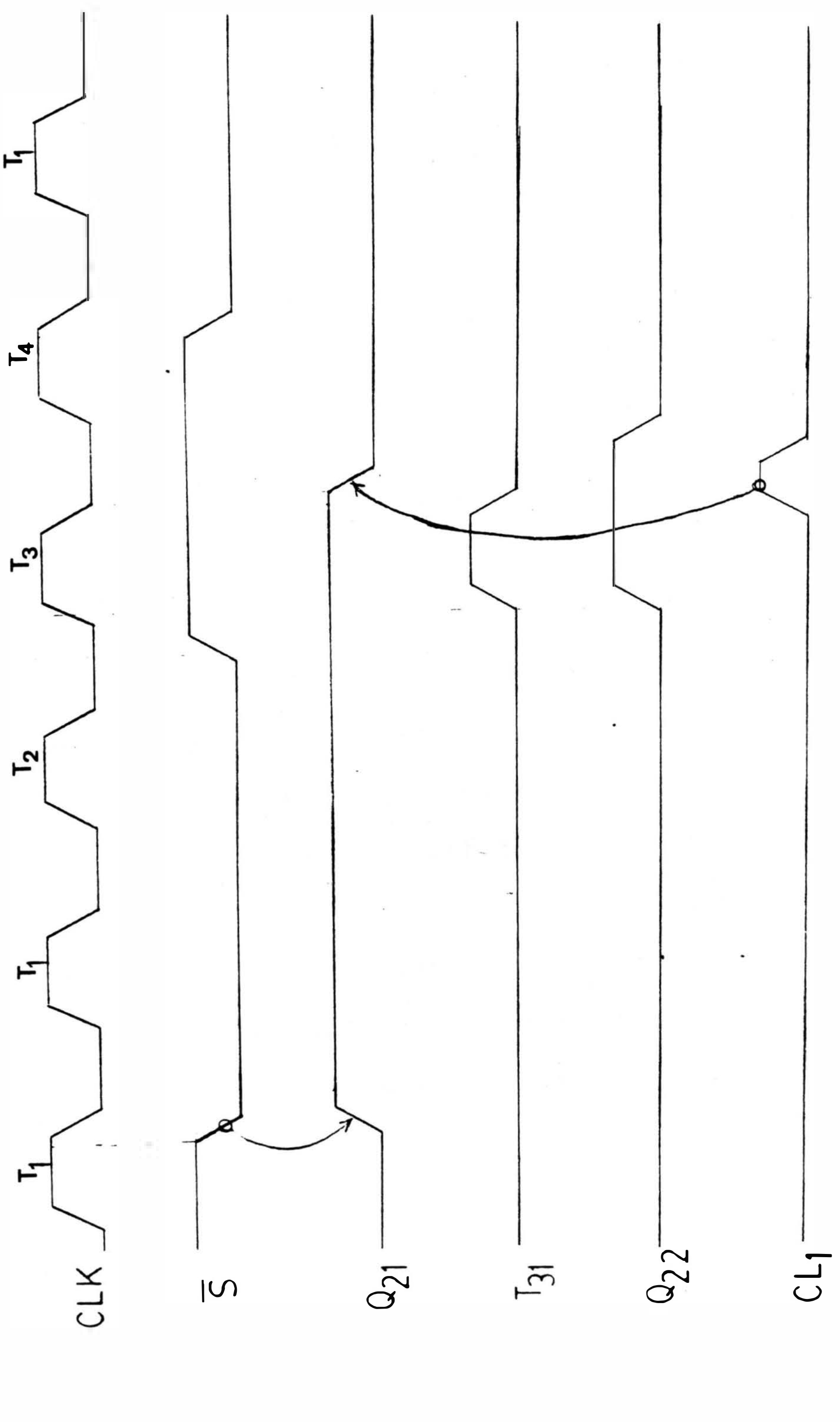


Figure 6.6 DEG Timing Waveforms

Q21 by CL1 forces Q22 and CL1 into low state, thus restoring initial conditions. If the first byte of the decoded instruction is a vector instruction, (indicated by the Data Line Monitor for "DF" block), then T31 of the following bus cycle is decoded. This T31 serves as input to the Subsequent Byte Decoder block.

C. Data Line Monitor for "DF".

As the name indicates, the Data Line Monitor for "DF" block monitors the data lines and checks the first byte of an instruction, following the reinitialization code, for "DF" (DF the hexadecimal code for first byte of a vector instruction). Figure 6.7 gives the hexadecimal codes for the 16 vector instructions. Observe that the first byte of all these instructions start with the hexadecimal code DF. This is why the first byte of the instruction is checked to see if it matches with the code DF.

Figure 6.8 shows the circuit of the Data Line Monitor for "DF" block. The inputs to the circuit are eight address/data lines from the CPU, Q1, and CLR2 from the Clearing and Control block, and T31 clock from Data Enable Generator.

Looking at the timing waveforms in Figure 6.9, the preset input F transits to a low state, if during T31 clock

FIRST BYTE	SUBSEQUENT BYTE
DF	F0
DF	F1
DF	F2
DF	F3
DF	F4
DF	F5
DF	F6
DF	F7
DF	F8
DF	F9
DF	FA
DF	FB
DF	FC
DF	FD
DF	FE
DF	FF

Figure 6.7 Hexadecimal Code for Vector Instructions

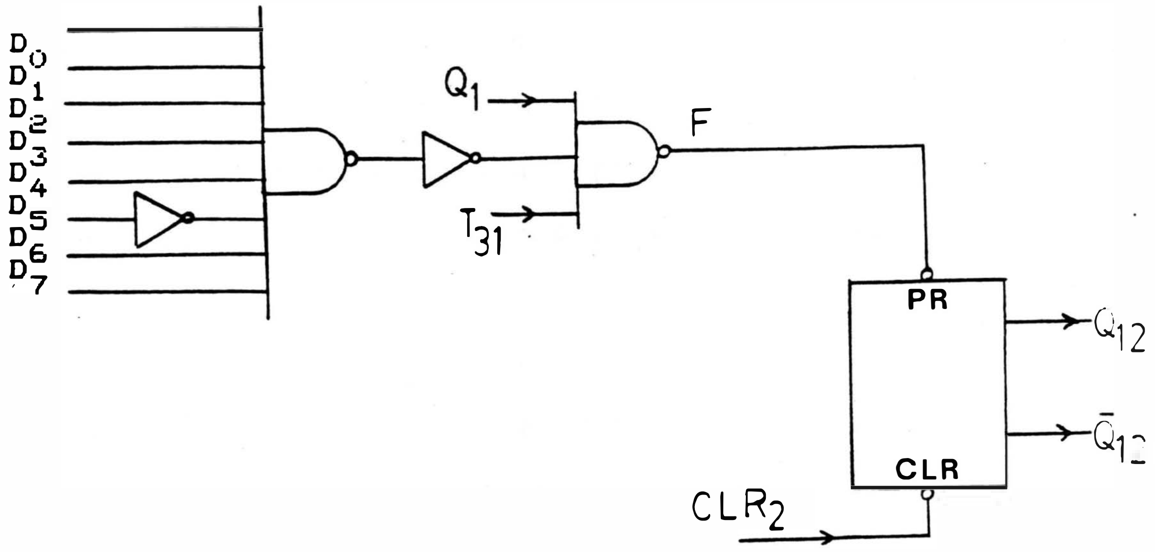


Figure 6.8 Data Line Monitor for "DF"

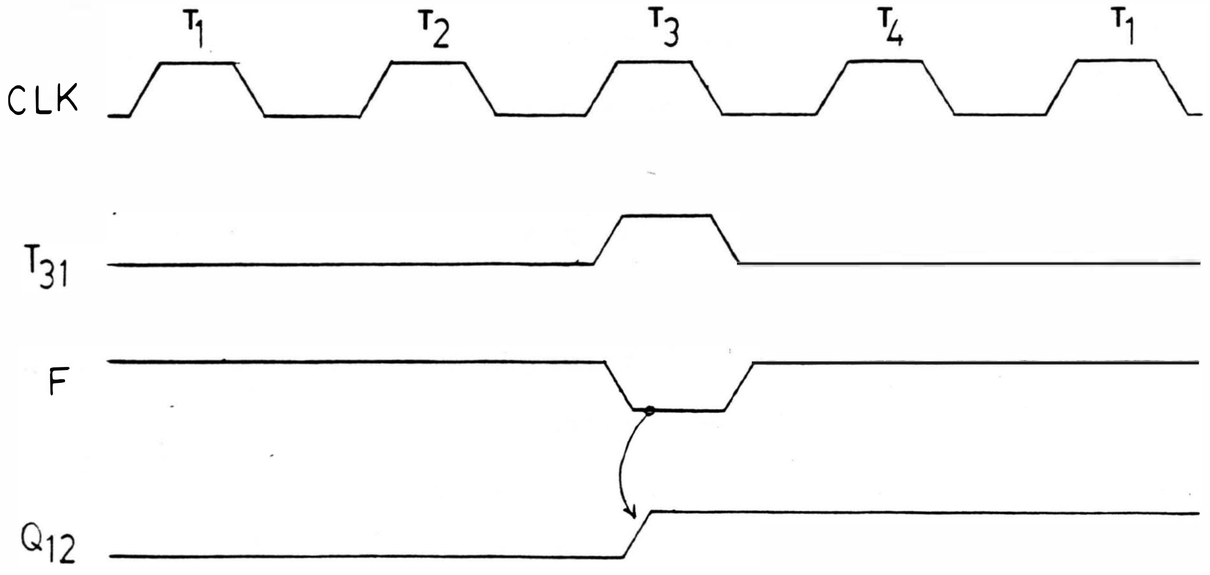


Figure 6.9 Timing Waveforms

the data on address/data line is DF. This low transition of F triggers the output Q12 into high state. The outputs Q12 and Q12 of this data monitoring circuit serve as inputs to the Subsequent Byte Decoder and Clearing and Control blocks respectively.

The logic 1 on Q12 is an indication of the possibility of the instruction being a vector instruction. This can be confirmed only after decoding the subsequent byte. At the end of decoding of the subsequent byte the CLR2 input from the Clearing and Control clears the flip-flop, forcing Q12 into low state.

D. Clearing and Control.

Clearing and Control provides the clear inputs required by the other blocks and controls the overall operations of the unit. The logic circuit needed is shown in Figure 6.10. The inputs to the circuit, as shown in the Figure, are status S and CL1 inputs from Data Enable Generator, and the Q12 signal from Data Line Monitor for "DF" block. A two bit counter counts the number of bus cycles and aids in generation of the CLR1 and the CLR2 outputs. These outputs clear the flip-flops of the other blocks at appropriate times depending on whether or not a vector instruction is found.

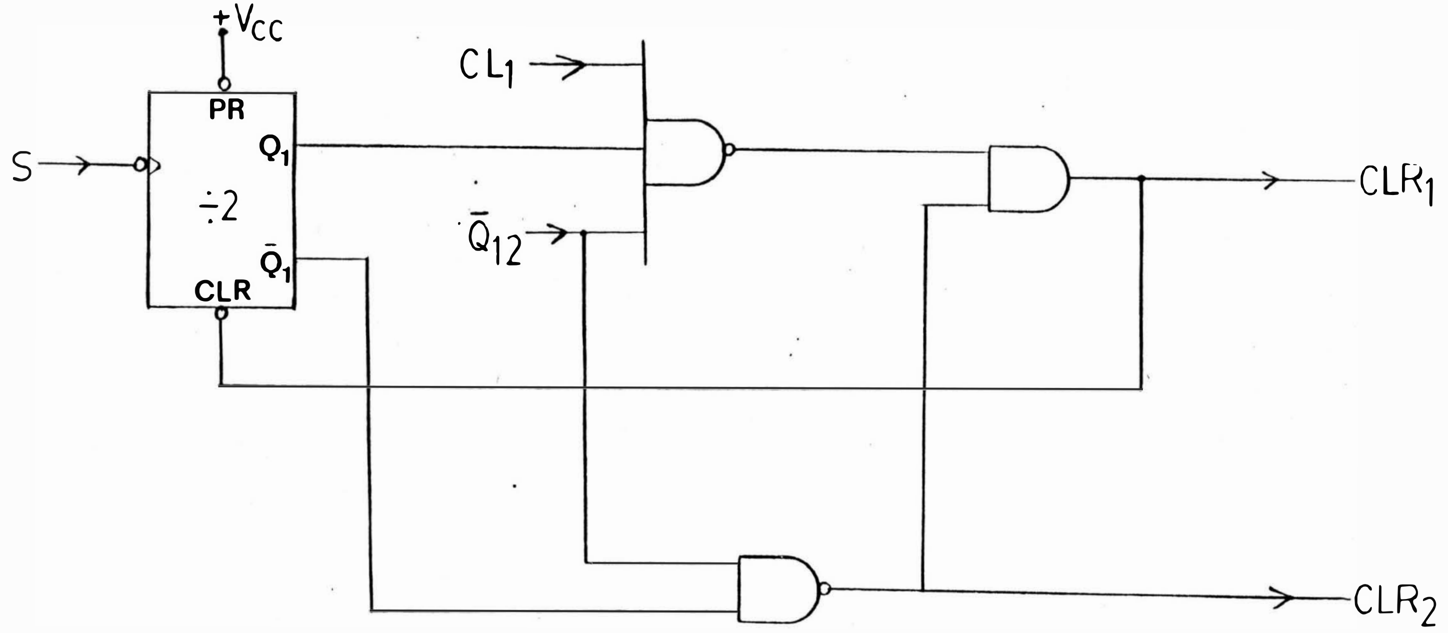


Figure 6.10 Clearing and Control Circuit

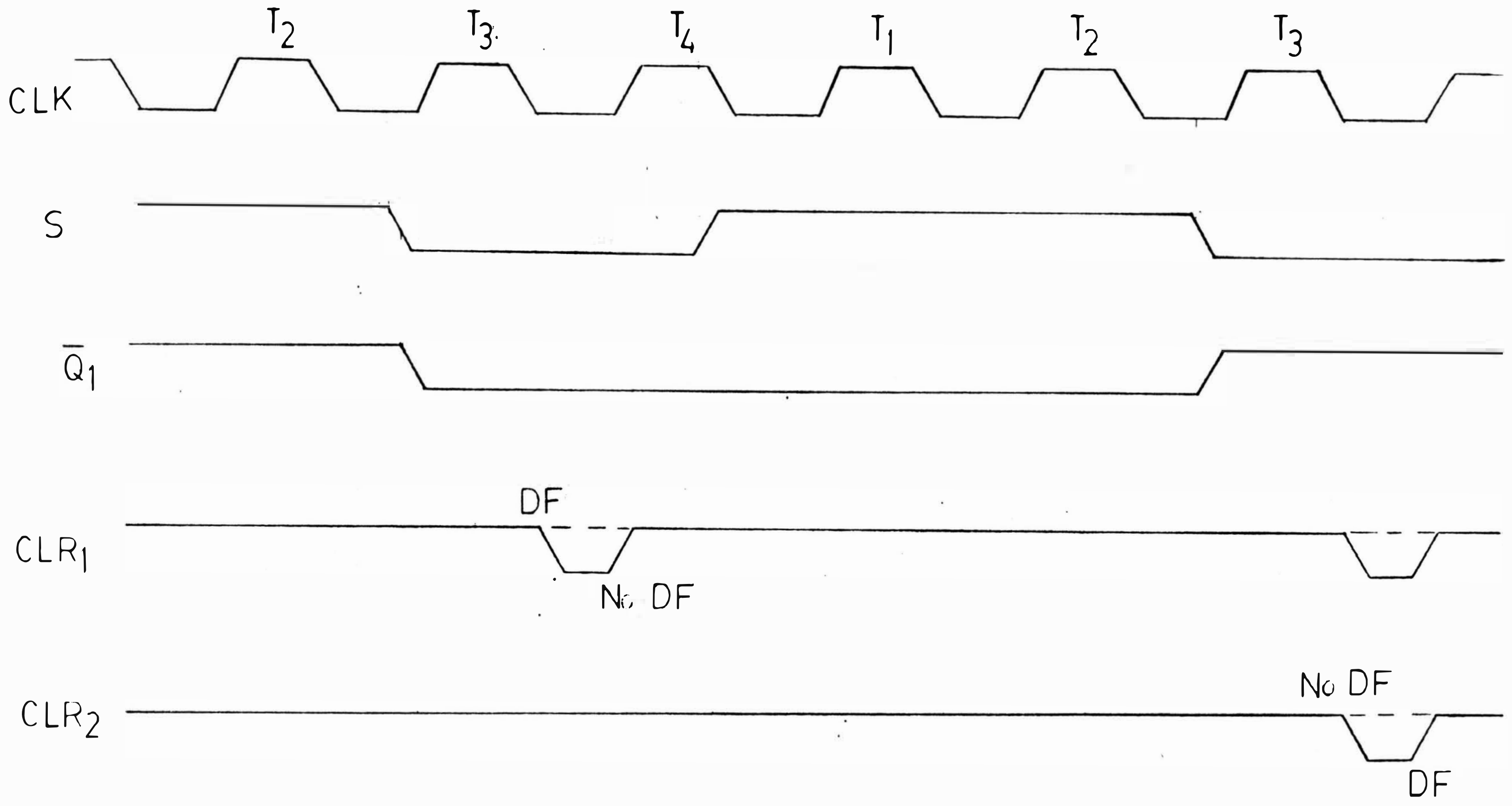


Figure 6.11 Clearing and Control Timing Waveforms

Figure 6.11 shows the timing waveform with respect to the main clock signal. The Q1 (counter) is triggered low by the Status input S during its transition into the high state (Figure shows trigger on negative slope). The output CLR1 low is generated at the beginning of the T4 clock of the first bus cycle only if DF is not found.

The input Q12 tells whether or not the first byte of a vector instruction DF was found. However, if the first decoded byte is DF then CLR1 transits into low state along with CLR2 during T4 of the second bus cycle. Thus all the blocks return to initial state at the end of the first instruction fetch bus cycle, if "DF" is not found, otherwise they will resume initial state only at the end of the subsequent bus cycle. The output Q1 serves as input to the Subsequent Byte Decoder block.

E. Subsequent Byte Decoder.

The Subsequent Byte Decoder circuit decodes the second byte of a vector instruction. The required logic circuit is shown in Figure 6.12. The inputs to the circuit are eight address/data lines from the CPU, Q12 from the Data Monitoring circuit, Q1 from the Clearing and Control block, and T31 from the Data Enable Generator.

The input Q12 indicates the successful decoding of the first byte of a vector instruction. If so, then the

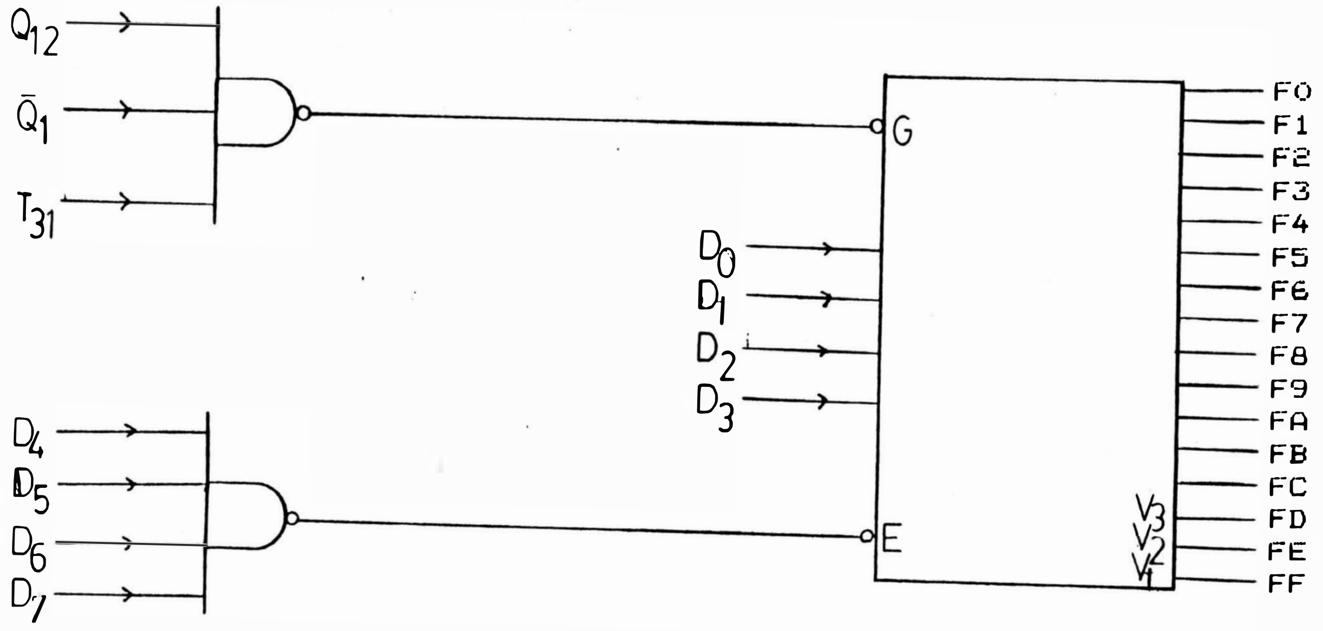


Figure 6.12 Subsequent Byte Decoder

Subsequent Byte Decoder monitors the address/data lines for the second byte of the same instruction. The T31 input indicates the presence of the instruction on the address/data line during the second instruction fetch bus cycle.

Looking back at Figure 6.7, it can be observed that the most significant nibble of the second byte of all the vector instructions is a hexadecimal code F. Hence the data lines, D4 to D7 are checked for this code F. A 4 x 16 decoder is used to decode the least significant nibble, which can be one of 0 through F codes. Thus, during T31 of the second bus cycle, the Subsequent Byte Decoder decodes the second byte of a vector instruction. The decoded output, which is one of the 16 vector instruction, is indicated by a logic 0 on the appropriate output pin of the 74L151 decoder. Thus the decoded instruction will be one of the 16 vector instructions DF F0 through DF FF.

Figure 6.13 shows the complete circuit diagram of the Vector Instructions Decoder unit. Refer to Appendix A for IC numbers and other details.

Address, data, status codes, Queue Status codes, and conditions of different control points at each clock pulse can be observed using the Tektronix 7D02 logic analyzer coupled with 7602 oscilloscope. Refer to Appendix B for an explanation.

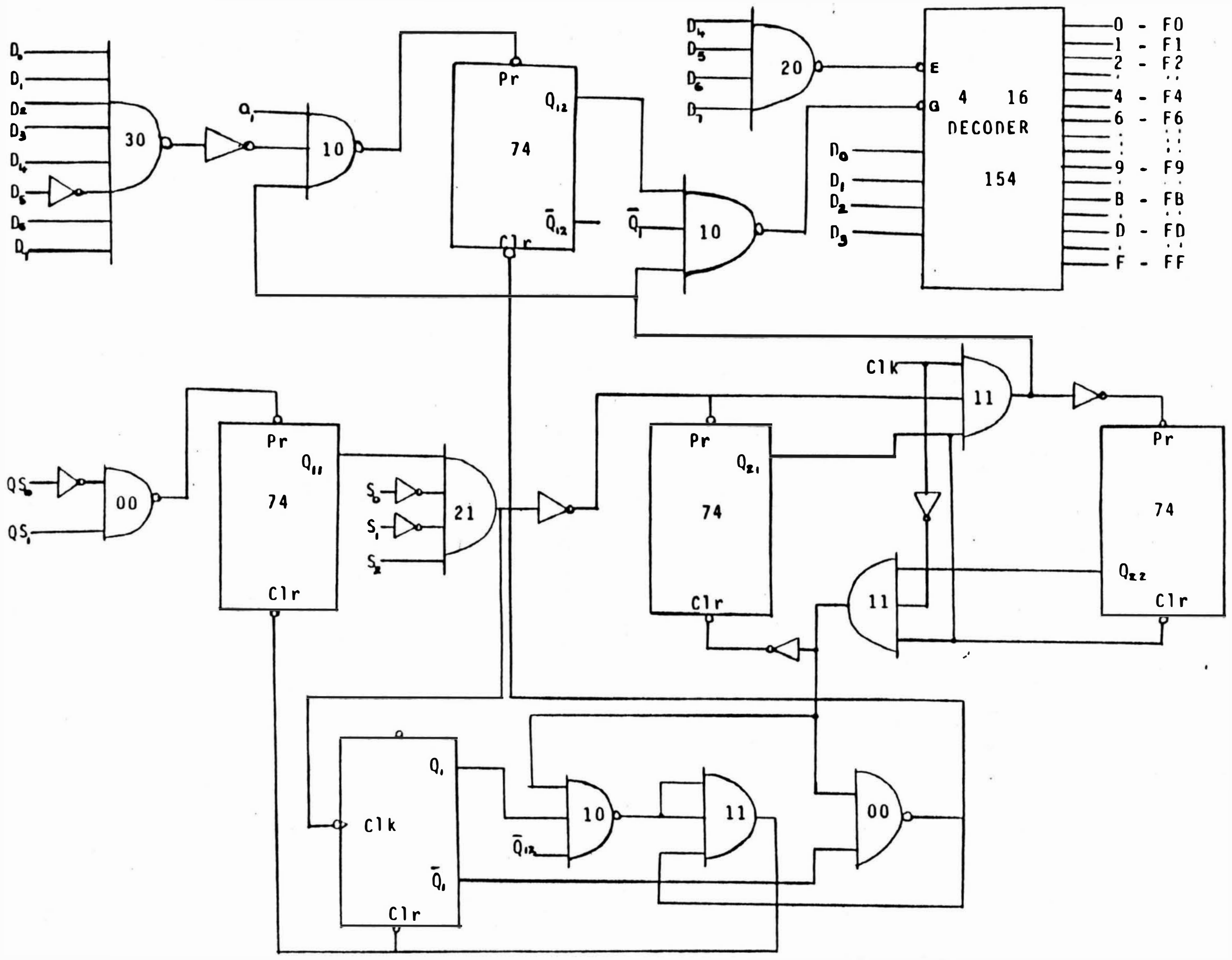


Figure 6.13 Vector Instructions Decoder Circuit

CHAPTER VII

SEQUENTIAL LOAD/STORE CONTROL

The Sequential LOAD/STORE Control unit controls the operations of the Vector Processor in serial mode. Serial mode operation requires that the Math Coprocessors be activated one at a time. This allows the loading or storing of a different number into or out of each Coprocessor. Serial mode also requires parallel loading of a LOAD/STORE instruction into the Coprocessors.

Serial mode can be entered through the Sequential LOAD/STORE Controller by the use of a vector instruction. As discussed in the previous chapter, there are 16 vector instructions that are output by the Vector Instruction Decoder. Two out of these 16 instructions are used to control the operations of Sequential LOAD/STORE Control. The vector instruction DF FF (hexadecimal) code is used to enter the serial mode and DF FE to return to scalar or normal mode.

The Sequential LOAD/STORE Control unit is divided into five functional blocks:

(1) Main Controller

(2) 87-1 Ready Control

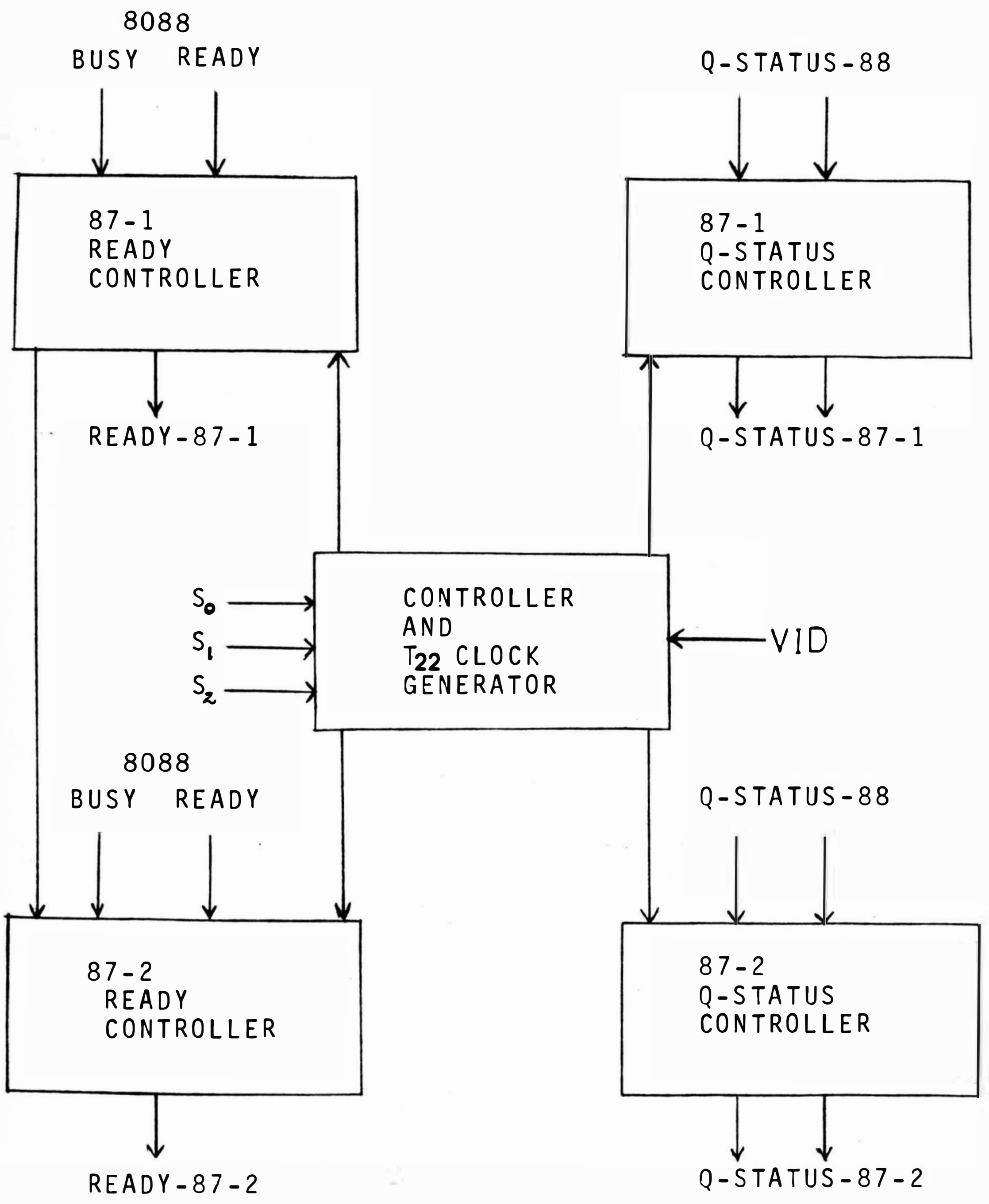


Figure 7.1 Sequential LOAD/STORE Control Circuit

(3) 87-1 Queue Status Control

(4) 87-2 Ready Control, and

(5) 87-2 Queue Status Control.

The Ready and Queue Status Controls of 87-2 (second 8087 Coprocessor) are repeated for the remaining Coprocessors with minor changes.

The Main Controller block controls the operation of the remaining blocks and synchronizes all the activities with the T2 clock of the CPU bus cycle. The control output C1 of the Main Controller is held low when the system is in scalar mode.

The 87-1 Ready Control maintains the Ready input to the first 8087 Coprocessor high at all times except in serial mode. In the serial mode of operation the Ready line is pulled low at the end of execution of the first instruction. The 87-1 Queue Status Control also follows a similar pattern.

The 87-2 Ready Controller keeps the Ready input to the second 8087 Coprocessor at logic 0 in scalar mode. Ready is allowed to receive R-88 (Ready output from the 8284 Clock Generator) in serial mode during parallel loading of a LOAD/STORE instruction and during execution of the loaded instruction. The 87-2 Queue Status Control activates the

Queue Status lines of the second 8087 Coprocessor only during instruction execution.

The above procedure can be better understood with the following Assembly instruction format sample:

```
ADDRESS          INSTRUCTION

0300
0301
0302             JMP NEXT1       ;
0304    NEXT1    FVECTOR-SQ      ;enter serial mode.
0306             FLOAD XXX       ;87-1 active
0308             WAIT            ;87-2 active during fetching.
0309             FLOAD XXX       ;87-1 Wait state, 87-2 active
030B             WAIT
030C             JMP NEXT2
030E    NEXT2    FSCALAR         ;return to scalar mode
0310
```

Note: XXX - any 8087 address mode.

The first FLOAD instruction following the vector instruction is loaded in parallel to all the 8087 Coprocessors (only two 8087 Coprocessors are shown in the above example). Remember, that the instruction fetch time is different from the instruction execution time. The loaded instruction will remain in the instruction queue of the Coprocessors until the Queue Status lines of the respective Coprocessors are activated. Refer to flow chart on Figure 9.3, of Chapter IX for Ready line and Queue Status lines conditions at each step during an Assembly language program execution.

Recall that all instructions in Assembly beginning with alphabet :F: are Escape instructions to the CPU. Hence, FLOAD is an Escape instruction ignored by the CPU. The Assembly instruction format example shown has only two FLOAD instructions. The number of FLOAD instruction can be varied depending on the number of Coprocessors required to be loaded. The first WAIT instruction following the vector instruction is used to terminate the parallel loading of instruction in serial mode.

A. Main Controller

The Main Controller block of the Sequential LOAD/STORE Control unit, controls and coordinates the activities of the remaining blocks. It generates a T22 clock that helps in synchronization of all operations with the T2 clock of the CPU bus cycle. For convenience the Main Controller is divided into two separate blocks: (1) the Control Block and (2) the T2 Clock Generator Block.

1. The Control Block.

The Control Block produces two control signals that control all other blocks. The required inputs, the circuit and the control outputs produced, are shown in Figure 7.2. The inputs are address/data lines from the CPU, T31, V1 (goes low on 'DF FF') and V2 (goes low on 'DF FE') from the Vector Instructions Decoder Unit and T22 from T22 Clock Generator block. The outputs are C1, C1 and SM1 signals.

Input V1 triggers Q31 (D-flip-flop) to high state on reception of the vector instruction "DF FF" (hexadecimal). High state of Q31 allows the monitoring of data lines for 9B (hexadecimal code of a WAIT instruction). T31, the decoded clock from the Data Enable Generator block of the Vector Instructions Decoder unit, indicates when the instruction is available on the address/data lines.

On reception of the WAIT (9B) instruction the counter triggers C1 (Q32) of the D-flip-flop to high state. The transition of C1 to high state indicates the end of the first 8087 Coprocessor instruction following the vector instruction.

The vector instruction DF FF causes low transition of V2 which triggers Q42 to a high state. Logic 1 on Q42 enables the T22 clock which clears all

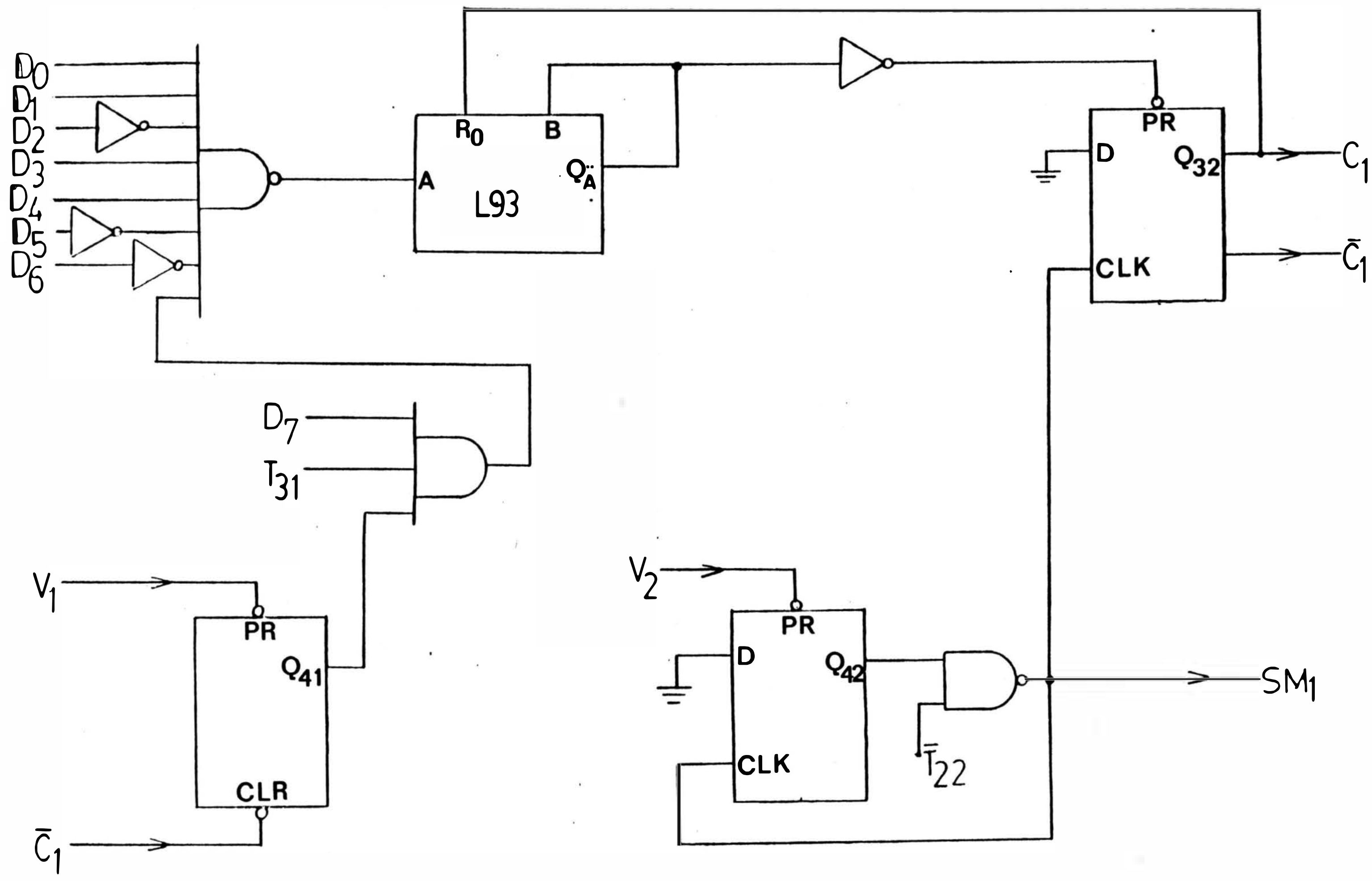


Figure 7.2 Circuit for Control Block

the flip flops and thus returns the system back to scalar mode.

2. T22 Clock Generator

The T22 Clock Generator decodes the T2 clock of the CPU bus cycle. This decoded clock, called T22, is used to synchronize the activities of Ready line controllers so as to satisfy setup time and hold time requirements of the Ready input.

The logic circuit of the T22 Clock Generator is shown in Figure 7.3. As indicted in the Figure, the inputs to the circuit are the status lines S0, S1, S2 from the 8088 CPU, clock input CLK from the 8284 Clock Generator and T31 from the Vector Instructions Decoder unit. The output is the decoded T2 clock of the CPU bus cycle, T22.

The circuit operation can be explained with the help of the timing waveforms shown in Figure 7.4. The transition of status lines from passive state to active state drives Tc low, allowing clock input to the counter. Outputs from the counter are fed to a decoder T12, T22, and T32 are the outputs of the decoder.

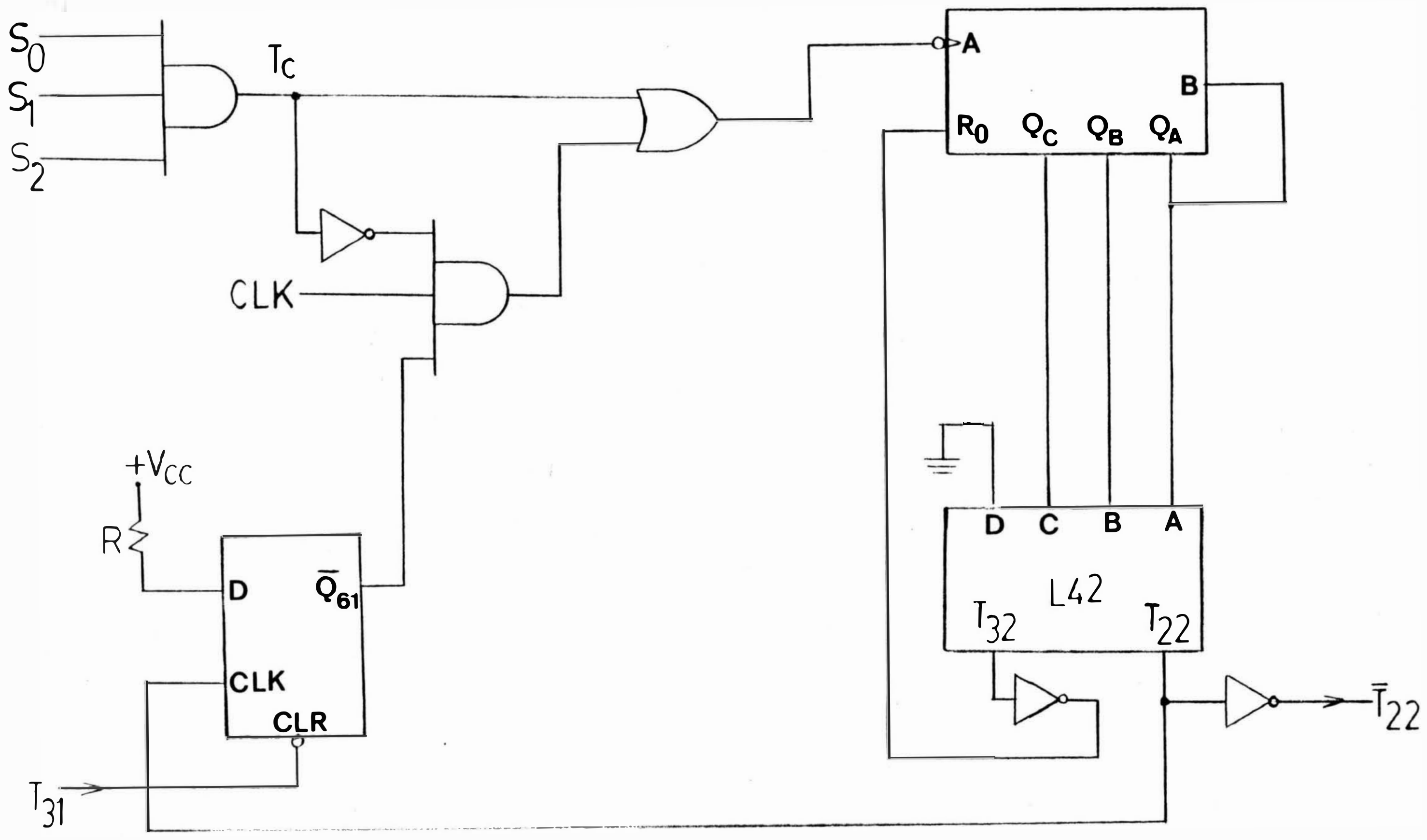


Figure 7.3 T_{22} Clock Generator

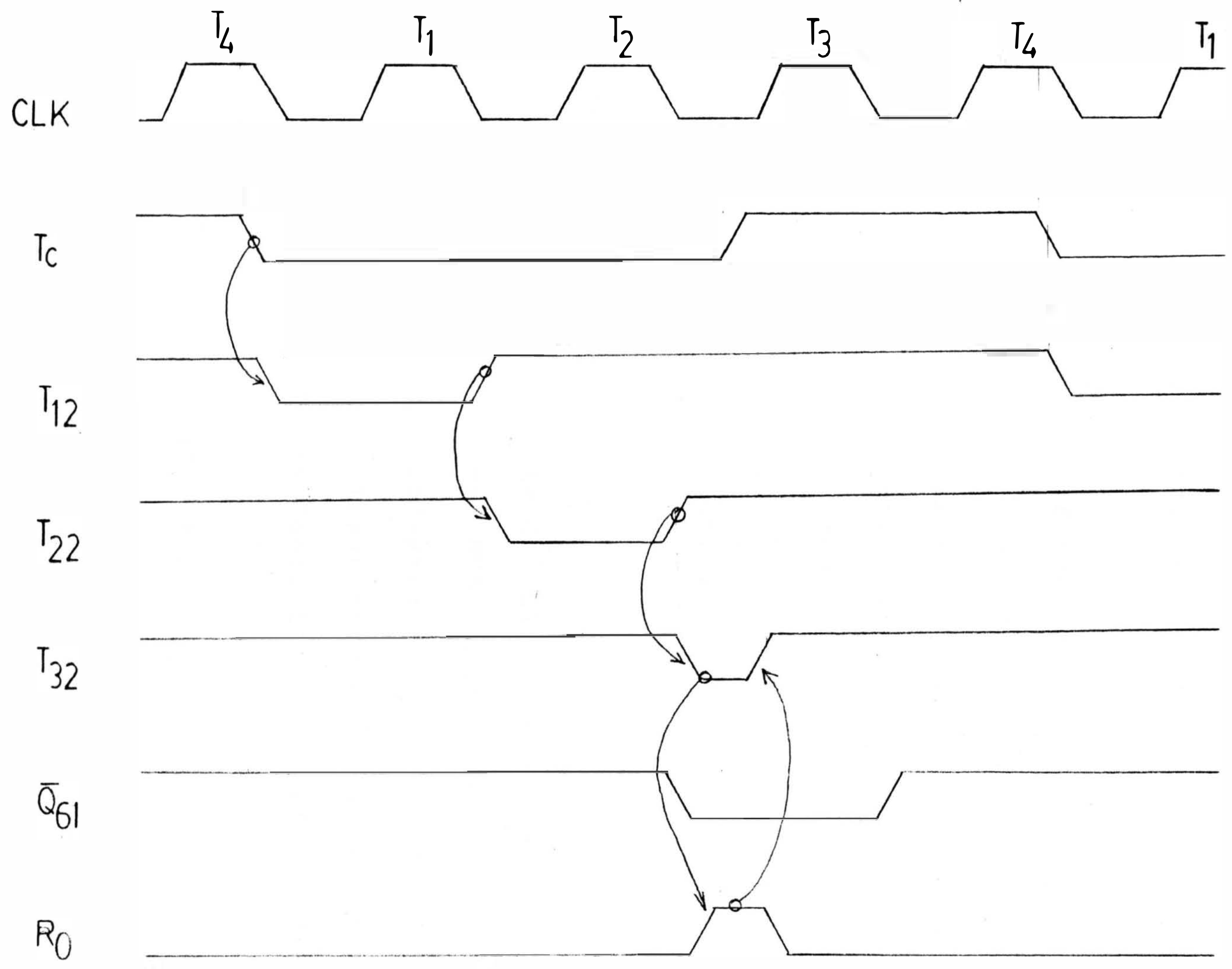


Figure 7.4 T_{22} Clock Generator Waveforms

High to low transition of Tc triggers T12 which returns to high state when triggered by the falling edge of the T1 clock. T12 in turn drives T22 low which is the required clock output. The low state of T22 ends when triggered by the falling edge of the T2 clock.

The T32 output of the decoder is used to clear the counter through Ro input, which results in termination of T32 itself. Q61 D-flip-flop triggered high by T31, enables the clock input to the counter. The clock input to the counter is disabled on transition of Q61 to low state, triggered on the positive edge of T22 clock.

B. 87-1 Ready Control.

87-1 Ready Control is a logic circuit that controls the Ready input of the first 8087 Coprocessor (87-1). The 8087 Coprocessor can be activated or forced into Wait state by controlling its Ready input.Figure 7.5 shows the logic circuit of the 87-1 Ready Control, required to implement the Ready logic.

The inputs to the 87-1 Ready Control circuit are: C1 from the Main Controller block, R-88, the Ready output of the 8284 Clock Generator, and B1 the Busy line output from the 87-1 Coprocessor. The output of the 87-1 Ready Control

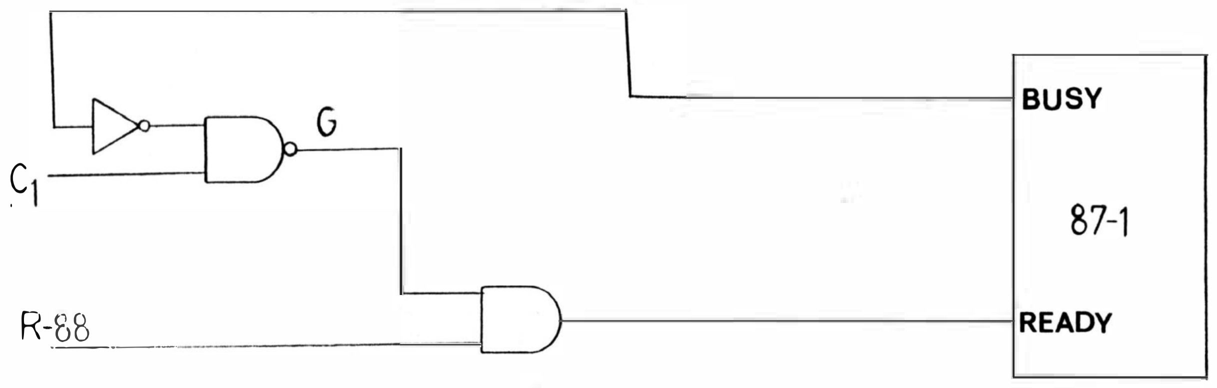


Figure 7.5   87-1 Ready Control

circuit drives the Ready input of the 87-1 Coprocessor. In scalar mode, logic 0 on C1 keeps the 87-1 in active state. The only time when the 87-1 is forced into Wait state is during serial mode operation. In serial mode, C1 switches to high state right after the parallel loading of the first instruction following the vector instruction. At this instant, Busy output B1 of the 87-1 will be high indicating the execution of the first LOAD/STORE instruction. At the end of execution, the Busy line B1 goes low which switches the Ready input of 87-1 to logic 0. A logic 0 on Ready input pushes the the 87-1 to Wait state. The 87-1 Coprocessor remains in Wait state unit C1 goes low. Upon transition of C1 to low state, Ready is set equal to R-88 activating the 87-1 Coprocessor.

C. 87-1 Queue Status Control.

87-1 Queue Status Control block controls the Queue Status lines QS0 and QS1 of the 87-1 Coprocessor. QS0 and QS1 are Queue Status information output by the 8088 CPU. QS0 QS1 = 00 indicates no operation. Refer back to Figure 4.10 for Queue Status codes. Queue status inputs to the 87-1 are held low when it is in Wait state. Active state 87-1 is allowed to receive Queue Status information from QS0 and QS1 lines of the CPU.

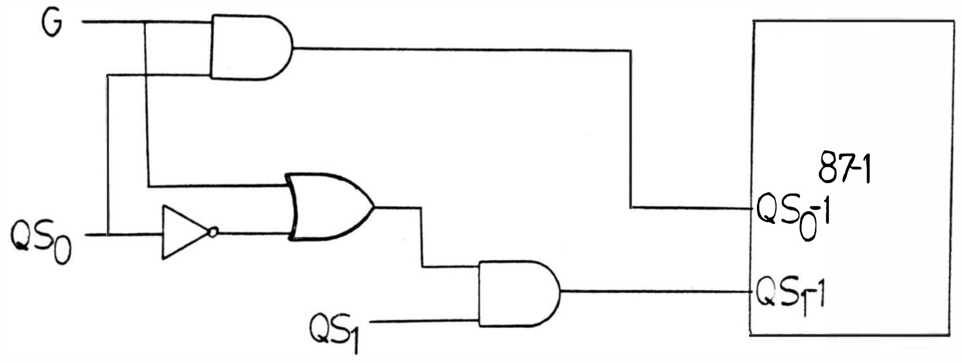


Figure 7.6 87-1 Queue Status Control

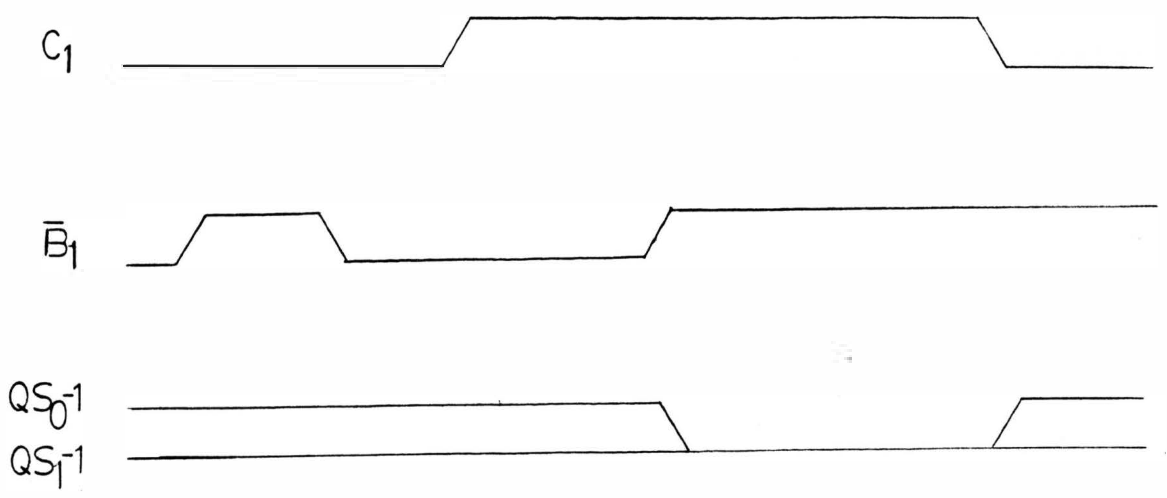


Figure 7.7 Timing Waveforms

Figure 7.6 shows the 87-1 Queue Status Control circuit. The inputs to the circuit are QS0 and QS1 from the CPU, Busy input B1 from the 87-1 and control input C1 from the Main Controller block.

Figure 7.7 shows the timing diagram for the circuit. As indicated in the timing diagram QS0-1 and QS1-1 are active until C1 and B1 both switch to logic 1. C1 and B1 switch to logic 1 only in serial mode right after the execution of the first Coprocessor instruction. Queue status lines QS0-1 and QS1-1 are held low until C1 switches back to logic 0. However, reinitialization code QS0QS1 = 01 are allowed even when 87-1 is in Wait state. This helps to clear the instruction queue before switching back to scalar mode. Thus the Queue Status lines QS0-1 and QS1-1 of the 87-1 Coprocessor, will be in active state in scalar mode, becomes inactive in serial mode after the execution of the first 8087 Coprocessor instruction and returns back to active state on termination of serial mode.

D. <u>87-2 Ready Control.</u>

The Ready input of the second 8087 Coprocessor (87-2) is controlled by the 87-2 Ready Control block. Unlike the first 8087 Coprocessor (87-1) the second 8087 (87-2) will be deactivated in the normal mode (scalar mode). The 87-2 Coprocessor is activated only in vector mode.

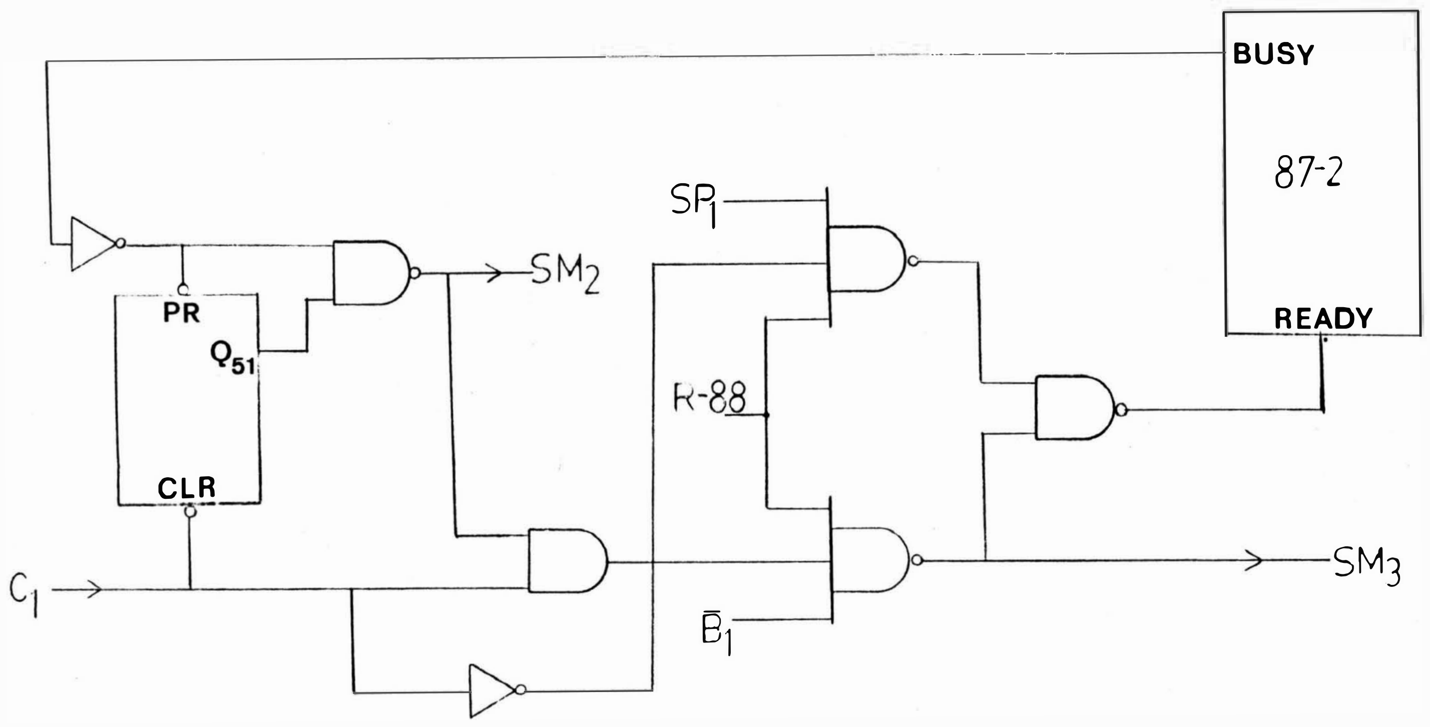


Figure 7.8   87-2 Ready Control

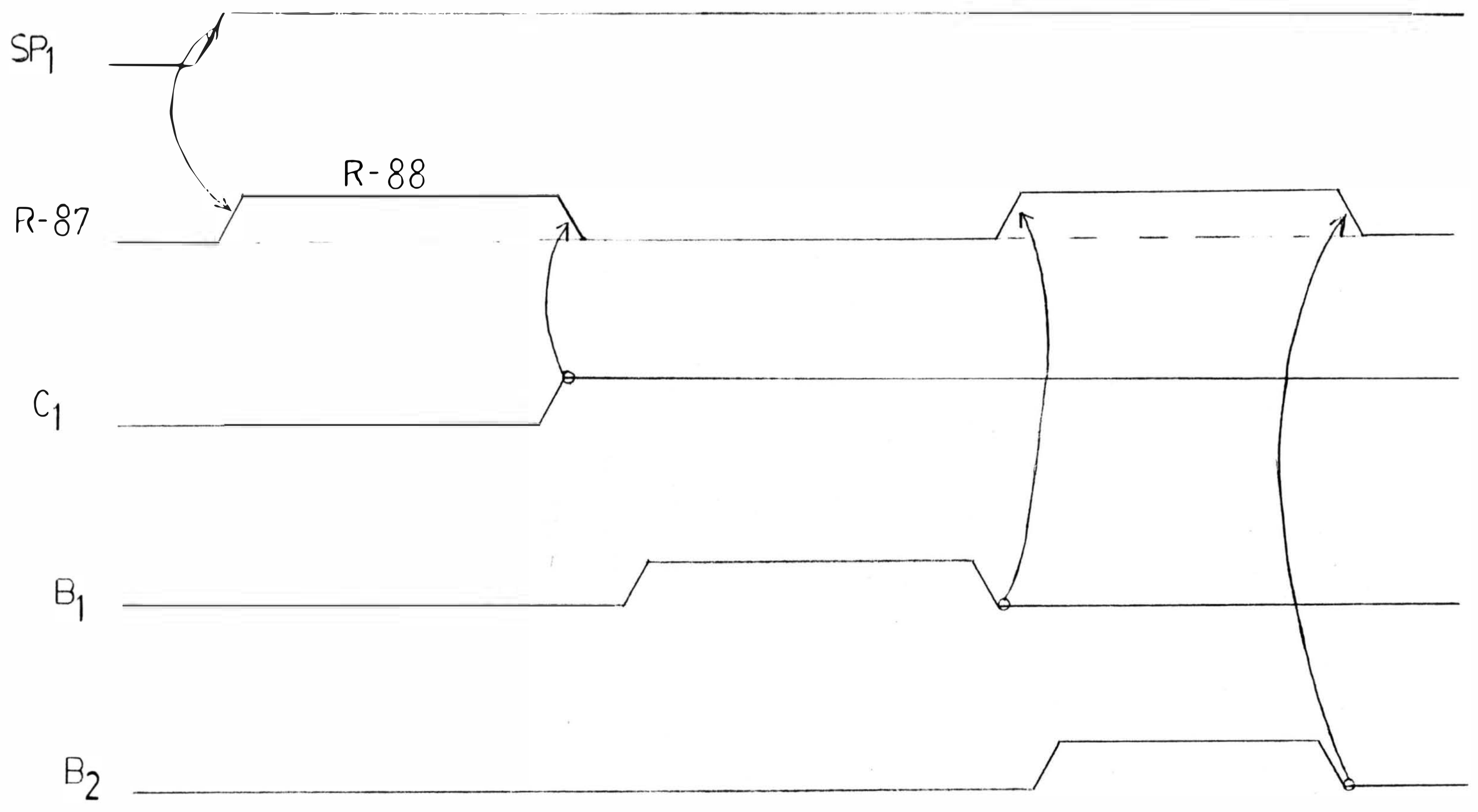


Figure 7.9 87-2 Ready Control Timings

The required circuit for the 87-2 Ready Control is as shown in Figure 7.8. Control input C1, R-88 input from 8284 Clock Generator, B1 from 87-1 and SP1 from parallel execution control unit are the inputs to this circuit. The Busy line B2 of 87-2 also serves as an input.

Inputs SP1 and C1 will be at logic 0 in scalar mode. This low state of SP1 and C1 maintains Ready input to 87-2 at logic 0 which forces 87-2 into Wait state. When serial mode is entered through the vector instruction, SP1 changes to logic 1 which sets the Ready input of the 87-2 equal to R-88.

Figure 7.9 shows the timing sequence that follow after the switching of SP1 to logic 1. As indicated in the timing diagram, the Ready line of 87-2 follows SP1 to high state. This switching Ready line is synchronized with the first instruction fetch cycle right after the vector instruction. Thus 87-2 is able to receive the instruction. At the end of the first instruction fetch, control C1 changes from low to high state forcing 87-2 Ready to logic 0. The 87-2 Coprocessor is activated again after the execution of the first instruction by the 87-1 Coprocessor. The completion of execution is indicated through the Busy line B1, upon which the 87-2 is activated again.

As shown in the timing diagram, (Figure 7.9), Ready input remains high until the end of execution, indicated through Busy line B2. At the end of execution, B2 is pulled low by the 87-2, which causes the Ready line to switch to logic 0, forcing 87-2 Coprocessor into Wait state.

The Busy line B2 of the second 8087 Coprocessor (87-2) can be used to invoke the next (third) 8087 Coprocessor (87-3) into active state. The control circuit required is very similar, with minor changes and is discussed in the next chapter.

E. 87-2 Queue Status Control.

The Queue Status lines QS0-2 and QS1-2 of the 87-2 Coprocessor is controlled using the 87-2 Queue Status Control circuit. This control circuit allows the 87-2 Coprocessor to receive Queue Status information from the 8088 CPU, when in active state. In the Wait state the 87-2 Coprocessor Queue Status inputs are held low by the 87-2 Queue Status Control. QS0 QS1 = 00 indicates no operation with the instruction queue. However, the reinitialize code QS0QS1 = 01 is allowed at all times.

The circuit of the 87-2 Queue Status Control is as shown in Figure 7.10. The inputs to the circuit, as shown in Figure 7.10 are: SM2 from the 87-2 Ready Control block, PM2 from the Parallel Execution Control unit, and C1, B1, QS0 and QS1.

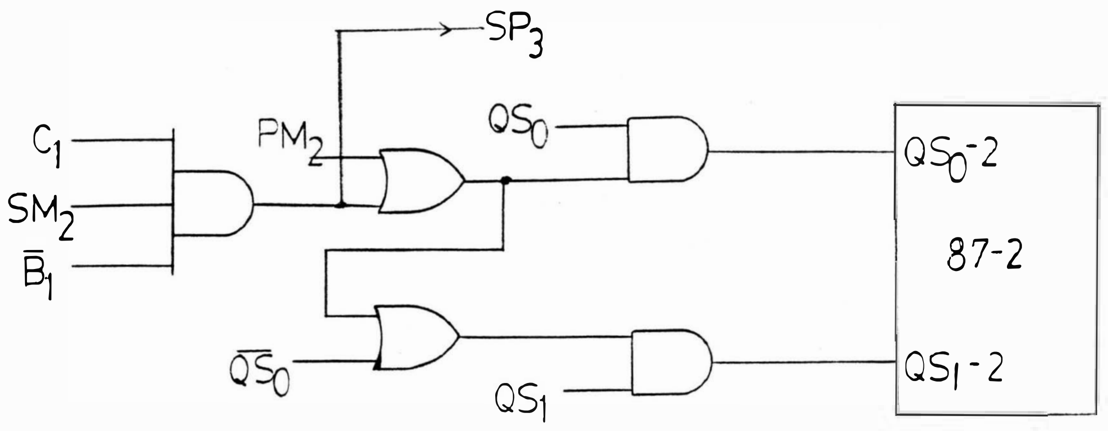


Figure 7.10   87-2 Queue Status Control

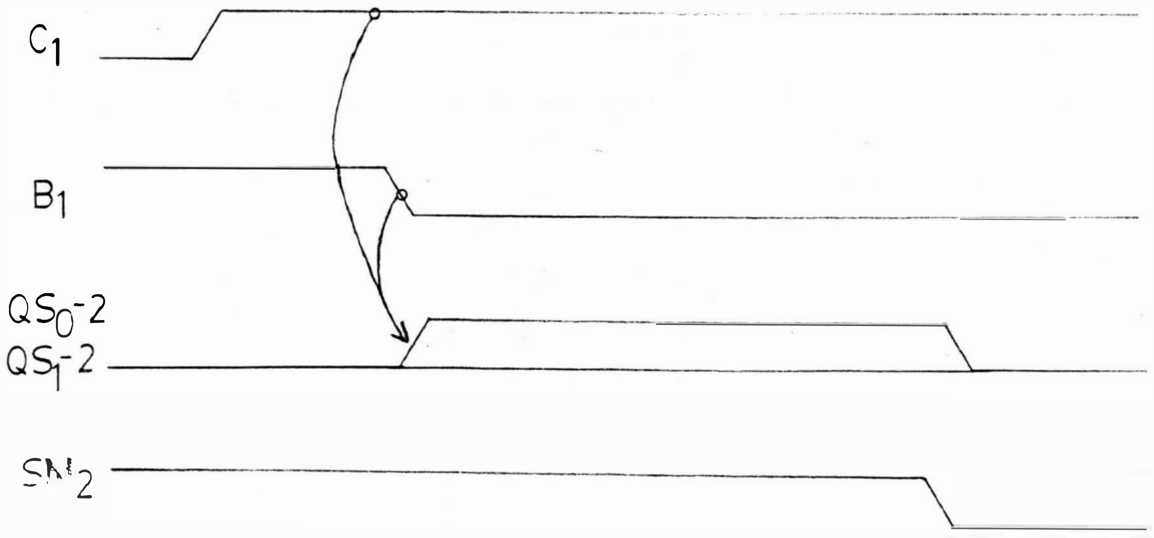


Figure 7-11   Timing Waveforms

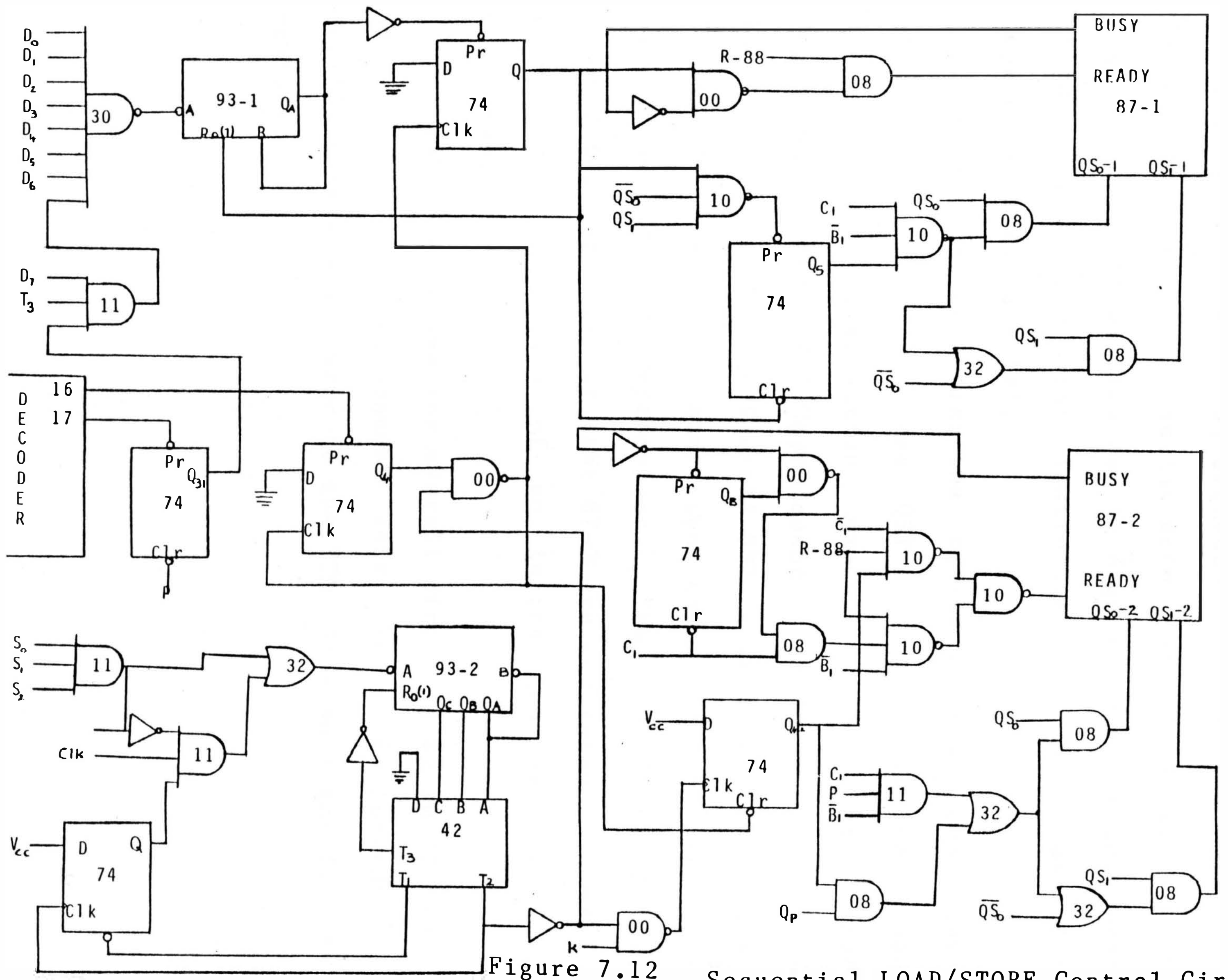


Figure 7.12 Sequential LOAD/STORE Control Circuit

The timing waveforms for the circuit, shown in Figure 7.11, indicates the timing sequence after the switching of C1 to high state. In scalar mode control C1 will be a logic 0. C1 changes from low to high state in serial mode at the end of the first instruction loading of all the Coprocessors in parallel.

As can be seen from the timing diagram, the Queue Status lines, QS0-2 and QS1-2 are activated upon completion of instruction execution by the first Coprocessor. The 87-2 Coprocessor receives Queue Status information from the CPU until SM2 changes to low state. Signal SM2 goes to low state upon completion of execution by the second 8087 Coprocessor. Thus the Queue Status lines QS0-2 and QS1-2 of the 87-2 are activated in serial mode during execution of an instruction by the 87-2 Coprocessor.

Figure 7.12 shows the overall circuit diagram of the Sequential LOAD/STORE Control unit. Refer to Appendix A for IC numbers and refer to the TTL Data book for IC pin numbers.

CHAPTER VIII

PARALLEL EXECUTION CONTROL

The Parallel Execution Control unit controls the operations of the vector processor in parallel mode. The data to be operated on are first loaded into the required number of Coprocessors using the Sequential LOAD/STORE Control unit, as explained in the previous chapter. The Parallel Execution unit, activated through a vector instruction, then allows the execution of 8087 Coprocessor instructions on these data in parallel.

The vector instruction DF FD (hexadecimal code) is used to activate the Parallel Execution Control unit. The system remains in the parallel mode until terminated by a vector instruction (DF FE) that returns the system back to scalar mode. In parallel mode, the Parallel Execution Control unit activates the Ready input and the Queue Status inputs of all the 8087 Coprocessors. Hence any 8087 instructions following thereon will be executed by all the Coprocessors. Any number of CPU instructions can be intermixed with the 8087 instruction in parallel mode. The only instructions that can not (should not) be used in parallel mode are the 8087 STORE instructions and the serial mode vector instruction.

Figure 8.1 shows the block diagram of the Parallel Execution Control, containing two control blocks: (1) Parallel Ready Control, and (2) Parallel Queue Status Control. On reception of a parallel mode vector instruction the Parallel Execution Control drives the Ready input of all the Coprocessors to logic 1, which activates the Coprocessor.

At the same instant the Parallel Queue Status Control activates the Queue Status inputs of the Coprocessors.

A. <u>Parallel Ready Control.</u>

Hardware circuitry required to implement the parallel Ready control logic is shown in Figure 8.2. The circuit shown is provided with inputs, V3 (goes low on 'DF FD') from the Vector Instruction Decoder, SM1, Q41 and T22 from the Sequential LOAD/STORE Control Unit and R-88 from the 8284 Clock Generator. The output of this circuit controls the Ready input of the 87-2 Coprocessor. The circuitry required to control the remaining Coprocessors is discussed at the end of the chapter.

The control actions of the Parallel Execution Control circuit begins with a vector instruction that drives V2 input low. The V3 low triggers Q71 (PM1) to high state

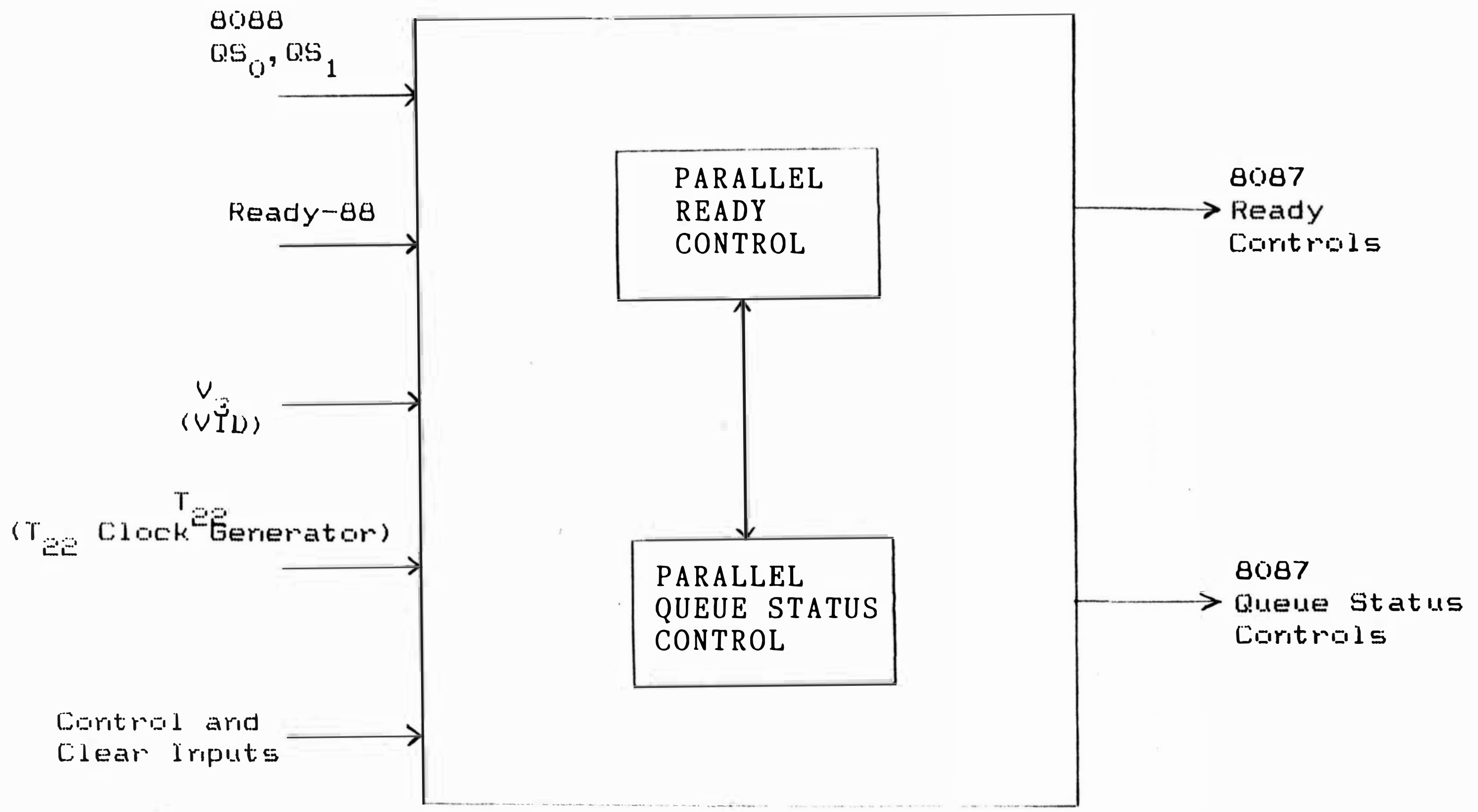


Figure 8.1 Parallel Execution Control Block Diagram

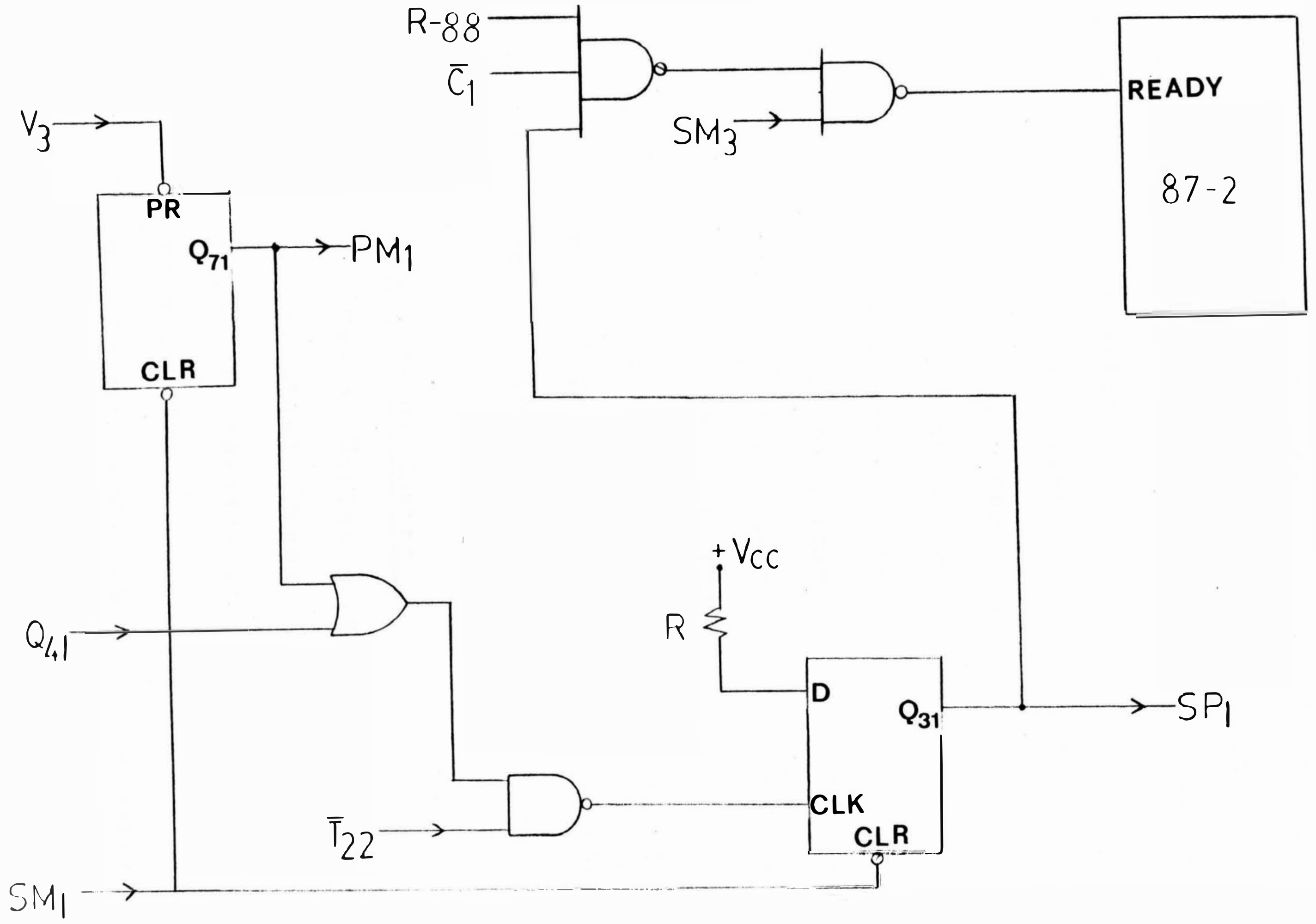


Figure 8.2 Parallel Ready Control

which in turn triggers Q31 high during the T22 clock. Q31 enables R-88 input to the Ready line which pulls the 87-2 Coprocessor to active state. It will be seen later that the same Q31 (SP1) output is used to activate the other Coprocessors in the sytem.

In parallel mode, with Ready input high, the Coprocessors execute all the 8087 instructions. The parallel mode is terminated through the same vector instruction that was used to terminate the serial mode operation. The vector instruction DF FE causes the low transition of SM1, a signal synchronized with T22. The SM1 low signal clears PM1 and Q31 (D-flip-flop) which in turn forces the Coprocessors into Wait state. The inputs C1 and SM2 shown in Figure 8.2 are serial mode controls and do not interfere with the parallel mode signals.

B. <u>Parallel Queue Status Control.</u>

The Parallel Queue Status Control follows a logic similar to the Parallel Ready Control circuit. The difference lies in the termination timing of Queue Status activity, compared to that of the Ready input. Figure 8.3 shows the logic circuit required for the parallel Queue Status control implementaton. For the circuit to operate, it needs QS0 and QS1, the Queue Status inputs from the CPU,

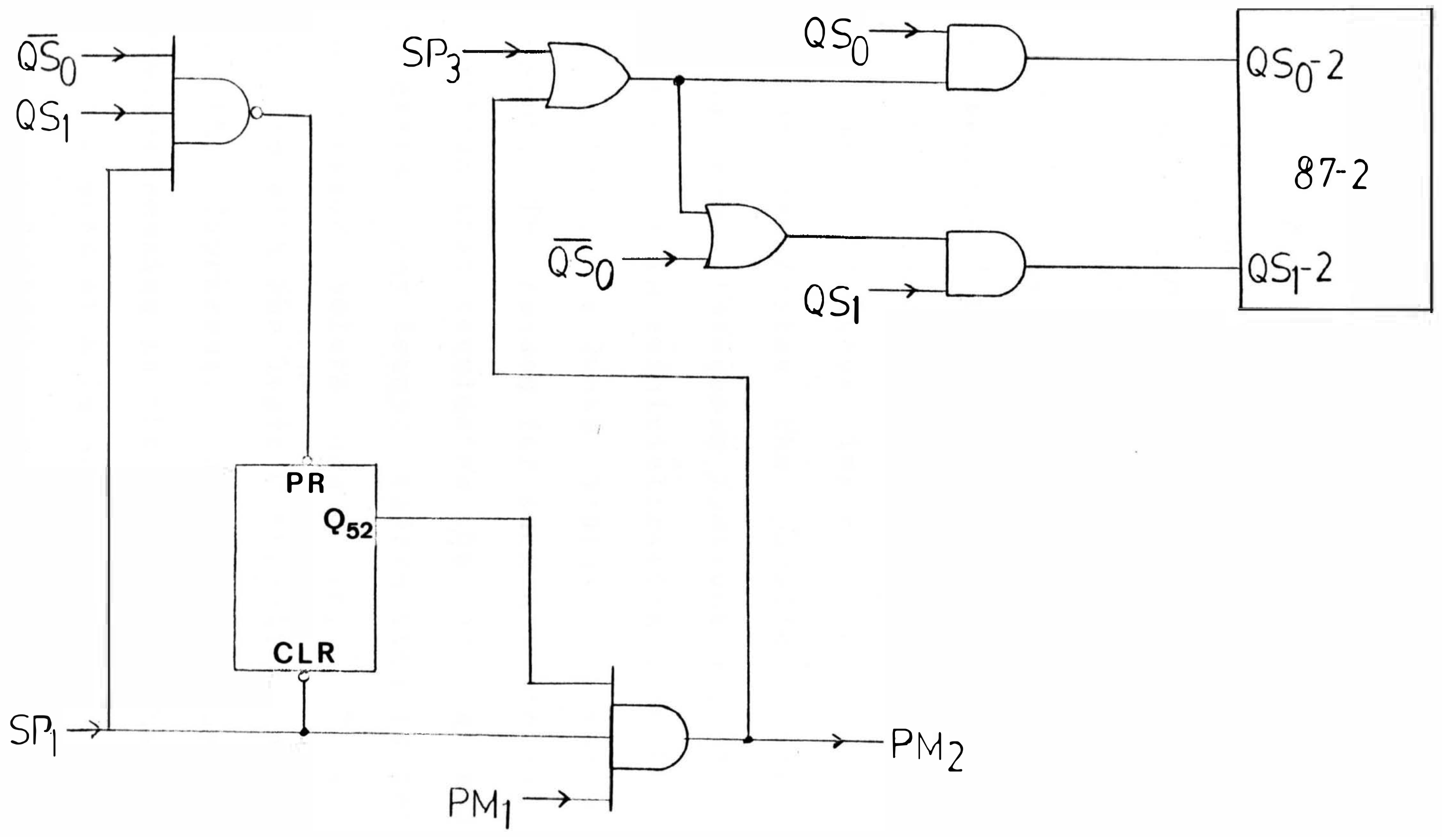


Figure 8.3 Parallel Queue Status Control

SP1 from the Sequential LOAD/STORE Control and PM1 from the Parallel Ready Control block.

As explained for the Parallel Ready Control circuit, the input signal PM1 is triggered high by the vector instruction. The high state of PM1 allows the QS0-2 and QS1-2 inputs of the 87-2 to receive Queue Status information from the QS0 and QS1 Queue Status lines of the CPU. In scalar mode PM1 is held low and hence the QS0-2 and QS1-2 status lines are held with low inputs which indicate "no operation" to the 87-2 Coprocessor.

The Queue Status lines of 87-2 are switched to logic 0 through an empty-the-queue instruction. The vector instruction that terminates the parallel mode will be preceded by an empty-the-queue instruction, for reasons explained earlier. This reinitialization code QS0QS1 = 01 is used to terminate the Queue Status information to the 87-2 Coprocessor. The reason for such a logic is that the vector instruction that terminates the parallel mode is an ESCAPE instruction. Any Escape instruction is decoded by the 8087 Coprocessor before neglecting. This decoding process coincides with the logic 0 transition of the Ready line, forcing the Coprocessor into Wait state. The Wait state Coprocessor remains in the Busy state that results in crashing of the system at a later stage. The deactivation of the Queue Status prevents the Coprocessor from executing

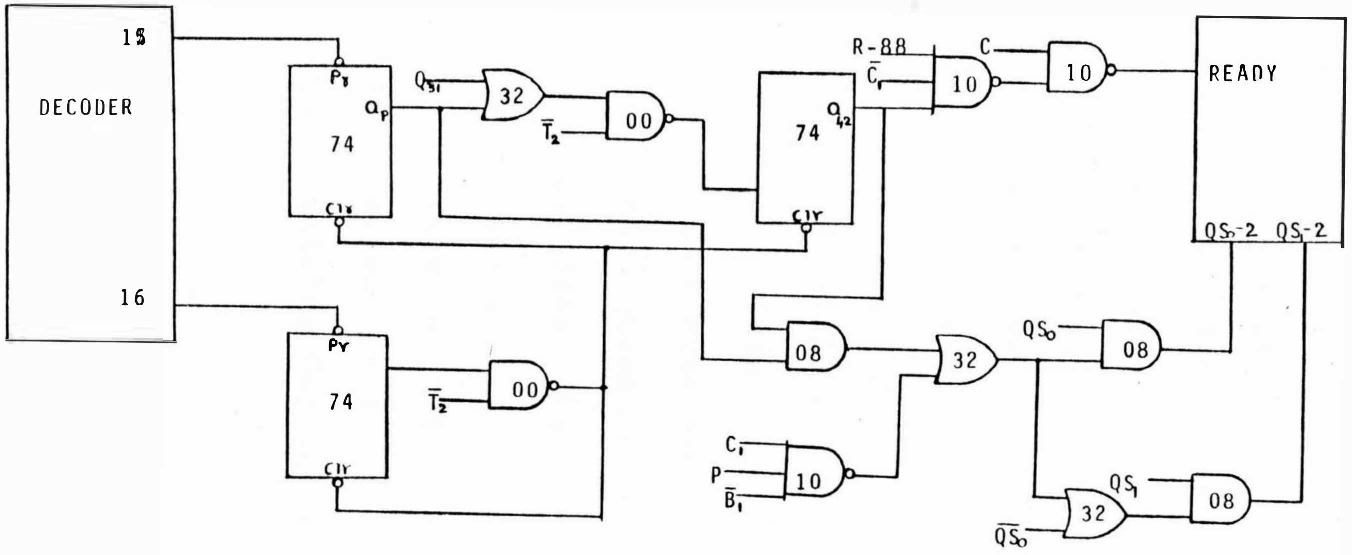


Figure 8.4 Parallel Execution Control Circuit

the vector instruction (DF FE). Figure 8.4 shows the overall circuit of the Parallel Execution Control Unit.

C. A Genralized Control Circuit.

All the control circuits discussed so far were only for the first two 8087 Math Coprocessors. This is because the control circuit required for the remaining Coprocessors is almost the same as that of the one discussed for the 87-2 Coprocessor, except one or two input changes. The number of 8087 Coprocessors in the system, however, depends on various factors discussed in the concluding chapter.

A Generalized Control Circuit for the remaining 8087 Coprocessors is shown in Figure 8.5. The circuit shown controls the Ready and the Queue Status inputs of an 8087 Coprocessor. The input signal Pn-1 and Bn of the 87-n Coprocessor (where n = 3,4,.......(say) 10), indicates the initiation and termination of 8087 Coprocessor activity respectively in serial mode. In scalar mode, like the 87-2 Coprocessor, all the remaining Coprocessors will be in the Wait state.

1. Serial Mode Operation.

In Serial mode, referring to Figure 8.4, with all other signals remaining the same, the input signal Pn-1 goes high at the end of the execution by the preceding

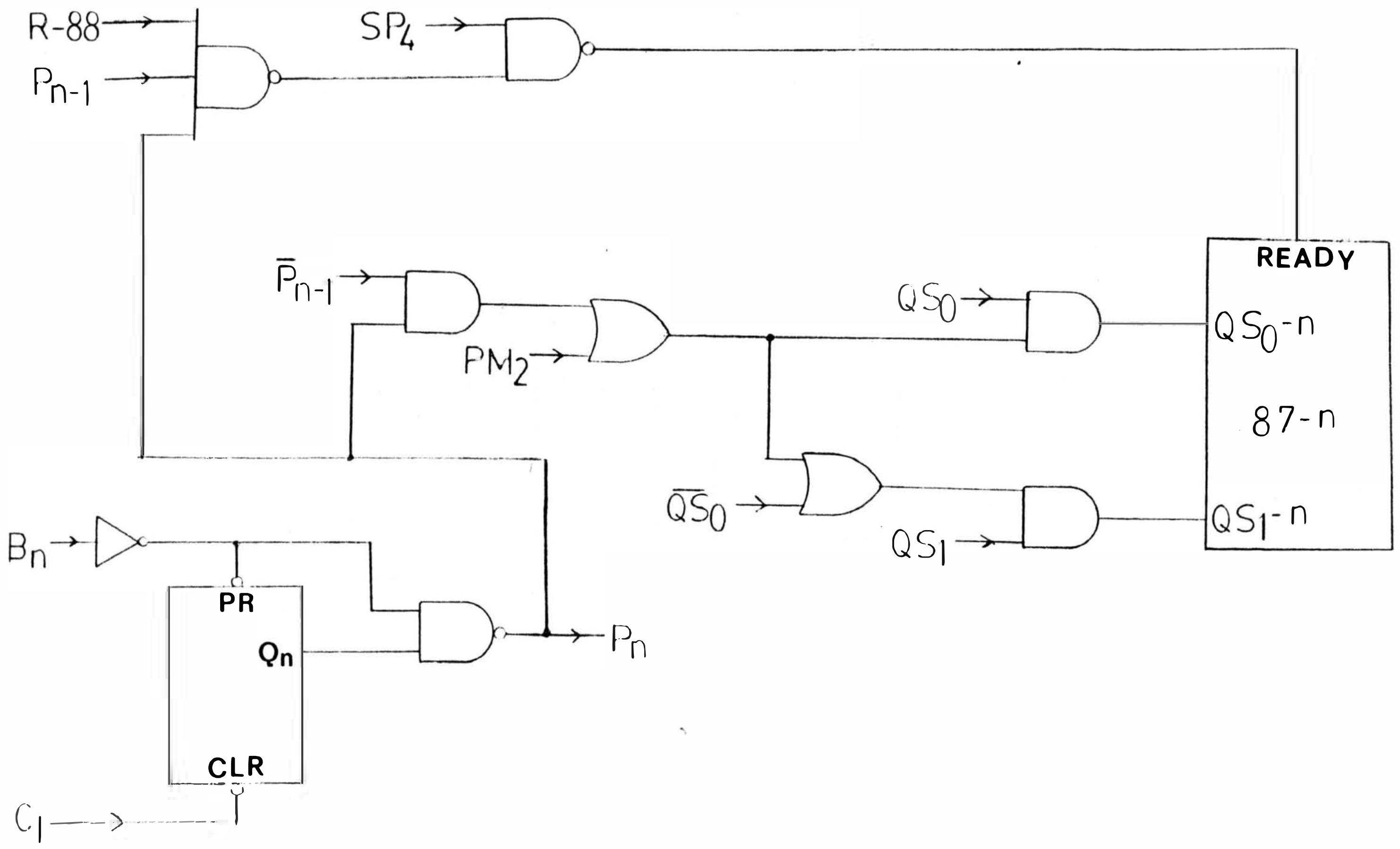


Figure 8.5 Generalized Control Circuit

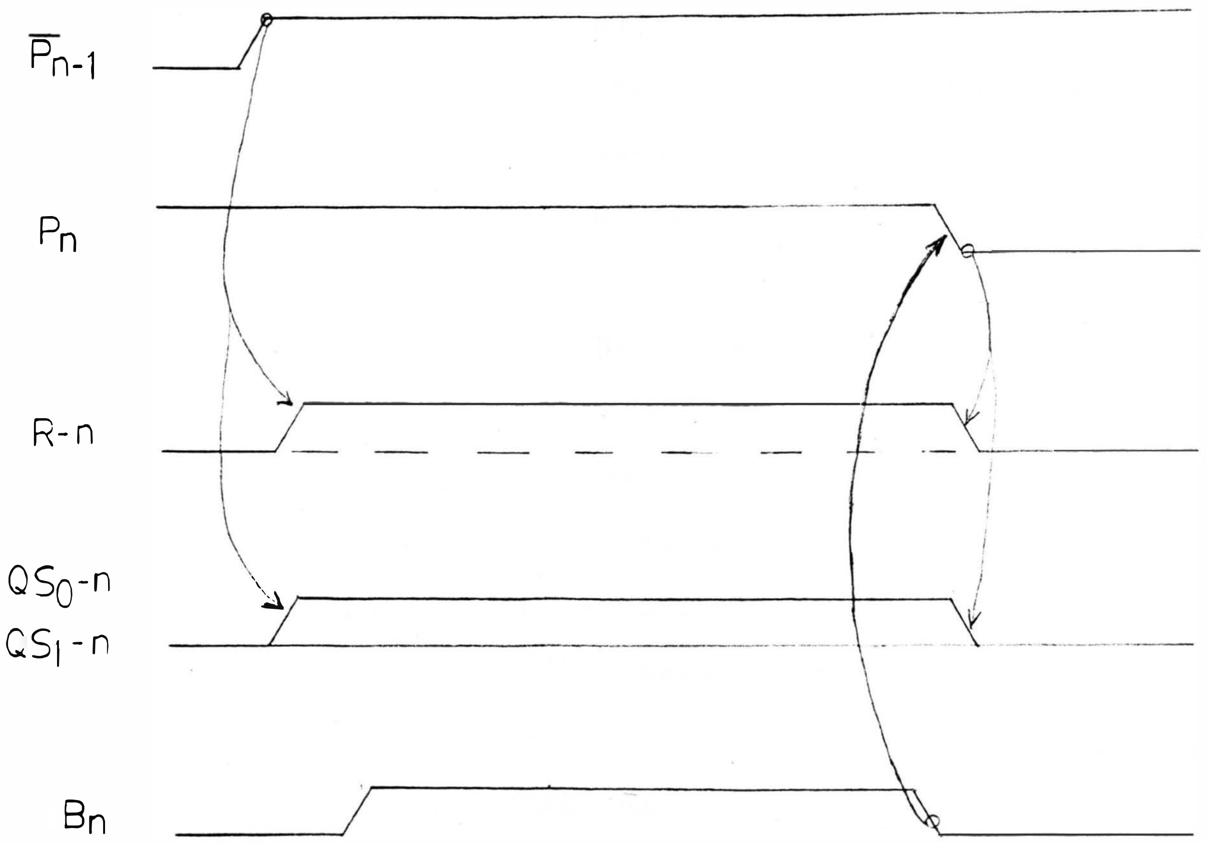


Figure 8.6 Generalized Control Timing Waveforms

8087 Coprocessor. The Busy signal Bn will be high when the Coprocessor is in operation. The high state of Pn-1 which forces the 87-(n-1) Coprocessor into Wait state also activates the 87-n Coprocessor.

Both the Ready and the Queue Status inputs of the 87-n Coprocessor are activated. The 87-n will remain in active state, until the end of the LOAD/STORE instruction, indicated through the Busy signal Bn. At the end of the execution Bn is pulled low which drives signal Pn low.

The Pn low signal causes the low transition of Ready and Queue Status signals, forcing the 87-n Coprocessor into Wait state. Figure 8.6 shows the timing waveform for the circuit in serial mode.

2. Parallel Mode Operation.

In parallel mode the signal SP1 high activates all the 8087 Coprocessors in the system, including 87-2. The termination of parallel mode Ready input occurs through the same signal SP1 going low, whereas the Queue Status lines are terminated a little earlier through the Q52 signal.

This chapter concludes the discussion of the hardware implementation of the Vector Controller that converts an IBM PC into The Vector Processor.

CHAPTER IX

ASSEMBLY LANGUAGE PROGRAM FORMAT FOR THE VECTOR PROCESSOR

The Vector Processor has its own insturction set containing sixteen vector instructions. At present The Vector Processor makes use of only three instructions out of these sixteen instructions. The remaining thirteen instructions are reserved for future use as vector instructions. The hexadecimal code of the three used vector instructions are :

1) DF FD - Parallel mode
2) DF FE - Scalar mode
3) DF FF - Serial mode

In Assembly language, these are named by Dr.Miron as -

FVECTOR-SQ	(DF FF)
FSCALAR	(DF FE)
FVECTOR-OP	(DF FD)

Under normal conditions, The Vector Processor will be in scalar mode. The vector environment is entered through FVECTOR-SQ instruction upon which the system assumes serial mode. A series of FLOAD Coprocessor instructions follows the serial mode vector instruction. After loading the data elements into the required number of Coprocessors the system should return back to scalar mode. This resets the Vector Controller either for parallel mode or a new sequential operation.

The FVECTOR-OP instruction is used to get into parallel mode. This is followed by Coprocessor instructions to operate on the data elements previously loaded. In parallel mode, any of the 8087 Coprocessor instructions may be used (except the FSTORE instruction) and these instructions are executed by all the Coprocessors simultaneously. The parallel mode is terminated through the FSCALAR instruction upon which the scalar mode is resumed.

The results of parallel mode operation are stored back into memory using an FVECTOR-SQ instruction followed by a series of Coprocessor STORE instructions.

The functions of these three instructions can be better understood with an example. Figure 9.1 shows the Assembly Language Program Format using vector instructions to perform the addition of three pairs of data elemtnts. As shown in Figure 9.1 the program starts with a jump instruction, followed by the FVECTOR-SQ. The three consecutive FLOAD instructions enables the loading of the first three data elements into the first three Math Coprocessors respectively. The second set of data element is fed by repeating the same set of instructions or by

```
; ASSEMBLY LANGUAGE PROGRAM FORMAT FOR ADDING THREE
; PAIRS OF DATA ELEMENTS

       JMP NEXT1              ;clear the queue.
NEXT1  FVECTOR-SQ             ;serial load vector instruction.
       FLD XXX                ;coprocessor load instructions
       WAIT                   ;to load the first three
       FLD XXX                ;coprocessors.
       WAIT                   ;XXX - indicates any of the
       FLD XXX                ;avialble addressing modes
       WAIT                   ;
       JMP NEXT2              ;
NEXT2  FSCALAR                ;return to scalar mode.
       JMP NEXT3              ;
NEXT3  FVECTOR-SQ             ;enter serial mode for loading
       FLD XXX                ;second set of data elements.
       WAIT                   ;
       FLD XXX                ;
       WAIT                   ;
       FLD XXX                ;
       WAIT                   ;
       JMP NEXT4              ;
NEXT4  FSCALAR                ;back to scalar mode
       JMP NEXT5              ;
NEXT5  FVECTOR-OP             ;enter parallel mode
       FADD                   ;add instruction executed by all
       JMP NEXT6              ;the three coprocessors.
NEXT6  FSCALAR                ;terminate parallel mode.
       JMP NEXT7              ;
NEXT7  FVECTOR-SQ             ;serial mode instruction for
       FST XXX                ;storing the results of add
       WAIT                   ;operation, from the three
       FST XXX                ;coprocessors, sequentially.
       WAIT                   ;
       FST XXX                ;
       WAIT                   ;
       JMP NEXT8              ;
NEXT8  FSCALAR                ;return to scalar mode
       END                    ;
```

Figure 9.1

changing the address and using a loop.The FSCALAR instruction brings back the Vector Processor into scalar mode, and resets the Vector Controller.

The FVECTOR-OP instruction activates all the Coprocessors simultaneously and hence the following FADD instruction will be executed by all the Coprocessors. Again the system returns to scalar mode with the FSCALAR instruction. The result of the FADD operation is stored back into the memory using FSTORE instruction. The storing is done sequentially using FVECTOR-SQ followed by three FSTORE instructions.

The number of operations executed in parallel mode also contributes to the inscrease in speed of system operation. The operation of The Vector Processor can be better appreciated through the program on Figure 9.2. This Assembly Language Program evaluates the following function:

$$\sqrt{\text{TAN } (A + C)}$$

for a four pair of data elements. As shown in the program , the FVECTOR-OP is followed by three 8087 Math Coprocessor instructions which will be executed by all the Coprocessors in parallel. This parallel operation saves the fetching, decoding,and execution time of the four instructions and hence increases the system speed, approximately by three

```
; ASSEMBLY LANGUAGE PROGRAM FORMAT FOR FINDING
; FUNCTION √TAN (A + C) OF FOUR SETS OF NUMBERS


        JMP NEXT1           ;
NEXT1   FVECTOR-SQ          ;serial mode vector instruction.
        FLD BX              ;load the first four Coprocessors
        WAIT                ;by incrementing register BX and
        FLD BX+02           ;loading from the location pointed
        WAIT                ;to by this register.
        FLD BX+04           ;
        WAIT                ;
        FLD BX+06           ;
        WAIT                ;
        JMP NEXT2           ;
NEXT2   FSCALAR             ;end loading.
        JMP NEXT3           ;
NEXT3   FVECTOR-OP          ;parallel mode vector instruction
        FLD CONSTANT        ;load the constant into all the
        WAIT                ;coprocessors.
        FADD                ;add the constant with the data
        WAIT                ;loaded in serial mode.
        FTAN                ;find the Tangent of the result.
        WAIT                ;
        FSQ                 ;find the square root of the
        WAIT                ;Tangent function.
        JMP NEXT4           ;
NEXT4   FSCALAR             ;terminate parallel mode.
        JMP NEXT5           ;
NEXT5   FVECTOR-SQ          ;enter serial mode.
        FST BX+08           ;store the result sequentially
        WAIT                ;into the memory location
        FST BX+0A           ;pointed to by the register BX.
        WAIT                ;
        FST BX+0C           ;
        WAIT                ;
        FST BX+0E           ;
        WAIT                ;
        JMP NEXT6           ;
MEXT6   FSCALAR             ;return to scalar mode.
        END                 ;
```

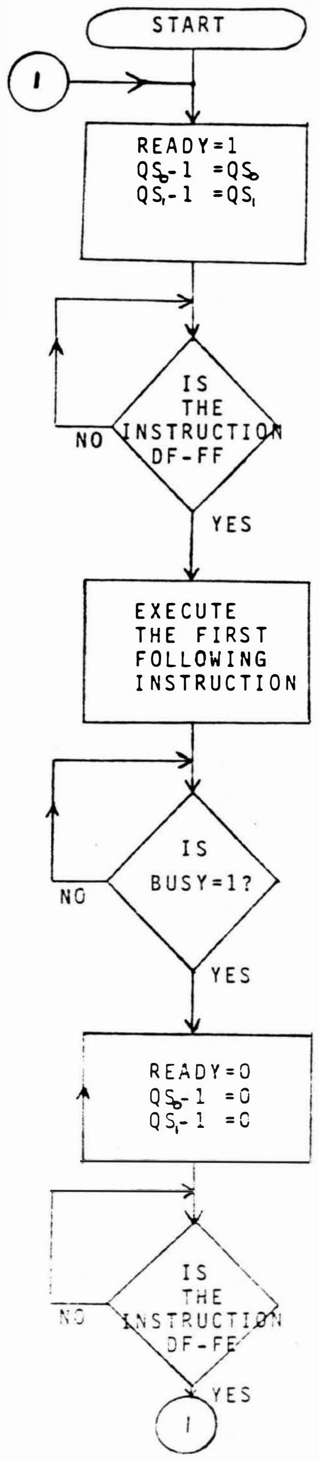
Figure 9.2

times. The load instruction in parallel mode is used to load a constant into all the Coprocessors.

Thus any Assembly Language Program will have to use the the three vector instructions for vector or parallel operations to get into and out of vector mode.

The Logical operation of the hardware part during these program execution will be as shown in the flow chart on Figure 9.3 and 9.4. The first flow chart on Figure 9.3 gives the conditions of the Ready and the Queue status lines of the first 8087 Math Coprocessor, which are controlled depending on the type of vector instruction.

Figure 9.4 shows the flow chart that gives the conditions of the Ready and the Queue status lines of the second Math Coprocessor (87-2). This flow chart also represents the conditions of the 3rd, 4th, or the 'n'th Coprocessor, except that the BUSY line of the n-1 Coprocessor serves as the input to the nth Coprocessor instead of the first coprocessor as shown in the Figure.Therefore these two flow charts, Figure 9.3 and 9.4 summarize the overall hardware and the software design concept of The Vector Processor.



- INITIAL CONDITIONS OF THE 8087-1

- WAIT FOR VECTOR SERIAL LOAD/STORE INSTRUCTION

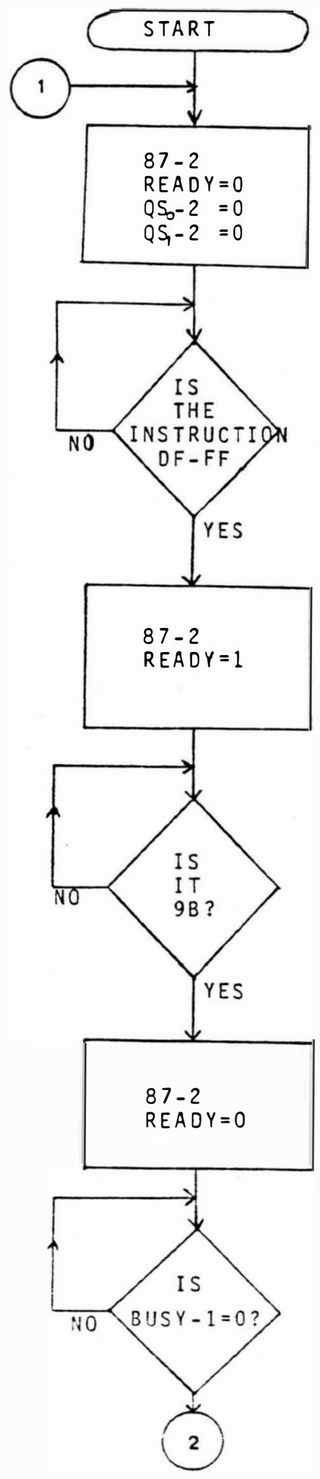
- EXECUTE THE FIRST LOAD/STORE INSTRUCTION

- WAIT UNTIL THE LOADING/STORING IS OVER

- ENTER WAIT STATE

- REMAIN IN WAIT STATE UNTIL THE VECTOR INSTRUCTION INDICATING THE END OF SEQUENTIAL LOAD/STORE

Figure 9.3 87-1 Ready and Queue Status Conditions Flow Chart



- INITIAL CONDITIONS OF 87-2; IN WAIT STATE

- WAIT FOR VECTOR SERIAL LOAD/STORE INSTRUCTION

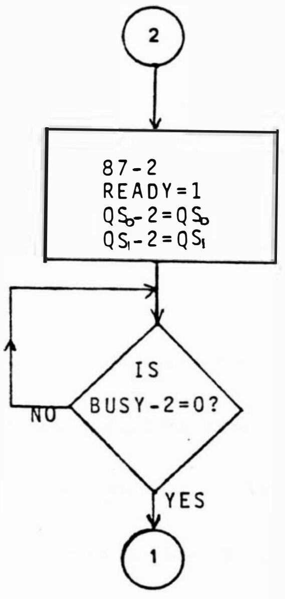
- RECEIVE THE LOAD/STORE INSTRUCTION

- 9B - THE WAIT INSTRUCTION INDICATES THE END OF LOAD/STORE INSTRUCTION

- 87-2 BACK TO WAIT MODE

- 87-2 WAITS UNTIL 87-1 FINISHES LOAD/STORE

Figure 9.4 87-2 Ready and Queue Status Conditions Flow Chart



- EXECUTE THE LOAD/STORE INSTRUCTION

- WAIT UNTIL THE END OF LOAD/STORE

Figure 9.4 87-2 Ready and Queue Status Conditions Flow Chart (continued)

CHAPTER X

CONCLUSION

This paper provides the hardware and the software dwsign necessary to convert a General Purpose Computer into The Vector Processor, using Math Coprocessors. Though this paper has used the IBM PC for the General Purpose Computer this design can be implemented on any other general purpose processor that provides the necessary interface signals for a Math Coprocessor.

Throughout this paper, no specific number for the Coprocessors that can be connected in parallel with the CPU has been mentioned. This is because the number of Math Coprocessors that can be used in parallel with the CPU depends on the hardware cost, the speed required, and the software application requirements.

The increase in number of Math Coprocessors also increases the system power requirements and the number of bus drivers, but gives rise to a dramatic increase in speed of the system.

Since it is the Math Coprocessor that operates in parallel, the Vector Processor gets all the advantages of having a Coprocessor in the system. These advantages include handling of floating-point numbers, increase in data types, and built-in facility of transcendental functions such as logarithm and tangent functions. Table 10.1 gives the amount of increase in speed of execution of certain instructions, caused by the use of a Math Coprocessor. A 5 to 10 times increase over this speed can be achieved (depending on the number of Coprocessors) by The Vector Processor. However, this paper does not discuss the exact amount of increase in speed as this varies from instruction to instruction of the Coprocessor and hence can be determined only with progress in software development for the system.

Suggestions For Further Research

As has been mentioned earlier, The Vector Processor uses only three instructions. Depending on the number of Coprocessors, an equal number of vector instructions may be used to indicate the number of Coprocessors to be activated.

As an example, for a system with eight Coprocessors in parallel, seven more instructions can be used so as to activate the exact number of Coprocessors required. With the present design, all the Coprocessors are activated regardless of the number required.

In chapter VI, the Vector Instructions Decoder decodes the T3 clock of a Read bus cycle and in Chapter VII, the

Instruction	Approximate Execution Time (μs) (5 MHz Clock)	
	8087	8086 Emulation
Multiply (single precision)	19	1,600
Multiply (double precision)	27	2,100
Add	17	1,600
Divide (single precision)	39	3,200
Compare	9	1,300
Load (single precision)	9	1,700
Store (single precision)	18	1,200
Square root	36	19,600
Tangent	90	13,000
Exponentiation	100	17,100

Figure 10.1 8087 Math Coprocessor Speed Comparison

Sequential LOAD/STORE Control generates T22 clock which is the T2 clock of a bus cycle. It might be convenient to combine these two and generate them with fewer components.

In Chapter VII, a '9B' counter is used to stop filling the operands queue of the Coprocessors. Instead it is advisable to use a status line divide-by-4 counter and stop the instruction storage into the operands queue at the end of the 4th count.

There will be occurences of new bugs, as software development for the system progress, which will result in further improvement of the hardware design. With this the author wishes to conclude the paper by expressing his best wishes for further research on this project initiated by Dr.Miron.

APPENDIX A

List of IC Numbers

SN7400	2-Input NAND Gates
SN7404	Hex Inverters
SN74L08	2-Input AND Gates
SN7410	3-Input NAND Gates
SN74L11	3-Input AND Gates
SN74L20	4-Input NAND Gates
SN74L30	8-Input NAND Gates
SN7432	2-Input OR Gates
SN7442A	4-Line-to-10-Line Decoders
SN74LS74	D-Type Flip-Flops
SN7476	J-K Flip-Flops
SN74L93	4-Bit Binary Counters
SN74L154	4-Line-to-16-Line Decoders
C8087	Intel Math Coprocessors

APPENDIX B

7D02 LOGIC ANALYZER

The 7D02 Logic Analyzer is a device used as design, debugging, and trouble shooting aid for use in the development of digital systems, especially microprocessor based systems. It is capable of supporting 8-bit and 16-bit microprocessor systems. This plug-in analyzer requires a 3 wide Tektronix 7603 oscilloscope mainframe. The basic 7D02 offers :

1) a wide (28 channel) acquisition memory
2) four word recognizers
3) two general purpose counters
4) a user configurable clock
5) data qualification circuitry
6) three memories and
7) a facility for using the 7D02 for wide variety of microprocessors.

The 28 channels of 7D02 are divided into 16 address bus lines, 8 data bus lines and 4 control lines. The 7D02 can be programmed to know the status of 28 channels or it is possible to know the status of each channel separately. The unit has a option called word recognizer which can be programmed to recognize user defined patterns on each of the channels.

There are four test programs stored in the program memory of the Logic Analyzer. The user can select any of these programs which are suitable for the system under test. It is important to know that only one test is active at a time.

The format of these tests is as follows:

```
TEST #
IF event clause
THEN DO statement command clause
OR IF event clause
THEN DO command clause
ELSE command clause (optional)
END TEST #
```

The physcial connection between the Logic Analyzer and the circuit under test is made through the PM 101 Personality Module. This module serves to analyze any system which does not have specific probe for testing. Electrically, the PM 101 can buffer up to 16 data lines, 24 address lines, and 10 control lines. It has a provision for qualifier clock which is helpful in checking the valid data on address, data, and control lines.

The tests followed in using the test program are as listed:

1. Connect the PM 101 personality module to the plug-in unit of 7D02.
2. After the power up diagnostics the screen will be blank until a command is entered. If the PM 101 is not properly connected then 'failure of the system' message is seen on the screen.
3. If there is no failure then connect the leads (other end of PM 101) to the system under test.
4. Enter the immediate mode by pressing Immediate key on 7D02 and then press Display and the Program. Then press Trigger to display the Test #1 program structure. After completing Test #1 the user can go to Test #2 or end Test #1 by pressing the START/STOP key. At this time the screen shows 'Running' message.
5. When the START/STOP key is pressed once again then the screen shows whatever data it has acquired from the system under test. By making use of the scroll key it is possible to see more data than can be displayed on the screen at one time.

The exact format of the test program is shown below:

```
           TEST #1
1 IF
1 WORD RECOGNIZER #1
1 DATA = XX
1 ADDRESS = XXXX
1 CO=X    C1=X    C2=X    C3=X
1 C4=X    C5=X    EXT TRIG IN = X
1 THEN DO
1 TRIGGER
1      0 - BEFORE DATA
1      1 - CENTERED
1      2 - AFTER DATA
1      3 - ZERO DELAY
1   0 - SYSTEM UNDER TEST CONT.
1   0 - STANDARD CLOCK QUAL.
END TEST 1
```

If one wants to trigger on a paritcular data, address or a control output then the value of that data can be entered in the places of don't cares (XXX), in the above program. So, whenever that particular value appears on the data , address or control line the screen displays the content of the data acquired by the acquisition memory.

For understanding and debugging of The Vector Processor, the Logic Analyzer may be used and programmed to trigger on the occurance of a vector instruction. The following procedure is recommonded:

1) Connect the data probes of the Logic Analyzer to the the data lines of The Vector Processor.
2) The qualifier probes C0, C1, C2, and C3 are connected to the status lines S0, S1, and queue status lines QS0, and QS1 respectively.
3) Connect the qualifier probe C4 to the decoder 74L154 (pin 17) of The Vector Processor.
4) Connect tha address probes, one out of the first 4 address lines to the Busy line of the 8087 or any of the required test points.

In the program make C4=1 and run the program. The Logic Analyzer will trigger on occurance of the vector instruction DF FF. By observing the Busy lines of all the 8087's through the address lines, one can see the serial loading and parallel execution of the Math Coprocessors.

REFERENCES

1. Intel Microsystem Components Handbook: Volume I and Volume II, Intel Corporation, California, 1984.

2. Morris, M.: Computer System Architecture, Prentice-Hall, Inc., New Jersey, 1982.

3. Intel iAPX 86/88, 186/188 User's Manual: Hardware Reference, Intel Corporation, California, 1985.

4. The TTL Data Book for Design Engineers, Texas Instruments, Inc., Texas, 1981.

5. Technical Reference Manual: IBM Personal Computer, IBM, Florida, 1983.

6. Intel iAPX 86/88, 186/188 User'Manual: Programmer's Reference, Intel Corporation, California, 1985.

7. Walter, A.T., A. Singh: 16-Bit Microprocessors, Prentice-Hall, Inc., New Jersey, 1985.

8. Tektronix 7D02 Logic Analyzer Instruction Manual, Tektronix, Inc., Oregon, 1981.

9. S.S.Reddi., E.A.Feustel.: A Restructurable Computer System, IEEE Transactions on Computers, VOL. C-27, No. 1, January 1978.

10. Disk Operating Systems Manual: IBM Personal Computer, Version 2.10, Microsoft, IBM, Florida, 1983.

11. Y.P.Raghuram,: Interfacing of Two Non-Identiacal Printers to the IBM PC, SDSU, 1986.

12. D. Miron,: Vector Processor, SDSU, 1985.

13. Tektronix PM101 General Purpose Personality Module: Instruction Manual, Tektronix, Inc., Oregon, 1980.

14. Walter, A.T. and A. Singh: 8088 Assembly Language for IBM PC, Printice Hall, Inc., New Jersey, 1984.

15. Macro Assembler Reference: IBM Personal Computer, Version 2.00, IBM, Florida, 1984.