R6500

MICROCOMPUTER SYSTEM

HARDWARE MANUAL

R6500

Rockwell International

$5.00
Document No. 29650 N31
Rev. 1 August 1978

NOTICE

Rockwell International reserves the right to change this product and its specifications at any time without notice to improve its design or performance.

No responsibility is assumed by Rockwell International for use of this information; nor for any infringement of patents or other rights of third parties which may result from the use of this information.

PREFACE

The Rockwell R6500 Microcomputer System combines the best features of second-generation microcomputers into a product line that is a leader in both price and performance. A growing array of products and a unique microprocessor (CPU) family provide the R6500 System user with answers to the most complex microcomputer design problems confronting today's programmers and designers.

Integrated circuit fabrication techniques have moved microcomputers to the forefront of complex, sophisticated components. The R6500 System benefits from an advanced but proven integrated circuit process technology which is directly responsible for the high-performance characteristics obtainable in the single-supply, 5-volt usage of the R6500 System.

The N-channel, silicon gate circuit technology which is applied throughout the R6500 System is further enhanced by use of "depletion loads," to provide greater speed, lower power, and smaller chip size than previous processing approaches. Ion implantation techniques are basic elements in providing control and stability of all processing parameters necessary to achieve the electrical characteristics of the R6500 product line. These characteristics provide a price/performance combination which establishes the R6500 family as the product offering best meeting the economic and technical demands of today's system designs.

A word of explanation is in order regarding the R6500 Microcomputer System's microprocessor (CPU) "family" concept, since it is around this family that the R6500 System is built.

The R6500 System's microprocessor (CPU) family includes a series of 8-bit devices which offer the customer a wide range of options and capabilities. For the single-application customer, a varied selection of devices is at his disposal in choosing the one which best meets his specific needs. The "microprocessor family" concept has an even greater impact on the user who has a variety of applications, each of which can best be served by a specific

member of the family. It is important to this user that all of the different microprocessors he selects maintain compatibility — both hardware (from the standpoint of bus and electrical specifications) and software. The R6500 System product line offers the first microprocessor (CPU) family to achieve such a level of compatibility because it was indeed conceptualized as a totally software and hardware compatible family of microprocessors (CPUs) offering a range of performance options from which the designer can select. The R6502 and R6512 are the two 40-pin members of the family, each offering 65K bytes of addressable memory. The R6503 through R6507 and R6513 through R6515 are 28-pin versions with various options of addressing capability and control functions from which to choose.

The R652X Series represents Peripheral Input/Output devices, the first being R6520 which is a direct replacement for the Motorola MC6820 Peripheral Interface Adapter (PIA). The second device of this series is the 6522 Versatile Interface Adapter (VIA). Subsequent members of this series will include devices with expanded I/O capabilities.

The R653X Series represents combinational devices — those consisting of various tradeoffs in RAM, ROM, I/O, and Timing. The first of these is the R6530 which contains 1K bytes of ROM, 64 bytes of RAM, an Interval Timer and 16 I/O lines. The second is the R6532 RAM/IO/Timer with 128 bytes of RAM, an Interval Timer and 16 I/O lines. Subsequent products in this series will provide the customer with different combinations and new implementations of I/O, Timing and Memory.

All of the R6500 product-line subsystems utilize the same fabrication techniques and meet identical electrical specifications. With this family of compatible products at his disposal, today's designer has available the elements necessary to develop a system configured to meet the most demanding tasks.

The R6500 family is compatible with standard Random-Access Memories (RAMs), including the 2102, 2111, 2114 and the 2115.

To allow for minimum I/O cost and maximum user flexibility, all of the R6500 products are compatible with the M6800 bus structure.

TABLE OF CONTENTS

LIST OF FIGURES

SECTION 1

THE R6500 MICROCOMPUTER SYSTEM

The past several years have seen the development of an exciting new concept in electrical design. The microcomputer started out as a relatively simple, difficult-to-use programmable device capable of handling simple control or computational problems. However, it has since matured into a powerful, inexpensive, easy-to-use device capable of controlling all but the most complex of systems.

Conventional system design is rapidly being revolutionized by the component that forms the heart of the microcomputer -- the large-scale, single-chip programmable microprocessor (CPU). Three primary attributes of microprocessor-based systems are bringing about this revolution:

1. Microprocessors allow a significant reduction in overall systems cost for products currently in production. Re-design of products around the microprocessor is permitting many manufacturers to develop or maintain a price advantage over competitors.
2. The reduction in cost of microcomputer systems is opening up vast new markets for microprocessors. A great number of systems which were simply impossible or were at best impractical, are being designed and marketed today using the modern, low-cost microprocessors.
3. At the same time that the price of microprocessors is dropping, their capability is rapidly expanding, thus allowing them to be designed into more systems than ever before.

Anyone contemplating a new design or trying to reduce cost in an existing design must first ask himself if a microprocessor will solve his problem.

The success of the microprocessor is based on the fact that it permits the design engineer and programmer to apply their expertise in solving a multitude of design problems using cost integrated effective circuits. A small number of large integrated circuits can be configured to solve design problems from the simplest to the most complex.

If the same integrated circuits are employed to solve a multitude of unique designs, the first question one must ask is, "What makes the designs unique?" The answer is: Programming. Although many different designs may share common hardware, each has its own unique program. This brings us to another very important characteristic of microcomputers. The integrated circuit which makes each system unique is the "Read-Only Memory" (ROM) which stores the system program. It is relatively easy for the integrated circuit manufacturer to establish the particular pattern which uniquely defines the data in a ROM. As a result, the typical charge for "designing" a ROM is generally less than 10% of the cost of designing a totally custom logic chip. Further, the user benefits from a high-volume standard product which is still unique for his own application due to the "customization" of one element of his system.

It will probably surprise many designers, approaching the subject of microcomputer design for the first time, to discover that designing a system around a microprocessor is much the same as designing around conventional logic. The total approach is the same; the process differs only in the implementation of each step.

A brief examination of the system design process will help to put microcomputer design in perspective and will also assist in clarifying the purpose of this manual. One can expect to perform the following steps in designing a microcomputer system:

1. Define the requirements of the system. What functions should it perform?
2. Define basic system components.
3. Complete design details.
4. Build and test prototypes.
5. Finalize design and begin production.

Step 1 is true for any system and, in general, for any product. Step 2 is the first point of departure for microprocessor-based designs. It is at this point that the designer must consider the possibility of using a microprocessor in his system. For the very cost-sensitive application he must look very carefully at total systems cost. Can a microprocessor do the job within the price constraints imposed? At the other end of the design spectrum, the system designer must evaluate the capability of microprocessors to assure himself that the available devices can in fact perform the required function. Will a microprocessor be fast enough to run the system? Will the system require more than one processor?

The purpose of this manual is to teach the designer how to effectively configure a microprocessor-based system and to evaluate the performance of the system. After this step, the design will be completed by development of the system program. Implementation of the system program is discussed in the Program Manual.

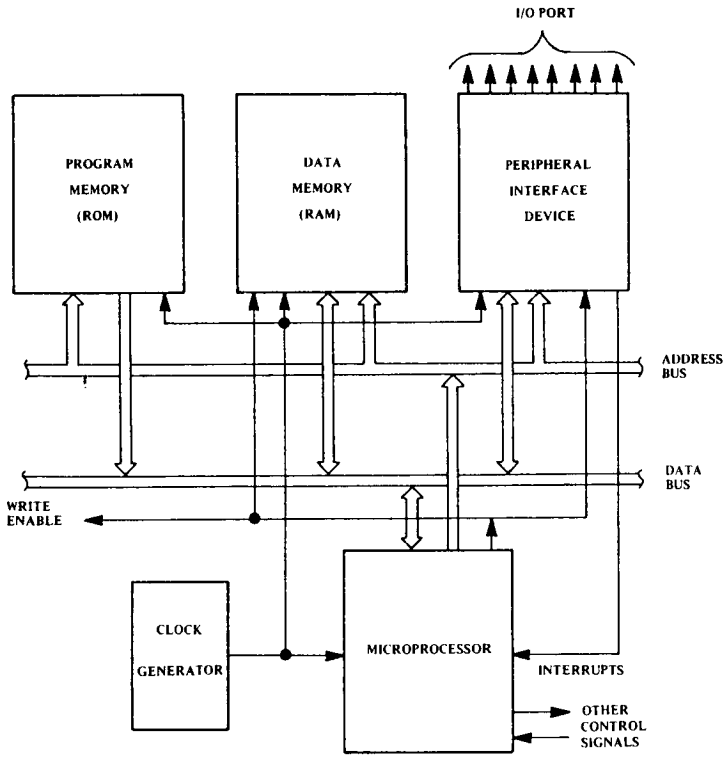
1.1 INTRODUCTION TO MICROCOMPUTER SYSTEMS

1.1.1 Organization of a Microcomputer System

Figure 1-1 illustrates the basic organization of a microcomputer system. It is important that the designer understand the operation of each component as well as the operation of each data path in the system. Each of these is discussed separately below. In addition, the following discussion describes the operation of the overall system and the use of the various signal paths.

1.1.2 Basic Operation

The microcomputer is a system which can be characterized as very simple in its detail and very complex in its overall operation. It carries out rather complex tasks by performing a large number of simple operations. Control of the system is primarily the responsibility of the processor. By putting out addresses to program memory, it controls the sequence of operations performed, and by interpreting and executing the instructions which it receives from the program memory it controls the actual operations carried out by the system. The processor is by far the most complex device in the system. For this reason, it is important to overall system cost that this part stay the



Organization of Microcomputer System
FIGURE 1-1

same for many different applications. In this way, the relatively high development cost can be shared by thousands of users. In addition, those thousands of users can all benefit from the economics of large-scale production.

The processor causes the system to perform the desired operations by reading the first instruction in the program, and performing the very simple task dictated by the specific pattern of bits in this instruction (referred to as "executing" that instruction). It then goes on to the next instruction in the program and executes it. This simple operation of fetching an instruction and executing it is performed over and over, each time on the next instruction in sequence. In this way the program instructs the processor to bring about the desired system operation.

1.1.3 Addressing Terms and Concepts

Before entering into a detailed discussion of the system operation, it would be useful to define a few terms and to introduce a few concepts concerning addressing. This should assist in an understanding of the detailed discussions which follow.

BIT

The term "Bit" is a general term referring to anything that can be assigned to binary value, i.e., anything that can be given a value of 0 or 1. Thus, an eight-bit data bus is a set of 8 lines which can be assigned a value of logic 0 or logic 1. On these lines, the logic values are represented by two different voltages or currents. Similarly, a 16-bit binary display can be built with 16 individual lamps. The logic 1 is represented by the lamp being on.

In this text, reference is made to an 8-bit data bus, a 16-bit address bus, 4 bits of data, 8-bit registers, etc. In all cases, definition of a bit remains the same.

ADDRESS SPACE

The concept of an address space is very useful in understanding microcomputer systems. The term "address space" refers to the total set of addresses which the microprocessor can generate. For example, if a processor had only 4 address lines, it could generate the addresses 0 - 15 (binary 0000

to binary 1111). This would not be adequate for any microcomputer operation and, consequently, the typical processor has between 12 and 16 address lines. Since each line can assume a value of 0 or 1, these devices can usually address from 4,096 to 65,536 separate addresses. Figure 1-2 contains a pictorial representation of the address space available in a typical 8-bit microcomputer with 16 address lines. In addition to the general address space, this figure introduce the PAGE concept discussed below.

THE ADDRESS PAGE

The concept of a PAGE in memory is very important in 8-bit microcomputer systems. The internal organization of an 8-bit processor is around 8-bit registers, 8-bit parallel data paths, etc. Most arithmetic operations, logic operations, etc. take place on 8 bits of data at a time. Similarly, the 16-bit counter which determines which instruction is being executed is actually divided into two 8-bit busses. One contains bits 0 - 7 (low-order address bits), and the other contains bits 8 to 15 (high-order address bits). With this in mind, one can think of the address space shown in Figure 1-2 as consisting of 256 blocks, each consisting of 256 specific address locations. Each of these blocks is referred to as a "PAGE" of memory. The high-order 8-bits of the address (ADH) therefore indicate in which page the address is located, and the low-order 8 bits (ADL) indicate a specific address on that page.

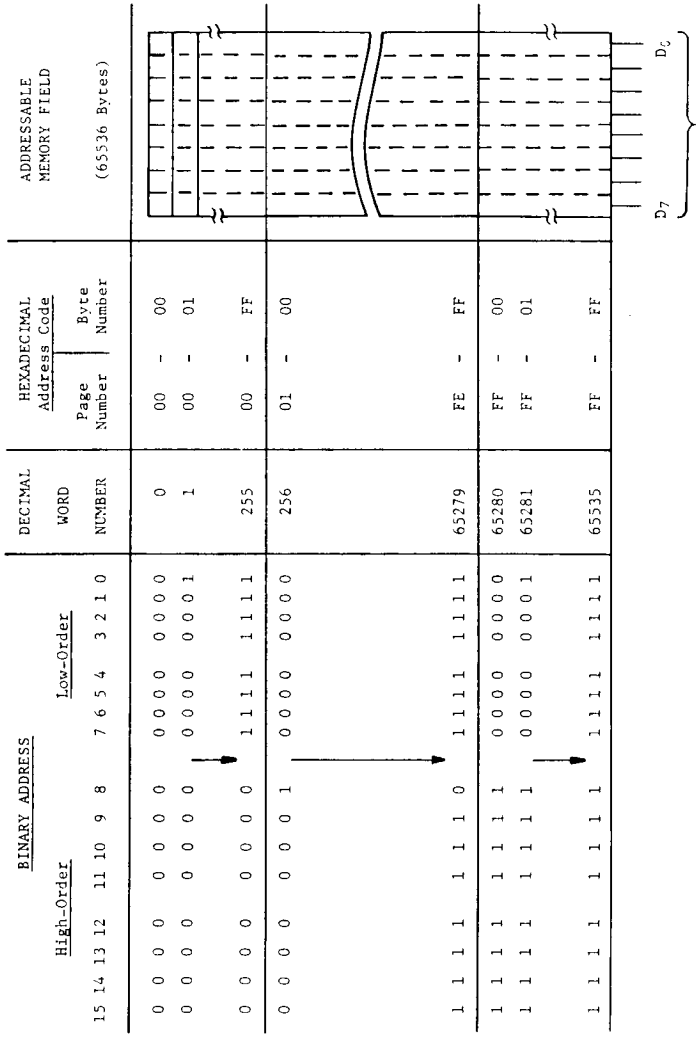
The first page in memory (ADH = 00) is referred to as page 0. The next-high-order page (ADH = 01) is referred to as page 1, etc.

1.1.4 System Components

The block diagram in Figure 1-1 shows the basic components which comprise all microcomputer systems. Each block in the diagram may consist of one or more integrated circuits and, in fact, several functions may be combined into single chips. However, the basic operation of each remains the same.

CLOCK GENERATOR

The clock generator produces a continuous waveform which is normally used to control all signal transitions within the system. It acts as the "heart" of the system. In the typical microcomputer system the address bus will change during one half of the clock cycle, and the data will be



Address Bus and Relation to Memory Field

FIGURE 1-2

transferred during the second half. In addition to interpreting the address, data and control lines, the processor and support chips must also examine the system clock to know when to put out data or when to latch in data generated by another device.

PROGRAM MEMORY

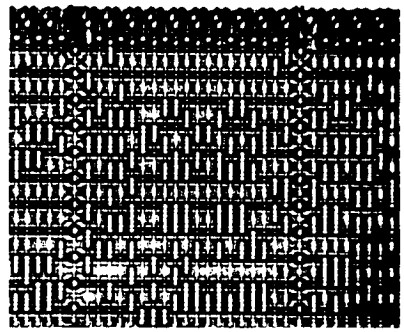
The program memory stores the sequence of instructions which comprise the system program. Like any memory, this unit puts a pattern of 1's and 0's on the data bus in response to the address on the address bus input. Each unique address selects a set of 8 binary bits and places these data on the data bus. Note that it does not matter where the address is generated or where the data are used; the memory simply obeys the rule that, given an address, it will put the corresponding 8 bits of data on the data bus.

A unique characteristic of most microprocessor-based systems is that the program is usually stored in Read-Only Memories (ROMs). The data are stored in a fixed pattern of bits in the memory. Figure 1-3 shows a section of a semiconductor ROM.

Since the data are stored in the physical configuration of the device, the data will not be lost when power is disconnected from the chip. In addition, it is necessary only to insert the device into its socket to provide the system program. The term "Read-Only Memory" refers to the fact that, in system operation, it is impossible for the processor to cause data to be stored in the device. The processor can "READ" the data stored in the device during the manufacturing process. "READING" a memory involves the simple process of supplying an address to the device to obtain the corresponding 8 bits of data on the data bus.

DATA MEMORY

For temporary storage of input data, the results of arithmetic operations, etc., the microcomputer uses a Read/Write Memory, commonly referred to as a RAM (Random-Access Memory). The processor can store data in the RAM (called "WRITING" the RAM), or it can read back the data it has stored. As in the ROM, each address corresponds to eight memory cells.



Portion of Read Only Memory Matrix
FIGURE 1-3

however, in a RAM the data must be placed into the memory by the processor and are stored in cross-coupled latches. Turning off the power to a RAM will cause the loss of all data stored there, and the data are said to be "volatile." Data in a ROM are not lost when power is disconnected from the device and the stored data are referred to as "non-volatile".

"WRITING" data into a RAM takes place when the Write-Enable signal goes to the write state. At this time the data on the data bus will be stored into the eight memory cells corresponding to the address on the address bus. The processor can READ this same data by supplying the proper address and keeping the Write-Enable line in the Read state.

INPUT/OUTPUT DEVICES

The Input/Output Devices are the circuits which interface the printer, keyboard, displays, etc. to the processor. These allow the processor to read data from the keyboard, to test the state of sensors and switches, and to display or to print the results of internal operations.

No matter where data are generated, they must be, in the form of 1's and 0's before the processor can work with them. Similarly, actions to be initiated by the processor must be triggered by 1's and 0's transferred by the processor to a set of output lines.

The transfer of data from the processor to an output device is usually accomplished by "WRITING" the data out in much the same manner as the processor writes data into RAM. Each set of 8 input or output lines (referred to as "PORT") is given an address, and the processor simply writes data to that address. For each "1" written out to the peripheral port an output is set high, and for each "0" the corresponding output is set low.

Although the basic concept of peripheral control is simple, the actual implementation of these interfaces can involve many sophisticated techniques designed to allow the processor to maximize its ability to control peripherals and perform internal operations concurrently. These techniques are discussed in detail in Section 3.

THE MICROPROCESSOR

At first glance it may seem strange to discuss the support chips in the microprocessor-based system before mentioning the processor. However, this approach is necessitated by the fact that most of the inputs and outputs on the processor are aimed at properly controlling the support chips and peripheral devices discussed above.

The address bus, the bidirectional data bus and the Write-Enable line allow the processor to exercise direct control over the rest of the system. The address bus puts out addresses to control the source or destination of data transfers. These addresses are derived from various sources within the processor. During the fetch of instructions from program memory, the addresses are usually derived from a counter which controls execution of sequential instructions. Addresses for data transfers between the processor and RAM are usually derived directly from the program or are calculated from the data in the program and data in internal registers.

The bidirectional data bus serves as a path for transferring data into and out of the processors. The direction of the data transfer is determined by the Write-Enable line.

Another special function found in modern microcomputer systems is the interrupt. This function allows the peripheral devices to directly affect the operation of the processor. When the interrupt signal is generated, the processor usually completes its current instruction and then, under program control, will respond to the interrupt. The importance of this function is that it allows the processor to execute the system program without requiring the system program to monitor the status of the peripheral device. The software which handles the operation of each peripheral will be executed only when required.

1.2 INTRODUCTION TO THE R6500 MICROCOMPUTER SYSTEM

The Rockwell R6500 microcomputer system consists of the 40-pin R6512, microprocessor; the 40-pin R6502 microprocessor, which has clock drivers on-chip; and eight 28-pin processors, the R6503 through R6507 and R6513 through R6515. Each of these devices is aimed at a specific range of applications. Therefore, it is important to develop an understanding of the capabilities of each and the differences between them.

The R6512 has application in existing M6800 systems where conversion to the R6500 system is to be performed. This processor requires the full high-level two-phase clocks of the M6800 system. The R6502 is expected to find application in all new designs which require a full 16-bit address bus. However, in the small cost-sensitive system, the 28-pin processors can represent a savings both in processor cost and in printed circuit board area.

1.2.1 The Microprocessors

The R6502 should be used in all new designs which require the capability of the 40-pin processors. The clock drivers can be driven with a single TTL level square wave or with the internal oscillator. The frequency of operation of the internal oscillator can be set by attaching an R-C combination to the chip and, if the clock stability is required, by attaching a crystal between the oscillator and ground. This feature totally eliminates the problems encountered in generating MC6800-type clock signals.

The R6502 provides a full 16-bit address bus, 8-bit bidirectional data bus, and two interrupts. In addition, the R6502 provides a sync signal which indicates those cycles in which the processor is fetching an operation code from program memory.

Eight 28-pin versions of the processor are available. They differ in the number of address lines and the clock generation methods required, the number of interrupts provided. Having all three options available allows the designer to tailor his processor to his particular application.

The R6504 and R6507 provide a total of 13 address pins and can, therefore, address a full 8K bytes in its memory space. The R6504 provides only one interrupt request input, $\overline{\text{IRQ}}$; the R6507 provides a RDY input instead of $\overline{\text{IRQ}}$. The non-maskable interrupt ($\overline{\text{NMI}}$) is not included in the pinouts of this device.

The R6503 and R6505 provide one less address line. In the R6503 the missing address line is replaced with a second interrupt input, $\overline{\text{NMI}}$, and in the R6505 it is replaced by the RDY signal. All other functions on these processors are the same. The details of each of the pins are discussed in the following sections.

The operation of the various busses, control signals, etc. is identical on all R650X products, with all processors obeying the system specifications discussed in Section 1.2.2.

The R6513 is the slave (clocks driven in) version of the R6503, the R6514 is the slave version of the R6504, and the R6515 is the slave version of the R6505.

1.2.2 Bus Structure

The R6500 microcomputer system is organized around two primary busses. Each bus consists of a set of parallel paths which can be used to transfer binary information between the devices in a system. The first bus, known as the ADDRESS BUS, is used to transfer the address generated by the processor to the address inputs of the memory and peripheral interface devices. The processor is the only source of addresses in a normal system, so this bus is

referred to as "unidirectional." The address bus consists of 16 lines on the R6502 and R6512, allowing those processors to access (READ or WRITE) up to a total of 65,636 memory words, registers, etc. In the R6503 through R6507 and R6512 through R6515 the address bus contains fewer lines; therefore, they operate with a smaller "address space." This is discussed in detail in Section 1.1.3.

The data bus in the R6500 microcomputer system consists of an 8-bit bidirectional data path. These lines transfer data from the processor to the selected memory word, etc. during a WRITE operation and from memory into the processor during a READ operation. All data and all instructions are transmitted on the data bus.

The direction of the data transfers is controlled by the READ/WRITE (R/W) line on the processor. This line performs the Write Enable function described in Section 1.1.4. As long as the R/W line is high (> 2.4V DC), all data transfers will take place from memory to the processor (READ operation). This line will go low only when the processor is going to WRITE data out to memory.

As in most microcomputer systems, the timing of all data transfers is controlled by the system clock. The clock itself is actually two non-overlapping square waves. This two-phase clock system can best be thought of as two alternating positive-going pulses. This text will refer to the clocks as "Phase 1" and "Phase 2." A "Phase 1" clock pulse is the positive pulse during which the address lines change, and a "Phase 2" clock pulse is the positive pulse during which the data is transferred. The timing of the signals on the Address Bus, Data Bus, and R/W line is shown in Figures 1-4 through 1-7. All signal transitions are specified with respect to the Phase 1 and Phase 2 clock signals. In particular, the address lines and the R/W line will stabilize during Phase 1, and all data transfers will take place during Phase 2.

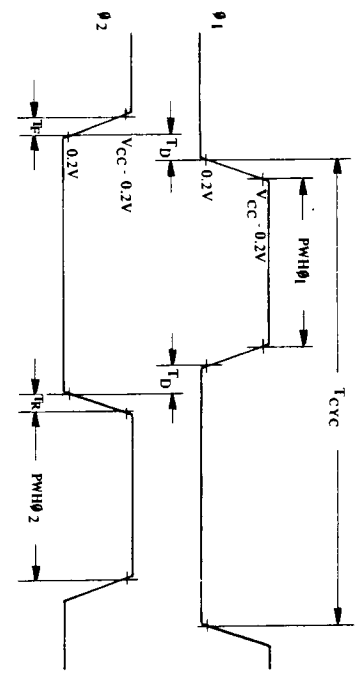
The specific timing specifications for operating at a 1-MHz clock rate are also given in Figure 1-4. Note that the sequence of operations will be the same for all processors. However, these timing specifications will change for devices which are specified to operate faster than 1.0 MHz.

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
Cycle Time	T_{CYC}	1.0 µs	--	--	µs
Clock Pulse Width Ø1 (Measured at V_{cc}-0.2v) Ø2	PWH Ø1 PWH Ø2	430 430	--	--	ns
Rise and Fall Times (Measured from 0.2V to V_{cc}-0.2V)	T_F, T_R	--	--	25	ns
Delay time between Clocks (Measured at 0.2V)	T_D	0	--	--	ns

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	UNIT
Read/Write Setup Time from R650X	T_{RWS}	--	100	300	ns
Address Setup Time from R650X	T_{ADS}	--	200	300	ns
Memory Read Access Time T_R T_{CYC} - (T_{ADS} - T_{DSU} - tr)	T_{ACC}	--	--	575	ns
Data Stability Time Period	T_{DSU}	100	--	--	ns
Data Hold Time	T_H	10	30	--	ns
Enable High Time for DBE Input	T_{EH}	430	--	--	ns
Data Setup Time from R650X	T_{MDS}		150	200	ns

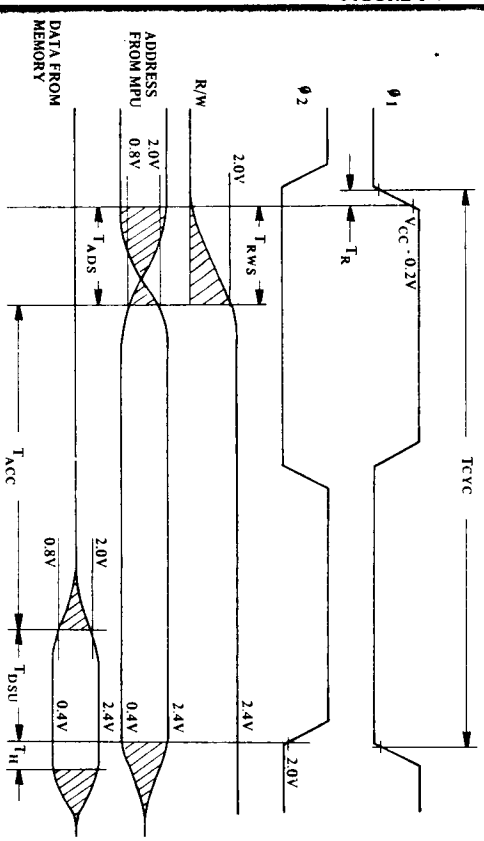
Clock and Read/Write Timing Table (1MHz Operation)

FIGURE 1-4



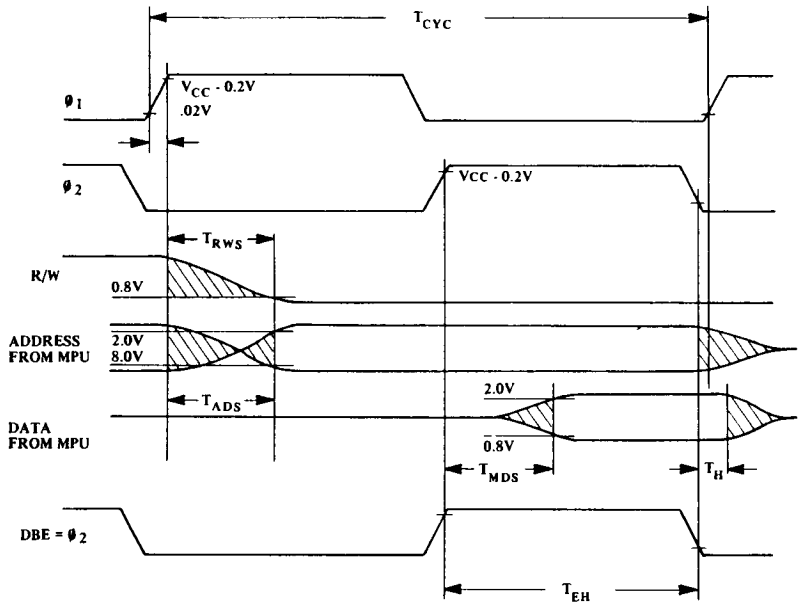
Two Phase Clock Timing

FIGURE 1-5



Timing for Reading Data from Memory or Peripherals

FIGURE 1-6



Timing for Writing Data to Memory or Peripherals
FIGURE 1-7

The address is guaranteed to be stable 300 ns after the leading edge of Phase 1, and the data must be stable 100 ns before the trailing edge of Phase 2. At 1.0-MHz operation, this allows the memory devices approximately 575 ns to make data available on the data bus. Although there are many factors which determine the actual data and address generated within the system, it is important to keep in mind that the basic operation shown in Figures 1-5, 1-6 and 1-7 does not change. These figures specify the system bus discipline which applies to all R6500 processors and support chips.

1.2.3 Interrupt Structure

Through the generation of processor interrupt signals, the peripheral devices (printers, keyboards, etc.) can request service from the processor. Although this technique is relatively simple in concept, the proper generation and control of interrupts is one of the most important problems which the designer will face. Total system capability can be greatly expanded if the processor is required to execute the peripheral software only when it is absolutely necessary. This is the goal of a well-planned interrupt structure. The interrupt structure is very much a systems sophistication problem since the entire system which must properly respond to the interrupt inputs. In fact, the actual signals to which the system must respond are usually applied to the inputs of a peripheral interface device. In this device, the interrupts are enabled, disabled and latched until the interrupt is processed. The peripheral interface device generates signals which meet the requirements of the processor interrupt inputs.

There are two interrupt input lines to the microprocessor, $\overline{IRQ}$ (Interrupt Request) and $\overline{NMI}$ (Non-Maskable Interrupt).

Since the requirements of the two interrupt inputs differ, they will be discussed separately below. The response of the processor to these inputs is very similar, however, after the interrupt is recognized. For this

reason, the internal operation of the processor during interrupt servicing is discussed in the detailed analysis of the processor chip. The present section of the manual concentrates on the system-level considerations which are required to assure proper operation of the interrupt structure.

APPLICATIONS FOR INTERRUPTS

One of the most important tasks facing the microcomputer system designer is the determination of those signals which will cause processor interrupts and those operations which will take place in response to the interrupts. A detailed discussion of these considerations is included in Section 3 of the manual; however, a few examples of interrupt-driven operations will be presented here to help the designer develop an understanding for why this technique is used extensively in microcomputer systems.

Example 1 -- A Fully-Decoded Keyboard

The problem of data entry is solved in many systems by a keyboard. In small systems, the interpretation of the binary code associated with each key can be determined by the processor. However, in large data terminals, the keyboard usually includes an encoder which generates the unique code corresponding to each key. When a key is closed, the corresponding code is placed on the output pins and a strobe signal is generated to indicate that a key has been pressed.

The keyboard represents a perfect candidate for interrupt-driven operation. The interrupts occur relatively infrequently and the operation to be performed is relatively simple. The keyboard strobe line is connected directly to an interrupt input on a peripheral interface device. Each time a strobe signal is generated, an interrupt occurs, the processor reads the data on the peripheral port into memory, analyzes these data, and then returns to the program that was in process. If no keys are pressed, the processor spends no time at all in servicing the keyboard.

Without the interrupts, the processor would have to read the keyboard data into memory periodically in order to detect an active key. This operation would be performed about every 50 to 100 ms. In addition to detecting an active key, the processor must make sure that each separate activation of a key is detected once and only once. (This is discussed later in this section.) This software is much more complex than the

simple interrupt routine. Another drawback of non-interrupt processing is that the processor is required to spend a periodic portion of its time on the keyboard. In many systems, this is not a problem, but in large terminals, etc., the time spent checking for keyboard strobes could be better spent in other operations. The designer must, therefore, ask himself if the system under development is such that the processor can perform the keyboard strobe checking function while still completing its other tasks.

Example 2 -- A Scanned Display

Although time is a major factor in determining the necessity of interrupts, the interrupt technique can also be extremely useful when performing parallel operations. A prime example of this can be found in a system which contains a digital display and/or printer.

A digital display is usually "scanned" in such a way that each digit is driven for a short period of time in sequence. The entire display is scanned at a rate faster than the eye can detect. However, it should be noted here that the display requires scan-related attention from the processor at fixed intervals. It is very difficult for the processor to calculate repetitive time intervals while it is performing its normal system program routines. The processor would much prefer to run the system program without consideration for the display time intervals -- only executing the display software only when it is required.

A solution to the above problem is the generation of processor interrupts at fixed intervals, employing an external counter or clock. Each time an interrupt occurs, the data for the next digit in the display are placed on an output port. The processor then returns to the program it had been executing.

Both of the operations described above represent solutions to system problems. Events which happen very infrequently, and events which must be performed in parallel with other events or in parallel with the main system program, should be seriously considered as candidates for interrupts. Additional considerations are described in Section 3 of this manual; however, it is important to note here that the typical system may have several sources of interrupts, each with its own timing and each with its own set of operations which must be performed in response to the interrupts.

INTERRUPT PRIORITIZING

After a careful analysis of the total system and a determination of all the sources of interrupts, the designer must ask himself, "What happens if more than one interrupt source requires attention at one time?" A priority level must be established between the various interrupt sources. Which ones must be taken care of within a very short period? Which ones can be put off for a while? This prioritizing and the technique for selecting among several concurrent interrupts is very important to the system operation and should be established early in the system development process.

The R650X-based system can employ several hardware methods of determining the highest-priority active interrupt. These usually involve using a special "priority encoder" which allows the processor to go directly to the software which services the highest-priority interrupt. After this is complete, it will go to the next higher priority and execute that software. However, the R650X family provides a much less expensive method of interrupt prioritizing -- the "polled" interrupt. With this technique, each time an active interrupt source is detected, the processor executes a "polled" interrupt program that interrogates the highest priority interrupt, then the next highest, and so on until an active interrupt is located. The program services that interrupt and returns to the "polled" interrupt program and continues to interrogate the next highest priority interrupt until all have been interrogated, or clears the interrupt disable to allow nested interrupts. The "polled" interrupt program is always executed when an interrupt occurs, so that all interrupts that occur concurrently will be serviced in order of priority level.

Several hardware techniques for prioritizing interrupts are discussed in Section 3 of this manual. The next section, however, describes the system interconnect which allows use of the simple "polled" interrupt.

SYSTEM INTERCONNECT FOR INTERRUPTS

In the simple "polled" interrupt technique for prioritizing interrupts, the interrupt software actually determines the highest-priority active interrupt. The $\overline{IRQ}$ or $\overline{NMI}$ interrupt request signals simply cause the processor to jump to the polling software.

For this reason, it is possible to "OR" the various interrupt signals together to form the signal for the processor. Any active interrupt source will then cause the processor to do the interrupt polling and

servicing operation. Provision for generation of this OR function is provided in the R6500 family peripheral interface devices. Since these peripheral adapters perform many of the enabling and latching functions necessary for proper interrupt servicing, the peripheral adaptor interrupt output then provides the actual signal which interrupts the processor. These interrupt outputs can be "wire-OR'd" by connecting them all together and then connecting this single line tc the processor. This input should then be pulled to +5V with a resistor. Any one of the interrupt outputs on the peripheral adaptors can then pull this interrupt low. This simple configuration is shown in Figure 1-8.

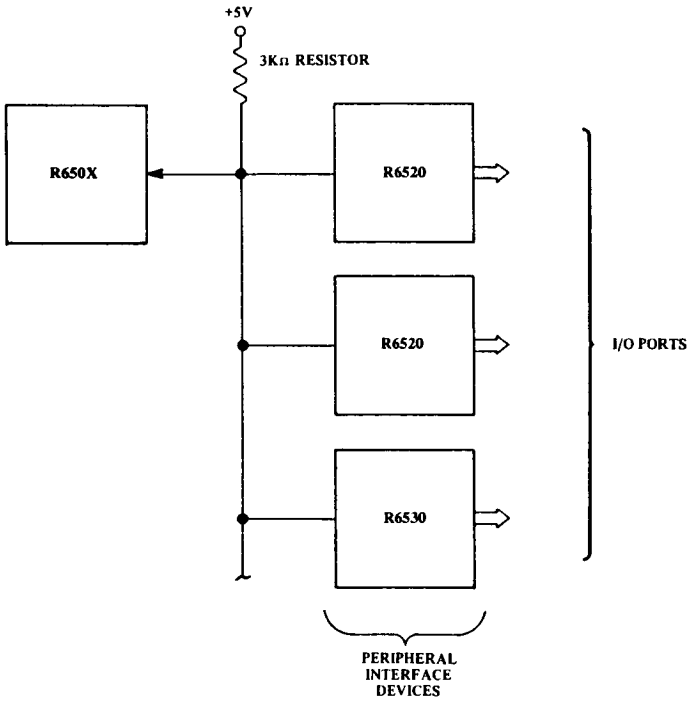
INTERRUPT SERVICING

Although a great deal has been said previously about the process of establishing interrupts and determining just what happens in response to an interrupt, it would be useful to detail the sequence which takes place when an interrupt is recognized by the processor. This will establish a basis for understanding of the details of the processor interrupt inputs.

An interrupt request is signaled by a GND (< 0.4V) signal on the interrupt request input. This interrupt will be recognized after the processor completes the instruction which it is currently executing. The next step is to store enough of the contents of the internal processor registers to assure that the processor can resume execution of the program which was interrupted. In particular, the Program Counter and the Processor Status Register are stored in a series of memory locations specified by another internal register, the Stack Pointer. As discussed in Chapter 9 of the Programming Manual, saving the contents of the Program Counter and Processor Status register uniquely defines, in memory, the state of the microprocessor at the time the interrupt occurred. The processor then goes to two fixed locations in memory to determine the address low and address high of the interrupt software.

The operation to this point is automatic and is determined by the internal processor logic. After the processor has properly set the address bus, execution of the interrupt program commences. Everything which occurs subsequently is determined by the system software.

The total interrupt software described above will consist of a complex combination of polling and interrupt servicing routines. However, unless



Interrupt Wired-OR Hardware Configuration from Peripheral Interface Devices to Microprocessor

FIGURE 1-8

a hardware prioritizing scheme is used, the actual system interconnections will not become any more complex than that shown in Figure 1-8.

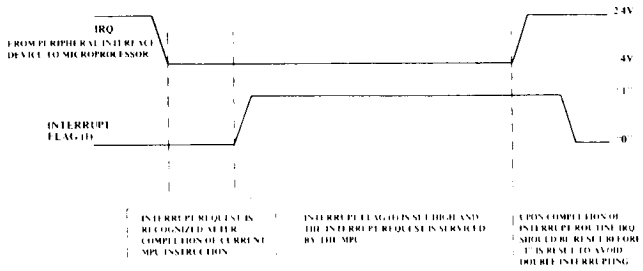
INTERRUPT REQUEST ($\overline{IRQ}$)

As stated earlier, the two interrupt lines for the microprocessor are $\overline{IRQ}$ and $\overline{NMI}$. The requirements for proper operation of the maskable Interrupt Request input ($\overline{IRQ}$) are more stringent than for the second interrupt input, $\overline{NMI}$. This is due primarily to the fact that $\overline{NMI}$ is edge-sensitive. With the $\overline{IRQ}$ input, the processor will be interrupted any time the signal on $\overline{IRQ}$ is GND (< 0.4V) and the internal Interrupt Disable flag is cleared. The Interrupt Disable flag (I) is a single bit in the internal Processor Status Register. The details of this register are described in Section 3.2 of the Programming Manual.

In the processing of interrupt request from the $\overline{IRQ}$ input, the I flag is extremely important. This is the element which assures that an interrupt will be recognized and serviced only once for each request and only when an interrupt is desired. This is described in detail below.

Figure 1-9 details the sequence of operations which should take place during the servicing of an $\overline{IRQ}$ interrupt. A positive or negative transition of the signal from the peripheral device (printer, keyboard, etc.) is detected on the edge-sensitive inputs to the peripheral interface device. If the interrupt is enabled within the peripheral interface device, the interrupt request output ($\overline{IRQ}$) on this chip will go low. The interrupt condition is latched within the peripheral interface device to allow sufficient time for the processor to poll the interrupt sources, assuring that the interrupt signal will not be cleared before the polling can be completed. This latch is reset by the processor as it executes the software associated with that interrupt. Details of this operation are described in Section 2.

The Interrupt Disable flag (I) is set automatically when the processor recognizes an interrupt. This assures that this same interrupt will not be recognized again. Resetting this flag can be performed manually with an instruction in the program or automatically with a "Return from Interrupt" instruction. It is very important that "I" not be cleared before the interrupt input is reset. Performing the "Clear I" instruction too early in the program can cause this same interrupt to be recognized again. The processor will then proceed to service this as if it were a new interrupt.



Sequence to Service IRQ

FIGURE 1-9

NON-MASKABLE INTERRUPT ($\overline{NMI}$)

The NMI input to the processor is edge-sensitive. To cause an interrupt to occur, there must be a negative transition of the signal on the $\overline{NMI}$ input. This negative transition will cause a single interrupt to occur. After servicing the interrupt, the processor will ignore this input until the $\overline{NMI}$ signal goes high (> +2.4V) and then back to ground.

The response to an $\overline{NMI}$ interrupt signal cannot be disabled within the processor. After the processor completes the instruction being executed, it will recognize the interrupt and will proceed to service the interrupt as described in the previous section. The proper discipline to employ in all interrupts is for the interrupt signal to be latched until the processor completes servicing the interrupt. This method of operation is assured if all the interrupts are connected to the interrupt inputs of the peripheral interface devices in the family.

Processing of multiple interrupts in a polled interrupt structure requires that all of the interrupts be polled before executing a "Return from Interrupt" instruction. This is necessitated by the "WIRE-OR" technique for combining the interrupts, since no knowledge exists of which line went to ground. If one of the interrupts is left unserviced, it will hold the $\overline{NMI}$ signal to ground, disabling the interrupts from all other sources since it is necessary for the $\overline{NMI}$ signal to go high (> 2.4V) and back low

again for an interrupt to occur. This is not true for the IRQ input since this latch is level-sensitive. Performing a "Return from Interrupt" before all $\overline{IRQ}$ interrupt sources are serviced will simply cause another $\overline{IRQ}$ interrupt to occur.

1.2.4 System Reset

One of the basic system control functions is the system RESET signal. Whether this signal is generated automatically by external power-on circuitry or manually from a push-button switch, the system components must obey a fixed set of rules to assure proper system operation. This is particularly true for the peripheral interface devices.

In the R650X-based systems, an assumption is made that RESET pins on all peripheral interface devices and on the processor will be held low during power-on until the supply voltages and the clocks have stabilized. This procedure assures that the peripheral pins will remain in a known state until the entire system is initialized and the processor is ready to assume control of the output lines, i.e., is ready to run the system program.

It should be mentioned that in the entire set of microcomputer chips, the contents of latches, registers, etc. are totally random after power is applied. On the peripheral output pins, random data can be disastrous. The only way to force these lines to a known condition is to apply the RESET signal. The designer can then make sure that the known condition will not cause spurious operations in the peripheral devices. The effect of RESET on the peripheral chips is discussed in the analysis of each chip.

In the processor, the single register which must be placed in a known state is the program counter. This is the register which selects the instructions to be executed. The RESET input causes the program counter to go to the first instruction in the system program. The specific details of this operation are discussed in Section 2.2.8.

There is one other very important function performed by the RESET input on the peripheral interface devices. Although the recognition of the processor interrupt signals is automatic and does not depend on software, the sequence of operations performed by the processor to totally service an interrupt is determined by the program. Until the various internal registers in the processor have been initialized, the processor is not ready to respond properly to any external interrupts. For this reason, it is important that the system RESET disable all external interrupt signals until they are enabled by the processor. The programmer can then make sure that the system has been properly initialized before the interrupts are enabled.

SECTION 2

THE MICROPROCESSOR FAMILY

2.1 FUNCTIONAL FEATURES

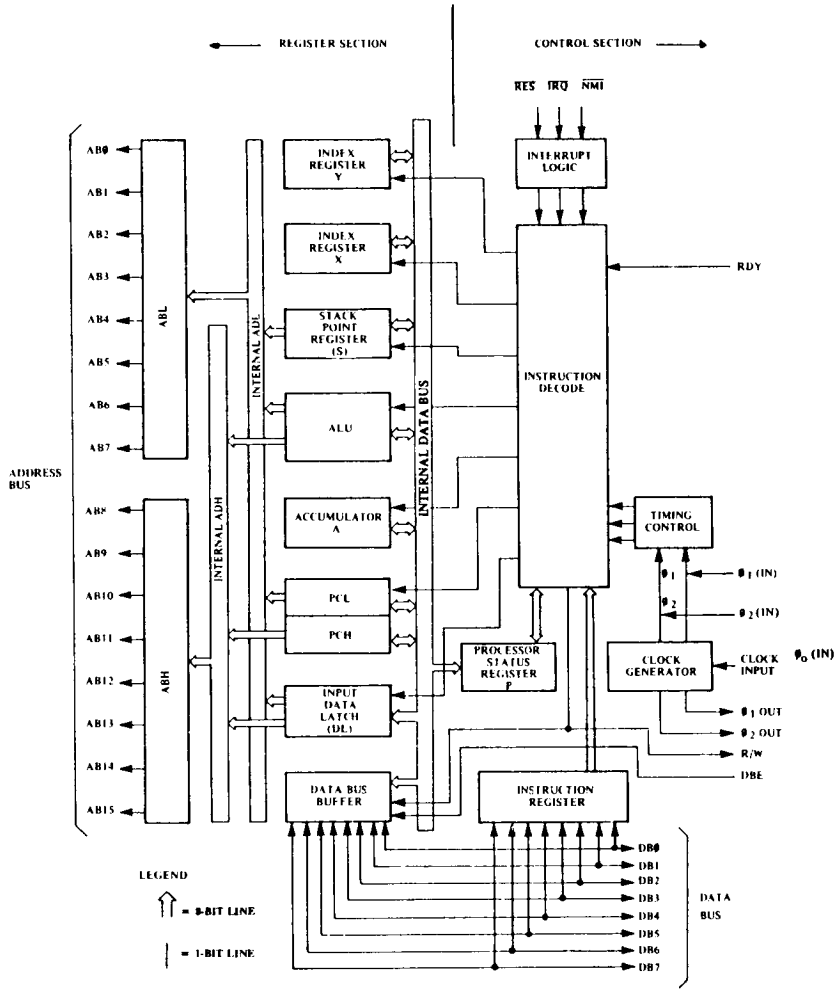
The microprocessors in the R6500 family have very similar internal architectures. A block diagram of the basic architecture is shown in Figure 2-1. This section of the manual begins with an analysis of this block diagram, discussing the function of the various registers, data paths, etc. A detailed discussion of the operation of the various pins on the chip follows.

The internal organization of the processor can be split into two sections. In general, the instructions obtained from program memory are executed by implementing a series of data transfers in one section of the chip (register section). The control lines which actually cause the data transfers to take place are generated in the other section (control section). Instructions enter the processor on the data bus, are latched into the instruction register, and are then decoded along with timing signals to generate the register control signals.

The timing control unit keeps track of the specific cycle being executed. This unit is set to "T0" for each instruction fetch cycle and is advanced at the beginning of each Phase One clock pulse. Each instruction starts in T0 and goes to T1, T2, T3, etc. for as many cycles as are required to complete execution of the instruction. Each data transfer, etc., which takes place in the register section is caused by decoding the contents of both the instruction register and the timing counter.

Additional control lines which affect the execution of the instructions are derived from the Interrupt logic and from the Processor Status register. The Interrupt logic controls the processor interface to the interrupt inputs to assure proper timing, enabling, sequencing, etc. which the processor recognizes and services.

The Processor Status register contains a set of latches which serve to control certain aspects of the processor operation, to indicate



NOTE: 1. CLOCK GENERATOR IS NOT INCLUDED ON R6512 THROUGH R6515.
2. ADDRESSING CAPABILITY AND CONTROL OPTIONS VARY WITH EACH OF THE R650X PRODUCTS

R650X Internal Architecture

FIGURE 2-1

the results of processor arithmetic and logic operations, and to indicate the status of data either generated by the processor or transferred into the processor from outside.

Since the real work of the processor is carried on in the register section of the chip, a detailed study will be made of this section. The components are:

- Data Bus Buffers
- Input Data Latch (DL)
- Program Counter (PCL, PCH)
- Accumulator (A)
- Arithmetic Logic Unit (ALU)
- Stack Pointer (S)
- Index Registers (X, Y)
- Address Bus Latches (ABL, ABH)
- Processor Status Register (P)

At 1 MHz, the data which come into the processor from the program memory, the data memory, or from peripheral devices, appear on the data bus during the last 100 ns of Phase 2. No attempt is made to actually operate on the data during this short period. Instead, it is simply transferred into the input data latch for use during the next cycle. The data latch serves to trap the data on the data bus during each Phase 2 pulse. The data can then be transferred onto one of the internal busses, and from there into one of the internal registers. For example, data being transferred from memory into the accumulator (A) will be placed on the internal data bus and will then be transferred from the internal data bus into the accumulator. If an arithmetic or logic operation is to be performed using the data from memory and the contents of the accumulator, data in the input data latch will be transferred onto the internal data bus as before. From there it will be transferred into the ALU. At the same time the contents of the accumulator will be transferred onto a bus in the register section and from there into the second input to the ALU. The results of the arithmetic or logic operation will be transferred back to the accumulator on the next cycle by transferring first onto the bus and then into the accumulator. All of these data transfers take place during the Phase 1 clock pulse.

The program counter (PCL, PCH) provides the addresses which step the processor through sequential instructions in the program. Each time the processor fetches an instruction from program memory, the contents of PCL are placed on the low-order 8 bits of the address bus and the contents of PCH are placed on the high-order 8 bits. This counter is incremented each time an instruction or data is fetched from program memory.

The accumulator is a general-purpose 8-bit register which stores the results of most arithmetic and logic operations. In addition, the accumulator usually contains one of the two data words used in these operations.

All logic and arithmetic operations take place in the ALU; this includes incrementing and decrementing of internal registers (except PCL and PCH). However, the ALU cannot store data for more than one cycle: if data are placed on the inputs to the ALU at the beginning of one cycle, the result is always gated into one of the storage registers or to external memory during the next cycle. Each bit of the ALU has two inputs. These inputs can be tied to various internal busses or to a logic zero; the ALU then generates the SUM, AND, OR, etc. function using the data on the two inputs.

The stack pointer (S) and the two index registers (X and Y) each consist of 8 simple latches. These registers store data which are to be used in calculating addresses in data memory. The specific operation of each of these is discussed in detail in the Programming Manual.

The address bus buffers (ABL, ABH) consist of a set of latches and TTL compatible drivers. These latches store the addresses which are used in accessing the peripheral devices (ROM, RAM, and I/O).

2.1.1 Functional Features of 28-Pin CPUs

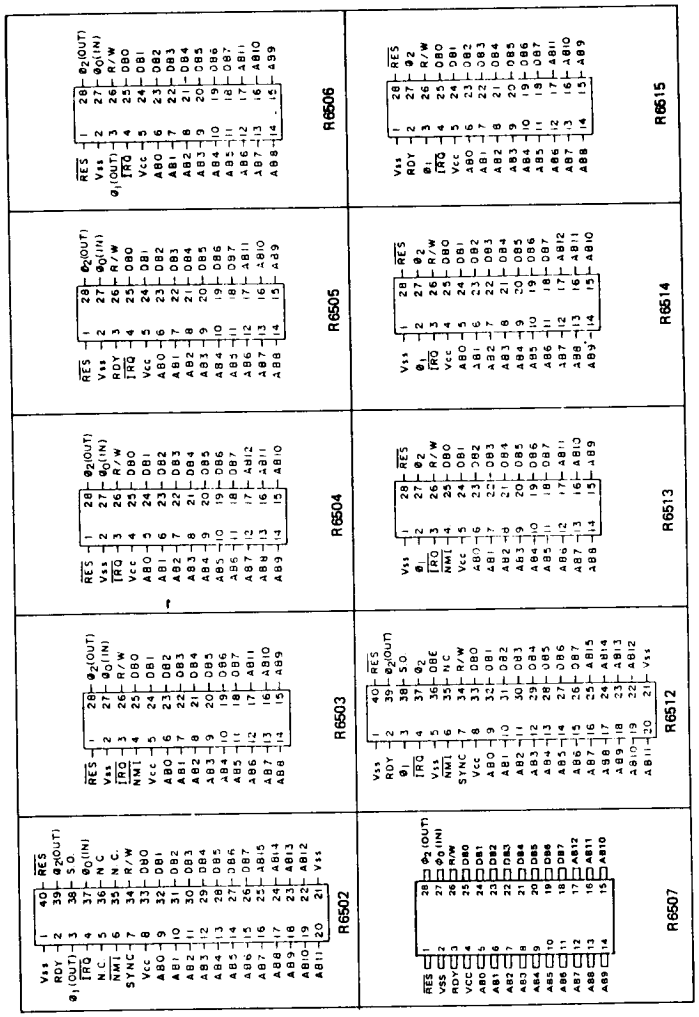
Table 2-1 summarizes the functional features of the R6503 through R6507 and R6513 through R6515. The operation of each function is exactly the same as on the R6502.

Figure 2-2 summarizes the pin designation for the eight processors, indicating the tradeoffs that exist between control signals and addressing capability due to pinout constraints. Like the R6502, five of the 28 pin microprocessors also have the on-the-chip oscillator and clock drivers.

TABLE 2-1
Functional Features of 28-Pin CPUs

Features	R6503, R6513	R6504, R6514	R6505, R6515	R6506	R6507
Addressing Capability	4096 Bytes (AB00 - AB11)	8192 Bytes (AB00 - AB12)	4096 Bytes (AB00 - AB11)	4096 Bytes (AB00 - AB11)	8192 Bytes (AB00 - AB12)
Interrupt Request Capability	$\overline{IRQ}$, $\overline{NMI}$	$\overline{IRQ}$	$\overline{IRQ}$	$\overline{IRQ}$	--
"Ready" Signal	--	--	RDY	--	RDY
*Timing Signals Required	Single Phase TTL Level $\emptyset_0$ (IN), or Crystal or RC	Single Phase TTL Level $\emptyset_0$ (IN), or Crystal or RC	Single Phase TTL Level $\emptyset_0$ (IN), or Crystal or RC	Single Phase TTL Level $\emptyset_0$ (IN) or Crystal or RC	Single Phase TTL Level $\emptyset_0$ (IN), or Crystal or RC
Other Control Signals	$\overline{RES}$, R/W	$\overline{RES}$, R/W	$\overline{RES}$, RW	$\emptyset_1$ (OUT), $\overline{RES}$, R/W	$\overline{RES}$, R/W

*6513, 6514 and 6515 are slave microprocessors requiring external $\emptyset_1$ and $\emptyset_2$ clock inputs



CPU Pinout Designations
FIGURE 2-2

2.2 SIGNAL LINES

Figure 2-2 summarizes the pinouts of the R6500 CPU's. These pins and their uses in microcomputer systems are discussed separately below.

2.2.1 Address Bus (AB00-AB15)

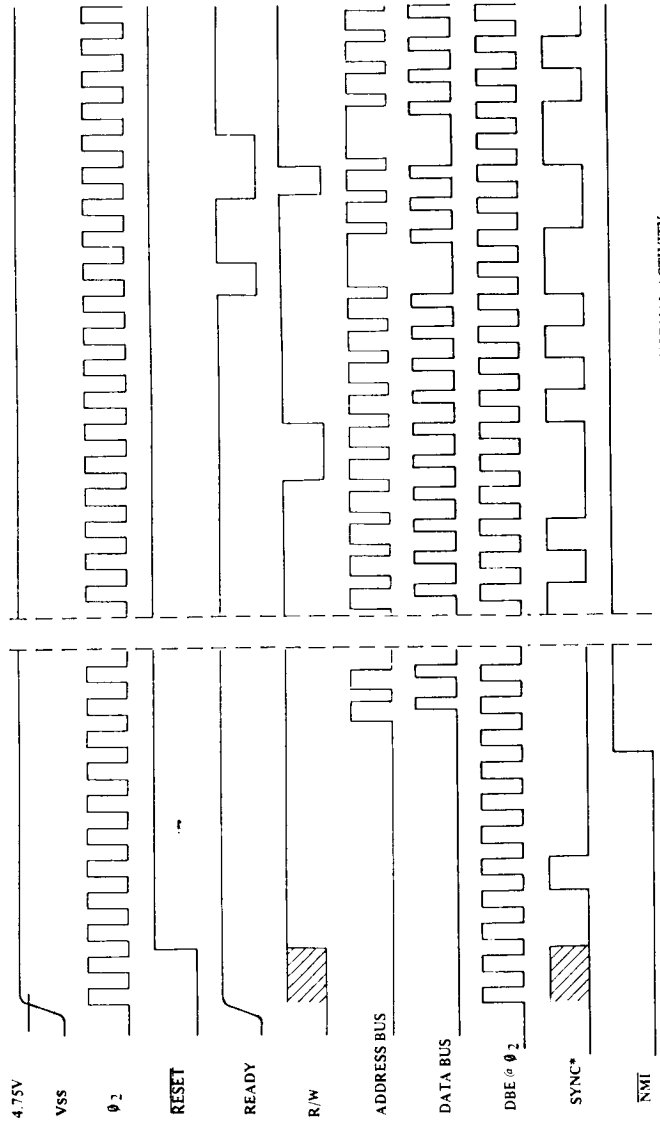
The address bus buffers on the R6500 family of microprocessors are push/pull type drivers capable of driving at least 130 pf and one standard TTL load.

The address bus will always contain known data as detailed in Appendix A. The addressing technique involves putting an address on the adress bus which is known to be either in program sequence, on the same page in program memory or at a known point in RAM. A brief study of Appendix A will acquaint the designer with the detailed operation of this bus.

The various processors differ somewhat in the number of address lines provided. In particular, the R6504, R6507 and R6514 provide 13 address lines (AB00 - AB12) and the R6503, R6505, R6506, R6513 and R6515 provide 12 address lines (AB00 - AB11). This total address space should prove to be more than sufficient for the small, cost-sensitive systems in which these devices should find their greatest application.

The specific timing of the address bus is exactly the same for all the processors. The address is valid 300 ns (at 1 MHz clock rate) into the Ø1 clock pulse and remains stable until the next Ø1 pulse: this specification will change only for processors which are specified to operate at a higher clock rate. Figure 2-3 details the relationship of address bus to other critical signals.

Because of the reduced number of address lines on the 28-pin processors, it is possible to write a program which attempts to access non-existent memory address space, i.e., the address bits 13, 14, or 15 set to logic "1." These upper address bits in the program will be ignored and the program will drop into existing address space. This assumes proper memory management when using devices of large addressing capability such that the addressed memory space will fit within the constraints of a device with smaller available memory addressing capability.



*SYNC IS AVAILABLE ON R6502 AND R6512 ONLY

R650X System Timing Diagram
FIGURE 2-3

2.2.2 Data Bus (DB0-DB7)

The processor data bus is exactly the same for the processors currently available and for the software-compatible processors which will be introduced in the near future. All instructions and data transfers between the processor and memory take place on these lines. The buffers driving the data bus lines have full "three-state" capability. This is necessitated by the fact that the lines are bidirectional.

Each data bus pin is connected to an input and an output buffer, with the output buffer remaining in the "floating" condition except when the processor is transferring data into or out of one of the support chips. All inter-chip data transfers take place during the Phase 2 clock pulse. During Phase 1 the entire data bus is "floating."

The data bus buffer is a push/pull driver capable of driving 130 pf and one standard TTL load at the rated speed. At a 1-MHz clock rate, the data on the data bus must be stable 100 ns before the end of Phase 2. This is true for transfers in either direction. Figure 2-3 details the relationship of the data bus to other signals.

2.2.3 Read/Write (R/W)

The Read/Write line allows the processor to control the direction of data transfers between the processor and the support chips. This line is high except when the processor is writing to memory or to a peripheral interface device.

All transitions on this line occur during the Phase 1 clock pulse (concurrent with the address lines). This allows complete control of the data transition which takes place during the Phase 2 clock pulse.

The R/W buffer is similar to the address buffers. They are capable of driving 130 pf and one standard TTL load at the rated speed. Again, Figure 2-3 details the relative timing of the R/W line.

2.2.4 Data Bus Enable (DBE)

On the R6512, a data bus enable signal is provided to allow external enabling of the data bus. This line is connected directly to the Phase 2 input clock signal for any normally operating system and is detailed in Figure 2-3.

The DBE signal affects only the data bus buffers. It does not affect processor timing and has no effect on the address or the R/W lines.

This input is provided primarily for use in systems which use non-R6500 devices for either the memory or the peripheral interface functions. In particular, it allows the data bus to be enabled for a period longer than the Phase 2 clock pulse for systems requiring greater processor hold time on the data bus. This application is covered in greater dedetail in Section 3.

2.2.5 Ready (RDY)

The RDY input delays execution of any cycle during which the RDY line is pulled low. This line should change during the Phase 1 clock pulse. This change is then recognized during the next Phase 2 pulse to enable or disable the execution of the current internal machine cycle. This execution normally occurs furing the next Phase 1 clock; timing is shown in Figure 2-3.

The primary purpose of the RDY line is to delay execution of a program fetch cycle until data are available from memory. This has direct application in prototype systems employing light-erasable PROMs or EAROMs. Both of these devices have relatively slow access times and require implementation of the RDY function if the processor is to operate at full speed. Without the RDY function a reduction in the frequency of the system clock would be necessary.

The RDY function will not stop the processor in a cycle in which a WRITE operation is being performed. If the RDY line goes from high to low during a WRITE cycle the processor will execute that cycle and will then stop in the next READ cycle (R/W = 1).

2.2.6 Non-Maskable Interrupt ($\overline{\text{NMI}}$)

The $\overline{\text{NMI}}$ input, when in the interrupted state, always interrupts the processor after it completes the instruction currently being executed. This interrupt is not "maskable" -- i.e., there is no way for the processor to prevent recognition of the interrupt.

The $\overline{\text{NMI}}$ input responds to a negative transition. To interrupt the processor, the $\overline{\text{NMI}}$ input must go from high (> +2.4V) to low (< +0.4V). It can then stay low for an indefinite period without affecting the processor operation and without another interrupt. The processor will

not detect another interrupt until this line goes high and then back to low. The $\overline{\text{NMI}}$ signal must be low for at least two clock cycles for the interrupt to be recognized, whereupon new program count vectors are fetched.

2.2.7 Interrupt Request ($\overline{\text{IRQ}}$)

The interrupt request ($\overline{\text{IRQ}}$) responds in much the same manner as $\overline{\text{NMI}}$. However, this function can be enabled or disabled by the interrupt inhibit bit in the processor status register. As long as the I flag (interrupt inhibit flag) is a logic 1, the signal on the $\overline{\text{IRQ}}$ pin will not affect the processor.

The $\overline{\text{IRQ}}$ pin is not edge-sensitive. Instead, the processor will be interrupted as long as the I flag is a logic "0" and the signal on the $\overline{\text{IRQ}}$ input is at GND. Because of this, the $\overline{\text{IRQ}}$ signal must be held low until it is recognized, i.e., until the processor completes the instruction currently being executed. If I is set when $\overline{\text{IRQ}}$ goes low, the interrupt will not be recognized until I is cleared through software control. To assure that the processor will not recognize the interrupt more than once, the I flag is set automatically during the last cycle before the processor begins executing the interrupt software, beginning with the fetch of program count.

The final requirement is that the interrupt input must be cleared before the I flag is reset. If there is more than one active interrupt driving these two lines (OR'ed together), the recommended procedure is to service and clear both interrupts before clearing the I flag. However, if the interrupts are cleared one-at-a-time and the I flag is reset after each, the processor will simply recognize any interrupts still active and will process them properly but more slowly because of the time required to return from one interrupt before recognizing the next. If the procedure recommended above is followed, each interrupt will be recognized and processed only once. Figure 2-4 provides several examples of interrupts, microprocessor recognition of each interrupt ($\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$), and processor selection of interrupts during overlapped requests.