DESIGN OF A MICROPROCESSOR-BASED CONTROL SYSTEM FOR THE MAGNETIC PARTICLE BRAKE ABOVE-KNEE PROSTHESIS

by

J. Mark Oaric

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF

BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1977

Signature of Author _____
Department of Mechanical Engineering May 23, 1977

Certified by _____
Thesis Supervisor

Accepted by _____
Chairman, Departmental Committee on Thesis

DESIGN OF A MICROPROCESSOR-BASED CONTROL SYSTEM FOR THE MAGNETIC PARTICLE BRAKE ABOVE KNEE PROSTHESIS

by

J. Mark Deric

Submitted to the Department of Mechanical Engineering on May 23, 1977 in partial fulfillment of the requirements for the Degree of Bachelor of Science in Mechanical Engineering.

ABSTRACT

The quest for an improvement in the dynamic behavior of prostheses for above-knee amputees has lead to the development of a versatile prosthesis emulator, based on a computer controlled magnetic particle brake (MPB). The MPB is capable of providing up to 300 in.-lbs. of dissipative torque at the knee. Its torque output can be made an arbitrary function of the inputs to the PDP-11 computer from the instrumented prosthesis. This system has been used to evaluate a variety of knee control schemes, both conventional and untested in a man-interactive setting. However, the required connection to a stationary computer limits the value of the system. Walking trials are constrained to be within the range of the umbilical cord to the computer. This limits accessability to amputees, limits the realism of the trials themselves, and provides no basis for a commercial prosthesis, once an optimum dynamic profile has been determined.

This thesis presents a Motorola MC6800 microprocessor-based controller design, in language understandable to the mechanical engineer, which eliminates those difficulties. The design specifications are formalized, the electronic hardware is fully described, and the simulation of a sample knee control algorithm is presented. The thesis concludes with a design evaluation and recommendations.

Thesis Supervisor: Woodie C. Flowers
Title: Associate Professor of Mechanical Engineering

ACKNOWLEDGEMENTS

There are a number of people who have made valuable contributions to the work of this thesis. Chris Bailey and Gary Sawyer of Motorola Inc., Lexington, Mass. office, gave freely of their time, answering questions, providing information, and lending technical support to the project. If those gentlemen are at all representative of applications engineers, in general, system designers must have an easy life. At MIT, Professor Derek Rowell and George Dalrymple, ME department microprocessor aces, provided advice and encouragement, not to mention data books, from the beginning. Professor Steve Burns and the people at the Biomedical Engineering Center for Clinical Instrumentation helped give the project initial direction by telling me about their work and experience with microprocessors. Professor David Jansson and the MIT Innovation Center provided me with an Exorcier, a Motorola microcomputer, designed as a system development tool, and a teletype for the software simulation. They were a sine qua non of the project. During the software development stage, Don Grimes, a graduate student and forefront researcher in gait control, led this programmer to the sacred Meaningful Algorithm. His help in specifying design parameters was also invaluable. To the people who work on the 8th floor of 545 Tech Square at the MIT Laboratory for Computer Science, especially Rich Zippel, Barry Trager, Dave Barton,

and Prof. Pete Szolovitz, I am indebted. They tolerated my almost continuous presence and were extremely helpful during the two and a half weeks of 24 hour days spent writing this.

Special thanks are owed to the following people for their contribution whether it was technical, spiritual or both: Woodie, who put up with a lot and may still be my friend. My advisor, Professor Ascher Shapiro; he was there when I needed him. Chuck Harrison, a fraternity brother and the fellow who turned me on to the wonders of the electron. Bob Kerns, an electrical engineering undergraduate and fraternity brother, who took up when Chuck left MIT. And, finally, to my parents, sisters, and to Donna...always.

May 1977 J. Mark Deric

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I—INTRODUCTION

1.1 The Problem

The work of this thesis is part of the MIT Knee Project, a research effort determined, eventually, to produce a blueprint for dynamically optimal prostheses for above-knee (A/K) amputees. So far, two computer controlled prostheses have been built and operated simulating the knee dynamics of a variety of conventional prostheses (i.e., constant friction, damping proportional to velocity, etc.) as well as some unconventional, previously untested types of knee control in a man-interactive setting. The first prosthesis is hydraulically powered and is capable of responding to computer commands with positive power at the knee. The second prosthesis, to which the work of this thesis is directed, is a passive device only

capable of providing dissipative torques at the knee by a computer controlled, magnetic particle brake (MPB)/transmission system. Both prostheses are fully instrumented to provide the computer with heel contact, toe contact, angular position, angular velocity and knee torque information. The MPB prosthesis, in addition, has torque feedback control for the MPB. This allows direct control of brake torque with the appropriate signal at the reference input of the feedback controller.

Although use of these two prostheses with an on-line PDP-11 in the MIT Knee laboratory has produced some significant results in terms of evaluating knee control and characterizing amputee behavior, the simulator system is limited by the size of the controller, the cabinet-sized PDP-11. With a portable controller, able to be carried inside the prosthesis or in a purse-sized carrying case, the passive MPB simulator would be made more mobile. Accessability to amputee test subjects would be enhanced since the system could be taken to clinics and hospitals. Walking trials, during which the amputees evaluate the implemented control modes, would become more realistic in that the amputee's range would not be limited by the umbilical cord from the stationary computer. Furthermore, a resident controller is a necessary development toward a commercially available optimum prosthesis.

The prosthesis controller must be versatile. It must be capable of simulating the control modes of current interest and it must be capable of implementing more complex, anticipated control schemes. Among the anticipated schemes are those involving

electromyographical (EMG) control and those which change control mode as the type of gait (i.e., walking, running, climbing stairs, etc.) is changed. Consequently, the device must be able to perform arithmetic computations and make logical decisions quickly.

This thesis proposes a microprocessor-based controller to meet the need for a versatile, portable control system for the MPB prosthesis.

1.2 Specifications

The technical specifications for the controller are as follows:

A. Costs less than $500

B. Capable of handling up to 6 analog inputs with 8-bit resolution

C. Capable of producing one analog output with 8-bit resolution

D. Able to be installed in the prosthesis or carried in a small satchel

E. Able to perform the most complex control algorithms in under 20 milliseconds

F. Control programs must be quickly interchangeable

G. Must operate for 12 hours on a modicum of batteries

These specifications were determined in a meeting of the Knee Project on 17 February 1977. 8-bit resolution of the inputs and outputs implies an accuracy limit of ±.4% of full scale for reading and writing data. It was decided that this accuracy is sufficient. The 20 millisecond turnaround time limit was experimentally determined to be the longest delay time not noticeable to the amputee. The cost was determined by budgetary constraints, and the remaining specifications define the portability requirement.

1.3 Dedicated-task Microprocessor Approach—Block Diagram Description

To incorporate the versatility of a digital computer and the portability of a transistor radio into the controller design, a microprocessor, the Motorola MC6800, was chosen as the fundamental building block for system design. In this application the microprocessor's sole function is to calculate MPB torque based on the inputs and according to the program stored in memory; hence, it is called a dedicated task microprocessor system. The scheme is presented in overview in the following two sub-sections.

1.3.1 Overview of Microprocessor and Memory

The microprocessing unit (MPU) and the memory are the heart of the system. The MPU is similar to the central processing unit or CPU of a full scale computer. However, its instruction set (i.e., its list of elemental operations) is more limited; its input/output (I/O) control is less sophisticated; and its speed is slower than that of the full CPU. The memory is the storage device for both program and data. The basic unit of stored information is the byte. (In most microprocessors, including the MC6800, a byte is an 8 digit binary number in which each digit is called a bit. The bits are labelled from the leftmost bit to the rightmost, b_7-b_0.) The memory is organized so that each one byte storage location has a binary address or addresses to which it is unique. There are two types of memory, read-only-memory (ROM) and random-access-memory (RAM). ROM is used primarily for program storage and fixed data storage, since the MPU cannot change its contents. RAM is used as a scratchpad for intermediate computational results and as a temporary buffer for short term storage; the MPU can change its contents. Easy reprogrammability can be accomplished by mounting the ROM in sockets. Then, the program can be changed by simply unplugging the current ROM and replacing it with another.

The MPU acts as the organizer for system operation as well as the arithmetic logic unit. When the system is started up, the program counter (PC), a register (i.e., a location containing a

binary number) in the MPU, is set to a value specifiable by the programmer. The binary number in the program counter corresponds to the address in memory of the first instruction. The MPU reads the byte corresponding to that address and interprets it as an instruction. The program counter is incremented, pointing to the next sequential location in memory. If the instruction requires arguments, they are read from the sequential memory locations, always incrementing the program counter after each program byte is read. Instructions tell the processor to A) manipulate internal MPU registers, B) access (read/write) any location in the memory, or C) manipulate the contents of external memory locations. (A detailed example of instruction execution is given in the last paragraph of 2.1.3.) When the execution is complete, the next instruction is taken from the address specified by the program counter. This may be the next sequential memory location or, if the last instruction changed the program counter, any other memory location. In this manner, a program is executed.

The memory acts only as support for the MPU. When the ROM is given an address, its job is to provide the MPU with the contents of the location. When RAM is given an address, its job, depending on whether the MPU is making a read or write, is to make the specified location availble for the action.

1.3.2 Overview of Input/Output Components

When the tin man got a heart, he already had eyes and a

mouth. To complete the controller system, input/output (I/O) capability must be provided. The system must be able to convert information in byte form (digital information) into continuously variable (analog) voltage or current and vice-versa. The latter is compatible with the MPB and the knee instrumentation, whereas, the former is not. Analog-to-digital (A/D) and digital-to-analog (D/A) converters controlled by the MPU via a programmable interface (i.e., information link) buffer perform these functions. The programmable buffer is called the peripheral interface adapter (PIA). It is able to simultaneously handle two channels of data, each moving a byte of data in either direction. Furthermore, the PIA can generate control signals for the peripherals (the D/A and A/D, in this case) at MPU command and it can present interrupt requests to the MPU if the microprocessor allows it. (An interrupt causes the program counter and all of the other internal registers to be saved. The program counter is then loaded with the starting address of the special interrupt program. This program attends to the special case for which the interrupt was generated. When the interrupt program is finished, the internal registers are reinstated and the main program is continued.) In the system configuration, the A channel of the PIA is used for input and the B channel is used for output. To expand the input section to handle 8 different inputs, an MPU controlled multiplexer is provided. The multiplexer controls which input goes to the A/D at any time.

To input data, the MPU first selects the appropriate multiplex channel. The MPU then programs the A side of the PIA for

the A/D conversion. It does this by writing into two PIA registers, which it sees as addressable memory locations. The now-programmed PIA sends a signal to the A/D telling it to start the conversion. When the successive approximation A/D is complete, the result is stored in the PIA data register. At the same time, the converter signals the PIA which in turn presents an interrupt request to the MPU. The MPU acknowledges the request and begins the interrupt program. This program writes into the PIA control register to inhibit further conversion and then reads the result from the data register. Control is returned to the main program.

To output data, the MPU programs the B side of the PIA only once in the beginning of the control program. As above, programming is done by writing into PIA registers which look like memory locations. The system is configured so that the D/A continuously converts the byte stored in the B side data register into the corresponding analog signal. As the MPU updates the contents of the data register, the analog output is automatically updated.

1.4 Purpose and Scope of this Thesis

The purpose of this thesis is fourfold. First, it provides a microprocessor-based control system design which meets the specifications outlined in Section 1.2. Secondly, it fully describes and documents the controller's hardware (the electronics) and its

operation in language understandable to the mechanical engineer. Chapter II is dedicated to the hardware. It is intended as a pedagogical tool as well as a technical document. Third, this thesis in Chapter III provides an example of software development (the development of the program stored in memory, i.e., the software). That chapter is a description of the implementation of the Ideal Swing-Phase control algorithm [4]. Implementation was simulated on the MC6800-based Exorciser (Motorola tradename) microcomputer system. Finally, Chapter IV evaluates the design in light of alternatives and in light of possible uses.

CHAPTER II—HARDWARE DESIGN

2.1 Background

Implementation of the scheme presented in Section 1.3 requires full understanding of the byte and its constituent bits, i.e., what can a byte represent, how is a byte represented electronically, and how may a byte be manipulated. When these questions are answered, it will be easy to view the controller as a machine which manipulates electronically represented bytes and uses them according to what they represent to perform its function.

2.1.1 What Can a Byte Represent

Depending on its context, the byte can represent a number of different things. In the simplest case, the eight bit binary number may represent an integer from 0 to 255. In this, the unsigned binary representation, b_7 is the most significant bit (MSB), and b_0 is the least significant bit (LSB). A byte may represent a signed integer in two ways. If the byte represents a signed binary number, it may have a value from -127 to +127. The magnitude is determined by b_6-b_0, evaluated as an unsigned binary number, b_6 being the MSB. If b_7, the sign bit, is 1, the number is negative, otherwise it is positive. If the byte is interpreted as a 2's complement number, it may have a value from -128 to +127. If b_7 is 1, the value is negative. Its magnitude is determined by changing the value of each bit (the Boolean negate operation, $\sim b_x \rightarrow b_x$), adding 1 to the result and evaluating the sum as an unsigned binary number. If b_7 is positive the byte value is the same as the unsigned binary value. A byte may represent any other quantity if it is interpreted as being multiplied by an implicit scale factor. For example, the 2's complement byte 1000000 (=-128) may represent -352 ft.-lbs. if the implicit scale factor is 2.75 ft.-lbs. The accuracy of this representation is limited by the scale factor. The byte can only stand for integral multiples of 2.75 ft.-lbs. Hence, a value of 4.132 ft.-lbs. can only be approximated as either 2.75 ft.-lbs. (by 00000001) or as 5.5 ft.-lbs (by 00000010). The byte is said to

resolve the full range of representations into increments of the scale factor. The accuracy limit is called the resolution. If the scale factor is divided by the full scale range, a percentage of the full scale to which the binary representation is accurate results. This is the percent resolution and is constant for a given number of bits in the byte. This use of resolution appears in the specifications.

A byte may be interpreted as an instruction. In the appendix (pages 74-76), a list of MC6800 instructions appears. On those tables in the columns labelled OP, two-digit hexadecimal (base 16) numbers are listed. Those hexadecimal numbers correspond to unsigned 8-bit binary numbers, which, when read as instruction bytes by the MPU, cause the appropriate operation to be performed. For example, when the MPU reads the byte 00111110 (hexadecimal 3E, where A-F represent 10-15) from the memory as an instruction, the wait for interrupt operation (op column listing 3E) is performed. (Note: The hexadecimal notation used in the Motorola document will be used to represent bytes or double bytes <two bytes strung together to constitute a 16-bit binary number> in the remainder of this thesis for the sake of text economy. This will be done without the use of a subscript indicating base 16; the context will indicate the notation. If one realizes that by grouping bits four at a time starting with the least significant four and evaluating the groups, he can obtain the hexadecimal value <as demonstrated below in Figure 1>, conversion can be done

binary 00111110 parses to 0011/1110
$0011_2=3_{16}$ ↓ ↓
$1110_2=E_{16}$ 3 E
the result is $3E_{16}$ as in the text.

Figure 1 - Hexadecimal-to-Binary Conversion

by inspection.)

A byte may represent a memory location. As mentioned earlier, each memory location has an address, which, in the case of the MC6800, is a 16-bit unsigned binary number. To each address, there must correspond only one memory location. Consequently, two bytes, one representing b_{15}-b_8 and the other b_7-b_0 of the 16-bit address, specify a memory location. In some cases, if b_{15}-b_8 are all zero, a single byte is sufficient to represent a memory location.

Finally, a byte may be viewed as a collection of bits which are interpreted individually or in groups. Bits have two different states, "1" or "0"; these states have been alternatively viewed as "true" or "false", "on" or "off", "high" or "low", and "yes" or "no". Expanding this to the general case, a bit may be used to represent anything which has only two states. A byte; in which b_0 indicates heel contact, b_1 represents toe contact, b_2-b_4 is interpreted as the last multiplex channel to be selected, and the other bits are unused; is an example of this representation. As a unit the byte is meaningless, but its constituent bits bear

information.

2.1.2 How a Byte is Represented Electronically

Within each integrated circuit component (IC), the electronic representation of bytes is irrelevant to the designer. However, when bytes are to be transferred from component to component, the electronic representation becomes important. Bytes may be represented serially or in parallel. In serial mode, a single circuit node sequentially represents each bit of the byte. That is, during the first time period, the node stands for b_7; during the second time period, the node stands for b_6, etc. The node exists as a conducting lead coming out of the IC (called a pin) and is thereby externally accessable. (Together, the pin and all the conductors wired to it are called a line.) In parallel mode, exclusively used in the controller, there are 8 circuit nodes, each corresponding to a bit of the byte. Consequently, there are 8 pins emanating from the IC (or chip) for each one byte, parallel, I/O port. In both representations, the value of each bit is determined by the voltage at its corresponding node. (In the serial case, "corresponding node" must include not only a physical place, but also a time element.) A voltage close to the positive power supply voltage represents a "1"; while a voltage close to ground (negative supply voltage) stands for "0". Lines carrying two discreet voltages are called digital lines. Using either mode, bytes can be input if voltages are externally applied to the appropriate pins at

the right time; and they can be output if voltages are internally supplied.

There are some important considerations to be dealt with in designing parallel I/O between components; the designer must understand some details of the transfer of each bit. First, he must have a model of the output stage which puts the high or low voltage at the output pin. Second, he must understand the characteristics of the circuit behind the input pin (or pins if more than one input is taken from a single output). And, third, he must have a model of the line connecting them. For the digital logic chips used in this design, the output stages when in the high state may be modelled as a Thevenin equivalent network (see Figure 2, below) in which the voltage source is 5 volts (the supply voltage) and the series resistance is determined by computation from the device specifications as follows:

$$R_{EQ} = (5 \text{ volts} - V_{OH}) \; / \; I_{OH(MAX)}$$

where $I_{OH(MAX)}$= maximum high level output current, and V_{OH}= the voltage output when $I_{OH(MAX)}$ was measured.

When the output is low, the model for this application becomes a low voltage source with a maximum current rating above which the device

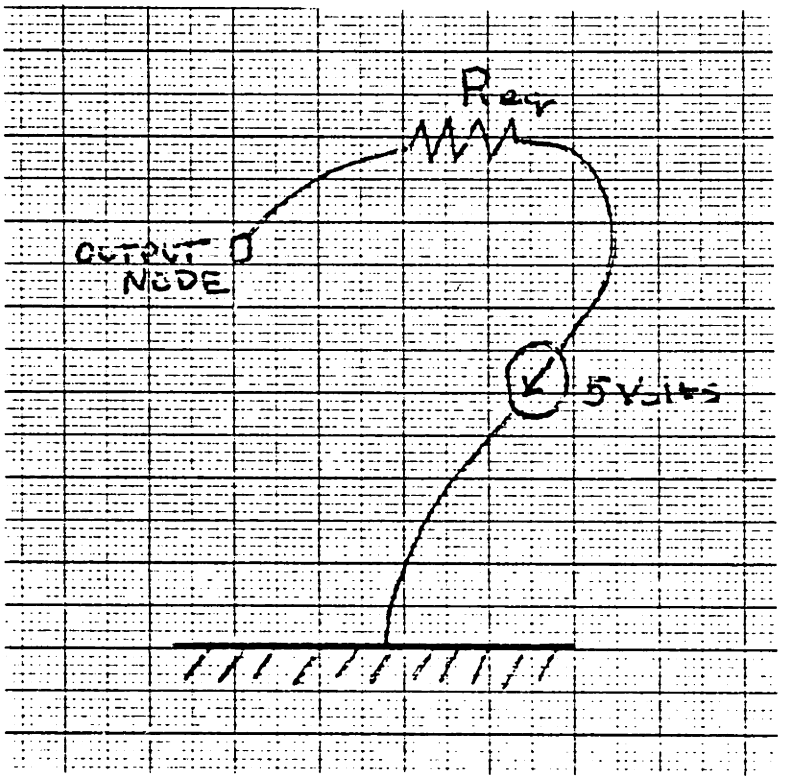


Figure 2—Output Stage Thevenin Equivalent

may fail. The inputs are characterized when the applied voltage wants to go high as capacitors which require charging. A "1" level is not garaunteed to be input to the device until the voltage on the capacitor reaches a minimum threshhold. In some cases the effective input capacitance is given in the device specs as C_{in}; however, if it is not, a safe approximation can be obtained by assuming a capacitance equal to test load capacitance the manufacturer used to determine the device's switching speeds. The threshhold voltage is always given in the specs as $V_{IH(min)}$. When the input pin is held low, it may be modelled as a current source. The current is given as $I_{IL(MAX)}$ in the device data sheet. Lines connecting pins are modelled as an infinite number of incremental resistances and capacitances in the low pass filter configuration. Fortunately, such systems represent pure delays, rather than waveform decomposers, as a single pole would. This is because in a system with infinite poles, phase is linear in frequency. In this application, since the system is relatively small and the lines will be short, the delay is taken to be negligable.

As a result of these models, the designer must be careful of two things to insure successful bit transactions. If a low value is to be transferred, he must design so that the source currents from all of the inputs (called loads) on the same line as the output do not add up to a total current greater than that which the output can safely sink. If a high value is transferred, he must make sure that the capacitance from all the loads does not slow the transfer down to the point where overall system timing is incorrect. When

transferring bits from a chip of one type logic to a chip of the same family (e.g., Transistor Transistor Logic, TTL, to TTL) the designer need not concern himself with the above details, as the manufacturer specifies that the output of any family device will drive up to so many loads of family devices and still meet the other specs. The number of inputs is called fan out. However, if the bit is to go from a chip of one family to a chip from another, unless the manufacturer states that his output will drive some number of another family's inputs and the specifications at which it will do so, the designer must consider the above.

2.1.3 How is a Byte Manipulated

Bytes are manipulated by the internal MPU hardware in response to both software (the instructions read as bytes from the memory) and externally applied hardware control signals. An external control signal may be viewed as an independent bit, not associated with any byte, to be input to the MPU. The bit is represented electronically as a voltage and is constantly input to the MPU via a pin corresponding to the control input which the bit represents. Depending on the state of the control signal, i.e., whether it is "high", "low" or in transition, action is taken or not taken. During one command, whether software instruction or hardware signal, the MPU may manipulate bytes in any combination of the following ways: input, output and alter in a prescribed manner (command dependent). The ordering of these byte manipulation tasks

is given by the internal MPU hardware's fixed interpretation of the command. It is given in terms of MPU cycles, also called clock cycles, the number of which vary from command to command. MPU cycles are determined by an external clock circuit, the output(s) of which oscillates from the high voltage 1, to the low voltage 0. Clock state transitions occur rapidly with a regular period producing an approximately square wave(s). The MC6800 requires two clock inputs, the ϕ_1 input, and the ϕ_2 input. ϕ_1 and ϕ_2 are 180° out of phase. The rising edge of the ϕ_1 clock, the transition from low to high (sometimes designated $\phi_1 = \uparrow$), delineates the beginning of a machine (MPU) cycle. In the Appendix (page 77), there appears a cycle-by-cycle summary of the MPU's I/O tasks for all of the MC6800 instructions. Since byte alteration tasks are entirely internal, there is no need to know their ordering.

For an example of command execution, consider the INC instruction (Appendix, page 78) in the extended address mode (see Chapter #3 for addressing modes). This instruction INCrements (by 1) the value in a memory location specified by the two bytes whose addresses follow the instructions address. (In general, the instruction is referred to as the operation; and its corresponding byte, the op code. The contents of the memory location to be used in the operation is called the operand; and its address, the operand address.) This instruction requires 6 MPU cycles. When ϕ_1 rises, starting the first clock cycle of INC, the MPU's two byte, parallel output port designated for address (called the address bus, having 16 lines, A_{16}-A_0, from the MPU) is set by the program counter

to the op code address in memory. Also, the MPU sets two independent output bits, which control the interface to memory. One is the Valid Memory Address (VMA) bit, which indicates that the address on the bus should be acted on by memory, if it is 1. If the VMA line voltage is low, corresponding to a 0 bit value, the address should be ignored. The other MPU generated control signal is on the Read/Write (R/W) line. If the line is high voltage, R/W = 1, the cycle is an MPU read data; if R/W = 0, it is an MPU write. In the first INC cycle, both VMA and R/W are high, indicating that this is a valid memory read cycle. The task timing is organized so that data is never set up on the pins of the parallel, one byte, I/O port (called the data bus) until half-way through the clock cycle, when ϕ_2 goes high. When this happens, the op code is set up on the data bus lines by the memory and is read by the MPU. The program counter is then incremented to point to the next memory location, op code address + 1. Reading the op code is the first cycle task of all software instructions. The ϕ_1 rise which starts the second clock cycle, tells the MPU to set the address bus to the value of the program counter to read in the first byte of the operand address. Hence, the VMA line and the R/W line are 1s. When the ϕ_2 clock goes high, the high order byte, bits A_{15}-A_8, is read. The program counter (PC) is incremented. The third cycle is similar, reading the low order operand address byte, bits A_7-A_0. At the start of the fourth cycle, the operand address is put on the address bus to read in the opperand. VMA and R/W are still high. ϕ_2 again governs the actual read. The program counter is not incremented since this read

was not of an instruction or an argument. (An argument is the byte(s) in the memory directly following an instruction, used to further define it. In this case, the argument is the operand address, not the operand itself.) During cycle 5, the VMA line is low, indicating that the MPU will not be doing I/O and that the address lines should be ignored. Presumably, it is during this cycle that the MPU increments the operand, since it is now in an internal register. The incremented value is output in the final cycle. Hence, when ϕ_1 goes high, the operand address goes on the bus, VMA is 1, and R/W is 0. ϕ_2's subsequent, high state causes the incremented value to be returned to the location from which the original data was taken.

2.2 The Processor—MPU, Clock, and Reset

This section starts the technical description of the controller hardware, shown schematicly in Figure 3. A complete description of each of the components can be found among references [2, 8, 11, 12, 13, 15]. In the schematic, the components represented as boxes are IC's; the labels inside the periphery correspond to pins. The labels include both the pin number, which describes the physical location of the pin on the chip (if applicable), and a mnemonic for the pin function. The symbolicly represented components are labelled once. It is assumed that the reader can identify the

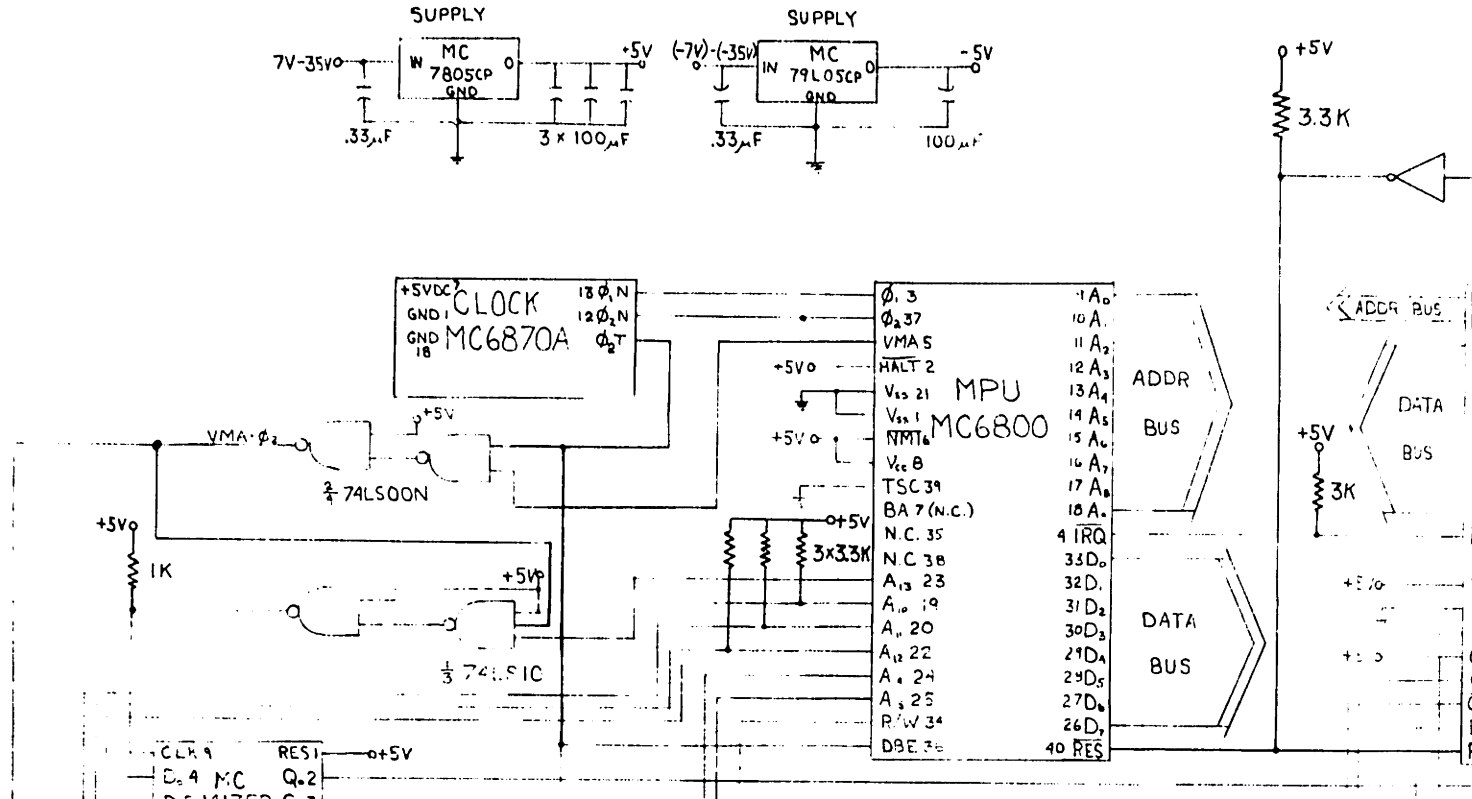


FIGURE 3 KNEE CONTROLLER HARDWARE SCHEMATIC

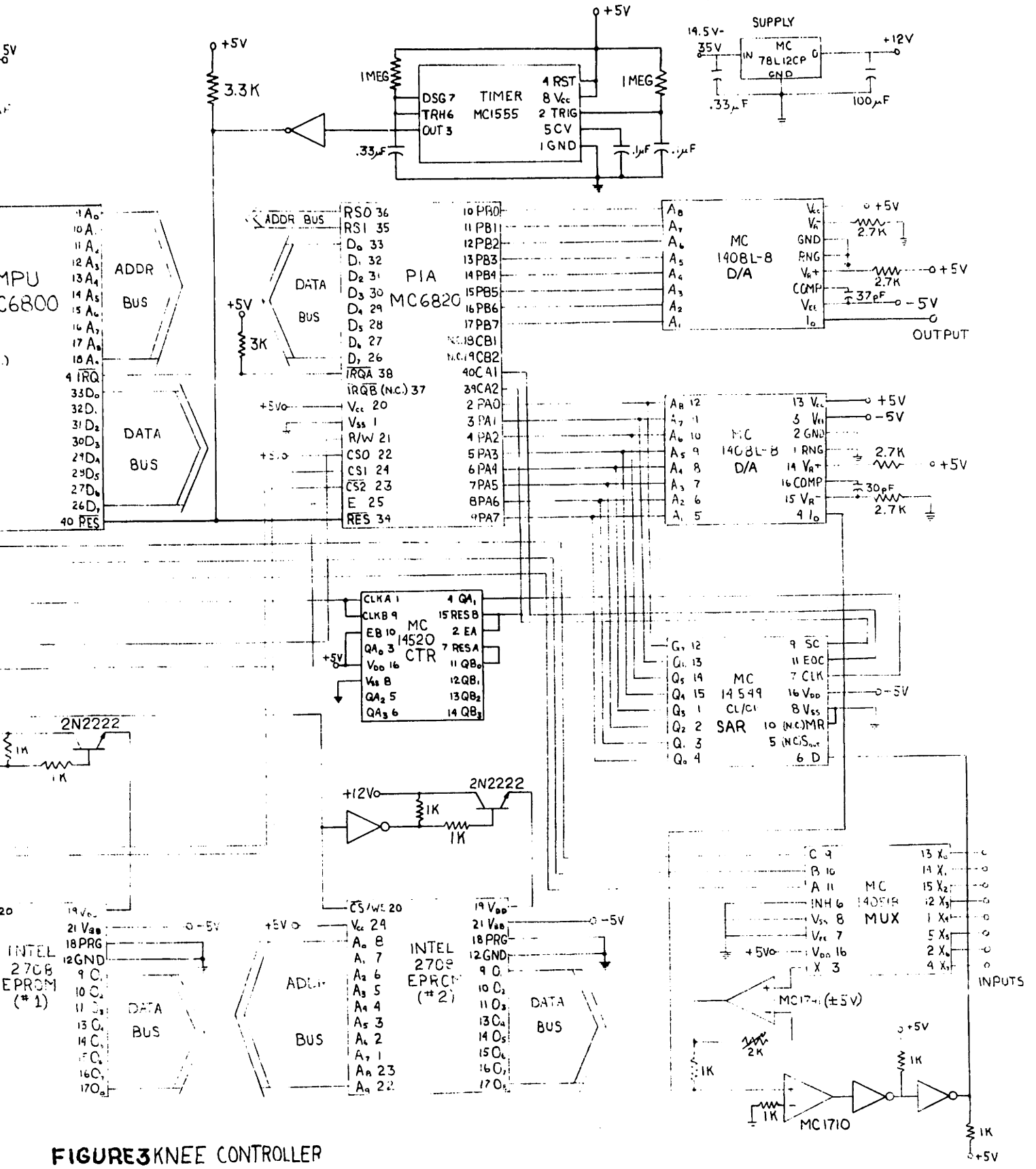


FIGURE 3 KNEE CONTROLLER HARDWARE SCHEMATIC

operational amplifier, the comparator, the transistors, and the passive components. The symbols labelled with a 74 as the first two non-fractional characters are transistor-transistor logic (TTL) components. Each one performs a logical evaluation of the digital inputs (as mentioned above 1=true, 0=false), according to what type of component it is, and outputs the digital result.

Based on price inquiry to Cramer Electronics in Newton, Mass., the price for the drawn system, including the mounting hardware (component sockets, unetched printed circuit board, etc.), would be under $400.

The pictured system, can be built on two printed circuit boards, each about 2.75 inches square, which, when stacked, would be about .75 inches high. This is based on a calculation of the area required for the larger components (all the components that would be packaged in dual-in-line, DIP, packages), multiplied by two to allow spacing and room for the smaller components.

In the schematic, the MPU chip and the clock module are labelled explicitly. The reset circuit is comprised of the timer IC, the passive components connected to it, the 7406 (symbolically represented, labelled elsewhere), and the resistor connected to it. Since the clock and the reset circuit, which sets the program counter to its initial value during power-on startup, are both essential to the MPU's operation, the three can be viewed together as performing the processor function. The following four sub-sections describe details of the processor "unit".

2.2.1 The Processor's Internal Workings—the Nitty Gritty

The voltages to the clock inputs of the MPU, labelled ϕ_1 and ϕ_2 in the schematic, have specified requirements in the time domain given in the appendix (page 81). These specifications describe what can be approximated as two square waves (in the frequency range of .1-1.0 MHz), 180^0 out of phase which never overlap (i.e., both waves can never be in the high state at the same time). The approximation accounts for the fact that a perfect square wave can not be realized in a real system because of finite state transition times, overshoot, and other unavoidable dynamic phenomena. The specifications, in this case and in others to be presented later, quantify the allowable (or even the required) deviation from the approximation. The specified clock input is obtained in the controller system by using the Motorola MC6870A Clock Module [8] (see output waveform in Appendix, page 82) designed to have pins ϕ_1N and ϕ_2N connected directly to the ϕ_1 and ϕ_2 pins of the MPU as shown in the schematic. The clock module was chosen to have the maximum running frequency of 1.0 MHz implying an MPU cycle time of 1 microsecond. The Clock Module also has available a ϕ_2 output for use by the other system components in synchronizing their actions with the MPU. This output, labelled ϕ_2T, is able to drive 5 standard TTL inputs. ϕ_2T leads ϕ_2N by a minimum of 15 nanoseconds to accomodate system delays (discussed in subsequent sections).

The reset input to the MPU, labelled RES with a bar over it

(indicates the negate of a "true" signal, i.e., a "false", causes the action which the mnemonic describes; called RES-bar), deals with power-on initialization. When the power goes on, the MPU wants to see the reset input held low for at least 8 machine cycles after the supply voltage, V_{CC}, reaches its steady state value of 5 volts. When the reset line goes high, the first clock cycle after the next ϕ_2 rising edge is used to read location $FFFE_{16}$. The details of read operations will be presented in the next subsection. The byte read is used as the most significant, or high order byte of the 16-bit program counter. The next MPU cycle is used to read location $FFFF_{16}$, which is used as the low order byte of the initial PC value. In the third cycle, the initial PC value is put on the address bus to bring in the first program instruction and the MPU is off and running. The circuit used to provide the reset input is a minor modification of the circuit Motorola suggests [9] (shown with its output waveform as Figure 4). The circuit is based on the 555 timer [13]. The timer's operation is based on two comparators, a flip-flop and a dicharge transistor. A block diagram of its internal structure is given as Figure 5. The comparator connected to the trigger input, pin 2, compares the analog voltage at that pin to 1/3 V_{CC}, if the input voltage is lower, it causes the flip-flop to set the timer output to "1" and makes the path from the dicharge pin (pin 7) to ground look like an open circuit. The comparator connected to the threshhold input, pin 6, compares the analog voltage at that pin to 2/3 V_{CC}, if it is higher, the internal flip-flop sets the output to "0" and makes the discharge pin to ground

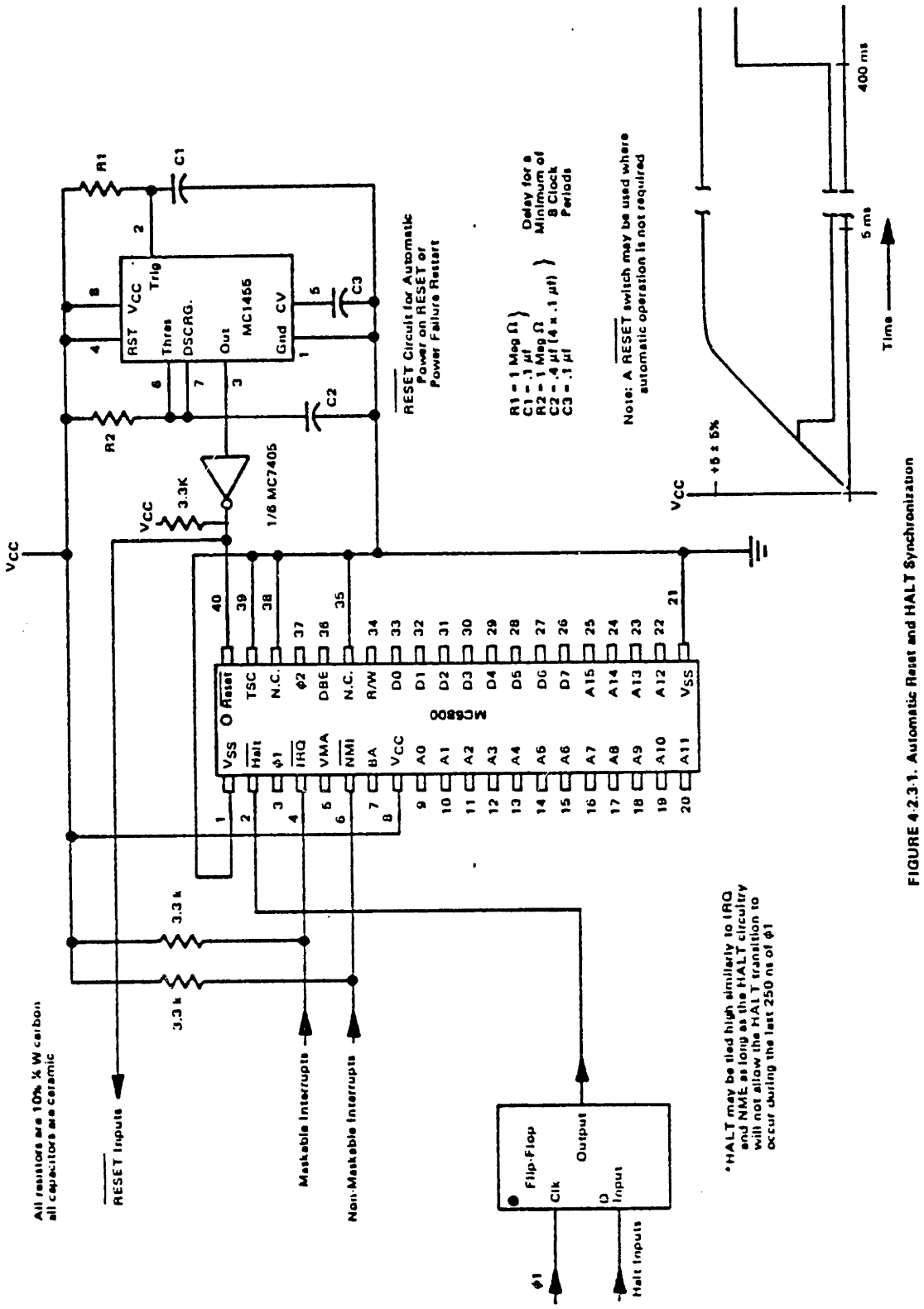


FIGURE 4-2.3-1. Automatic Reset and HALT Synchronization

Figure 4--Motorola's Suggested Reset Circuit (Source: [9])

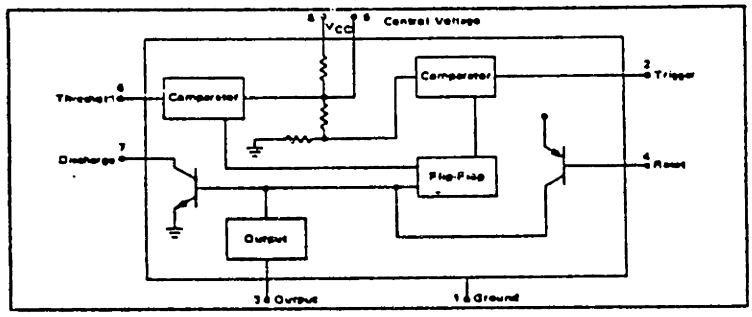


Figure 5--555 Timer Block Diagram (Source: [11])

path look like a closed circuit. Hence, with the external components as shown in the schematic, the timer output behaves as follows: 1) When the power is turned on and reaches a sufficiently high level that the timer can operate (min. voltage which guarantees operation listed in spec. sheet as 4.5 volts), both capacitor voltages will still be below 1/3 V_{CC}. 2) Hence, the output is set to "1" and the dicharge path is open. 3) This allows the threshhold RC circuit to charge the capacitor. The time constant for this RC is 400 ms. It will charge up to 2/3 V_{CC} in about one time constant. 4) When the RC gets to that value, the output is reset to low and the discharge path is closed. The threshhold capacitor is discharged. 5) By this time, the trigger RC circuit has gone four of its time constants; so, the capacitor voltage is very close to V_{CC}. Hence, both the trigger and threshhold inputs are stable in this configuration and no more action will occur until the power is next turned on. To produce the waveform in Figure 4, the output from the timer must be negated ("1" voltage made zero and "0" made one, the Boolean negate described

above). This is done using a 7406 inverter, which outputs the digital opposite, "negates", the input. This is a digital logic component from the TTL family; so, there is no problem driving it with the 555 which is specified to drive TTL. The resistor following the inverter is used because the output of the 7406 is a special type called "open collector". Open collector outputs make available at the output pin of the chip the collector of an NPN transistor to ground as shown in Figure 6. The transistor represents a short to ground when the output is low and an open circuit to ground when the output is high. Consequently, with the external resistor, the output pin, when high, acts as the model presented above with a Thevenin equivalent resistance equal to the resistor, and, when low, acts as the model above with the current limit equal to the collecter-emmitter current limit of the transistor. This output drives the reset of the MPU and the PIA with no problem, holding the line down for 400 ms, allowing the clock to start up (crystal clocks take about 100 ms after full voltage to start) and count 8 cycles (.008 ms at 1 MHz) with room to spare.

Internally, the MPU performs the system organization and arithmetic logic functions according to the software instructions and the hardware inputs of which reset is an example. Performance of these functions requires the MPU to work with bytes internally, to deal with memory, and to deal with controller system I/O. The hardware view of the latter two functions are presented from the MPU standpoint in following sub-sections. The internal byte manipulation hardware need not be considered. The only hardware

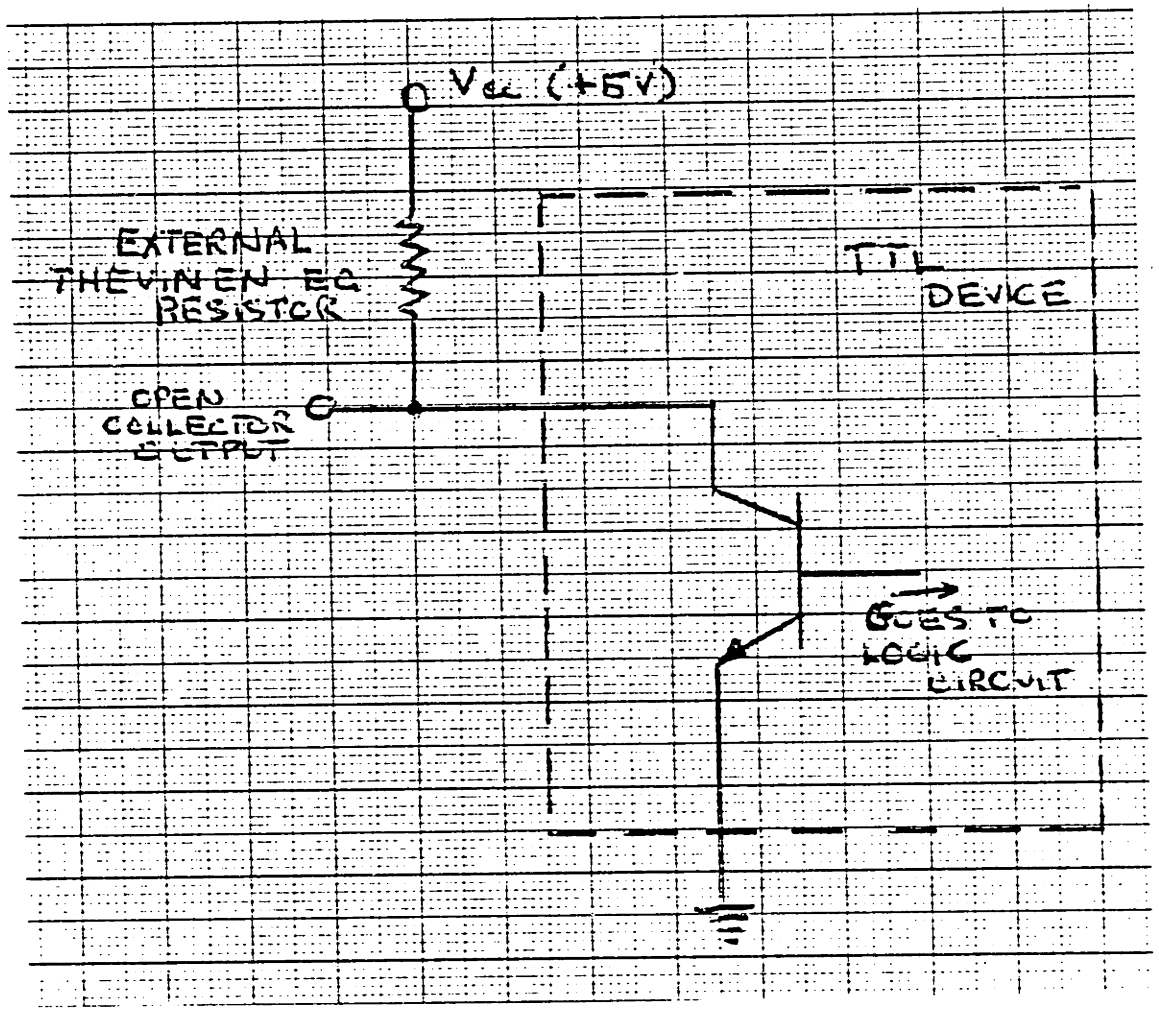


Figure 8—Open-Collector Output

inputs used in the controller system, besides the reset (which can be viewed as within the processor "unit"), are the interrupt request and the data bus enable. When the MPU pin labelled IRQ-bar (interrupt request, active low), is brought low externally, an interrupt of the MPU's action may be caused after the current software instruction is processed. The internal handling of an interrupt, briefly explained in 1.3.2, is presented in more detail in the last paragraph of 3.1.1, after the internal machine structure is described. The data bus enable (DBE in the schematic) is used to tell the MPU when the data bus should be read from or written to. The input to DBE is taken from the ϕ_2T output. When it is high, during the second half of the machine cycle, the data pins are in their normal read/write mode. When the DBE is low, the data pins go into a high impedance, inactive mode. In this state, the MPU data pins look like very high impedance inputs from the outside, and they are unable to read data on the bus. Other hardware inputs on the MPU chip include the halt input (labelled HALT in the schematic), the non-maskable interrupt (NMI), and the three-state control. These unused inputs are available on the MC6800 largely for applicational versatility and are commonly used in larger systems for single cycle execution, and direct memory access. In this systyem, these inputs are connected to the appropriate voltage level for normal operation.

2.2.2 How the Processor "Sees" Memory

The processor has 16 address pins, allowing it to access 2^{16} or 65,536 different memory locations. Since this system only has 2K (2,048) bytes of ROM (2 Intel 2708s [2]), 128 bytes of RAM (Motorola MCM6810-L [8]), and 4 locations for PIA registers (Motorola MC6820 [8]) on the address bus, each memory location can be specified uniquely using only 12 pins. (2^{12} is 4,096, greater than the number of memory locations.) Hence, only address pins A_{15}, A_{14}, and A_9-A_0 are used to address memory. The other address pins are not connected to the memory location selection circuitry. This arrangement is described graphically in Figure 7, the MPU's memory map shown below. In the memory map, the bit values of each of the address lines are shown for different types of memory; X's stand for variable bits, which when set to 1 or 0 specify a location in the memory type, and D's stand for don't care, that is, if all the other bits are set to values according to the map, the location is specified regardless of the values of the D's.

Mem Type	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
RAM	0	0	D	D	D	D	D	D	X	X	X	X	X	X	X	X
ROM	1	X	D	D	D	D	X	X	X	X	X	X	X	X	X	X
PIA	0	1	D	D	D	D	D	D	D	D	D	D	D	D	X	X

Figure 7--Memory Map

Consequently, when the MPU sets address FFFF on the bus, it addresses the same location as it would had it put out address C3FF.

From the MPU's standpoint, each location has several addresses; however, as stressed earlier, each address only has a single corresponding location.

During an MPU read cycle (a machine cycle in which a read from memory is to occur), the MC6800, with an address delay, t_{AD}, of no more than 300 nanoseconds from when ϕ_1N reaches 4.7 volts, will have the address lines, the R/W line, and the VMA line properly set. Motorola states that this will occur even in the case of the maximum load, one standard TTL load (I_{IL}=1.6mA, C_{in}>15pF) [8]. The MPU guarantees that by the address delay time the MPU will have those lines up to a voltage of at least 2.4 into the TTL load if the output is to be high, and will have them down to below .4 into the TTL if the output is low. This works well with TTL threshholds, since standard TTL reads voltages greater than 2.0 as "1", and voltages less than 0.7 as "0". In the read operation, the address lines carry the variable address, while the R/W and VMA carry 1s. In reponse to these signals, the MPU expects the memory to supply the data bus with the value of the location before 100 nanoseconds prior to the time when ϕ_2N falls below 4.7 volts. 100 nanoseconds is the data setup time, $t_{DSR(min)}$, required for the read operation; 100 nanoseconds of correct data must be provided to the MPU before ϕ_2N falls putting the data bus in the high impedance state via DBE. Near the end of the read cycle, the MPU holds the address lines, the R/W, and the VMA line at their values for at least 50 nanoseconds after ϕ_2N goes below .3 volts. This is called the address hold time, t_{AH}. The MPU expects that the memory will hold the data on

the bus for at least the minimum output data hold time, t_H (=10ns), after ϕ_2N goes below .3 volts. Since, the actual value read is the value on the bus when it is disabled, this insures that the data does not change before that occurs. The read timing is shown below in Figure 8 (Source: [8]).

During a write cycle, the address setup information presented for a read still applies; however, the R/W line is set low to indicate a write. After this, the MPU sets the data to be written into memory on the data bus. This is done in no more than 225 nanoseconds, the data delay time for a write (t_{DDW}), after the DBE line from ϕ_2T goes above 2.0 volts enabling the data pins. The MPU holds the data on the bus until at least t_H, 10 nanoseconds, after the DBE falls below .8 volts. With this information, the MPU expects the memory to read the output and to store it in the addressed location. The write timing information is presented below as Figure 9 (Source: [8]).

2.2.3 How the Processor Views System Output

The MPU controls the system output through the B side of the PIA by writing into its internal registers. These B side registers can be accessed in either direction (input or output) in the same way as memory, except that there are three internal registers and they have only two addresses. The address of the B side control register, CRB, is 4003. The functioning of various aspects of the PIA is determined by the values of the low order 6 bits in the CRA

MC6800

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	−0.3 to +7.0	Vdc
Input Voltage	V_{in}	−0.3 to +7.0	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature Range	T_{stg}	−55 to +150	°C
Thermal Resistance	θ_{JA}	70	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high impedance circuit.

READ/WRITE TIMING Figures 2 and 3, f = 1.0 MHz, Load Circuit of Figure 6.

Characteristic	Symbol	Min	Typ	Max	Unit
Address Delay	t_{AD}	–	220	300	ns
Peripheral Read Access Time $t_{acc} = t_{ut} - (t_{AD} + t_{DSR})$	t_{acc}	–	–	540	ns
Data Setup Time (Read)	t_{DSR}	100	–	–	ns
Input Data Hold Time	t_H	10	–	–	ns
Output Data Hold Time	t_H	10	25	–	ns
Address Hold Time (Address, R/W, VMA)	t_{AH}	50	75	–	ns
Enable High Time for DBE Input	t_{EH}	450	–	–	ns
Data Delay Time (Write)	t_{DDW}	–	165	225	ns
Processor Controls*					
Processor Control Setup Time	t_{PCS}	200	–	–	ns
Processor Control Rise and Fall Time	t_{PCr}, t_{PCf}	–	–	100	ns
Bus Available Delay	t_{BA}	–	–	300	ns
Three State Enable	t_{TSE}	–	–	40	ns
Three State Delay	t_{TSD}	–	–	700	ns
Data Bus Enable Down Time During φ1 Up Time (Figure 3)	t_{DBE}	150	–	–	ns
Data Bus Enable Delay (Figure 3)	t_{DBED}	300	–	–	ns
Data Bus Enable Rise and Fall Times (Figure 3)	t_{DBEr}, t_{DBEf}	–	–	25	ns

*Additional information is given in Figures 12 through 16 of the Family Characteristics – see pages 17 through 20.

FIGURE 2 – READ DATA FROM MEMORY OR PERIPHERALS

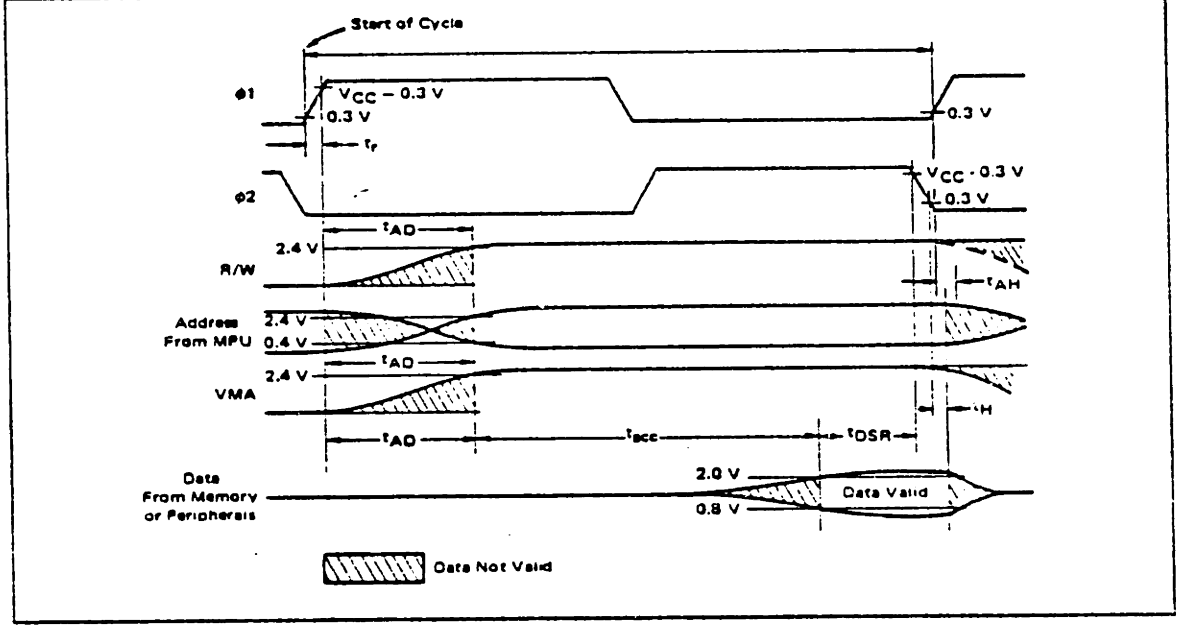


Figure 8—MPU Read Timing (Source: [8])

MC6800

FIGURE 3 – WRITE IN MEMORY OR PERIPHERALS

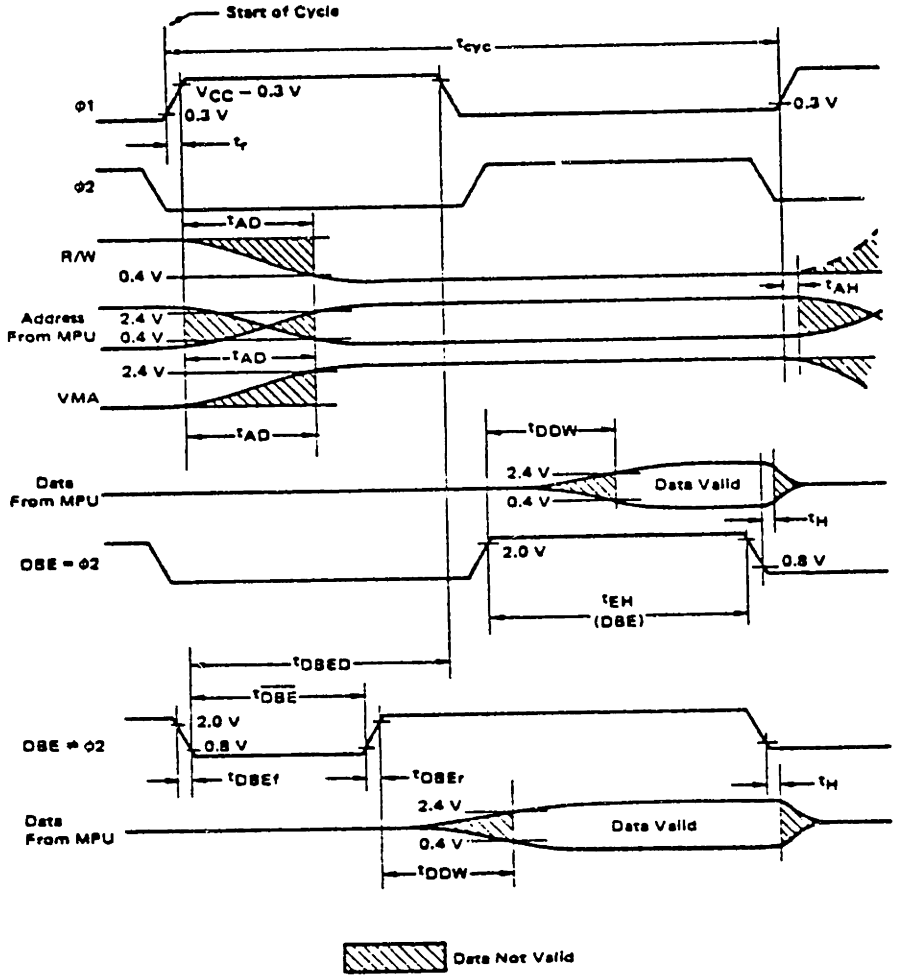


FIGURE 4 – TYPICAL DATA BUS OUTPUT DELAY versus CAPACITIVE LOADING

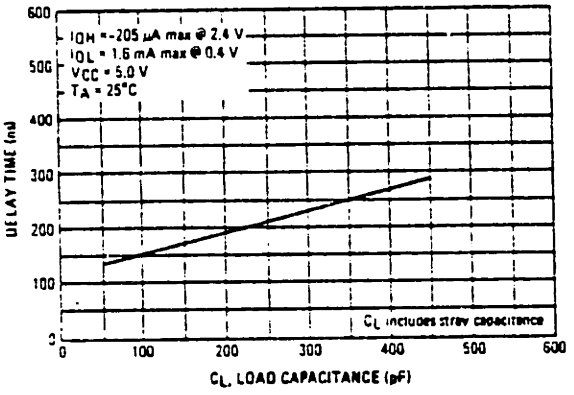


FIGURE 5 – TYPICAL READ/WRITE, VMA, AND ADDRESS OUTPUT DELAY versus CAPACITIVE LOADING

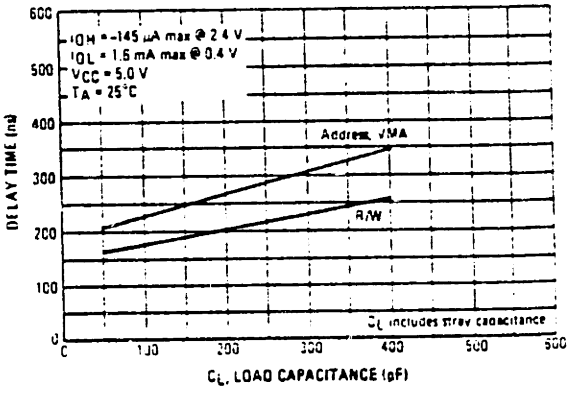


Figure 9—MPU Write Timing (Source: [8])

byte (the details of which will be dealt with in Sections 2.4 and 2.5). It is only these 6 bits of the CRB which can be directly written into by an MPU write. If the value of b_2 of the byte in CRB (CRB_2) is set to 1, the other PIA B address, 4002, represents a second internal register, the output peripheral register. This register contains the value on the peripheral line, in this case the line to the D/A. If the value of CRB_2 were 0, the address, 4002, would allow MPU access to the third B side internal register, the data direction register (DDRB). Each bit in DDRB corresponds to a bit in the peripheral output register; if a DDRB bit is 0, the corresponding bit in the peripheral register is used as an input, if 1, as an output. The MPU accomplishes system output by writing into the CRB and the DDRB, setting up the peripheral register as an output port. As the MPU subsequently writes into the peripheral register, the written value is put on the peripheral lines and becomes the new digital signal which the D/A converts into the output. Hence, after the B side of the PIA has been programmed once, system output appears to the MPU to be the same as a memory write.

2.2.4 How the Processor Views System Input

The MPU views the A/D through the A side of the PIA. Just as in the case of the B side, there are three registers and two addresses; their operation is similar to that of the B side. CRA has address 4001 and DDRA/peripheral register A have address 4000.

As opposed to the system output, however, the input control requires setting the PIA to signal the A/D to begin and to signal the MPU with an interrupt when the A/D routine is complete. Hence, the PIA must be conditioned in a more complicated way and each time an input is to be read. Programming the PIA is still done using only reads and writes into the appropriate registers (the details of which will be dealt with later). The final write into CRA starts the A/D on the conversion algorithm. When the algorithm is completed, as a result of the setting of the PIA, that device signals the MPU by bringing the IRQ (interrupt request) line down to a 0. In this system, since this is the only possible cause for an interrupt request, the MPU recognizes the request and immediately starts attending to the A/D. Using a write into CRA, the MPU halts further conversion; then it reads the result of the routine, now on the PIA's peripheral lines, by reading the peripheral register. The use of the A/D is governed, then, by MPU reads and writes. It may be viewed from the MPU as a special part of memory.

The other aspect of the system input which the MPU must control is the analog input multiplexer. This is done using the otherwise unused address lines, A_{13}-A_{10}, in a dummy read or write statement. The multiplexing circuit is configured such that when address line A_{13} is high, the binary value represented by A_{12}-A_{10} (A_{10} being the least significant bit) is stored and applied to the multiplexer. This binary number, representing decimal 0-7, when applied to the multiplexer causes the analog channel with the same number to be input to the A/D. For example, the MPU's reading of

location 2400 not only causes the internal storage of the contents of the lowest memory address (also known as location 0000, since the bits which represent the 2 and 4 are in don't care columns), but it causes the multiplexer to be set to channel 1. Any memory access cycle in which the most significant hexadecimal digit of the address is 2, 3, 6, 7, A, B, E, or F has a one on the A_{13} pin and will set the multiplexer to the channel represented by A_{12}-A_{10}. In this manner the MPU selects the input channel.

2.3 Memories—RAM, ROM, and Select Circuits

The memory in the system is based on three chips, two ROM and one RAM. Each ROM contains 1024 locations addressable through 10 address pins on the chip. The RAM has a 128 byte capacity, with locations selectable through 7 address pins. On the three chips, the address inputs are labelled A_0 through A_8 or A_9, depending on the chip. These pins are connected to the address bus. That is, A_0 on the memories is on the same line as A_0 from the MPU and so on. In order to make a memory location unique for a given address, A_{15} and A_{14} are used to select only one chip from the three (and the PIA which is also on lines A_1 and A_0 of the address bus). This is conceptually simple in that two bits can represent four different things. How it works is the high two address bits can be viewed as being decoded to a one-of-four output, in which each one is applied

to the chip select input of one of the ICs. Hence, for any given value of the high address bits, only one chip is selected. The remaining connected address pins uniquely specify the memory location.

With the hardware schematic, the memory map, and the MPU read and write timing information (pages 28, 37, 40, and 41, respectively) in mind, the byte transfer interactions from the RAM's viewpoint are as follows (RAM requirements for read and write are presented as Figures 10 and 11, respectively. The actual worst case signals with which it is presented in this system configuration for read and write are presented as Figures 12 and 13): Whether in read or write mode, the RAM wants to see a stable address at its address pins at least 20ns before it is enabled (selected). It wants to see the stabilized R/W signal, also, before it is selected. When it is selected, its behavior is dependent on whether the mode is read or write. Looking at the pre-transfer setup, using the ϕ_1 clock's positive going crossing of the 4.7 volt threshhold as time=0, it has been shown above that the address information and the R/W line from the MPU is stabilized before time=300ns. This occurs with maximum load. The RAM, the ROMs, and the PIA (the four loads on the address bus) represent a load which is significantly less than the standard TTL load used to determine the delay time; they are all NMOS devices which have less input capacitance, less input low current (by almost three orders of magnitude), and the same (1 volt higher for the ROM) input high threshhold. Thus, the address delay on the bus should easily meet the 300ns specification, and the RAM will see

MCM6810A

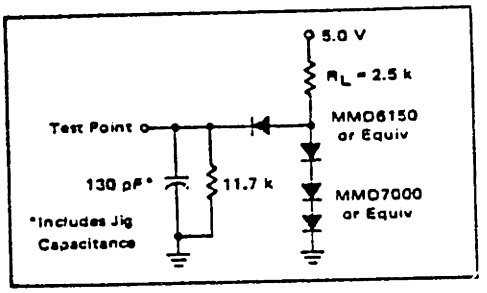
AC OPERATING CONDITIONS AND CHARACTERISTICS

(Full operating voltage and temperature unless otherwise noted.)

AC TEST CONDITIONS

Condition	Value
Input Pulse Levels	0.8 V to 2.0 V
Input Rise and Fall Times	20 ns
Output Load	See Figure 1

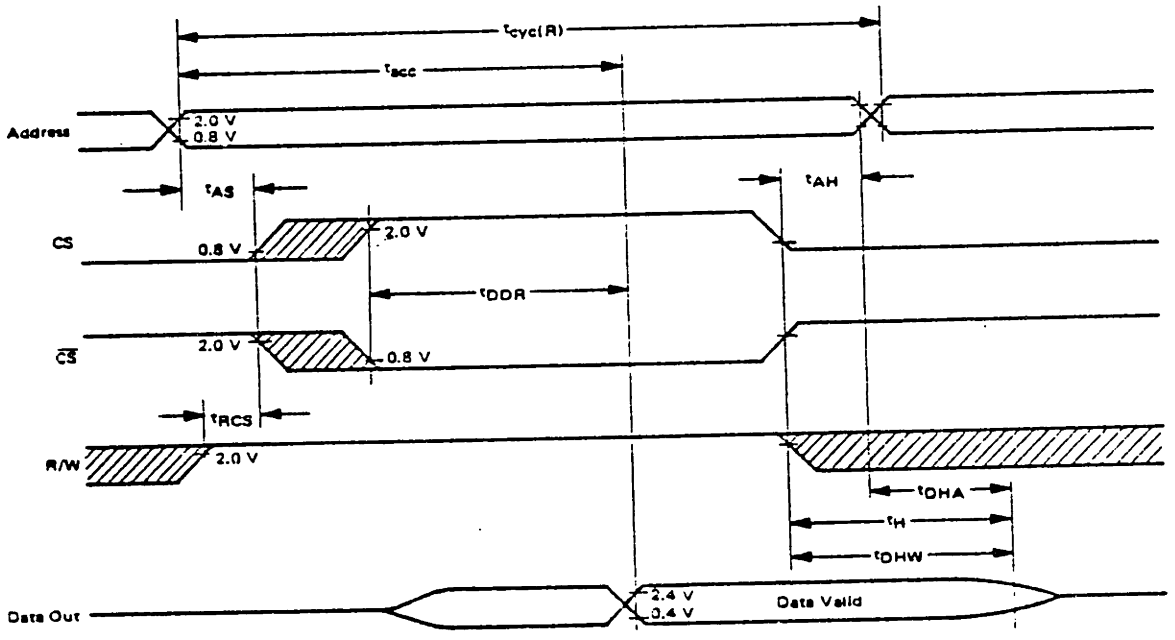
FIGURE 1 – AC TEST LOAD



READ CYCLE

Characteristic	Symbol	MCM6810AL Min	MCM6810AL Max	MCM6810AL1 Min	MCM6810AL1 Max	Unit
Read Cycle Time	$t_{cyc(R)}$	450	–	350	–	ns
Access Time	t_{acc}	–	450	–	350	ns
Address Setup Time	t_{AS}	20	–	20	–	ns
Address Hold Time	t_{AH}	0	–	0	–	ns
Data Delay Time (Read)	t_{DDR}	–	230	–	180	ns
Read to Select Delay Time	t_{RCS}	0	–	0	–	ns
Data Hold from Address	t_{DHA}	10	–	10	–	ns
Output Hold Time	t_H	10	–	10	–	ns
Data Hold from Write	t_{DHW}	10	80	10	60	ns

READ CYCLE TIMING



= Don't Care

Note: CS and $\overline{CS}$ can be enabled for consecutive read cycles provided R/W remains at V_{IH}.

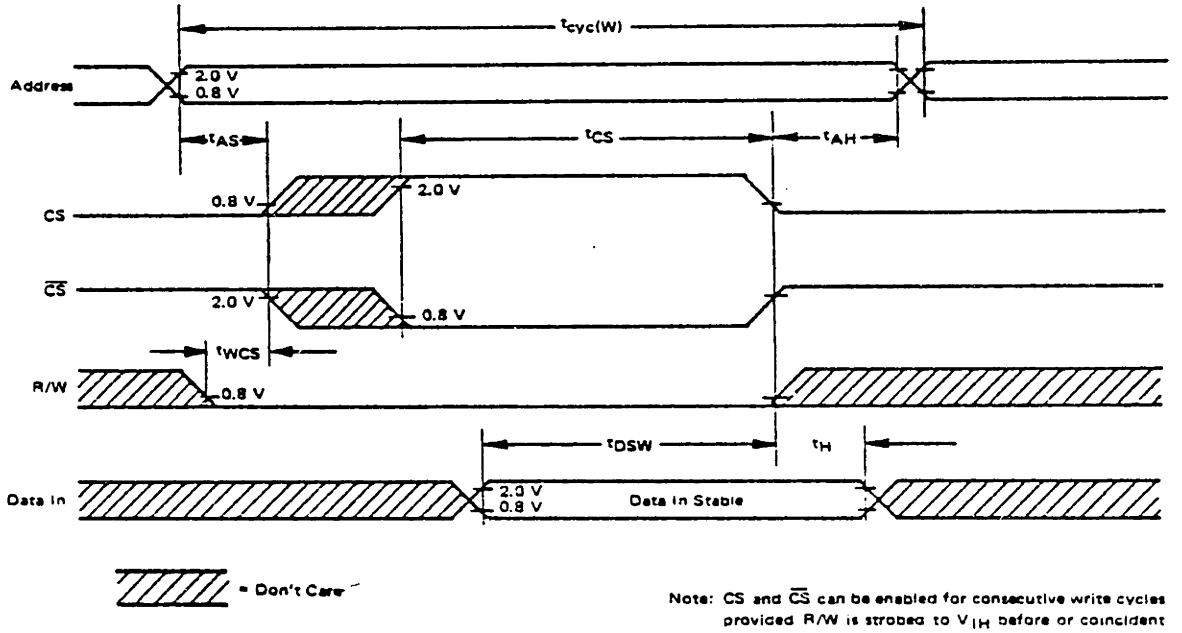
Figure 10—RAM Read Timing (Source: [8])

MCM6810A

WRITE CYCLE

Characteristic	Symbol	MCM6810AL		MCM6810AL1		Unit
		Min	Max	Min	Max	
Write Cycle Time	$t_{cyc(W)}$	450	–	350	–	ns
Address Setup Time	t_{AS}	20	–	20	–	ns
Address Hold Time	t_{AH}	0	–	0	–	ns
Chip Select Pulse Width	t_{CS}	300	–	250	–	ns
Write to Chip Select Delay Time	t_{WCS}	0	–	0	–	ns
Data Setup Time (Write)	t_{DSW}	190	–	150	–	ns
Input Hold Time	t_H	10	–	10	–	ns

WRITE CYCLE TIMING



Note: CS and $\overline{CS}$ can be enabled for consecutive write cycles provided R/W is strobed to V_{IH} before or coincident with the Address change, and remains high for time t_{AS}.

Figure 11—RAM Write Timing (Source: [8])

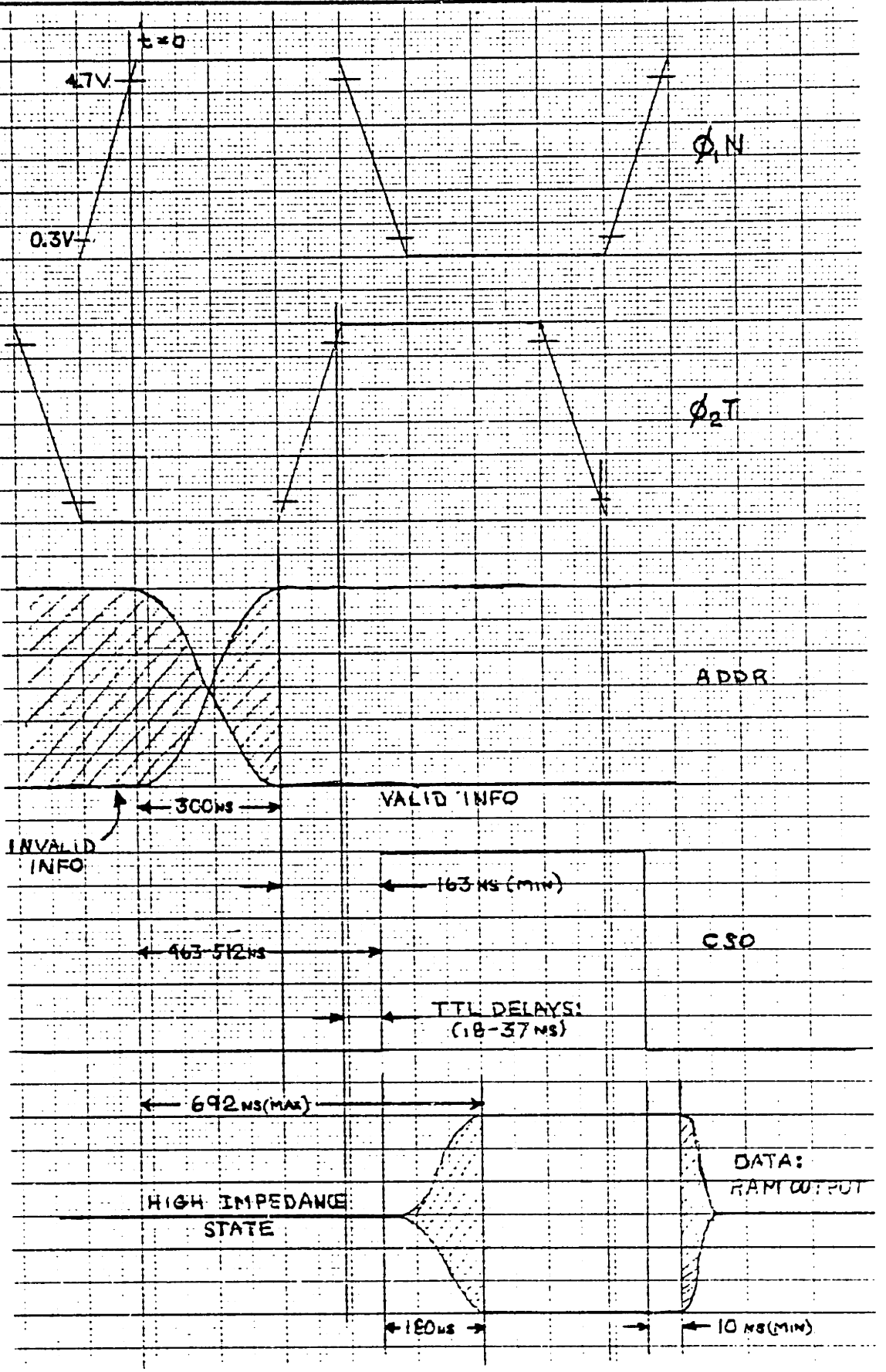


Figure 12—Actual System Read Timing

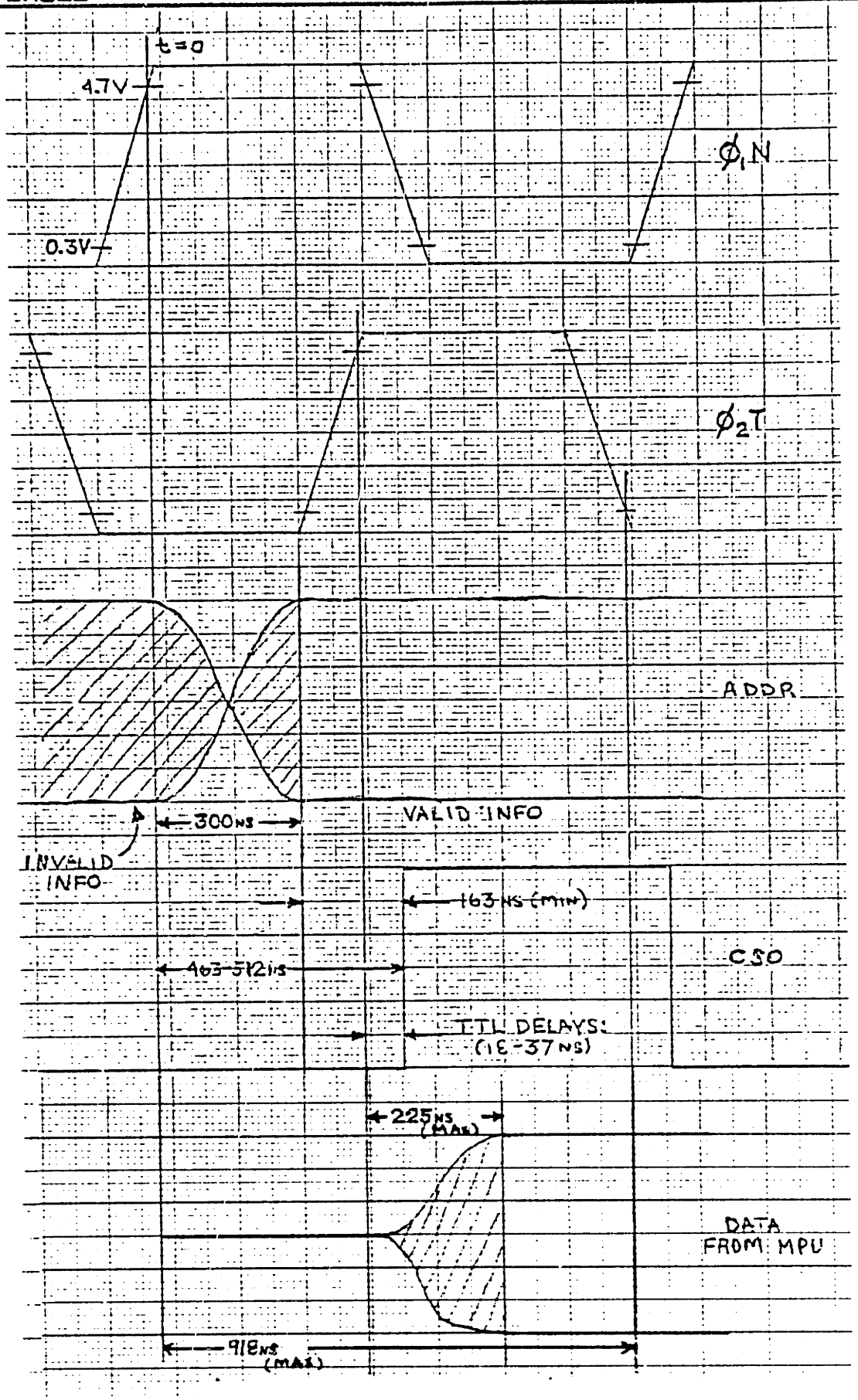


Figure 13—Actual System Write Timing

the address and the R/W before 300ns. When the RAM sees the chip select signals is a more complicated matter due to the delays from the symbolically represented low-power Schottky (LS) TTL gates which manipulate the control bit outputs from the processor. Furthermore, the actual internal chip select is only generated when all of the external chip select pins are in the selecting state. (Chip select pins are labelled on the schematic starting with the letters CS or CS-bar.) For the CS0 select, the RAM sees the logical result of VMA AND ϕ_2T. This means that the chip will not be selected until there is a valid memory address and the ϕ_2 half of the machine cycle has started. The logical result is obtained using two 2-input NAND gates. Each NAND gate performs the logical AND operation and negates the result to produce the output. The first gate produces the NAND of VMA and ϕ_2T; the second gate produces the NAND of the result and 1, which is equivalent to a negate. So, the overall result is the negate of the NAND or an AND. Neither the loading on ϕ_2T, nor the loading on VMA is greater than the specified maximum load; so, they will appear as specified in their output data. The timing of the CS0 high input is delayed from t=0 by 445-475ns due to the delay from ϕ_1N=1 to ϕ_2T=1. The two NAND gates add 18-37ns delay; this is within their specs, since the loading they see is small compared to their capability. Hence, the total delay is between 463ns and 512ns. The CS1-bar and CS2-bar inputs are driven directly by the address lines, A_{15} and A_{14}, which are loaded within spec; so, their delay from t=0 is 300ns maximum. Hence, the chip is selected no earlier than at t=463ns, allowing no less than 163ns

address (and R/W) setup time, easily meeting the 20ns requirement.

In a read cycle, the maximum time the RAM will take from CS=1 to valid data on the bus is 180ns. This implies that relative to t=0, the latest time at which valid data will appear to the MPU is 692ns. This easily meets the MPU data setup time requirement, t_{DSR}. After CS falls, the RAM holds the data on the bus for at least 10ns after the address becomes invalid. Thus, the MPU's data hold, t_H, requirement is met. Data is read.

In a write cycle, the RAM expects a select pulse width of at least 250ns, which the ϕ_2T AND VMA easily provides. In addition, it requires a data setup time, t_{DSW}, of at least 150ns before the chip is disabled. The MPU requires a maximum of 225ns after ϕ_2T is high to have valid data on the bus. Relative to t=0, this may be no later than 700ns; however, since the chip disable does not go low until 918 at the earliest, the setup requirement is met. The MPU easily meets the RAM's 10ns hold time requirement, since it becomes disabled much later than the RAM and, even so, has at least a 10ns hold period after that.

The reader may consider himself lucky to be spared such an analysis for the cases of the PIA and the two ROMs, which are more complicated due to the address decoding NAND gates and the delay inherent in the power switching circuit involving the 2N2222 transistor. The analysis has been done and the MPU timing requirements and the memory timing requirements have been met with room to spare. One may perform the above analysis for the other memory elements using information presented in the data sheets for

the involved components [2, 8, 15].

There are two features of the memory design which distinguish it from a typical microprocessor memory design. First, the power supply to the ROM is switched, such that the device which normally requires 800 mW all the time only requires that amount when it is selected. When it is not selected the requirement is reduced to 20 mW. So, even if one ROM or the other is selected 25% of the time, which is a high estimate, the power savings is 1600 mW - 235 mW or 1365 mW. The second feature of the ROM is that it is a special ultraviolet eraseable/reprogrammable chip. Since this application is one in which the software, contained in the ROM, is not fully developed and certainly not finalized, the reprogrammability feature can be important. The erase/reprogram instructions for the EPROM (eraseable programmable read only memory) are presented in reference [2]. Additionally, there are facilities at MIT for reprogramming the Biomedical Engineering Center for Clinical Instrumentation (now located in Bldg. 20) has hardware for computer controlled EPROM programming, called promming.

2.4 PIA, A/D, and Multiplexer: System Input

As mentioned earlier, the PIA is capable of being programmed by MPU writes into the internal registers. The CRA is the primary programming register for the A side, which will be exclusively dealt

with in this section. The state of the CRA determines the behavior of the two 1-bit peripheral device control ports labelled CA1 and CA2 on the schematic. CA1 may be used as an interrupt request input only; its behavior is controlled by bits 0 and 1 of CRA (CRA_0 and CRA_1). In the configuration used for the A/D conversion, CRA_0=CRA_1=1 (other configurations and their meanings are available in reference [8]). This causes the interrupt request flag in the CRA, CRA_7, one of the MPU inaccessable bits, to be set high when the CA1 pin is brought high. It also causes the IRQ-bar line to the MPU to be pulled down when CRA_7 goes high, generating an interrupt request. When the PIA is not being used for an A/D, CRA_0 is set to 0. This does not disable setting of CRA_7, but it does disable the IRQ-bar line to the MPU so no interrupt requests are generated. Once CRA_7 is set, since it is inaccessable by the MPU directly, a peripheral register read is required to reset it (i.e., make it 0). CA2 may be used as an interrupt request generating input, or it may be used as a peripheral controlling output signal, depending on the state of CRA_5. In this system it is used as an output to initiate the hardware A/D conversion sequence, CRA_5 is set to 0. The behavior of CA2 as an output is further controlled by CRA_4 and CRA_3, a summary of the various settings and their effects can be found in ref. [8]. In this application, for initiation of the A/D routine, CRA_4 is set to a 1 so the value of the output on CA2 is the same as the value in CRA_3. The secondary programming register, DDRA, is set such that the peripheral register is used as an input.

The system uses a modification of an external hardware

successive approximation type A/D converter presented in ref. [1]. This method of conversion was chosen because of its relatively fast speed and low cost (see Figure 14, Source: [1], which compares A/D techniques for the 6800). The basis for the successive

Characteristic	Successive Approximation			Dual Ramp			
	8-Bit Software	10-Bit Software	8-Bit Hardware	12-Bit Software	3½-Digit Software	4½-Digit Software	3½-Digit Hardware
External Hardware	8-Bit DAC Op Amp Comparator	10-Bit DAC Op Amp Comparator	8-Bit DAC SAR* Op Amp Comparator	MC1405	MC1405	MC1405	MC1405 MC14435 MC14558 (for 7-segment display)
Conversion Rate	700 μs Constant	1.25 ms Constant	60 μs for MPU, plus A/D Conversion Time	165 ms (max) Variable	60 ms (max) Variable	600 ms (max) Variable	183 μs (min) for MPU, plus A/D Conversion Time
Interrupt Capability	Allowed	Allowed	Allowed	Not Allowed	Not Allowed	Not Allowed	Allowed
Number of Memory Locations Required (Including PIA Configuration)	106	145	42	84	296	328	58
Serial Output Available	Yes	Yes	Yes	No	No	No	No

*Successive Approximation Register

FIGURE 1 – Relative Merits of A/D Conversion Techniques

Figure 14—A/D Method Comparison Chart (Source: [1])

approximation A/D is as follows: 1) The successive approximation register (labelled SAR in the schematic) is signalled to start conversion by the MPU controlled CA2 output of the PIA going high. 2) The dual 4-bit counter (a device containing two 4-bit binary registers. Each register can be reset, and can be incremented by the rising edge of a digital signal to its clock input, if the enable is held high and the reset is low.) is signalled to count from zero on its A side. It had not been counting on its A side, since the A side enable was held low by CA2 and the B side, which had been operating, was resetting it every other clock pulse, when QB_0 was high. The change in state of the counter, came about by the

rise of CA2 to 1. This reset the B side to disable any further A side reset inputs, and enabled the previously reset A side to commence count. In this new state, the A side counter output, QA_1, the second least significant digit, has the effect of dividing the the input clock frequency by four, providing a 250 kHz clock to the SAR. The first rising edge from this clock comes 2 microseconds after CA2 goes high, giving the SAR the necessary setup time (a listed parameter for the SAR [13]) between its SC pin high and the first clock rising edge. 3) When the SAR recieves the rising edge of the first clock pulse, it sets its most significant bit, Q_0, high in less than 1200ns. 4) When the D/A (MC1408L-8 [11]) recieves this high bit it sets up an output current (which settles to steady-state in 400ns), which causes a voltage drop across the 1KΩ resistor, such that an input voltage of greater than 1/2 full-scale (1/2 full scale=1.35 volts) is required to keep the 710 comparator's plus input above 0 volts. 5) The comparator in 40ns outputs a high level if the input voltage is greater than 1/2 full scale and otherwise outputs a zero. 6) The data from the comparator is input to the SAR and on the next rising clock edge; if the data is high, Q_0 is set to 1 for the rest of the conversion. If the data is low, Q_0 is set to 0 for the rest of the conversion. In addition, the rising clock edge causes the next lower significant bit to be set to 1 and the process iterates again, each time checking to see if the current approximation plus a 1 in the least significant bit tried so far represents a voltage greater than the input voltage. 7) When the last bit is tried, and the next rising clock pulse comes along,

latching the value of the least significant bit of the input byte, the SAR output labelled EOC, for end of conversion, goes high. This is input to the PIA through the CA1 pin and causes an interrupt request to be generated to the MPU. The MPU services the request by immediately writing to CRA to bring the CA2 pin low. This halts further conversion. The processor then reads the data from the peripheral register, thus completing the input routine.

There are a few points of note. First, the input has an implicit scale factor. At the very least the binary number represents its integer value times 2.7 volts (full-scale input)/255 (full-scale binary input). If, however, the input was caused by a potentiometer measuring an angular displacement with a transduction ratio of 1° per volt, the binary number would have an implicit scale factor of 2.7°/255. The second note is an explanation of why the clock period is cut down to 4000ns when the delay time around the loop is only 1640ns. This is because the response of the operational amplifier to a change in current load is not known. That is not one of the parameters given in the data books. Personal communication with Motorola engineers resulted in the consensus that it was of "secondary importance". The best strategy is to build the circuit and try it, clocking the SAR with the Q_0 output of the counter, which only divides by two. If it works, the hardware A/D conversion time, now 32 microseconds, can be cut in half.

The multiplexer must be set to the appropriate channel before the conversion is done. The multiplex channel number is stored in a quad "D"-type flip-flop [13] so that it can be made

available to the multiplexer even when the address bits which defined it no longer contain its value. A "D" flip-flop is a device which stores and makes available at its output whatever data was at its input when its clock input goes from low to high. The quad flip-flop has four one bit inputs, which are stored and presented at the output (called latched) when the clock goes high. To use this device to latch the values of the address bits A_{12}-A^0 when A_{13} is high, the lines A_{12}-A_{10} are applied to the input of the flip-flop. Clocking is done by the AND of (VMA AND ϕ_2T) and A_{13}. When the MPU makes a dummy read or write to activate the multiplex latch, the channel defining addresses are appropriately set. Then, no sooner than 121 nanoseconds later, the composite AND from above goes high, latching the address. This is enough delay time between data and clock to allow for the setup requirement of the flip-flop which at absolute maximum is 120 nanoseconds. The resistors on the address line and on the clock line are designed to be larger enough that when the output is low, the output current sinking ability of the driver is not exceeded; but small enough that, when the output is high, the effective Thevenin resistance of the output is decreased. Use of resistors in this manner is to speed switching times. They are called "pull-up" resistors. The multiplexer, itself is nothing more than a bunch of swiches on a chip. When a channel is not selected, the resistance from pin three to the input is very high (on the order of 50MΩ); when it is selected, the resistance is around 1050 Ω. This is why the variable resistor is included in the op-amp's feedback loop, to limit input voltage error produced by the

input bias current dropping across the multiplexer.

2.5 PIA and D/A: System Output

Compared to the input, the output is simple. CB1 and CB2 are not used to control peripherals on the B side, so programming the CRB, which is analogous to CRA, is easy. (Note: not shown in the schematic is the fact that CB1 and CB2 should be connected to ground through 3.3KΩ resistors. In fact, all unused inputs should be connected to ground through resistors to prevent damage to components by built up static charge, which in the case of high impedance inputs can reach very high potentials.) All the bits of CRB except CRB_2 may be zeros for the actual output operation. The reason why bit 2 must be on is to make the output register immediately accessable to memory. System output is performed by the D/A continuously, converting the current value in the peripheral output register to an output current proportional to its digital value. As the MPU updates the peripheral register, the analog output is automatically updated.

2.6 Power Supply

The power supply for the system is expected to be batteries carried by the amputee as he walks. Since some of the components in this system, notably, the TTL, are sensitive to voltage, regulators are used to keep the supply to the chips at a constant level. The

regulators have capacitors to ground on both their inputs and outputs so that noise, both system generated and externally generated will be kept out of the power supply. Each regulator was chosen to be able to provide current to all of the system devices using its voltage at their absolute maximum current rating. The absolute maximum power the system can draw is 3.8 watts, although, this figure is not realistic. A typical system (i.e., based on the typical power reqirements of the components) would have a power requirement of about 2.1 watts. Mercury batteries can put out about 3.5 watt-hours per ounce; so, 7.2 ounces of batteries could keep a typical system running for 12 hours.

CHAPTER III—SOFTWARE DEVELOPMENT: AN EXAMPLE

This chapter provides an example of the implementation of the Ideal Swing Phase Control algorithm based on the Ideal Torque and Ideal Velocity curves developed by C.W. Radcliffe et al. at the University of California Biomechanics Laboratory. The algorithm was presented by David Lampe in his Master's thesis (Reference [4]), in which he describes its implementation in the PDP-11 in the Knee Lab. Its implementation, with some improvements, on an MC6800 based microcomputer, the Motorola Exorciser [5,6,7], is described here. The Exorciser implementation is designed to emulate the program were it to be in the EPROM of the actual system. The shortcomings of the emulation are as follows: A) Keyboard entry for input/output was required due to lack of the special input/output hardware described above. B) Hardware signals were unavailable, so their effect was simulated using software routines which would not be included in the

actual system.

3.1 The Problem: Algorithm Goal vs. Instruction Capability

There are a limited number of types of manipulations which can be done by the 6800 in a single instruction. There are also a required number of operations which must be performed to do the control algorithm. The programmer must understand both of these sets in order to bridge the gap between them. He must code patches of instructions, using the available operations, which perform the needed operations to allow the algorithm. The next two sub-sections look at the two sets, and the following section describes the bridge.

3.1.1 Instructions: The Program Building Blocks

The MPU has five internal registers, other than the PC, which it uses for byte manipulation. There are two one byte accumulators, accumulator A (ACCA) and accumulator B (ACCB). They are used to contain bytes which are currently being operated on. They are filled by read instructions called load operations; the MPU is said to load the accumulator. There is a stack pointer, a double byte register, which is used to define an area of memory to be used as a last-in-first-out (LIFO) buffer. For example, the

stack pointer (SP) is set to some value before it is used, defining the topmost address of the LIFO stack. When the instruction, push accumulator A, is encountered, the contents of ACCA are stored in the memory location pointed to by the SP, and the SP is decremented. This may go on many times for different values in ACCA or ACCB. When the instruction, pull ACCB, is encountered, the SP is incremented and the last value pushed is loaded into ACCB. The fourth register is the double byte index register (IX). This is used when memory need be addressed, but the location is not known until it is computed by the MPU. For example, if look-up tables were used to find sine(x), where x is a computed value, each memory location in the table might correspond to an x value; and the contents to the sine(x). When x is computed, the IX would be set to the value in which the appropriate sine was stored. The next instruction might be a load ACCA in the indexed addressing mode. That instruction would contain in its first byte, the fact that it is an index addressed load of ACCA; the second byte would contain a 2's complement offset value to be added to the IX to determine the location to be accessed. The offset value does not change the contents of the IX. In the above example, the offset byte would be 00. The final register is the condition code register (CCR). This register is only six bits long, each bit representing a condition of some aspect of the processor. CCR_0 is 1 if the last operation resulted in a carry from the most significant bit of the bytes involved. CCR_1 is 1 if the last instruction resulted in an overflow of the 8-bit byte length, and so on. Complete specification of the

conditions which set the various CCR bits appear in reference [10]. The contents of the CCR are used to determine whether or not conditional branching to different parts of the program should be done.

Operand location is specified by the op code and the argument(s) following it in memory. This is done through five addressing modes of which indexed addressing, described above, is one. The trivial addressing mode is implied addressing mode. Instructions able to use that mode are defined in one byte, since they do not use values from external memory to perform their function. Immediate addressing uses the contents of the memory location following the instruction as the operand itself. Direct addressing uses the byte following the instruction itself to access memory locations 0000-00FF; the value of that byte is the low order byte of the address. Extended addressing is used to access any location in memory. The first byte following the instruction proper gives the high order byte of the operand address while the location after that gives the low order byte. There is actually a sixth addressing mode, used only by branch instructions, called relative addressing mode. In this mode, the location after the instruction provides a 2's complement byte, which can be added to (2+<the address of the branch instruction>), to provide the destination of the branch.

In the appendix, pages 74-80, there appears a complete table of the MC6800 instructions, which should be readable with the above background and the legend of symbols used.

The one actual hardware instruction, the interrupt, works as follows: 1) the contents of the current MPU registers are pushed on the stack such that the return from interrupt command (see page 76) can restore them. 2) The program counter is set to the value contained in the locations specified by address FFF8 and FFF9; the former contains the high order byte and the latter, the low order byte. 3) Normal instruction execution is continued.

3.1.2 The Ideal Profile Algorithm

The "Ideal" Profile control scheme dictates that during the swing phase of the walking cycle (defined here to mean the time from when the knee starts bending, while still touching the ground, to the time when it returns to the fully extended position, not necessarily touching the ground), the torque at the knee, τ, should be given by the expression

$$\tau = (\tau(\theta)/\omega^2(\theta)) \times (\omega_{actual})^2$$

where θ=the knee angle input from the prosthesis (full extension=0°); ω_{actual}=the angular velocity of the lower leg, relative to the upper=$d\theta/dt$; and $\tau(\theta)$ and $\omega(\theta)$ equal the "Ideal" values for torque and angular velocity, respectively, presented as a function of knee angle by the C.W. Radcliffe and the Berkley group. These parameters are available in the Lampe thesis. During the rest of the cycle, the control routine is different. When the leg

reaches full extension, the knee torque goes to the maximum value to lock it there until heel contact is made. Heel contact unlocks the brake, $\tau=0$, allowing the swing phase to begin.

3.2 The Implementation

If one looks at the expression $(\tau(\theta)/\omega^2(\theta))$, and realizes that it can be viewed as a function of θ, the operations required to perform the algorithm are simplified greatly. The first operation necessary is a read of the varibles, θ, ω_{actual}, and heel contact. θ is measured by a precision rotary potentiometer, which is turned by knee movement. The algorithm was implemented assuming that the full range of angles was between 0° and 127°; hence, the operational amplifier feeding the A/D should be set to output 0 volts at 0°, and 2.7 volts (full-scale for the A/D) at 128°. The input byte then would have an implicit scale factor of .50°. ω_{actual} is assumed to have a range from -12.8 rad/sec to +12.7 rad/sec. The value is derived by op-amp differentiation of the potentiometer output. Since the the A/D can not handle negative input voltage, the differentiator's signal should be biased to make the range of ω_{actual}s produce voltages using the full input range. After the input, the MPU is programmed to digitally remove the bias; hence, the scale factor of the prepared byte is .1 rad/sec. Heel contact is measured using a switch which should provide a voltage to the A/D above 1.35

volts, if the heel is touching; and under that amount if it is not. Of course, the threshhold is software determined, and can be changed; this value is the assumption of the implementation.

The next operation required is a check to see whether or not the knee is in the swing-phase. If it is, the program continues; if not, the trivial output for the appropriate case is generated. If in the swing phase, ω_{actual} is squared. For the sake of high speed, the multiply subroutine uses a partial lookup table. The multiply time is around 280μs. The multiply routine works by storing the products of 4-bit by 4-bit multiplies in address locations 8000-80FF. If 4-bit numbers, X and Y are to be multiplied, their product can be found in address 80XY or 80YX. The multiply routine takes advantage of this when multiplying two 8-bit 2's complement numbers (all internal data is taken to be in 2's complement by the MPU for the purpose of arithmetic operations) by first taking their absolute value, storing the sign of the result in the process. It then parses each one into half bytes and manipulates them, as shown below in Figure 15.

(In this figure, capital letters represent hexadecimal digits.)

If a=ZY and b=WV, then if c=VxY=TU, d=WxY=RS, e=ZxV=PQ, and f=ZxW=MN, the product, axb, is the result of the following sum.

```
  00TU
+ 0RS0
+ 0PQ0
+ MN00
```

Figure 15--Basis for Partial Multiply

When the multiply is complete, the program looks up the value of $(\tau(\theta)/\omega^2(\theta))$, stored in memory in 14 bits (to insure that the result from the MPU is limited in accuracy onlt by the input resolution). The scale factor of the 14 bit word is 3.588×10^{-3} in.-lb.-sec^2, chosen so that the final output of the MPU could easily be scaled. As the next operation, the MPU multiplies the lookup value by the value of ω_{actual}^2, using a procedure similar to the one presented in Figure 15. The product is then scaled to make the output byte FF correspond to an MPB torque of 300 in.-lbs., its maximum. The result is then output, producing a current proportional to MPB torque desired. The proportionality constant is 6.17μA per in.-lb.

The actual program appears in the appendix, pages 83-89. The first column in the assembly listing presented there is the statement number; the second, the address; the third, the op code; and the fourth, the argument. The text includes a mnemonic for the command (see appendix, page 74, for description), its argument, and comments to assist the reader in following the flow. The addrssing

mode is given in the assembly listing coded in the following way: no argument--implied; #--immediate; $(meaning the following number is base 16)followed by two digits--direct; $followed by four digits--extended; ,X--indexed; with branch instruction--relative. The multiply table look-up data, also, appears in the appendix, pages 90-95, along with the FORTRAN program used to generate it. The lookup table for $(\tau(\theta)/\omega^2(\theta))$ is presented on pages 96-98. This table was derived from reference [4], by graphical interpretation. Finally, on pages 99-103, a sample trace of the program is presented, showing the values in the PC, IX, ACCA, ACCB, CCR, and SP after each instruction has been executed. From the appendices containing the program and the text here, the interested reader should be able to follow the program workings step-by-step. If there is any difficulty, reference [7] will be helpful.

This program, in the actual system, would take about 2.5ms to cycle once. This implies an update frequency of roughly 400Hz; Lampe claims that his implementation on the PDP-11 only ran at 167Hz. The program requires about 40% of the available ROM space, and about 12% of the available RAM space in the system. This is without optimization of space requirement, which could free considerable memory area at the expense of little time.

CHAPTER IV--EVALUATION AND RECOMMENDATIONS

Although evaluation is difficult until the system is built and tested, it is clear that this design provides a basis for microprocessor knee control that meets or exceeds all of the specifications laid down in Section 1.2. The strengths of the design are in the areas of input capability, software development, speed, and program capacity. The input specification is exceeded by two inputs, permitting the use of inputs to vary control parameters on site, without the necessity of reprogramming. Furthermore, the availability of additional inputs provides less restriction on possible control scheme ideas. The software development comes rather easily, too, because of the relative antiquity of the MC6800. There are many people familiar with its operation; and Motorola has done an admirable job backing it up with a substantial number of software development tools. These tools include devices like the

Exorciser and the Resident Editor/Assembler, used in the work of this thesis and available for the asking at MIT. The speed of the MPU is such that control schemes eight times as complex as the most complicated one implemented now (i.e., the "Ideal" profile algorithm) may be realized. This indicates the viability of the MPU design in applications involving multiple gait mode control and EMG control. The viability is further indicated by the fraction of total available memory space used by the system in implementing the "Ideal" profile algorithm.

The design's weakness is in the area of power consumption. The power consumption of a 6800 system is necessarily less favorable than that of some of the more recent microprocessors. For example, the RCA COSMAC 1800 MPU requires 8mW to operate, as opposed to the 6800, which requires about 600mw. That savings alone, not to count power savings from less hungry system devices (like the clock module <the 1800 does not have one, it uses an external crystal with an on board oscillator> and the PIA) found with newer MPU systems, would reduce the battery weight by 25%.

The system presented is certainly a starting point. The application is a good one. Hopefully, this author's first microprocessor design will be built and used by the Knee Project. The familiarity gained by using an MPU system will better enable the project to utilize faster, more powerful, less-power consuming microprocessors when they become more available; and in the meantime, the system will eliminate the limitations of the current prosthesis emulator.

REFERENCES

1. Aldridge, Don, "Analog-to-Digital Conversion Techniques with the M6800 Microprocessor System", Application Note AN-757, Motorola Semiconductor Products Inc., 1975.

2. Greene, B., "Application of the Intel 2708 8K Eraseable PROM", Applications Note AP-17, Intel Corporation, 1976.

3. Henry, T.W., "A/D Conversion Series--Part 4: High Speed Digital-to-Analog and Analog-to-Digital Techniques", Application Note AN-702, Motorola Semiconductor Products Inc., 1974.

4. Lampe, David R., "Design of a Magnetic Particle Brake Above-Knee Prosthesis Simulator System", S.M. and Mechanical Engineering Thesis, Department of Mechanical Engineering, M.I.T., Cambridge, Mass., February 1976.

5. M6800 Co-Resident Assembler Reference Manual, First Edition, Motorola, Inc., 1976.

6. M6800 Co-Resident Editor Reference Manual, First Edition, Motorola, Inc., 1977.

7. M6800 EXORciser User's Guide, Second Edition, Motorola Semiconductor Products Inc., 1975.

8. M6800 Microcomputer System Design Data, Motorola, Inc., 1976.

9. M6800 Microprocessor Applications Manual, First Edition, Motorola Semicorductor Products Inc., 1975.

10. M6800 Programming Reference Manual, Motorola Semiconductor Products Inc., 1975.

11. MC1508L-8/MC1408L-8/MC1408L-7/MC1408L-6 Data Sheet, Motorola Semiconductor Products Inc., 1975.

12. Semiconductor Data Library, Volume 5/Series B - CMOS, Motorola Semiconductor Products Inc., 1976.

13. Semiconductor Data Library, Volume 6/Series A - Linear Integrated Circuits, Motorola Semiconductor Products Inc., 1975.

14. Torrero, Edward A., "Focus on Microprocessors", Electronics Design, Vol. 22, No. 18, pp. 52-69, Sept. 1, 1974.

15. The TTL Data Book for Design Engineers, First Edition, Texas Instruments Incorporated, 1973.

APPENDIX

PRINT ON THE FOLLOWING PAGES IS PARTIALLY ILLEGIBLE

TABLE 3 – ACCUMULATOR AND MEMORY INSTRUCTIONS

OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	BOOLEAN/ARITHMETIC OPERATION (All register labels refer to contents)	5 H	4 I	3 N	2 Z	1 V	0 C
Add	ADDA	3B	2	2	3B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	0B	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmtrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	09	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A · M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	04	3	2	E4	5	2	F4	4	3				B · M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A · M	•	•	↕	↕	R	•
	BITB	C5	2	2	05	3	2	E5	5	2	F5	4	3				B · M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A − M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	01	3	2	E1	5	2	F1	4	3				B − M	•	•	↕	↕	↕	↕
Compare Acmtrs	CBA													11	2	1	A − B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				M̄ → M	•	•	↕	↕	R	S
	COMA													43	2	1	Ā → A	•	•	↕	↕	R	S
	COMB													53	2	1	B̄ → B	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 − M → M	•	•	↕	↕	↕	↕
	NEGA													40	2	1	00 − A → A	•	•	↕	↕	↕	↕
	NEGB													50	2	1	00 − B → B	•	•	↕	↕	↕	↕
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add. of BCD Characters into BCD Format	•	•	↕	↕	↕	↕
Decrement	DEC							6A	7	2	7A	6	3				M − 1 → M	•	•	↕	↕	•	•
	DECA													4A	2	1	A − 1 → A	•	•	↕	↕	•	•
	DECB													5A	2	1	B − 1 → B	•	•	↕	↕	•	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A ⊕ M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	08	3	2	E8	5	2	F8	4	3				B ⊕ M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	↕	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	↕	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	↕	•
Load Acmtr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	•	•	↕	↕	R	•
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B + M → B	•	•	↕	↕	R	•
Push Data	PSHA													36	4	1	A → Msp, SP − 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	B → Msp, SP − 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, Msp → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, Msp → B	•	•	•	•	•	•
Rotate Left	ROL							69	7	2	79	6	3				M	•	•	↕	↕	↕	↕
	ROLA													49	2	1	A	•	•	↕	↕	↕	↕
	ROLB													59	2	1	B	•	•	↕	↕	↕	↕
Rotate Right	ROR							66	7	2	76	6	3				M	•	•	↕	↕	↕	↕
	RORA													46	2	1	A	•	•	↕	↕	↕	↕
	RORB													56	2	1	B	•	•	↕	↕	↕	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M	•	•	↕	↕	↕	↕
	ASLA													48	2	1	A	•	•	↕	↕	↕	↕
	ASLB													58	2	1	B	•	•	↕	↕	↕	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M	•	•	↕	↕	↕	↕
	ASRA													47	2	1	A	•	•	↕	↕	↕	↕
	ASRB													57	2	1	B	•	•	↕	↕	↕	↕
Shift Right, Logic	LSR							64	7	2	74	6	3				M	•	•	R	↕	↕	↕
	LSRA													44	2	1	A	•	•	R	↕	↕	↕
	LSRB													54	2	1	B	•	•	R	↕	↕	↕
Store Acmtr.	STAA	—			97	4	2	A7	6	2	B7	5	3				A → M	•	•	↕	↕	R	•
	STAB				D7	4	2	E7	6	2	F7	5	3				B → M	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A − M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B − M → B	•	•	↕	↕	↕	↕
Subtract Acmtrs.	SBA													10	2	1	A − B → A	•	•	↕	↕	↕	↕
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A − M − C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B − M − C → B	•	•	↕	↕	↕	↕
Transfer Acmtrs	TAB													16	2	1	A → B	•	•	↕	↕	R	•
	TBA													17	2	1	B → A	•	•	↕	↕	R	•
Test, Zero or Minus	TST							6D	7	2	7D	6	3				M − 00	•	•	↕	↕	R	R
	TSTA													4D	2	1	A − 00	•	•	↕	↕	R	R
	TSTB													5D	2	1	B − 00	•	•	↕	↕	R	R

LEGEND:

- OP Operation Code (Hexadecimal)
- ~ Number of MPU Cycles
- # Number of Program Bytes
- + Arithmetic Plus
- − Arithmetic Minus
- · Boolean AND
- Msp Contents of memory location pointed to be Stack Pointer
- + Boolean Inclusive OR
- ⊕ Boolean Exclusive OR
- M̄ Complement of M
- → Transfer Into
- 0 Bit = Zero
- 00 Byte = Zero

Note – Accumulator addressing mode instructions are included in the column for IMPLIED addressing

CONDITION CODE SYMBOLS:

- H Half-carry from bit 3
- I Interrupt mask
- N Negative (sign bit)
- Z Zero (byte)
- V Overflow, 2's complement
- C Carry from bit 7
- R Reset Always
- S Set Always
- ↕ Test and set if true, cleared otherwise
- • Not Affected

MC6800

TABLE 4 – INDEX REGISTER AND STACK MANIPULATION INSTRUCTIONS

		IMMED			DIRECT			INDEX			EXTND			IMPLIED				COND. CODE REG.					
																		5	4	3	2	1	0
POINTER OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	BOOLEAN/ARITHMETIC OPERATION	H	I	N	Z	V	C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$X_H - M, X_L - (M+1)$	•	•	↕	↕	↕	•
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	•	•	↕	↕	•	•
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	•	•	↕	↕	•	•
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	6	2	FE	5	3				$M \rightarrow X_H, (M+1) \rightarrow X_L$	•	•	↕	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M+1) \rightarrow SP_L$	•	•	↕	↕	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M+1)$	•	•	↕	↕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M+1)$	•	•	↕	↕	R	•
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

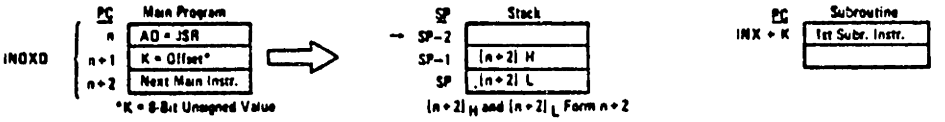
TABLE 5 – JUMP AND BRANCH INSTRUCTIONS

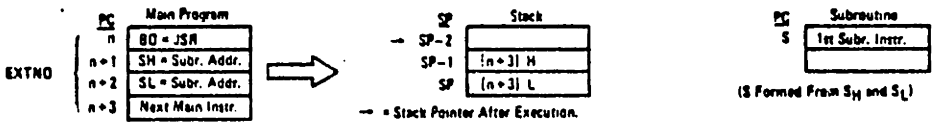
		RELATIVE			INDEX			EXTND			IMPLIED				COND. CODE REG.						
															5	4	3	2	1	0	
OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	BRANCH TEST	H	I	N	Z	V	C	
Branch Always	BRA	20	4	2										None	•	•	•	•	•	•	
Branch If Carry Clear	BCC	24	4	2										C = 0	•	•	•	•	•	•	
Branch If Carry Set	BCS	25	4	2										C = 1	•	•	•	•	•	•	
Branch If = Zero	BEQ	27	4	2										Z = 1	•	•	•	•	•	•	
Branch If ≥ Zero	BGE	2C	4	2										N ⊕ V = 0	•	•	•	•	•	•	
Branch If > Zero	BGT	2E	4	2										Z + (N ⊕ V) = 0	•	•	•	•	•	•	
Branch If Higher	BHI	22	4	2										C + Z = 0	•	•	•	•	•	•	
Branch If ≤ Zero	BLE	2F	4	2										Z + (N ⊕ V) = 1	•	•	•	•	•	•	
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1	•	•	•	•	•	•	
Branch If < Zero	BLT	2D	4	2										N ⊕ V = 1	•	•	•	•	•	•	
Branch If Minus	BMI	2B	4	2										N = 1	•	•	•	•	•	•	
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	•	•	•	•	•	•	
Branch If Overflow Clear	BVC	28	4	2										V = 0	•	•	•	•	•	•	
Branch If Overflow Set	BVS	29	4	2										V = 1	•	•	•	•	•	•	
Branch If Plus	BPL	2A	4	2										N = 0	•	•	•	•	•	•	
Branch To Subroutine	BSR	8D	8	2											•	•	•	•	•	•	
Jump	JMP				6E	4	2	7E	3	3				See Special Operations	•	•	•	•	•	•	
Jump To Subroutine	JSR				AD	8	2	BD	9	3					•	•	•	•	•	•	
No Operation	NOP										01	2	1	Advances Prog. Cntr. Only	•	•	•	•	•	•	
Return From Interrupt	RTI										3B	10	1		———	———	———	———	———	———	
Return From Subroutine	RTS										39	5	1		•	•	•	•	•	•	
Software Interrupt	SWI										3F	12	1	See Special Operations	•	•	•	•	•	•	
Wait for Interrupt*	WAI										3E	9	1		•	•	•	•	•	•	

*WAI puts Address Bus, R/W, and Data Bus in the three-state mode while VMA is held low.

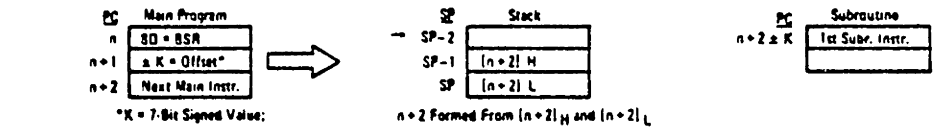
SPECIAL OPERATIONS

JSR, JUMP TO SUBROUTINE:



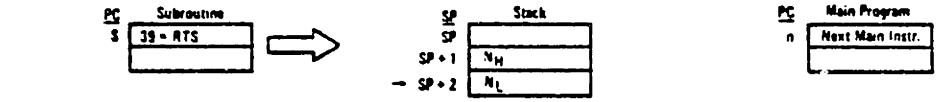


BSR, BRANCH TO SUBROUTINE:



JMP, JUMP:

RTS, RETURN FROM SUBROUTINE:



RTI, RETURN FROM INTERRUPT:

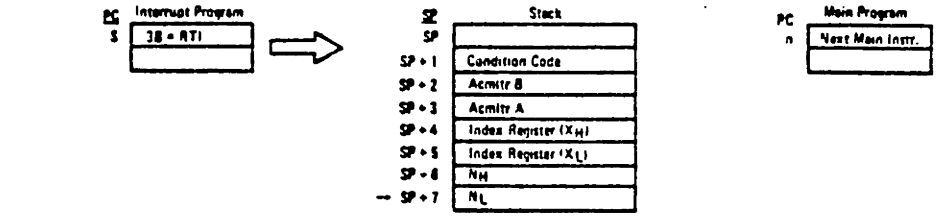


TABLE 6 — CONDITION CODE REGISTER MANIPULATION INSTRUCTIONS

OPERATIONS	MNEMONIC	IMPLIED OP	~	=	BOOLEAN OPERATION	COND. CODE REG. 5 H	4 I	3 N	2 Z	1 V	0 C
Clear Carry	CLC	0C	2	1	0 → C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1 → V	•	•	•	•	S	•
Acmltr A → CCR	TAP	06	2	1	A → CCR	——— (12) ———					
CCR → Acmltr A	TPA	07	2	1	CCR → A	•	•	•	•	•	•

CONDITION CODE REGISTER NOTES: (Bit set if test is true and cleared otherwise)

1 (Bit V) Test: Result = 10000000?
2 (Bit C) Test: Result = 00000000?
3 (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
4 (Bit V) Test: Operand = 10000000 prior to execution?
5 (Bit V) Test: Operand = 01111111 prior to execution?
6 (Bit V) Test: Set equal to result of N⊕C after shift has occurred.
7 (Bit N) Test: Sign bit of most significant (MS) byte = 1?
8 (Bit V) Test: 2's complement overflow from subtraction of MS byte?
9 (Bit N) Test: Result less than zero? (Bit 15 = 1)
10 (All) Load Condition Code Register from Stack. (See Special Operations)
11 (Bit I) Set when interrupt occurs. If previously set, a Non Maskable Interrupt is required to exit the wait state.
12 (All) Set according to the contents of Accumulator A.

MC6800

SUMMARY OF CYCLE BY CYCLE OPERATION

Table 8 provides a detailed description of the information present on the Address Bus, Data Bus, Valid Memory Address line (VMA), and the Read/Write line (R/W) during each cycle for each instruction.

This information is useful in comparing actual with expected results during debug of both software and hardware as the control program is executed. The information is categorized in groups according to Addressing Mode and Number of Cycles per instruction. (In general, instructions with the same Addressing Mode and Number of Cycles execute in the same manner; exceptions are indicated in the table.)

TABLE 8 – OPERATION SUMMARY

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
IMMEDIATE						
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	2	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Operand Data
CPX LDS LDX	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Operand Data (High Order Byte)
		3	1	Op Code Address + 2	1	Operand Data (Low Order Byte)
DIRECT						
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	1	Address of Operand	1	Operand Data
CPX LDS LDX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	1	Address of Operand	1	Operand Data (High Order Byte)
		4	1	Operand Address + 1	1	Operand Data (Low Order Byte)
STA	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Destination Address
		3	0	Destination Address	1	Irrelevant Data (Note 1)
		4	1	Destination Address	0	Data from Accumulator
STS STX	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	0	Address of Operand	1	Irrelevant Data (Note 1)
		4	1	Address of Operand	0	Register Data (High Order Byte)
		5	1	Address of Operand + 1	0	Register Data (Low Order Byte)
INDEXED						
JMP	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	1	Index Register Plus Offset	1	Operand Data
CPX LDS LDX	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	1	Index Register Plus Offset	1	Operand Data (High Order Byte)
		6	1	Index Register Plus Offset + 1	1	Operand Data (Low Order Byte)

MC6800

TABLE 8 – OPERATION SUMMARY (Continued)

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
INDEXED (Continued)						
STA	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	0	Index Register Plus Offset	1	Irrelevant Data (Note 1)
		6	1	Index Register Plus Offset	0	Operand Data
ASL LSR ASR NEG CLR ROL COM ROR DEC TST INC	7	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	1	Index Register Plus Offset	1	Current Operand Data
		6	0	Index Register Plus Offset	1	Irrelevant Data (Note 1)
		7	1/0 (Note 3)	Index Register Plus Offset	0	New Operand Data (Note 3)
STS STX	7	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
		5	0	Index Register Plus Offset	1	Irrelevant Data (Note 1)
		6	1	Index Register Plus Offset	0	Operand Data (High Order Byte)
		7	1	Index Register Plus Offset + 1	0	Operand Data (Low Order Byte)
JSR	8	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer	0	Return Address (Low Order Byte)
		5	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		6	0	Stack Pointer – 2	1	Irrelevant Data (Note 1)
		7	0	Index Register	1	Irrelevant Data (Note 1)
		8	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (Note 1)
EXTENDED						
JMP	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Jump Address (High Order Byte)
		3	1	Op Code Address + 2	1	Jump Address (Low Order Byte)
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Operand Data
CPX LDS LDX	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Operand Data (High Order Byte)
		5	1	Address of Operand + 1	1	Operand Data (Low Order Byte)
STA A STA B	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Destination Address (High Order Byte)
		3	1	Op Code Address + 2	1	Destination Address (Low Order Byte)
		4	0	Operand Destination Address	1	Irrelevant Data (Note 1)
		5	1	Operand Destination Address	0	Data from Accumulator
ASL LSR ASR NEG CLR ROL COM ROR DEC TST INC	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Current Operand Data
		5	0	Address of Operand	1	Irrelevant Data (Note 1)
		6	1/0 (Note 3)	Address of Operand	0	New Operand Data (Note 3)

MC6800

TABLE 8 – OPERATION SUMMARY (Continued)

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
EXTENDED (Continued)						
STS STX	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	0	Address of Operand	1	Irrelevant Data (Note 1)
		5	1	Address of Operand	0	Operand Data (High Order Byte)
		6	1	Address of Operand + 1	0	Operand Data (Low Order Byte)
JSR	9	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Subroutine (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Subroutine (Low Order Byte)
		4	1	Subroutine Starting Address	1	Op Code of Next Instruction
		5	1	Stack Pointer	0	Return Address (Low Order Byte)
		6	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		7	0	Stack Pointer – 2	1	Irrelevant Data (Note 1)
		8	0	Op Code Address + 2	1	Irrelevant Data (Note 1)
		9	1	Op Code Address + 2	1	Address of Subroutine (Low Order Byte)
INHERENT						
ABA DAA SEC ASL DEC SEI ASR INC SEV CBA LSR TAB CLC NEG TAP CLI NOP TBA CLR ROL TPA CLV ROR TST COM SBA	2	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
DES DEX INS INX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Previous Register Contents	1	Irrelevant Data (Note 1)
		4	0	New Register Contents	1	Irrelevant Data (Note 1)
PSH	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	1	Stack Pointer	0	Accumulator Data
		4	0	Stack Pointer – 1	1	Accumulator Data
PUL	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Stack Pointer	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer + 1	1	Operand Data from Stack
TSX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Stack Pointer	1	Irrelevant Data (Note 1)
		4	0	New Index Register	1	Irrelevant Data (Note 1)
TXS	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Index Register	1	Irrelevant Data
		4	0	New Stack Pointer	1	Irrelevant Data
RTS	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (Note 2)
		3	0	Stack Pointer	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer + 1	1	Address of Next Instruction (High Order Byte)
		5	1	Stack Pointer + 2	1	Address of Next Instruction (Low Order Byte)

MC6800

TABLE 8 – OPERATION SUMMARY (Continued)

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
INHERENT (Continued)						
WAI	9	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	1	Stack Pointer	0	Return Address (Low Order Byte)
		4	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		5	1	Stack Pointer – 2	0	Index Register (Low Order Byte)
		6	1	Stack Pointer – 3	0	Index Register (High Order Byte)
		7	1	Stack Pointer – 4	0	Contents of Accumulator A
		8	1	Stack Pointer – 5	0	Contents of Accumulator B
		9	1	Stack Pointer – 6 (Note 4)	1	Contents of Cond. Code Register
RTI	10	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (Note 2)
		3	0	Stack Pointer	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer + 1	1	Contents of Cond. Code Register from Stack
		5	1	Stack Pointer + 2	1	Contents of Accumulator B from Stack
		6	1	Stack Pointer + 3	1	Contents of Accumulator A from Stack
		7	1	Stack Pointer + 4	1	Index Register from Stack (High Order Byte)
		8	1	Stack Pointer + 5	1	Index Register from Stack (Low Order Byte)
		9	1	Stack Pointer + 6	1	Next Instruction Address from Stack (High Order Byte)
		10	1	Stack Pointer + 7	1	Next Instruction Address from Stack (Low Order Byte)
SWI	12	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (Note 1)
		3	1	Stack Pointer	0	Return Address (Low Order Byte)
		4	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		5	1	Stack Pointer – 2	0	Index Register (Low Order Byte)
		6	1	Stack Pointer – 3	0	Index Register (High Order Byte)
		7	1	Stack Pointer – 4	0	Contents of Accumulator A
		8	1	Stack Pointer – 5	0	Contents of Accumulator B
		9	1	Stack Pointer – 6	0	Contents of Cond. Code Register
		10	0	Stack Pointer – 7	1	Irrelevant Data (Note 1)
		11	1	Vector Address FFFA (Hex)	1	Address of Subroutine (High Order Byte)
		12	1	Vector Address FFF8 (Hex)	1	Address of Subroutine (Low Order Byte)
RELATIVE						
BCC BHI BNE BCS BLE BPL BEQ BLS BRA BGE BLT BVC BGT BMI BVS	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Branch Offset
		3	0	Op Code Address + 2	1	Irrelevant Data (Note 1)
		4	0	Branch Address	1	Irrelevant Data (Note 1)
BSR	8	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Branch Offset
		3	0	Return Address of Main Program	1	Irrelevant Data (Note 1)
		4	1	Stack Pointer	0	Return Address (Low Order Byte)
		5	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		6	0	Stack Pointer – 2	1	Irrelevant Data (Note 1)
		7	0	Return Address of Main Program	1	Irrelevant Data (Note 1)
		8	0	Subroutine Address	1	Irrelevant Data (Note 1)

Note 1. If device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.
Note 2. Data is ignored by the MPU.
Note 3. For TST, VMA = 0 and Operand data does not change.
Note 4. While the MPU is waiting for the interrupt, Bus Available will go high indicating the following states of the control lines: VMA is low; Address Bus, R/W, and Data Bus are all in the high impedance state.

MC6800

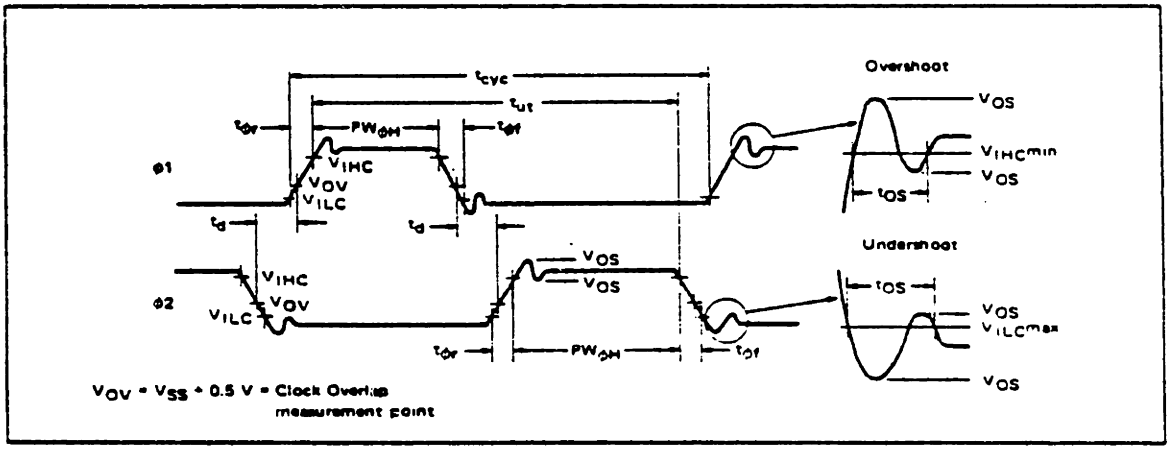
ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 V ± 5%, V_{SS} = 0, T_A = 0 to 70°C unless otherwise noted.)

Characteristic		Symbol	Min	Typ	Max	Unit
Input High Voltage	Logic	V_{IH}	V_{SS} + 2.0	–	V_{CC}	Vdc
	ϕ1,ϕ2	V_{IHC}	V_{CC} – 0.3	–	V_{CC} + 0.1	
Input Low Voltage	Logic	V_{IL}	V_{SS} – 0.3	–	V_{SS} + 0.8	Vdc
	ϕ1,ϕ2	V_{ILC}	V_{SS} – 0.1	–	V_{SS} + 0.3	
Clock Overshoot/Undershoot – Input High Level		V_{OS}	V_{CC} – 0.5	–	V_{CC} + 0.5	Vdc
– Input Low Level			V_{SS} – 0.5	–	V_{SS} + 0.5	
Input Leakage Current		I_{in}				μAdc
(V_{in} = 0 to 5.25 V, V_{CC} = max)	Logic*		–	1.0	2.5	
(V_{in} = 0 to 5.25 V, V_{CC} = 0.0 V)	ϕ1,ϕ2		–	–	100	
Three-State (Off State) Input Current	D0-D7	I_{TSI}	–	2.0	10	μAdc
(V_{in} 0.4 to 2.4 V, V_{CC} = max)	A0-A15,R/W		–	–	100	
Output High Voltage		V_{OH}				Vdc
(I_{Load} = –205 μAdc, V_{CC} = min)	D0-D7		V_{SS} + 2.4	–	–	
(I_{Load} = –145 μAdc, V_{CC} = min)	A0-A15,R/W,VMA		V_{SS} + 2.4	–	–	
(I_{Load} = –100 μAdc, V_{CC} = min)	BA		V_{SS} + 2.4	–	–	
Output Low Voltage (I_{Load} = 1.6 mAdc, V_{CC} = min)		V_{OL}	–	–	V_{SS} + 0.4	Vdc
Power Dissipation		P_D	–	0.600	1.2	W
Capacitance #	ϕ1,ϕ2	C_{in}	80	120	160	pF
(V_{in} = 0, T_A = 25°C, f = 1.0 MHz)	TSC		–	–	15	
	DBE		–	7.0	10	
	D0-D7		–	10	12.5	
	Logic Inputs		–	6.5	8.5	
	A0-A15,R/W,VMA	C_{out}	–	–	12	pF
Frequency of Operation		f	0.1	–	1.0	MHz
Clock Timing (Figure 1)						
Cycle Time		t_{cyc}	1.0	–	10	μs
Clock Pulse Width		$PW_{\phi H}$				ns
(Measured at V_{CC} – 0.3 V)	ϕ1		430	–	4500	
	ϕ2		450	–	4500	
Total ϕ1 and ϕ2 Up Time		t_{ut}	940	–	–	ns
Rise and Fall Times	ϕ1,ϕ2	$t_{\phi r}$, $t_{\phi f}$	5.0	–	50	ns
(Measured between V_{SS} + 0.3 V and V_{CC} – 0.3 V)						
Delay Time or Clock Separation		t_d	0	–	9100	ns
(Measured at V_{OV} = V_{SS} + 0.5 V)						
Overshoot Duration		t_{OS}	0	–	40	ns

*Except $\overline{IRQ}$ and $\overline{NMI}$, which require 3 kΩ pullup load resistors for wire-OR capability at optimum operation.

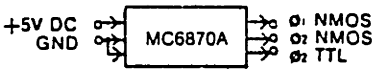
#Capacitances are periodically sampled rather than 100% tested.

FIGURE 1 – CLOCK TIMING WAVEFORM

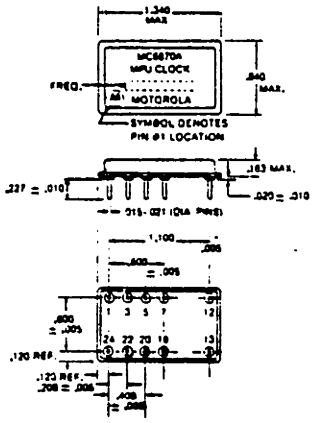


MC6870A

limited function microprocessor clock
250 kHz to 2.5 MHz



DIMENSIONS

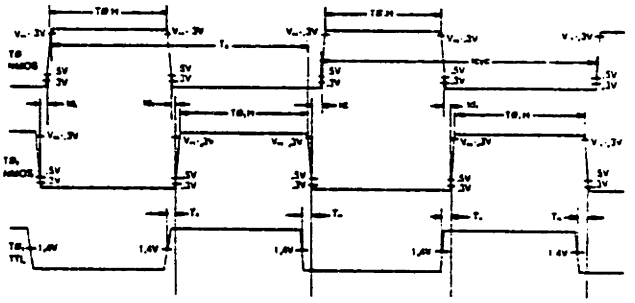


PIN	CONNECTION
1	GND
3	NC
5	Ø2 TTL
7	V_{cc} (+5VDC)
12	Ø1 NMOS
13	Ø2 NMOS
18	GND
20	NC
22	NC
24	NC

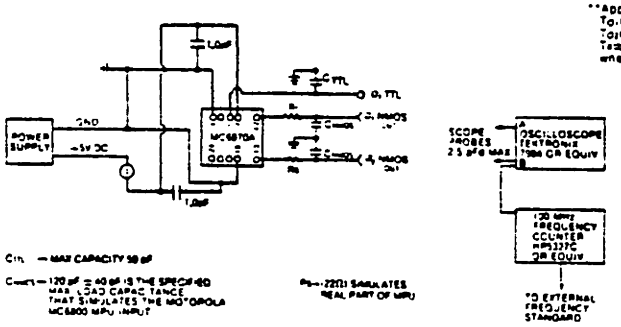
Note: All dimensions are in inches

WAVEFORM TIMING

(ALL TIME IN NANOSECONDS)



TEST CIRCUIT



Cin = MAX CAPACITY 50 pF

C_{max} = 120 pF = 40 pF IS THE SPECIFIED MAX. LOAD CAPACITANCE THAT SIMULATES THE MOTOROLA MC6800 MPU INPUT

R_{in} = 220Ω SIMULATES REAL PART OF MPU

specifications

Rating	Symbol	Value	Unit
Supply Voltage	V_{cc}	5.00±5%	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature	T_{stg}	−55 to +125	°C
Power Supply Drain (max.)	I_{cc}	100	mA

ELECTRICAL CHARACTERISTICS (V_{cc} = 5.0 ± 5%, V_{ss} = 0, T_A = 0° to 70°C, unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
Frequency					
Operating Frequency	f_o	250		2.5	MHz
Frequency stability (inclusive of calibration tolerance at +25°C, operating temperature, input voltage change, load change, aging, shock and vibration)			±.01		%
NMOS Outputs at 1.0 MHz Operation**					
Pulse Width (meas. at V_{cc} = −.3V dc level)	TØ1H TØ2H	430 450			ns ns
Logic Levels	V_{OLC} V_{OHC}	V_{ss} - 1 V_{cc} - .3	− −	V_{cc} + 3 V_{cc} + 1	Vdc Vdc
Rise and Fall Times	t_r t_f	5 5	12 12	50 50	ns ns
*Overshoot/Undershoot Logic "1" Logic "0"	V_{os}	V_{ss} - 5 V_{ss} - 5		V_{cc} + 5 V_{cc} + 5	Vdc Vdc
Pulse duration of any overshoot or undershoot	T_{os}			40	ns
Period @ 0.3V dc Level	t_{cyc}		1.00		µs
Edge Timing @ V_{cc}=0.3V dc	T_X	940			ns
NMOS Relationship @ +0.5V dc Level	t_{d1} t_{d2}	0 0		80	µs
TTL Outputs					
In ref. to Ø2 NMOS @ 0.3V dc					
Ø2 TTL @ +1.4V dc	T_A T_{AB}	15 10	30 25	45 40	ns ns
Logic Levels	V_{OH} V_{OL}	2.4	3.2 3	 .4	Vdc Vdc
Rise and Fall Times .4V and 2.4V 2.4V and .4V	t_r t_f			15 15	ns ns
Logic "0" Sink (1 Gate)	I_{OL}			−1.6	mA
Logic "1" Source (1 Gate)	I_{OH}			−40	µA
Current Output Shorted	I_{SC}	−18		−57	mA
Load					
NMOS—Load Capacity Ø1, Ø2	C_{max}	80	120	160	pf
TTL—No. of Loads				5	
TTL—Load Capacity	C_{AB}			50	pf

*Into specified test load

**Apply the following parameters for frequencies other than 1.0 MHz
T_{ON}=P-0.5 (P-140) ns
T_{OH}=P-2.5 (P-100) ns
T_{AB}=P-60 ns
where P=desired period of operation in nanoseconds

PRINT ON THE FOLLOWING PAGES IS PARTIALLY ILLEGIBLE


```
PAGE  001   IDL.PRFL


00001                       NAM     IDL.PRFL
00002 8290                  ORG     $8290
00003                       OPT     M
00004                       OPT     S



00006                   * ---------------------------------------
00007 8290 86 FF            LDA A   #$FF
00008 8292 B7 8800          STA A   PIADDR   SETS DATA DIRECTION REG
00009                   * B TO ALL ONES CONFIGURING THE B HALF FOR OUTPUT.
00010                   *PIADDR IN THE REAL SYSTEM WOULD HAVE ADDR=$4002
00011                   *(PROVIDED CRB-2=0). IN THIS SIMULATION THE ADDR IS
00012                   * $8800. HENCE,
00013       8800        PIADDR EQU   $8800
00014 8295 86 04            LDA A   #$04
00015 8297 B7 8801          STA A   PIACCR   THIS SETS CRB-2=1 MAKING
00016                   *$4002 THE ADDRESS FOR THE OUTPUT PERIPHERAL REG
00017                   * (SIMULATOR ADDR $8800).
00018                   *IN REAL SYSTEM PIACCR ADDR=$4003. SIMULATOR
00019                   * ADDR $8801.
00020       8801        PIACCR EQU   $8801
00021 829A 20 2E            BRA     START
00022                   *----------THE ABOVE SETS THE B SIDE OF THE---------
00023                   *            PIA FOR OUTPUT



00025                   *---------SUBROUTINES (EXCPT MULT)-----------
00026 829C 86 34        CONVRT LDA A  #$34      A TO D CONV ROUTINE
00027 829E B7 8802          STA A   PIAACR   ENABLES CLR INTERRUPT
00028                   *FLAG CRA-7 BY DUMMY READ
00029                   *PIAADR REAL SYSTEM ADDR $4001 (SIMUL ADDR $8802).
00030       8802        PIAACR EQU   $8802
00031 82A1 86 3F            LDA A   #$3F
00032 82A3 F6 8803          LDA B   PIAADR   DUMMY READ: CLR INT FLAG
00033                   *PIAADR REAL SYS ADDR $4000 (SIMUL ADDR $8803)
00034       8803        PIAADR EQU   $8803
00035 82A6 B7 8802          STA A   PIAACR   START CONV: CA1 UNMASKED,
00036                   *+ EDGE ACTIVE; CA2--HIGH
00037 82A9 01               NOP              IN REAL SYS A WAI (WAIT
00038                   * FOR INTERRUPT) WOULD GO HERE. SINCE SIMUL
00039                   * HAS NO INT HARDWARE IT MUST BE SIMULATED
00040                   *BY THE FOLLOWING ROUTINE ENDING IN A NOP
00041                   * AT WHICH A MAID BREAKPOINT WILL BE ENTERED.
00042 82AA CE 82EE          LDX     #NEXT
00043 82AD DF FF            STX     $FE
00044 82AF 96 FF            LDA A   $FF
00045 82B1 36               PSH A
00046 82B2 96 FE            LDA A   $FE
00047 82B4 36               PSH A
00048 82B5 36               PSH A
```

```
PAGE  002  IDL.PRFL

00049 82E6 36              PSH A
00050 82E7 36              PSH A
00051 82E8 36              PSH A
00052 82E9 36              PSH A
00053 82EA 01              NOP                 END WAI SIMULATION
00054                  *ENTER MAID BREAKPOINT
00055 82EB 39          NEXT    RTS
00056 82EC 86 36       INTRPT  LDA A  #$36     D TO A INTRPT PROC
00057 82EE E7 8802             STA A  PIACR    CA1 MASKED CA2 LOW
00058 82C1 E6 8803             LDA A  PIADR
00059 82C4 97 0A               STA A  $0A       TEMPORARY INPUT STO LOC 000A
00060 82C6 3B                  RTI
00061                  *-----------------------------------------

00063                  *--------START ITERATING CTRL ALGORITHM----------
00064 82C7 B7 00FF     START   STA A  MUXC      DUMMY WRITE: SETS MULTIPLEX
00065                  * TO CHANNEL 0 (REAL SYS ADDR $2000) TO READ
00066                  * HEEL SWITCH INFORMATION. MUX NOT SIMULATED.
00067          00FF    MUXC    EQU    $FF
00068 82CA 8E 007F             LDS    #$7F      SET STK PTR FOR SUBRTNS
00069 82CD 8D CD               BSR    CONVRT    HEEL SWITCH IN LOC 000A
00070 82CF 96 0A               LDA A  $0A
00071                  *.........PREPARE HEEL DATA.............
00072 82D1 5F                  CLR B
00073 82D2 48                  ASL A
00074 82D3 59                  ROL B
00075 82D4 D7 0B               STA B  $0B       HEEL INFO: LOC $0B
00076                  *.........................................
00077 82D6 B7 00FF             STA A  MUX1      DUMMY WRITE: SET MUX TO
00078                  * CHAN 1. (REAL SYS ADDR $2400) NOT SIMULATED.
00079          00FF    MUX1    EQU    $FF
00080 82D9 8D C1               BSR    CONVRT    READ THETA FROM CHAN 1.
00081 82DB 96 0A               LDA A  $0A
00082                  *.......PREPARE ANGLE DATA..............
00083 82DD 44                  LSR A
00084 82DE 81 41               CMP A  #$41
00085 82E0 2F 7C               BGT    CONST
00086                  *.........................................
00087 82E2 97 0C               STA A  $0C       STO THETA: LOC 000C
00088 82E4 B7 00FF             STA A  MUX2      DUMMY WRITE: SET MUX TO
00089                  * CHAN 2. (REAL SYS ADDR 2800) NOT SIMULATED.
00090          00FF    MUX2    EQU    $FF
00091 82E7 8D B3               BSR    CONVRT    READ OMEGA FROM CHAN 2.
00092 82E9 96 0A               LDA A  $0A
00093                  *.........PREPARE ANCLAR VEL DATA..........
00094 82EB 8B 80               ADD A  #$80      ELIMINATES BIAS REQUIRED
00095                  *                       TO KEEP INPUT ABOVE 0 VOLTS.
00096 82ED 97 00               STA A  $00
00097 82EF 97 01               STA A  $01       PREPARE FOR SCALING
00098 82F1 BD 9100             JSR    $9100     CALL MULTIPLY
00099 82F4 58                  ASL B
```

```
PAGE  003   IIL.PRFL


00100 82F5 49             ROL A
00101 82F6 54             LSR B
00102 82F7 97 0D          STA A  $0D       STO MSB OF RESULT IN LOC 0D
00103 82F9 D7 0E          STA B  $0E       STO LSBOF RESULT LOC 0F
00104                *.....................................
00105 82FB 96 0B          LDA A  $0B
00106 82FD 9B 0C          ADD A  $0C
00107 82FF 27 65          BEQ    LOCK      BRANCH IF FULL EXTENSION
00108                *                        AND NO HEEL CONTACT
00109 8301 96 0C          LDA A  $0C
00110 8303 27 69          BEQ    UNLOCK    BRANCH IF FULL EXTENSION W/HC
00111 8305 D6 0C          LDA B  $00
00112 8307 2D 07          BLT    EXTENS    BRANCH IF EXTENSION
00113 8309 C6 81    FLEX  LDA B  #$81
00114 830B 48             ASL A            MULTIPLY ANGLE BY 2
00115 830C 8B 8A          ADD A  #$8A
00116 830E C9 00          ADC B  #$00
00117 8310 20 05          BRA    TCALC
00118 8312 C6 82    EXTENS LDA B  #$82
00119 8314 48             ASL A            MULTIPLY ANGLE BY 2
00120 8315 8B 0C          ADD A  #$0C
00121 8317 D7 12    TCALC SIA B  $12
00122 8319 97 13          STA A  $13
00123 831B DE 12          LDX    $12       LOAD X WITH LOOKUP ADDR
00124 831D A6 00          LDA A  0,X
00125 831F 97 00          STA A  $0
00126 8321 96 0E          LDA A  $0E
00127 8323 97 01          STA A  $1        SETUP FOR MULTIPLY
00128 8325 BD 8100        JSR    $8100     FIND D: CALL MULTIPLY
00129 8328 58             ASL B
00130 8329 49             ROL A
00131 832A 48             ASL A
00132 832B 97 0F          STA A  $0F       STO D: MSB IN LOC 0F
00133 832D 96 0D          LDA A  $0D
00134 832F 97 01          STA A  $1        SETUP FOR MULTIPLY
00135 8331 BD 8100        JSR    $8100     FIND F: CALL MULTIPLY
00136 8334 58             ASL B
00137 8335 49             ROL A
00138 8336 97 10          STA A  $10       STO MSB F: LOC 10
00139 8338 D7 11          STA B  $11       STO LSB F: LOC 11
00140 833A DE 12          LDX    $12
00141 833C A6 01          LDA A  1,X
00142 833E 97 00          STA A  $0        SETUP FOR MULTIPLY
00143 8340 BD 8100        JSR    $8100     FIND E: CALL MULTIPLY
00144 8343 58             ASL B
00145 8344 49             ROL A
00146 8345 48             ASL A
00147 8346 DB 1C          LDA B  $1C       PREPARE FOR ADD: LD MSB F
00148 8348 9B 0F          ADD A  $0F       ADD D
00149 834A C9 00          ADC B  #$0       CARRY
00150 834C 9B 11          ADD A  $11       ADD LSB F
00151 834E C9 00          ADC B  #$0       CARRY
00152 8350 54             LSR B
00153 8351 46             ROR A            SHIFT TO RIGHT JUSTIFY
```

```
PAGE  004   IDL.PRFL


00154 8352 54            LSR B
00155 8353 46            ROR A              DIVIDE BY 2
00156                *NOW THE OUTPUT SHOULD BE IN ACC A UNLESS
00157                *THE PROGRAM CALLS FOR MPB SATURATION
00158 8354 C1 00         CMP B  #$0         CHECK FOR SATURATION
00159 8356 26 0E         BNE    LOCK        SATURATION IMPLIES LOCKING
00160 8358 B7 8800       STA A  PIADDR      OUTPUT DATA
00161 835B 7E 82C7       JMP    STRT        GO TO STRT
00162 835E 86 0C    CONST LDA A  #$0C
00163 8360 B7 8800       STA A  PIADDR      OUTPUT 15 IN-LBS
00164 8363 7E 82C7       JMP    STRT        GO TO STRT
00165 8366 86 FF    LOCK  LDA A  #$FF
00166 8368 B7 8800       STA A  PIADDR      OUTPUT 300 IN LBS
00167 836B 7E 82C7       JMP    STRT        GO TO STRT
00168 836E 86 00    UNLOCK LDA A #$00
00169 8370 B7 8800       STA A  PIADDR      OUTPUT 0 IN-LBS
00170 8373 7E 82C7       JMP    STRT        GO TO STRT
00171                *-----------------------------------------------


00173                *-------SET RESTART AND INTERRUPT VECTORS---------
00174 87F8               ORG    $87F8
00175 87F8 82EC          FDB    INTRPT,0,0,$8290
      87FA 0000
      87FC 0000
      87FE 8290
00176                *-----------------------------------------------
00177                    END
PIADDR 8800
PIADCR 8801
CONVRT 829C
PIAACR 88C2
PIAADR 8803
NEXT   82EB
INTRPT 82EC
STRT   82C7
MUX0   00FF
MUX1   00FF
MUX2   00FF
FLEX   8309
EXTENS 8312
TCALC  8317
CONST  835E
LOCK   8366
UNLOCK 836E

TOTAL ERRORS 00000
```

```
PAGE  001   MULTIPLY


00001                        NAM    MULTIPLY
00002                        OPT    M
00003                        OPT    S
00004 8100                   ORG    $8100
00005 8100 C6 00             LDA B  #$00
00006 8102 D7 02             STA B  $02       SET SIGN BYTE FOR + A
00007 8104 96 00             LDA A  $00       READ FACTOR A INTO ACC A
00008                     *------------FIND ABS(A)------------
00009 8106 2A 05             BPL    BP1       BRANCH TO BP1 IF +
00010 8108 C6 80             LDA B  #$80
00011 810A D7 02             STA B  $2        SET SIGN BYTE FOR - A
00012 810C 40                NEG A            PRODUCE + OF A
00013                     *-----------------------------------
00014 810D 97 03          BP1    STA A  $3       STO ABS(A): LOC 0003
00015                     *-----------PARSE ABS(A) INTO MOST SIG-----
00016                     *            HALF BYTE (MSHB) AND LSHB
00017 810F 44                LSR A
00018 8110 44                LSR A
00019 8111 44                LSR A
00020 8112 44                LSR A
00021 8113 97 C4             STA A  $4        STO MSHB(ABS(A)): LSHB LOC C4
00022 8115 96 C3             LDA A  $3
00023 8117 84 0F             AND A  #$0F
00024 8119 97 C3             STA A  $3        STO LSHB(ABS(A)): LSHB LOC 03
00025                     *-----------------------------------------
00026 811B 96 01             LDA A  $1        READ FACTOR B INTO ACC A
00027                     *-----------ABS(B)-----------------------
00028 811D 2A 07             BPL    BP2       BRANCH TO BP2 IF +
00029 811F 40                NEG A            PRODUCE + OF B
00030                     *.........DETERMINE SIGN OF ANSWER.........
00031 8120 D6 02             LDA B  $2
00032 8122 C8 80             EOR B  #$80
00033 8124 D7 02             STA B  $2        STO SIGN OF ANS: LOC 0002
00034                     *.........................................
00035                     *-----------------------------------------
00036 8126 97 C5          BP2    STA A  $5       STO ABS(B): LOC 0005
00037                     *--------PARSE ABS(B) AS ABS (A)--------
00038 8128 84 F0             AND A  #$F0
00039 812A 97 C6             STA A  $6        STO MSHB(ABS(B)): LOC MSHB C6
00040 812C 90 C5             SUB A  $5
00041 812E 40                NEG A
00042 812F 48                ASL A
00043 8130 48                ASL A
00044 8131 48                ASL A
00045 8132 48                ASL A
00046 8133 97 C5             STA A  $5        STO LSHB(ABS(B)): LOC MSHB C5
00047                     *-----------------------------------------
00048                     *---------------LOOKUP C----------------
00049 8135 9A 03             ORA A  $3        FORM LOW ADDR BYTE FOR C LKP
00050 8137 97 C9             STA A  $9
00051 8139 86 80             LDA A  #$80
00052 813B 97 C8             STA A  $8        SET HIGH ADDR BYTE: ALL LKPS
00053 813D DE C8             LDX    $8
00054 813F A6 00             LDA A  0,X       READ C INTO ACC A
```

```
PAGE  002   MULTIPLY

00055 8141 97 07          STA A  $7        STO C* LOC 0007
00056                    *-----------------------------------------------
00057                    *-------------LOOKUP E-------------------------
00058 8143 96 06          LDA A  $6
00059 8145 9A 03          ORA A  $3
00060 8147 97 09          STA A  $9        STO LOW ADDR BYTE FOR E LKP
00061 8149 DE 08          LDX    $8
00062 814B A6 00          LDA A  0,X       READ E
00063 814D 97 03          STA A  $3        STO E: LOC 0003
00064                    *-----------------------------------------------
00065                    *-------------LOOKUP F-------------------------
00066 814F 96 06          LDA A  $6
00067 8151 9A 04          ORA A  $4
00068 8153 97 09          STA A  $9        STO LOW ADDR BYTE FOR F LKP
00069 8155 DE 08          LDX    $8
00070 8157 A6 00          LDA A  0,X
00071 8159 97 06          STA A  $6        STO F: LOC0006
00072                    *-----------------------------------------------
00073                    *-------------LOOKUP D-------------------------
00074 815B 96 04          LDA A  $4
00075 815D 9A 05          ORA A  $5
00076 815F 97 09          STA A  $9        STO LOW ADDR FOR D LOOKP
00077 8161 DE 08          LDX    $8
00078 8163 A6 00          LDA A  0,X       STO D: LOC ACCA
00079                    *-----------------------------------------------
00080                    *----------SETUP D FOR ADD----------------------
00081 8165 5F             CLR B            CLEAR ACC B FOR SHIFTING
00082                    *...........SPLIT SHIFT....................
00083 8166 44             LSR A
00084 8167 56             ROR B
00085 8168 44             LSR A
00086 8169 56             ROR B
00087 816A 44             LSR A
00088 816B 56             ROR B
00089 816C 44             LSR A
00090 816D 56             ROR B
00091                    *-----------------------------------------
00092 816E 97 04          STA A  $4        STO MSHE(D): LOC LSHE 0004
00093 8170 D7 05          STA B  $5        STO LSHE(D): LOC MSHE 0005
00094                    *-------------SETUP E FOR ADD------------------
00095 8172 96 03          LDA A  $3        READ E INTO ACC A
00096 8174 5F             CLR B            CLEAR B FOR SHIFTING
00097                    *.............SPLIT SHIFT....................
00098 8175 44             LSR A
00099 8176 56             ROR B
00100 8177 44             LSR A
00101 8178 56             ROR B
00102 8179 44             LSR A
00103 817A 56             ROR B
00104 817B 44             LSR A
00105 817C 56             ROR B
00106                    *-----------------------------------------
00107                    *---------------------------------------
00108 817D DB 07          ADD B  $7        C+LSHE(E)
```

```
PAGE  003   MULTIPLY


00109 817F 99 06          ADC A   $6       MSHB(E)+F+CARRY
00110 8181 DB 05          ADD B   $5       ADD LSHB(D)
00111 8183 99 04          ADC A   $4       ADD MSHB(L)+CARRY
00112 8185 97 03          STA A   $3       STO MSB(ANS): LOC 0003
00113 8187 D7 04          STA B   $4       STO LSB(ANS): LOC 0004
00114 8189 7E 0103        JMP     $103
00115                     END
EP1    810D
EP2    8126

TOTAL ERRORS 00000
```

```
      /L$$
      DIMENSION Q(16)
      READ(5,10) Q
      WRITE(5,40)
10    FORMAT(16A1)
      DO 30 J=1,16
      DO 30 K=1,16
      L=(J-1)*(K-1)
      M=IFIX(FLOAT(L)/16.)
      N=L-M*16
      M=M+1
      N=N+1
30    WRITE(5,20) Q(J),Q(K),L,Q(M),Q(N)
      STOP
20    FORMAT(6X,2A1,12X,I4,11X,2A1)
40    FORMAT('1',' 4X4 TABLE LOOK UP GENERATOR'//5X,'ADDR',5X,
     1 'DECIMAL PRODUCT',5X,'DATA')
      END
*
```

4X4 TABLE LOOK UP GENERATOR

ADDR	DECIMAL PRODUCT	DATA
00	0	00
01	0	00
02	0	00
03	0	00
04	0	00
05	0	00
06	0	00
07	0	00
08	0	00
09	0	00
0A	0	00
0B	0	00
0C	0	00
0D	0	00
0E	0	00
0F	0	00
10	0	00
11	1	01
12	2	02
13	3	03
14	4	04
15	5	05
16	6	06
17	7	07
18	8	08
19	9	09
1A	10	0A
1B	11	0B
1C	12	0C
1D	13	0D
1E	14	0E
1F	15	0F
20	0	00
21	2	02
22	4	04
23	6	06
24	8	08
25	10	0A
26	12	0C

27	14	0E
28	16	10
29	18	12
2A	20	14
2B	22	16
2C	24	18
2D	26	1A
2E	28	1C
2F	30	1E
30	0	00
31	3	03
32	6	06
33	9	09
34	12	0C
35	15	0F
36	18	12
37	21	15
38	24	18
39	27	1B
3A	30	1E
3B	33	21
3C	36	24
3D	39	27
3E	42	2A
3F	45	2D
40	0	00
41	4	04
42	8	08
43	12	0C
44	16	10
45	20	14
46	24	18
47	28	1C
48	32	20
49	36	24
4A	40	28
4B	44	2C
4C	48	30
4D	52	34
4E	56	38
4F	60	3C
50	0	00
51	5	05
52	10	0A
53	15	0F
54	20	14
55	25	19
56	30	1E
57	35	23
58	40	28
59	45	2D
5A	50	32
5B	55	37
5C	60	3C
5D	65	41
5E	70	46
5F	75	4B
60	0	00
61	6	06
62	12	0C
63	18	12
64	24	18
65	30	1E
66	36	24
67	42	2A
68	48	30

69	54	36
6A	60	3C
6B	66	42
6C	72	48
6D	78	4E
6E	84	54
6F	90	5A
70	0	00
71	7	07
72	14	0E
73	21	15
74	28	1C
75	35	23
76	42	2A
77	49	31
78	56	38
79	63	3F
7A	70	46
7B	77	4D
7C	84	54
7D	91	5B
7E	98	62
7F	105	69
80	0	00
81	8	08
82	16	10
83	24	18
84	32	20
85	40	28
86	48	30
87	56	38
88	64	40
89	72	48
8A	80	50
8B	88	58
8C	96	60
8D	104	68
8E	112	70
8F	120	78
90	0	00
91	9	09
92	18	12
93	27	1B
94	36	24
95	45	2D
96	54	36
97	63	3F
98	72	48
99	81	51
9A	90	5A
9B	99	63
9C	108	6C
9D	117	75
9E	126	7E
9F	135	87
A0	0	00
A1	10	0A
A2	20	14
A3	30	1E
A4	40	28
A5	50	32
A6	60	3C
A7	70	46
A8	80	50
A9	90	5A
AA	100	64

AB	110	6E
AC	120	78
AD	130	82
AE	140	8C
AF	150	96
B0	0	00
B1	11	0B
B2	22	16
B3	33	21
B4	44	2C
B5	55	37
B6	66	42
B7	77	4D
B8	88	58
B9	99	63
BA	110	6E
BB	121	79
BC	132	84
BD	143	8F
BE	154	9A
BF	165	A5
C0	0	00
C1	12	0C
C2	24	18
C3	36	24
C4	48	30
C5	60	3C
C6	72	48
C7	84	54
C8	96	60
C9	108	6C
CA	120	78
CB	132	84
CC	144	90
CD	156	9C
CE	168	A8
CF	180	B4
D0	0	00
D1	13	0D
D2	26	1A
D3	39	27
D4	52	34
D5	65	41
D6	78	4E
D7	91	5B
D8	104	68
D9	117	75
DA	130	82
DB	143	8F
DC	156	9C
DD	169	A9
DE	182	B6
DF	195	C3
E0	0	00
E1	14	0E
E2	28	1C
E3	42	2A
E4	56	38
E5	70	46
E6	84	54
E7	98	62
E8	112	70
E9	126	7E
EA	140	8C
EB	154	9A
EC	168	A8

	ED	182	B6
	EE	196	C4
	EF	210	D2
	F0	0	00
	F1	15	0F
	F2	30	1E
	F3	45	2D
	F4	60	3C
	F5	75	4B
	F6	90	5A
	F7	105	69
	F8	120	78
	F9	135	87
	FA	150	96
	FB	165	A5
	FC	180	B4
	FD	195	C3
	FE	210	D2
	FF	225	E1
STOP	--		

MOTOROLA Semiconductors

MC6800

(0 to 70°C; L or P Suffix)

MC6800C

(-40 to 85°C; L Suffix only)

MICROPROCESSING UNIT (MPU)

The MC6800 is a monolithic 8-bit microprocessor forming the central control function for Motorola's M6800 family. Compatible with TTL, the MC6800, as with all M6800 system parts, requires only one +5.0-volt power supply, and no external TTL devices for bus interface.

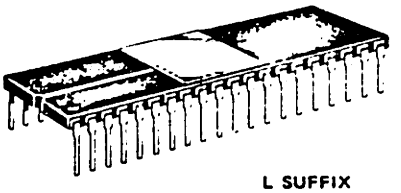
The MC6800 is capable of addressing 65K bytes of memory with its 16-bit address lines. The 8-bit data bus is bidirectional as well as 3-state, making direct memory addressing and multiprocessing applications realizable

- Eight-Bit Parallel Processing
- Bi-Directional Data Bus
- Sixteen-Bit Address Bus — 65K Bytes of Addressing
- 72 Instructions — Variable Length
- Seven Addressing Modes — Direct, Relative, Immediate, Indexed, Extended, Implied and Accumulator
- Variable Length Stack
- Vectored Restart
- Maskable Interrupt Vector
- Separate Non-Maskable Interrupt — Internal Registers Saved In Stack
- Six Internal Registers — Two Accumulators, Index Register, Program Counter, Stack Pointer and Condition Code Register
- Direct Memory Addressing (DMA) and Multiple Processor Capability
- Clock Rates as High as 1 MHz
- Simple Bus Interface Without TTL
- Halt and Single Instruction Execution Capability

MOS

(N-CHANNEL, SILICON-GATE)

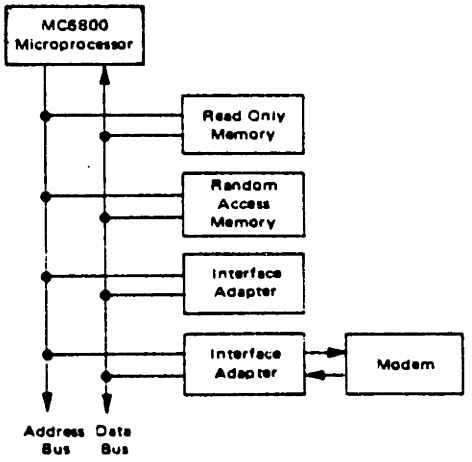
MICROPROCESSOR



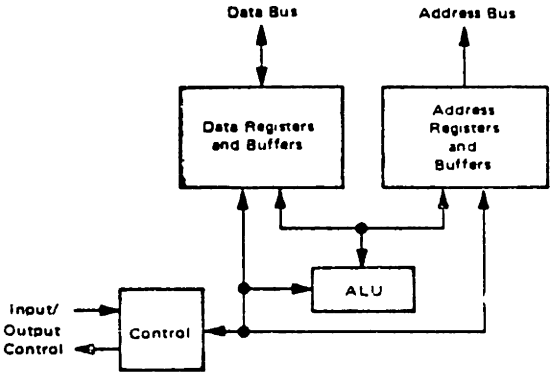
L SUFFIX
CERAMIC PACKAGE
CASE 715

NOT SHOWN: P SUFFIX
PLASTIC PACKAGE
CASE 711

M6800 MICROCOMPUTER FAMILY
BLOCK DIAGRAM



MC6800 MICROPROCESSOR
BLOCK DIAGRAM



Look-up table for $[\tau/\omega^2](\theta)$

Knee Angle units: (degrees)	τ/ω^2 units: (in.-lb.-sec.^{2})	14-bit representation (implicit scale factor: 3.588×10^{-3} in.-lb.-sec.^{2})
flexion	flexion	flexion
0	SPECIAL CASE	SPECIAL CASE
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	.0516	000E
13	.0473	000D
14	.0905	0019
15	.0868	0018
16	.1250	0023
17	.1200	0021
18	.1538	002B
19	.1479	0029
20	.1780	0032
21	.1715	0030
22	.1983	0037
23	.1913	0035
24	.2155	0030
25	.2378	0042
26	.2585	0048
27	.2778	0040
28	.2956	0052
29	.3122	0057
30	.3382	005E
31	.3527	0062
32	.3780	0069
33	.4031	0070
34	.4150	0074
35	.4395	007A
36	.4638	0101
37	.4883	0108
38	.5207	0111
39	.5680	011E
40	.6391	0132

41	.6864	013F
42	.7813	015A
43	.8301	0167
44	.9322	0204
45	1.093	0230
46	1.397	0305
47	1.781	0370
48	2.216	0469
49	2.711	0573
50	3.133	0669
51	3.537	075A
52	3.915	0843
53	3.803	0824
54	3.407	0735
55	3.029	064C
56	2.736	057A
57	2.498	0538
58	2.264	0477
59	1.959	0422
60	1.745	0366
61	1.457	0316
62	1.070	022A
63	.549	0119
64	0	0
65	2.493	0537
66	SPECIAL CASE	SPECIAL CASE

EXTENSION	EXTENSION	EXTENSION
0	SPECIAL CASE	SPECIAL CASE
1	43.56	5E68
2	13.3	1C7F
3	7.688	105E
4	5.498	0B7C
5	4.698	0A1D
6	4.221	0918
7	4.194	0911
8	4.167	0909
9	4.058	0872
10	4.006	085C
11	3.905	0840
12	3.784	081E
13	3.692	0805
14	3.558	075F
15	3.431	073C
16	3.309	071A
17	3.266	070E
18	3.130	0668
19	3.088	065C
20	2.878	0622
21	2.735	057A

22	2.519	053E
23	2.380	0517
24	2.242	0471
25	2.045	043A
26	1.910	0414
27	1.794	0374
28	1.726	0361
29	1.607	0340
30	1.488	031F
31	1.408	0308
32	1.286	0266
33	1.163	0244
34	1.092	0230
35	0.987	0213
36	0.908	017D
37	0.822	0165
38	0.757	0153
39	0.712	0146
40	0.663	0139
41	.630	0130
42	.684	013F
43	.694	0141
44	.730	014B
45	.708	0145
46	.694	0141
47	.651	0135
48	.634	0131
49	.589	0124
50	.567	011E
51	.444	007C
52	.270	004B
53	0	0
54	0	0
55	0	0
56	0	0
57	0	0
58	0	0
59	0	0
60	0	0
61	0	0
62	0	0
63	0	0
64	0	0
65	0	0
66	SPECIAL CASE	SPECIAL CASE

PRINT ON THE FOLLOWING PAGES IS PARTIALLY ILLEGIBLE


```
        :9
PXBUG 1.1 MAIN
*$1
END ADDR 8297 829A
*8290;C
P-8292 X-8031 A-FF B-00 C-C8 S-FF8A
P-8295 X-8031 A-FF B-00 C-C8 S-FF8A
P-8297 X-8031 A-C4 B-CC C-C0 S-FF8A
P-829A X-8031 A-C4 B-C0 C-C0 S-FF8A
P-82C7 X-8031 A-04 B-C0 C-C0 S-FF8A
P-82CA X-8031 A-04 B-00 C-C0 S-FF8A
P-82CL X-8031 A-04 B-00 C-C0 S-007F
P-829C X-8031 A-C4 B-00 C-CC S-007D
P-829F X-8031 A-34 B-C0 C-C0 S-007D
P-82A1 X-8031 A-34 B-00 C-C0 S-007D
P-82A3 X-8031 A-3F B-0C C-C0 S-007D
P-82A6 X-8031 A-3F B-3D C-C0 S-007D
P-82A9 X-8031 A-3F B-3D C-C0 S-007D
P-82AA X-8031 A-3F B-3D C-C0 S-007D
P-82AL X-82EE A-3F B-3D C-C8 S-007D
P-82AF X-82EF A-3F B-3D C-C8 S-007D
P-82B1 X-82EB A-EF B-3D C-C8 S-007D
P-82B2 X-82EF A-EE B-3D C-C8 S-007C
P-82B4 X-82EE A-82 B-3D C-C8 S-007C
P-82B5 X-82EE A-82 B-3D C-C8 S-007E
P-82B6 X-82FE A-82 B-3D C-C8 S-007A
P-82B7 X-82FE A-82 B-3D C-C8 S-0079
P-82B8 X-82EF A-82 B-3D C-C8 S-0078
P-82B9 X-82EE A-82 B-3D C-C8 S-0077
P-82BA X-82EE A-82 B-3D C-C8 S-0076
*8803/3D C1
*$1
END ADDR 82BA 82D9
*82BC;C
P-82BE X-82BF A-36 B-3D C-C0 S-0076
P-82C1 X-82B5 A-36 B-3D C-C0 S-0076
P-82C4 X-82EF A-C1 B-3D C-C0 S-0076
P-82C6 X-82FE A-C1 B-3D C-C0 S-0076
P-82BF X-8282 A-82 B-82 C-C2 S-007D
P-82CF X-8282 A-82 B-82 C-C2 S-007F
P-82D1 X-8282 A-C1 B-82 C-C0 S-007F
P-82D2 X-8282 A-C1 B-00 C-C4 S-007F
P-82D3 X-8282 A-C2 B-0C C-C0 S-007F
P-82D4 X-8282 A-C2 B-C0 C-C4 S-007F
P-82D6 X-8282 A-C2 B-00 C-C4 S-007F
P-82D9 X-8282 A-C2 B-C0 C-C0 S-007F
*82BA;V
*82DF;V
*;F
```

```
P-82EA X-82EF A-82 B-01 C-C8 S-0076
*8803/C1 0E
*82EC;G
P-82DE X-8282 A-82 B-82 C-C2 S-007F
*ST
END ADDR 82D9 82E7
*;P
P-82DD X-8282 A-0E B-82 C-C0 S-007F
P-82DE X-8282 A-07 B-82 C-C0 S-007F
P-82E0 X-8282 A-07 B-82 C-C9 S-007F
P-82E2 X-8282 A-07 B-82 C-C9 S-007F
P-82E4 X-8282 A-07 B-82 C-C1 S-007F
P-82E7 X-8282 A-07 B-82 C-C1 S-007F
*82E9;V
*;P
P-82EA X-82EB A-82 B-0E C-C9 S-0076
*8803/0E 3D
*82EC;G
P-82E9 X-8282 A-82 B-82 C-C2 S-007F
*ST
END ADDR 82E7 82F1
*;P
P-82EE X-8282 A-3D B-82 C-C0 S-007F
P-82ED X-8282 A-ED B-82 C-C8 S-007F
P-82EF X-8282 A-ED B-82 C-C8 S-007F
P-82F1 X-8282 A-ED B-82 C-C8 S-007F
*82F4;V
*;P
P-82F4 X-8034 A-11 B-89 C-C8 S-007F
*ST
END ADDR 82F1 8325
*;P
P-82F5 X-8034 A-11 B-12 C-C3 S-007F
P-82F6 X-8034 A-23 B-12 C-C0 S-007F
P-82F7 X-8034 A-23 B-09 C-C0 S-007F
P-82F9 X-8034 A-23 B-09 C-C0 S-007F
P-82FB X-8034 A-23 B-09 C-C0 S-007F
P-82FD X-8034 A-0C B-09 C-C4 S-007F
P-82FF X-8034 A-07 B-09 C-CC S-007F
P-8301 X-8034 A-07 B-09 C-C0 S-007F
P-8303 X-8034 A-07 B-09 C-C0 S-007F
P-8305 X-8034 A-07 B-09 C-C0 S-007F
P-8307 X-8034 A-07 B-ED C-C8 S-007F
P-8312 X-8034 A-07 B-ED C-C3 S-007F
P-8314 X-8034 A-07 B-82 C-C8 S-007F
P-8315 X-8034 A-CE B-82 C-C0 S-007F
P-8317 X-8034 A-1A B-82 C-EC S-007F
P-8319 X-8034 A-1A B-82 C-EF S-007F
P-831B X-8034 A-1A B-82 C-E0 S-007F
P-831D X-821A A-1A B-82 C-E8 S-007F
P-831F X-821A A-09 B-82 C-E0 S-007F
P-8321 X-821A A-09 B-82 C-EC S-007F
P-8323 X-821A A-09 B-82 C-EC S-007F
P-8325 X-821A A-09 B-82 C-EC S-007F
```

```
*8328;V
*;P
P-8328 X-8090 A-C0 B-51 C-C0 S-C07F
*S1
END ADDR 8325 8331
*;P
P-8329 X-8090 A-C0 B-A2 C-CA S-C07F
P-832A X-8090 A-0C B-A2 C-C4 S-007F
P-832B X-8090 A-C0 B-A2 C-C4 S-007F
P-832D X-8090 A-C0 B-A2 C-C4 S-007F
P-832F X-8090 A-23 B-A2 C-C0 S-007F
P-8331 X-8090 A-23 B-A2 C-C0 S-007F
*8334;V
*;P
P-8334 X-8030 A-C1 B-3E C-CC S-C07F
*S1
END ADDR 8331 8340
*;P
P-8335 X-8030 A-C1 B-76 C-C0 S-C07F
P-8336 X-8030 A-02 B-76 C-C0 S-C07F
P-8338 X-8030 A-C2 B-76 C-C0 S-C07F
P-833A X-8030 A-02 B-76 C-C0 S-C07F
P-833C X-8021A A-C2 B-76 C-C8 S-C07F
P-833E X-8021A A-11 B-76 C-CC S-C07F
P-8340 X-8021A A-11 B-76 C-C0 S-C07F
*8343;V
*;P
P-8343 X-8031 A-02 B-53 C-C0 S-C07F
*S1
END ADDR 8340 82CE
*;P
P-8344 X-8031 A-02 B-A6 C-CA S-C07F
P-8345 X-8031 A-04 B-A6 C-C0 S-C07F
P-8346 X-8031 A-08 B-A6 C-C0 S-C07F
P-8348 X-8031 A-C8 B-02 C-C0 S-C07F
P-834A X-8031 A-C8 B-C2 C-C0 S-C07F
P-834C X-8031 A-C8 B-C8 C-CC S-C07F
P-834E X-8031 A-7F B-C2 C-C0 S-C07F
P-8350 X-8031 A-7F B-C2 C-C0 S-C07F
P-8351 X-8031 A-7E B-C1 C-C0 S-C07F
P-8352 X-8031 A-3F B-01 C-C0 S-C07F
P-8353 X-8031 A-3F B-0C C-C7 S-007F
P-8354 X-8031 A-9F B-C0 C-C9 S-007F
P-8356 X-8031 A-9F B-C0 C-C4 S-C07F
P-8358 X-8031 A-9F B-C0 C-C4 S-C07F
P-835B X-8031 A-9F B-C0 C-C8 S-C07F
P-82C7 X-8031 A-9F B-C0 C-C8 S-C07F
P-82CA X-8031 A-9F B-C0 C-C8 S-C07F
P-82CD X-8031 A-9F B-C0 C-C0 S-C07F
*
```

```
EXEC 1.1 MAIN
*0000/03 A2
0001/00 D1
*T
END ADDR 8189
*8100;G
P-8102 X-80F5 A-11 B-00 C-F4 S-FF8A
P-8104 X-80F5 A-11 B-00 C-F4 S-FF8A
P-8106 X-80F5 A-A2 B-00 C-F8 S-FF8A
P-8108 X-80F5 A-A2 B-00 C-F8 S-FF8A
P-810A X-80F5 A-A2 B-80 C-F8 S-FF8A
P-810C X-80F5 A-A2 B-80 C-F8 S-FF8A
P-810D X-80F5 A-5E B-80 C-F1 S-FF8A
P-810F X-80F5 A-5E B-80 C-F1 S-FF8A
P-8110 X-80F5 A-2F B-80 C-F0 S-FF8A
P-8111 X-80F5 A-17 B-80 C-F3 S-FF8A
P-8112 X-80F5 A-0B B-80 C-F3 S-FF8A
P-8113 X-80F5 A-05 B-80 C-F3 S-FF8A
P-8115 X-80F5 A-05 B-80 C-F1 S-FF8A
P-8117 X-80F5 A-5E B-80 C-F1 S-FF8A
P-8119 X-80F5 A-0E B-80 C-F1 S-FF8A
P-811B X-80F5 A-0E B-80 C-F1 S-FF8A
P-811D X-80F5 A-D1 B-80 C-F9 S-FF8A
P-811F X-80F5 A-D1 B-80 C-F9 S-FF8A
P-8120 X-80F5 A-2F B-80 C-F1 S-FF8A
P-8122 X-80F5 A-2F B-80 C-F9 S-FF8A
P-8124 X-80F5 A-2F B-00 C-F5 S-FF8A
P-8126 X-80F5 A-2F B-00 C-F5 S-FF8A
P-8128 X-80F5 A-2F B-00 C-F1 S-FF8A
P-812A X-80F5 A-20 B-00 C-F1 S-FF8A
P-812C X-80F5 A-20 B-00 C-F1 S-FF8A
P-812E X-80F5 A-F1 B-00 C-F9 S-FF8A
P-8130 X-80F5 A-0F B-00 C-F1 S-FF8A
P-8131 X-80F5 A-1E B-00 C-F0 S-FF8A
P-8132 X-80F5 A-3C B-00 C-F0 S-FF8A
P-8133 X-80F5 A-78 B-00 C-F0 S-FF8A
P-8135 X-80F5 A-F0 B-00 C-FA S-FF8A
P-8137 X-80F5 A-F0 B-00 C-F8 S-FF8A
P-8139 X-80F5 A-FF B-00 C-F8 S-FF8A
```

```
P-813E X-8CF5 A-8C E-00 C-F8 S-FF8A
P-813D X-8CF5 A-80 E-00 C-F8 S-FF8A
P-813F X-8CFE A-80 E-00 C-F8 S-FF8A
P-8141 X-8CFE A-D2 E-CC C-F8 S-FF8A
P-8143 X-8CFE A-D2 E-00 C-F8 S-FF8A
P-8145 X-8CFE A-20 E-CC C-FC S-FF8A
P-8147 X-8CFE A-2E E-00 C-FC S-FF8A
P-8149 X-8CFF A-2E E-0C C-FC S-FF8A
P-814E X-8C2E A-2F E-00 C-F8 S-FF8A
P-814D X-8C2E A-1C E-CC C-FC S-FF8A
P-814F X-8C2E A-1C E-00 C-FC S-FF8A
P-8151 X-802E A-20 E-CC C-FC S-FF8A
P-8153 X-802E A-25 E-00 C-FC S-FF8A
P-8155 X-802E A-25 E-00 C-FC S-FF8A
P-8157 X-8025 A-25 E-CC C-F8 S-FF8A
P-8159 X-8025 A-0A E-CC C-FC S-FF8A
P-815E X-8025 A-0A E-CC C-FC S-FF8A
P-815D X-8025 A-05 E-0C C-FC S-FF8A
P-815F X-8025 A-F5 E-0C C-F8 S-FF8A
P-8161 X-8025 A-F5 E-0C C-F8 S-FF8A
P-8163 X-8CF5 A-F5 E-CC C-F8 S-FF8A
P-8165 X-8CF5 A-4E E-CC C-FC S-FF8A
P-8166 X-8CF5 A-4E E-CC C-F4 S-FF8A
P-8167 X-8CF5 A-25 E-CC C-F3 S-FF8A
P-8168 X-8CF5 A-25 E-8C C-FA S-FF8A
P-8169 X-8CF5 A-12 E-8C C-F3 S-FF8A
P-816A X-8CF5 A-12 E-CC C-FA S-FF8A
P-816E X-8CF5 A-09 E-CC C-FC S-FF8A
P-816C X-8CF5 A-09 E-6C C-F0 S-FF8A
P-816D X-8CF5 A-04 E-6C C-F3 S-FF8A
P-816E X-8CF5 A-04 E-EC C-FA S-FF8A
P-817C X-8CF5 A-04 E-EC C-F0 S-FF8A
P-8172 X-8CF5 A-C4 E-EC C-F8 S-FF8A
P-8174 X-8CF5 A-1C E-FC C-FC S-FF8A
P-8175 X-8CF5 A-1C E-00 C-F4 S-FF8A
P-8176 X-8CF5 A-0E E-CC C-F0 S-FF8A
P-8177 X-8CF5 A-0E E-0C C-F4 S-FF8A
P-8178 X-8CF5 A-07 E-CC C-FC S-FF8A
P-8179 X-8CF5 A-07 E-00 C-F4 S-FF8A
P-817A X-8CF5 A-03 E-CC C-F3 S-FF8A
P-817E X-8CF5 A-03 E-8C C-FA S-FF8A
P-817C X-8CF5 A-01 E-8C C-F3 S-FF8A
P-817D X-8CF5 A-01 E-CC C-FA S-FF8A
P-817F X-8CF5 A-01 E-92 C-D9 S-FF8A
P-8181 X-8CF5 A-CC E-92 C-DC S-FF8A
P-8183 X-8CF5 A-CC E-42 C-D3 S-FF8A
P-8185 X-8CF5 A-11 E-42 C-FC S-FF8A
P-8187 X-8CF5 A-11 E-42 C-FC S-FF8A
P-8189 X-8CF5 A-11 E-42 C-F0 S-FF8A
*CCCC/A2
CC01/D1
CC02/CC
CC03/11
CC04/42
```