

Pittsburg State University

Pittsburg State University Digital Commons

Electronic Thesis Collection

12-1987

A Microprocessor-based multivariable interactive control system

Sayed Mehdi Khayam-Nekouei
Pittsburg State University

Follow this and additional works at: <https://digitalcommons.pittstate.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Khayam-Nekouei, Sayed Mehdi, "A Microprocessor-based multivariable interactive control system" (1987).
Electronic Thesis Collection. 42.
<https://digitalcommons.pittstate.edu/etd/42>

This Thesis is brought to you for free and open access by Pittsburg State University Digital Commons. It has been accepted for inclusion in Electronic Thesis Collection by an authorized administrator of Pittsburg State University Digital Commons. For more information, please contact mmccune@pittstate.edu, jmauk@pittstate.edu.

A MICROPROCESSOR-BASED MULTIVARIABLE INTERACTIVE
CONTROL SYSTEM

A Thesis Submitted to the Graduate School in Partial
Fulfillment of the Requirements for the
Degree of Master of Science

By
Sayed Mehdi Khayam-Nekouei

PITTSBURG STATE UNIVERSITY

Pittsburg, Kansas

December, 1987

ACKNOWLEDGEMENT

The author wishes to express his sincere appreciation to Dr. Bill Studyvin for his encouragement, helpful comments, constructive criticisms, patience, and timely suggestions which made this thesis possible.

The author also wishes to express his gratitude to Dr. John Iley, Dr. W. Larry Williamson, Dr. Porter and Mr. Jim Lookadoo for their constructive criticisms and suggestions.

A special note of gratitude is extended to my lovely and faithful wife for being a source of inspiration throughout this period.

ABSTRACT

This study outlines the various types of control systems and reviews the necessary mathematical techniques to solve the problem of multivariable interactive control. The characteristics as well as the state representation for control processes involving either p or v type canonical structures are discussed.

Next, the characteristics of multivariable interactive discrete control systems are discussed in detail. The advantages of flexibility and speed of microprocessors are used as powerful tools to implement a microprocessor-based control system for a selected model.

The associated hardware and software of a microprocessor-based control system are described. It is also shown how a microprocessor-based system can be employed to control discrete processes.

To demonstrate a practical application of a microprocessor-based system in a multivariable interactive discrete process, the algorithm and software (Assembly Language) is developed for a special engine control system selected as the model.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION.....	1
1.1. Introduction to the Problem.....	1
1.2. Control System Overview.....	2
1.3. Purpose of the Study.....	6
1.4. Limitation of the Study.....	6
2. REVIEW OF LITERATURE	7
3. MULTIVARIABLE INTERACTIVE CONTROL SYSTEM.....	10
3.1. Interaction Analysis Technique.....	10
3.2. Design Approach.....	14
3.3. Description of Transfer Function Representations of Canonical Structure.....	14
3.4. Description of the Matrix Polynomial Representations of Canonical Structure.....	17
3.5. Description of State Representations of Canonical Structures.....	18
3.6. Multivariable Interactive Discrete Control System.....	19
3.6.1. ON-OFF Control.....	20
4. THE MICROPROCESSOR IN CONTROL APPLICATIONS	24
4.1. Associated Hardware.....	24
4.2. Associated Software.....	29
4.3. Software Specifications of MC68HC11.....	34
4.4. The Microprocessor in a Control Loop.....	37
4.4.1. Digital-to-Analog (D/A) Conversion.....	41
4.4.2. Analog-to-Digital (A/D) Conversion.....	49
4.4.3. Input Multiplexing.....	54
4.4.4. Signal Processing Cycle.....	55
4.4.5. Digital Interfacing.....	57
4.5. Microprocessor-Based Implementation of Multivariable Interactive Discrete Control System.....	61
4.6. Microprocessor-Based Implementation of the Special Engine Control System.....	62
4.6.1. Algorithms.....	63
4.6.2. I/O Truth Table.....	64
4.6.3. Development of the Program in Assembly.....	64
5. RESULTS, CONCLUSIONS, AND RECOMMENDATIONS.....	71
5.1. Results	71
5.2. Conclusions.....	73
5.3. Recommendations.....	75
BIBLIOGRAPHY	77

APPENDIX 1 84

LIST OF FIGURES

FIGURE	PAGE
1. Control Problem Overview	5
2. A Two-Output Multivariable Interactive System	11
3. The Bristol Array for a Two-Loop System	12
4. Illustration of the Special Engine's Multivariable Interactive Process	15
5. Canonical Structures of Multivariable Interactive Processes for a Two-Variable Process	16
6. Discrete-State Systems	22
7. Simplified CPU Architecture	25
8. Microcomputer Architecture	27
9. Block Diagram of the MC68HC11 Single-Chip Microcomputer	30
10. Typical Microprocessor-Based Products	31
11. The computer Process	32
12. Flowcharting Symbols	35
13. Flow Chart for a Microprocessor-Based System Used for the Automobile Engine Control System.....	36
14. A Continuous Control Loop Containing a Microprocessor	38
15. Outline of the Operation of a D/A Convertor	42
16. The input-output behavior of an ideal D/A converter	43
17. The Input-Output Behavior of a Non-Ideal D/A Convertor	43
18. Analog Output-One D/A Convertor Per Channel	44
19. Analog Output-One Sample and Hold Per Channel	44
20. Analog Output Using Serial Digital Transmission to a Group of Actuators	45
21. The Digital-to-Analog Converter (D/A)	47
22. A Sample and Hold as well as A/D Converter for a Data Input System	48
23. A Timing Sequence for a Data Input Process	53

24. A Typical Sample-and-Hold Circuit	53
25. The Process Computer as Sampled-Data Controller	56
26. Control Loop with a Computer as a Sampled-Data Controller	56
27. Outline of an Asynchronous Communications Interface Adaptor (ACIA)	59
28. A Microcomputer in a Basic Control Loop	59
29. Control of an A/D Converter by a PIA Chip	60
30. Essential Features of Engine Control	60

List of Tables

Table	Page
1. The Comparison of Classical and Modern Controls	4
2. The Truth Table for the Special Engine Control System Which Used as A Model ..	65

CHAPTER 1

INTRODUCTION

1.1 Introduction to the Problem

During the last decade extensive developments in digital control theory have been made. The rapid drop in the cost of microcomputer hardware identifies it as a potential device available for implementation of control systems. With increase in the number of microprocessor-based devices for process control applications it can be expected for the future that modern digital control techniques will expand to applications with microprocessors. (17, p. 1641)

The advent of Large Scale Integration (LSI) microprocessors promises to remove past limitations and expand the computer applications in control significantly. (20, p. 8) Microprocessor-based control systems are relatively inexpensive to build and are potentially more flexible and more reliable than equivalent systems built with electromechanical or electronic logic. (12, p. 29) The flexibility of microprocessor-based systems drastically expands its applications to a wide range of modern control systems.

Microprocessors can be used for effective, fast and accurate control of any process involving more than one variable. The systems which have more than one input and more than one output, where perturbation of any one input results in a response from more than one output, are called multivariable interactive systems. If perturbation of any input results in a response from more than one output then it is due to some internal coupling or transmission path in the system. Usually one particular system output will respond more than the others when a particular input is manipulated. The response of the other outputs to this input perturbation is called interaction. (18,p. 197-198)

Any control system may be classified as either closed loop or open loop. The term closed loop essentially designates the same concept as the term feedback. The concept of

closed-loop control is to provide feedback of a measured value of the output variable to be compared with the setting of a control command value, to amplify the difference between them, and to alter the output based on this difference. In open-loop control systems, it is not possible to alter the output based on the difference between the control command and the output. The selected problem which is a special engine control system involves discrete control of multiple variables which interact with one another. For the problems mentioned above, microprocessors offer an attractive solution for discrete control of the system.

1.2 Control Systems Overview

Control problems may be classified into several categories according to the type of control approach, signal format, control objective, and closed or open loop. These classifications are shown in Fig. 1.

(a) Types of Controls

Classical control: Control problems can also be classified into classical control and modern control. The description "modern control" is misleading because it implied that the classical controls are "not modern" and "antiqued". In Table I, a detailed list of their differences is presented for clarification. It can be seen that their objectives and mathematical operations are significantly different. In classical control problems, relatively less signal processing operations are used. In modern control problems, sophisticated signal processing operations are often used which require very high computational capabilities.

Modern control: The interest of microprocessor applications in control is in the category of real time digital controls. Although their applications today are mainly in classical types of control problems. It seems eminent that more control applications of microprocessors will take place because its computational capabilities are being increased at a rapid rate.

Multivariable control is considered as a modern control system. Their differences from the classical problems have been summarized in Table 1. The important items to be considered about modern control are:

(1) In modern control, signals usually contain several pieces of information (state variables) and must be represented by a vector of several dimensions (state vectors).

Processing of these vectors requires more complicated matrix operations in time domain.

(2) In modern control, signals are often contaminated by noise. It is desirable that the quality of signal is first improved by some type of signal processing operations before any control operations can be made. Frequently, some information of the signal cannot be directly measured. They have to be estimated by some type of signal processing operation. These signal processing operations are generally known as "optimal (or sub-optimal) estimations." (20,p. 9 and 11)

(b) Signal Format

Continuous (analog) control: In which the independent variable signal is continuous.

Discrete (sampled data and digital) control: In which the independent variable is discrete and the signal is sampled. The special engine control system which is designed and simulated has a discrete nature.

(c) Control Objective

Four types of control operations are shown in Fig. 1. Two are real time and include analysis/simulation and synthesis/design.

The other two can be either real time or non-real time although it is desirable that they can be carried out in real time (or on-line). They are signal processing operation and control operation. The purposes of signal processing operations are typically as follows:

.to improve the quality of signal-such as the use of filters to enhance the signal-to-noise ratio;

TABLE 1 THE COMPARISON OF CLASSICAL AND
MODERN CONTROLS

	CLASSICAL CONTROL	MODERN CONTROL
OBJECTIVE	Good system performance: rise time, settling time, overshoot, stability gain/phase margin, etc.	Optimize system design to meet more complex performance requirements. Minimize or maximize performance index.
SIGNAL	Scaler (Single Variable) Deterministic (Mostly)	Vector/scaler (multivariable, state space) Stochastic/deterministic
SYSTEM	Linear Time invariant	Nonlinear/linear Time variant/invariant
SIGNAL PROCESSING ACTION	Signal well measured and characterized. (Passive, active filters) (Digital, sampled data filter)	Signal is contaminated by noise. Optimal Estimation may be required
CONTROL	Simple servo-loops with fixed flow paths (servo-control)	more complex feedback and feed forward loops with conditional signal flow paths. (optimal control)
REMARKS	Both signal processing and control actions can be described by differential integral equation or $H(S)$ or $H(Z)$.	Both signal processing and control actions are described by algorithms of matrix. They are often inter-active process.

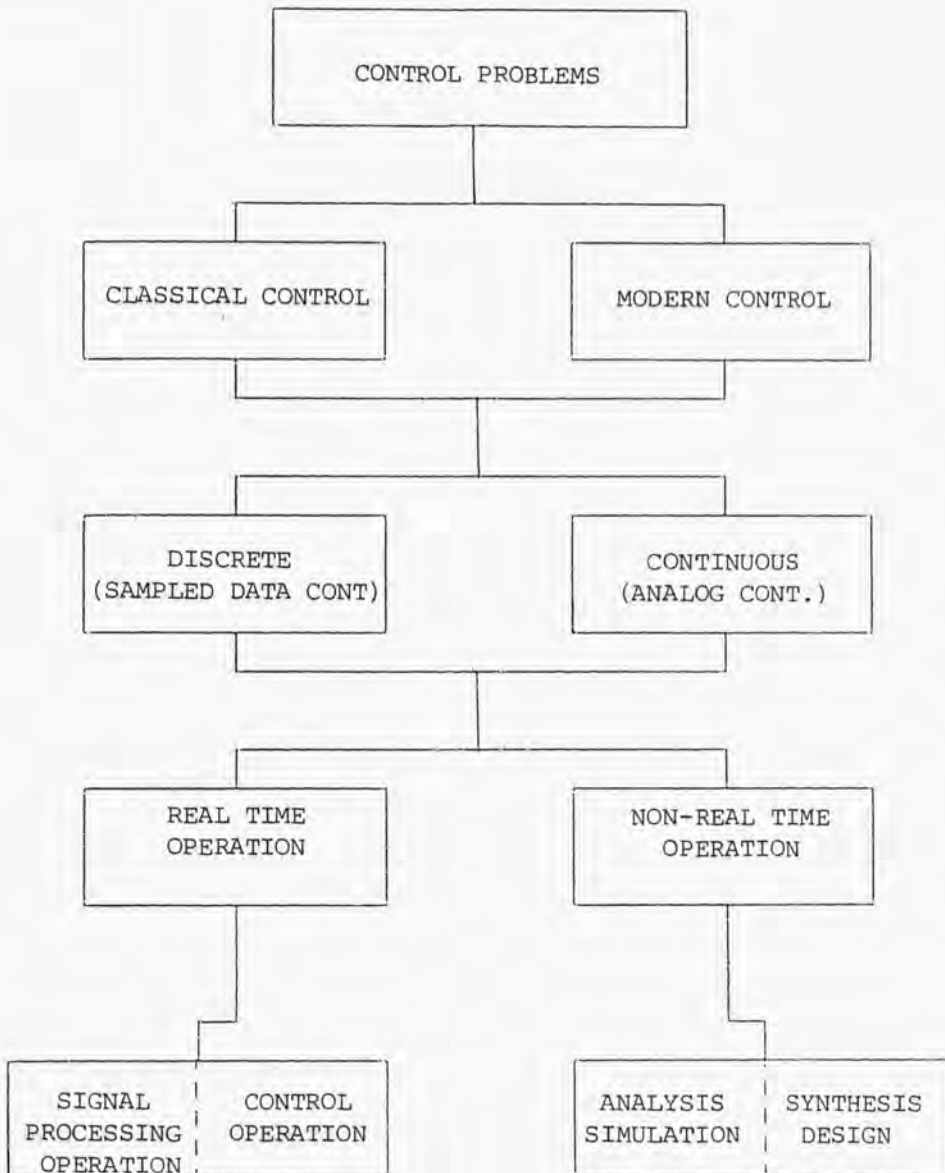


Fig. 1. Control problem overview.

- .to change the measured signal into another form more suitable for its uses-such as the transformation of coordinate systems;
- .to estimate, discriminate or recognize targets;
- .to transmit the signal.

The purpose of control operations are the use of processed signals to accomplish the feedback control objectives.

The best way to solve multivariable interactive discrete control problems with the aid of microprocessors is to define the problem in the form of truth table. Write the Boolean equations, draw the flow chart, before attempting to write the assembly program for the system.

1.3 Purpose of the Study

Objectives for this study are:

- (a) Identify a technique for analysis and design of multivariable interactive control systems which have interaction bonds.
- (b) Apply the technique to an industrial discrete control system model involving multivariable interaction.
- (c) Develop a microprocessor-based system to solve the industrial discrete control system model.

1.4 Limitation of the Study

This study is limited to a discrete system of multivariable interactive control based on an engine control system used as the model.

CHAPTER 2

REVIEW OF LITERATURE

The use of specific devices for automatic control or regulation is an old art. A systematic theory, however, has been developed only within the last few decades. In particular, the linear control theory has advanced rapidly and is now being recognized as a powerful tool for solving a variety of control problems. Attention was first restricted to simple control systems with conventional P, PI, or PID regulators. Increasing complexity of systems requiring control along with availability of digital computers initiated the idea of discrete control. The need or desire to optimize the performance of such control systems has eventually led to the advancement of optimal control theory.

Each single stage of the evolution of linear control theory can be characterized by the mathematical techniques then employed for the analysis and synthesis of control systems. The classical or frequency-domain approach has evolved from the frequency response analysis and the main tool is the theory of complex functions. In particular, the traditional Laplace transform method gave rise to the z-transform theory. The pertinent material is covered in any of the following books: Ragazzini and Franklin (1958), Jury (1958), Tou (1959), Tschauner (1960), and Tsytkin (1963). Systems are described by transfer functions or in any other equivalent way which reflects just the external or input-output properties of the system. This mode of description entails some difficulties concerning stability and realization.

The modern or time-domain approach revolves around the axiomatic concept of state and the main mathematical tools are differential equations and vector spaces. The inception of these methods is usually attributed to increasing complexity of systems requiring control and to the advent of large-scale general-purpose digital computers. The methods are exact in defining the notion of dynamical system and are tailored to describe

the structure and internal properties of the system. They are applicable to multivariable systems and to time-varying situations as well. Though promising success at first, they came in the end to seem somewhat disappointing. This is for the most part due to the necessity of finding state-variable models and to the implicit assumptions that all state variables are accessible for direct measurement. This assumption is justified in mechanical or electrical systems but it is only exceptionally satisfied for plants encountered in chemical, gas, paper and other industries. Then the need for state reconstruction is lost. Last, not least, this approach leads to relatively complicated matrix manipulations, like the change of basis in the state space of the solution of matrix Riccati equations. The existing literature is ample and diverse; the reader may wish to sample Bellman (1957), Zadeh and Desoer (1963), Tou (1964), Sage (1968), Kalman, Falb, and Arbib (1969), Meditch (1969), Anderson and Moore (1971), Ackermann (1972), and Kwakernaak and Sivan (1972).

This status quo was responsible for the comeback of transfer function methods. This trend became evident in early seventies through the works of Rosenbrock (1970), MacFarlane (1972), Wolovich (1974), Wonham (1974), Desoer and Vidyasagar (1975), and others. As a result of synthesis of the two approaches, combining the advantages of both, has been achieved. A new feature is the use of polytypic matrices to cope with multivariable systems. The emphasis is placed on exposing the algebraic nature of various system manipulations. This point of view was pioneered by Kalman, Falb, and Arbib (1969) and in recent years has witnessed a growing cognizance of the intrinsic presence of algebra in system theory.

In keeping with this most recent trend, a more algebraic approach to discrete linear control is offered. Systems are described by input-output data, typically by the transfer matrices; however, these matrices are regarded as algebraic objects rather than complex functions in order to allow for systems defined over arbitrary fields or even rings. Then, the principal idea is to reduce the synthesis procedure to solving linear Polynomial

equations. This mathematical technique provides natural and elegant means for solving a variety of control problems in a unified way and leads directly to simple computational algorithms.

The earliest attempts to employ Polynomial equations in the control system design go back to Jury (1958), Tou (1959), Chang (1961), and notably to Volgin (1962) and Astrom (1970). It took time to understand and fully appreciate their role even in the simplest situations involving single variable systems. (10,p. 17-19)

Discrete control systems are used in a great range of applications in the modern world, from household appliances to the most sophisticated guided missile systems. Computers, and particularly microprocessor-based systems, are being applied in every increasing numbers to all areas of control.

The two-value nature of the variables in discrete (ON/OFF) control systems makes interface to the computer particularly simple, but control operations can still become quite complex. (9,p. 138)

During the past ten years the growth of the microprocessor-based control systems could be appropriately described as volcanic. Today it impacts nearly all aspects of our daily lives to one extent or another. Likewise, the research applications of microprocessors have proliferated. It is used to measure and control laboratory analog signals in instruments ranging from simple single-pan balances to complex particle-beam accelerators.

The microprocessor is an integral part of internal equipment in a vast assortment of applications. Microprocessor-based interactive discrete control system is one such application.

Before a microprocessor can be of any use in a control system, it must be properly programmed and interfaced to the system of interest, many of which are becoming increasingly sophisticated.

CHAPTER 3

MULTIVARIABLE INTERACTIVE CONTROL SYSTEM

3.1 Interaction Analysis Technique

A method for evaluating the degree of interaction between control loops involves use of the relative gain concept or the Bristol array. (6,p. 17, 124) For clarification a two-output-variable interactive system is shown in Fig. 2. The Bristol array for this process is given in Fig. 3. The concept is similar for a system of any size. In each square of the table, the numerator is the ratio of the change in an output variable to the change in an input variable with all other output variables held constant with all other loops under perfect control. Thus, when there is zero interaction between the inputs of one loop and the outputs of another loop, the ratio in all loops on the diagonal, box (1,1) and box (2,2), will be one. The off-diagonal terms, boxes (1,2) and (2,1), will be zero. No matter how many boxes there are in Fig. 3, each row and each column will add to one. Thus, in a system with any number of loops, if the diagonal terms are one, the off-diagonal terms must all be zero. Besides, the relative gain approach of Bristol is based on steady-state gains. To apply Bristol array for interaction analysis of multivariable control system, the following steps should be considered:

1. Form array A of measured variables and manipulated variables such as the one given in Equ. 3.1 and calculate the uncontrolled response for each pair.

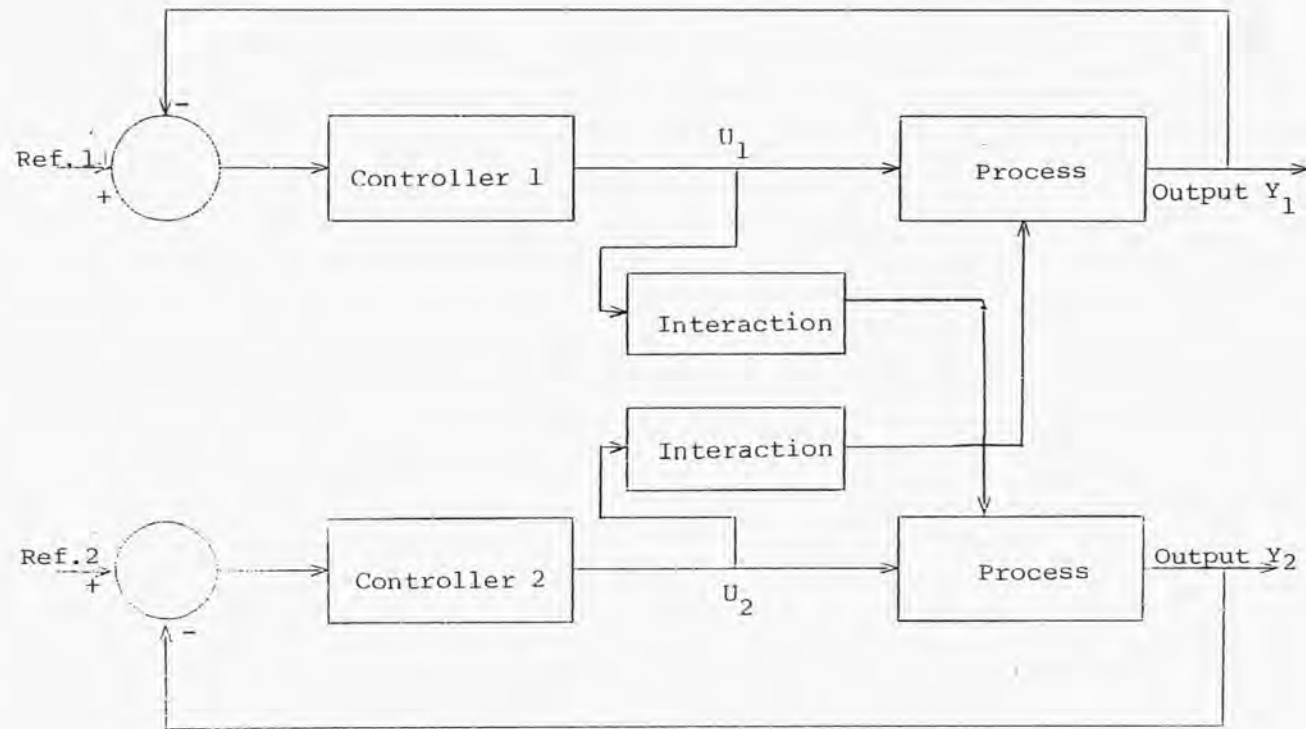


Fig. 2. A two-output multivariable interactive system.

<p>Box (1,1)</p> $\frac{\text{Change in } Y_1}{\text{Change in } U_1 \text{ (} U_2 = \text{constant)}} = \frac{\text{Change in } Y_1}{\text{Change in } U_1 \text{ (} Y_2 = \text{constant)}}$	<p>Box (1,2)</p> $\frac{\text{Change in } Y_1}{\text{Change in } U_2 \text{ (} U_1 = \text{constant)}} = \frac{\text{Change in } Y_1}{\text{Change in } U_2 \text{ (} Y_2 = \text{constant)}}$
<p>Box (2,1)</p> $\frac{\text{Change in } Y_2}{\text{Change in } U_1 \text{ (} U_2 = \text{constant)}} = \frac{\text{Change in } Y_2}{\text{Change in } U_1 \text{ (} Y_1 = \text{constant)}}$	<p>Box (2,2)</p> $\frac{\text{Change in } Y_2}{\text{Change in } U_2 \text{ (} U_1 = \text{constant)}} = \frac{\text{Change in } Y_2}{\text{Change in } U_2 \text{ (} Y_1 = \text{constant)}}$

Fig. 3. The Bristol array for a two-loop system.

		Manipulated Variables			
		m1	m2.....mn		
C-1	a11	a12.....	a1n		
C2	a21	a22.....	a2n		
..	Controlled Variables	(3-1)
..		
Cn	an1	an2.....	ann		

2. Calculate the B matrix.

$$B = (A^{-1})^T \tag{3-2}$$

3. Form the Bristol array λ by multiplying corresponding terms of A and B. It should be remembered that this is not conventional matrix multiplication.

4. Select manipulated-variable-controlled-value pairs by selecting those with positive relative gain closest to 1.0.

5. Consider the properties of the relative gain array (RGA):

- a. Rows and columns of λ sum to 1.0.
- b. If $a_{ij} = 0$, then $\lambda_{ij} = 0$,
- c. Pairing on a negative relative gain array (RGA) element results in either an

unstable system or an inverse responding system,

Where A is array of measure variables.

B is inverse of controlled responses.

λ is relative gain array.

However, using discrete approach the multivariable interactive discrete processes can be controlled efficiently with the aid of microprocessors which are discussed in the next chapter.

3.2 Design Approach

In the design of multivariable control systems, an adequate process model is crucial. The most useful applications of the theory have been in the selection of variables in applications involving interaction. (6, p. 17, 124)

As shown in Fig. 4 the inputs and outputs of multivariable processes influence each other, resulting in mutual interactions of the direct signal paths R-F, T-A, and L-S. The internal structure of multivariable processes has a significant effect on the design of multivariable control systems. This structure can be obtained by theoretical modeling if there is sufficient knowledge of the process. The structures of technical processes are very different such that their input-output relations cannot be described in terms of only a few standardized structures. However, the real structure can often be transformed into a Canonical Model Structure using similarity transformations or simply block diagram conversion rules. The following sections consider special structures of multivariable processes based on the transfer function representation, matrix polynomial representation and state representation. These structures are the basis for the designs of multivariable control system. (8,p. 316)

The relative gain array (RGA) method of Bristol is also very useful for multivariable processes involving interactions. (6,p. 17, 124)

3.3 Description of Transfer Function Representation of Canonical Structure

The most important Canonical Structures used to describe the multivariable process input/output behavior are shown in Fig 5. (8,p. 317)

In case of P-Canonical Structure each input acts on each output. And the summation points are at the outputs. Changes in one transfer element influence only the corresponding output, and the number of inputs and outputs can be different. The characteristics of the V-Canonical Structure is that each input acts directly only on one corresponding output and each output acts on the other inputs; this structure is defined only

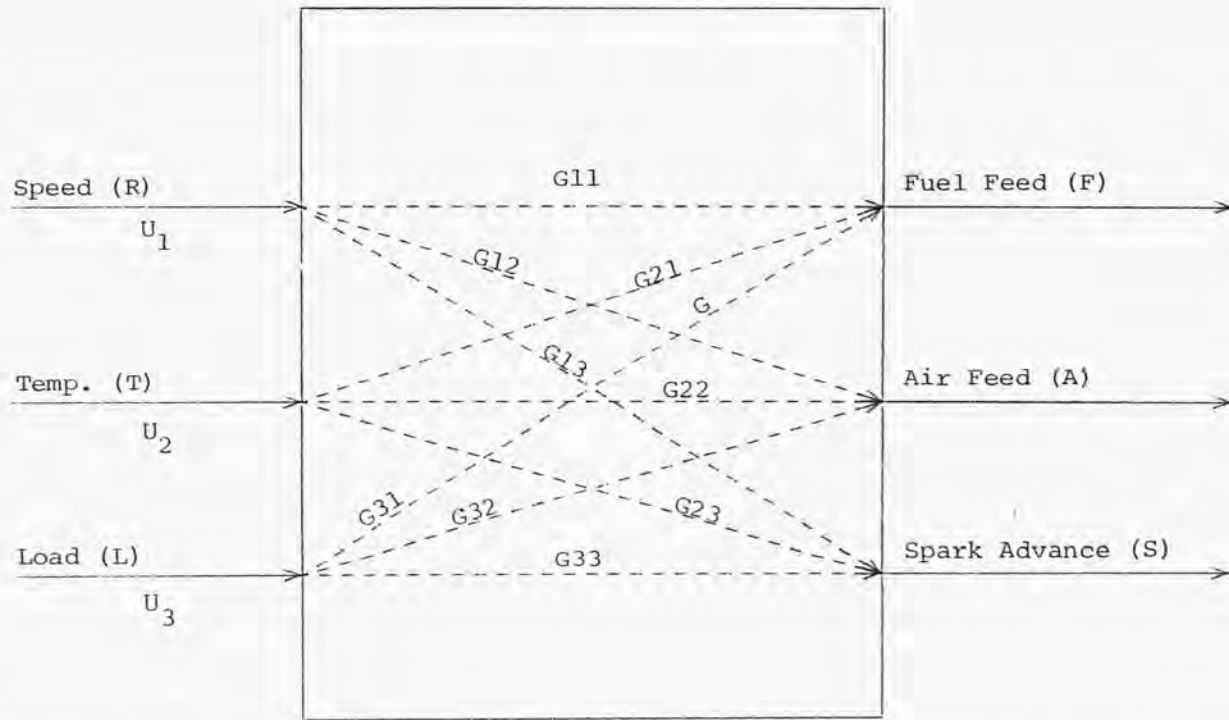
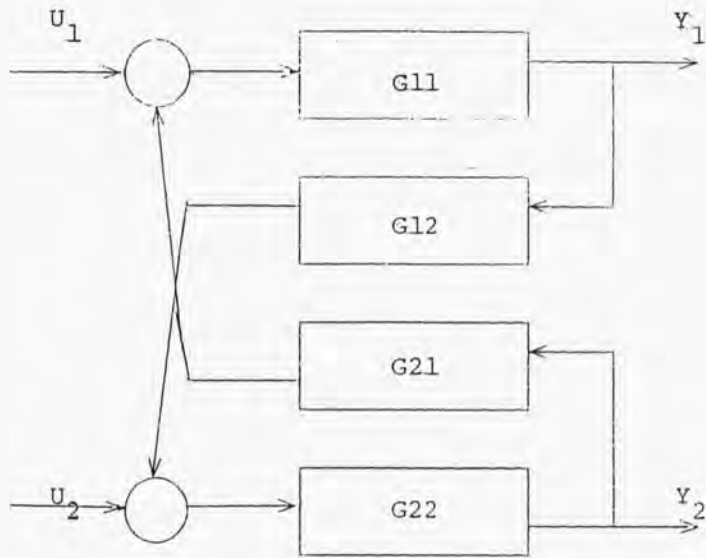
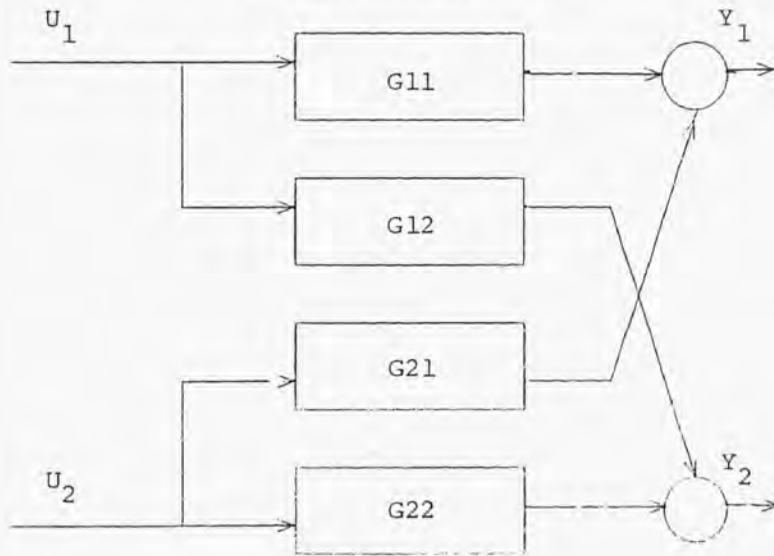


Fig. 4. Illustration of the special engine's multivariable interactive process.



V-Canonical Structure



P-Canonical Structure

Fig. 5. Canonical Structures of Multivariable Interactive Processes for a Two variable Process.

for the same number of inputs and outputs. Change in one transfer element influences the signal of all other elements.

Both Canonical forms can be converted to each other, but realizability must be considered. If the behavior of multivariable processes has to be identified on the basis of non-parametric models, as for example using non-parametric frequency responses or impulse responses, then one obtains only the transfer behavior in a P-Canonical Structure. If other internal structures are considered, proper parametric models and parameter estimation methods must be used. (8,p. 318)

The overall structure describes only the signal flow paths. The actual behavior of multivariable processes is determined by the transfer functions of the main and coupling elements including both their signs and mutual position. One distinguishes between symmetrical multivariable processes, where

$$\begin{aligned} G_{ii}(z) &= G_{jj}(z) & i=1,2,\dots \\ G_{ij}(z) &= G_{ji}(z) & j=1,2,\dots \end{aligned} \quad (3-3)$$

and non-symmetrical multivariable processes, where

$$\begin{aligned} G_{ii}(z) &\neq G_{jj}(z) \\ G_{ij}(z) &\neq G_{ji}(z) \end{aligned} \quad (3-4)$$

With regard to the settling times of the decouples main control loops, slow process elements G_{ii} can be coupled with fast process elements G_{ij} . With lumped parameter processes signals can only appear at the input or output of energy, mass or momentum storages. The main and coupling elements often contain the same storage components, so that a main transfer element and coupling transfer element possess some common transfer function terms. Hence $G_{ii} \ G_{ij}$, or $G_{ii} \ G_{ji}$ can often be observed.

3.4 Description of the Matrix Polynomial Representations of Canonical Structure

An alternative to the transfer function representation of linear multivariable system is the matrix polynomial representation. (11, p. 239)

$$\underline{A}(z^{-1}) \underline{Y}(z) = \underline{B}(z)^{-1} \underline{U}(z)$$

with

(3-5)

$$\underline{A}(z^{-1}) = \underline{A}0 + \underline{A}1z^{-1} + \dots + \underline{A}mz^{-m}$$

$$\underline{B}(z)^{-1} = \underline{B}1z^{-1} + \dots + \underline{B}mz^{-m}$$

If $\underline{A}(z)$ is a diagonal polynomial matrix, the matrix polynomial representation for a process with two inputs and two outputs will be

$$\begin{bmatrix} \underline{A}11(z^{-1}) & 0 \\ 0 & \underline{A}22(z^{-1}) \end{bmatrix} \begin{bmatrix} \underline{Y}1(z) \\ \underline{Y}2(z) \end{bmatrix} = \begin{bmatrix} \underline{B}11(z^{-1}) & \underline{B}21(z^{-1}) \\ \underline{B}12(z^{-1}) & \underline{B}22(z^{-1}) \end{bmatrix} \begin{bmatrix} \underline{U}1(z) \\ \underline{U}2(z) \end{bmatrix} \quad (3-6)$$

This corresponds to a P-Canonical Structure with a common denominator polynomials of $G11(z)$ and $G21(z)$ or $G22(z)$ and $G12(z)$. More general structures arise if off-diagonal polynomials are introduced into $\underline{A}(z^{-1})$.

3.5 Description State Representation of Canonical Structures

For a linear multivariable processes with p inputs $\underline{U}(k)$ and r outputs $\underline{Y}(K)$, the following equations apply:

$$\underline{X}(k+1) = \underline{A} \underline{x}(k) + \underline{B} \underline{u}(k)$$

$$\underline{Y}(k) = \underline{C} \underline{x}(k) + \underline{D} \underline{u}(k) \quad (3-7)$$

where

p is number of inputs

r is number of outputs

$\underline{x}(k)$ is $(mx1)$ state vector

$\underline{u}(k)$ is $(px1)$ control vector

$\underline{y}(k)$ is $(rx1)$ output vector

\underline{A} is (mxm) systems matrix

\underline{B} is $(m \times p)$ control matrix

\underline{C} is (rxm) output (measurement) matrix

\underline{D} is (rxp) input-output matrix.

The state representation of multivariable systems has several advantages over the transfer matrix notation. For example, arbitrary internal structures with a minimal number of parameters and non-controllable or non-observable process parts can also be described. Furthermore, on switching from single-input/single-output processes to multivariable processes only parameter matrices \underline{B} , \underline{C} and \underline{D} have to be written instead of parameter vectors \underline{b} and \underline{c}^T and the parameter d . Therefore, the analysis and design of single-input/single-output control systems can easily be extended to multi-input/multi-output control systems. However, a larger number of canonical structures exists for multivariable processes in state form. These techniques which are discussed would be applicable if the system is intended to operate in continuous mode. However, since the special engine control problem has a discrete nature it is essential to be specific and discuss the discrete control of multivariable process in the following section.

3.6 Multivariable Interactive Discrete Control System

A discrete-state system is one for which at every instant of time the state of the system is defined by the values of a set of variables, each of which can only be defined to be in one of two conditions or states. The variables themselves may be continuous in value, but insofar as the control system is concerned, their values are only required to be known relative to two states. Two distinct types of control strategies are associated with discrete-state systems. One type is used to control the value of one or more variables in the system. The second type of control is sequential in nature and refers to the progress of the system through a defined set of discrete states, in time, to accomplish some overall objective. (9,p. 138)

The special engine control system model is intended to be implemented based on two-state (ON/OFF) control systems. The control algorithm of such a system is based on a

determination of the state of the input and using this to determine the proper output state. The output respond levels are based on the state of the inputs. The ON/OFF condition of inputs is determined based on voltage level (between logic 1 and 0) referred to as V_{sp} .

For ON State $V_{in} > V_{sp}$

For OFF State $V_{in} < V_{sp}$

(3-8)

Where V_{in} = input (measurement)

V_{sp} = Limit Value Between Hi and Low State (set point)

ON/OFF = Two possible output states.

Even though the input variables may actually be continuous but the discrete nature of the system is based on two facts: (1) the value of input variables relative to a limit, and (2) that the output can only have two states. Of course, in special engine control systems the inputs are inherently discrete and have only two states, such as being either on or off. (9,p. 139)

3.6.1 ON-OFF Control

In the on-off control mode, the final correcting device has only two positions or operating states. For this reason, on-off control is also known as two-position or bang-bang control. If the error signal is positive, the control system sends the final correcting device to one of its two positions. If the error signal is negative, the controller sends the final correcting device to the other position. On-off control can be conveniently visualized by considering the final correcting device to be a solenoid-actuated valve. When a valve is actuated by a solenoid, it is either fully open or completely closed; there is no middle ground. (14,p. 288-290)

The position control supplies energy in pulses to the process. This causes a cycling of the controlled variables. The amplitude of the cycling depends on three factors: the capacitance of the process, the dead-time lag of the process, and the size of the load

changes the process is capable of handling. The amplitude of the oscillation is decreased by either increasing the capacitance, decreasing the dead-time lag, or decreasing the size of the load change that can be accommodated. For these reasons, two-position control is only used on processes that have a capacitance large enough to counteract the combined effect of the dead-time lag and the load-change capability of the process. (2,p. 245)

Being limited to two positions, the two-position control either supplies too much or too little correction to the system. Thus, the controlled variables must continuously move between the two limits required to cause the controlling elements to move from one fixed position to the other. The range through which the controlled variables must move is called differential gap. The "oscillation" of the controlled variables between two limits is one important characteristic of two-position control and one which sometimes limits its usefulness. However, two-position control is relatively simple and inexpensive and, for this reason, is widely used. (7,p. 172-173)

The application of ON/OFF control to systems with a continuous controlled variable, have two important practical conditions and consequences:

1. Such systems will usually always require a deadband or hysteresis about the set point to prevent rapid fluctuations of the output when the input is near the set point.
2. Such systems will usually always exhibit an oscillation of the controlled variables within the deadband. The period of this oscillation increases with decreasing deadband width.

The use of microprocessor for ON/OFF control would clearly be impractical for a single variable. For example, it would be difficult to justify using a microprocessor-based control system to turn on and off the compressor of a refrigerator. However, when there are many such independent, single variables to control in a system, it may be practical to use a microprocessor-based control system. In such a case the control system would have a hardware configuration like that shown in Fig. 6. The comparators are used to convert

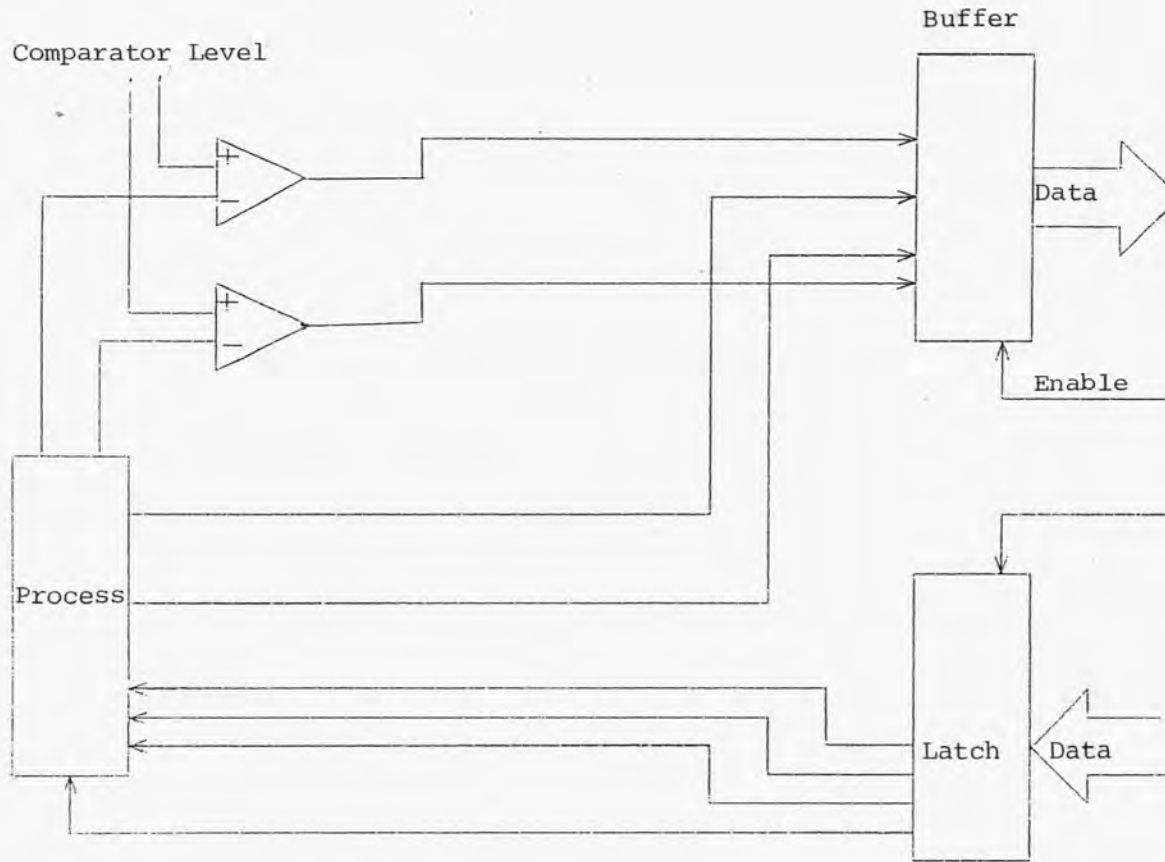


Fig. 6. A Discrete State-System.

continuous variables into a two-state input, while inherently discrete variables are input directly after conversion to proper digital signals. A tri-state buffer is used for interface. The output uses a latch to provide updated ON/OFF state information to the output.

The software consists of a series of decision blocks that evaluate the input states and update the output state. Any required hysteresis can be provided by hysteresis comparators or by timing loops in the software that prevent output state changes from occurring too rapidly. (9,p. 140-141)

CHAPTER 4

THE MICROPROCESSOR IN CONTROL APPLICATIONS

4.1 Associated Hardware

The digital processor or central processing unit (CPU), first as the minicomputer and now in the form called microprocessor, has become an excellent all-purpose electronic control unit (3,p.33)

Microprocessors are integrated circuits that have the ability to perform many functions and consist of numerous registers, counters and decoders. Buses serve to transfer information internally between the registers and to external devices. A system clock and timing circuitry causes functions to be sequenced properly to move data from one area to another at the right time. However, no information can flow within the microprocessor until it is instructed to do so. The hardware approach begins with a diagram showing the interconnections of a component in the system. (21,p. 3-1, 4-1)

The primary functions of the CPU of a microcomputer are to:

1. Fetch, decode, and execute program instructions in the proper order.
2. Transfer data to and from memory and to and from input/output sections.
3. Respond to external interrupts.
4. Provide overall timing and control signals for the entire system.

Most microprocessor CPU's contain at least the elements diagrammed in Fig. 7.

The main sections include the various registers, the arithmetic and logic unit, the instruction decoder, the all-important control and timing section, along with inputs and outputs. Most CPU's actually contain several special registers as well as many specialized input and output not detailed in Fig. 7.

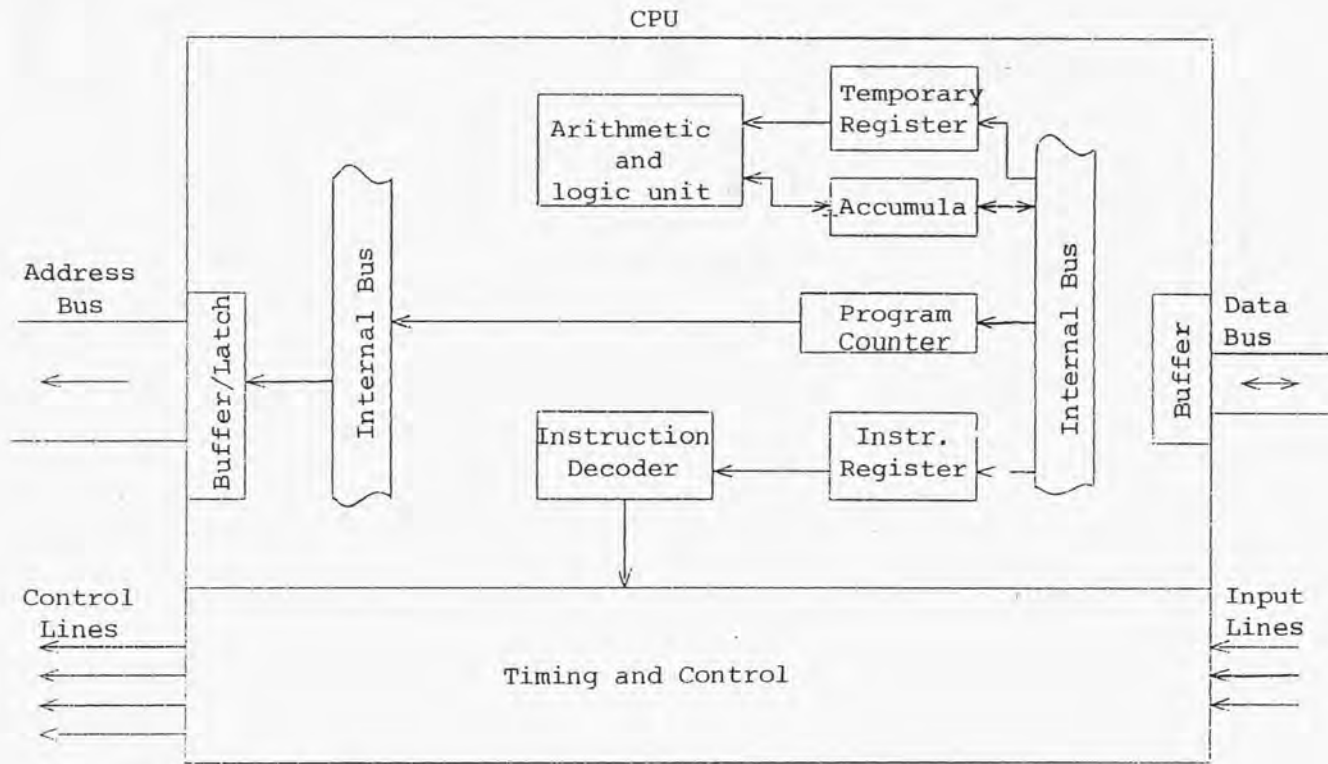


Fig. 7. Simplified CPU architecture.

The CPU's arithmetic and logic unit performs operation such as add, shift/rotate, compare, increment, decrement, negate, AND, OR, XOR, complement, clear and preset. (21,p. 79)

The microcomputer architecture shown in Fig. 8 shows two types of semiconductor memory used in this system. The ROM is the permanent memory which probably contains the monitor program for the system. The ROM has address input along with chip-select and read-enable input lines. The ROM also has 8 three-state outputs connected to data bus. Each memory word is then 8 bits wide. Of course, the ROM would also have power supply connections, although they are many times omitted from the block diagrams.

The architecture in Fig. 8 also shows a RAM as temporary read/write storage device. The RAM has address inputs along with chip-select and read/write enable inputs. The RAM has 8 three-state outputs connected to the data bus. This RAM inputs, outputs stores data as 8-bit words. RAM power supply connections are also shown.

The microcomputer system diagrammed in Fig. 8 uses a keyboard as the input device. Power connections to the keyboard are shown along with the data lines to a special IC called a keyboard interface. The interface circuit stores data and coordinates the keyboard inputs.

At the proper time, the keyboard interface interrupts the microprocessor via the special interrupt line. This interrupt signal causes the microprocessor to (1) finish executing the current instruction, (2) suspend normal operation, and (3) jump to a special group of instructions in its monitor program that handles the data input from the keyboard. The keyboard interface circuit has address, chip-selects and control inputs for activating the unit. When activated, the keyboard interface unit will put keyboard data on the data bus. The microprocessor accepts the new input data via the data bus. When the interface three-state outputs are not activated, they return to their high-impedance state.

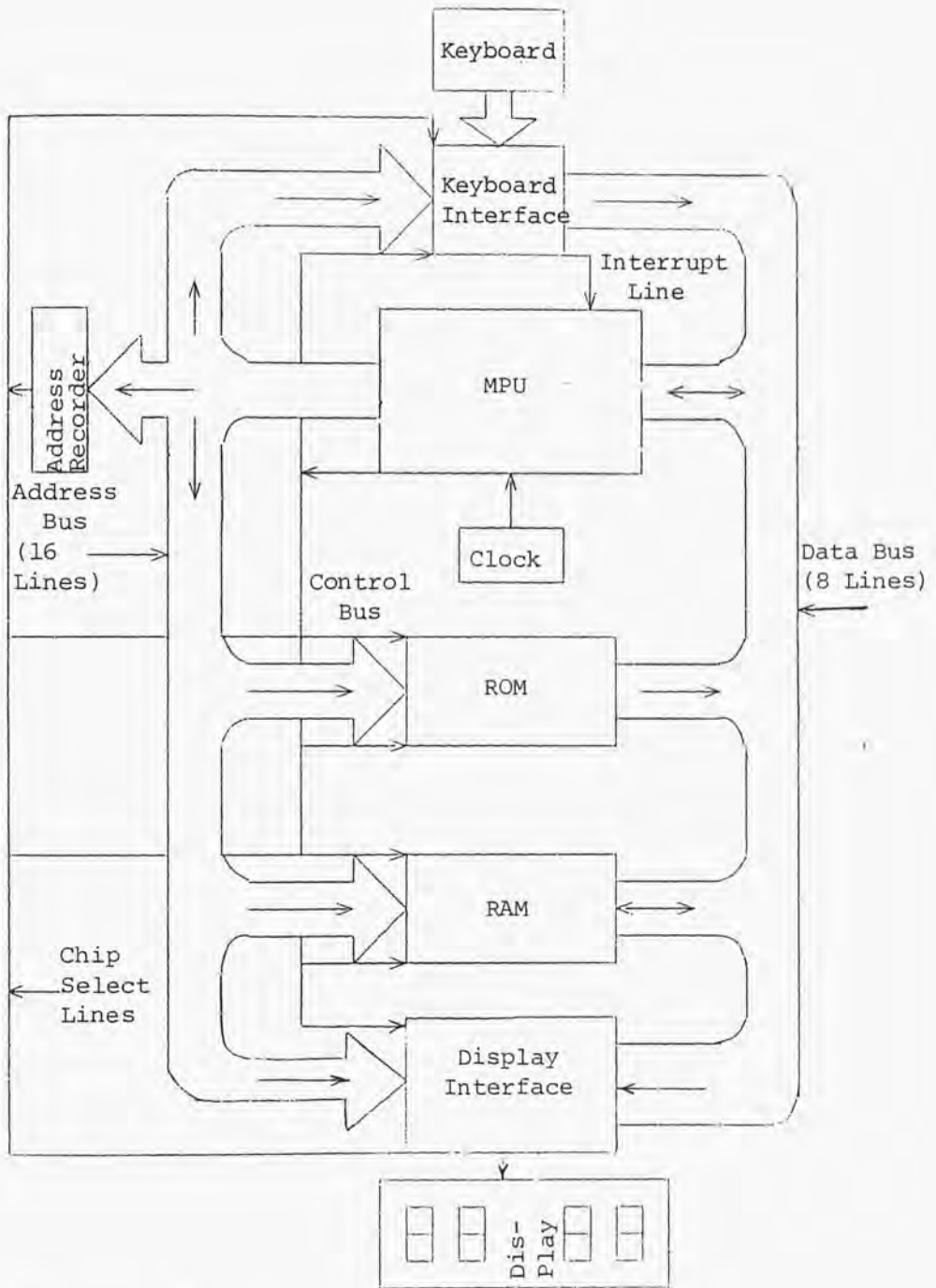


Fig. 8. Microcomputer architecture.

The microcomputer in Fig. 8 uses a group of seven-segmented displays for output. The display is connected to the power supply on the right. A special interface circuit or IC is used to store data and drive the displays in Fig. 8. When activated by the address, chip-select, and enable inputs, the interface accepts data off the data bus and stores it. The interface then drives the displays continuously showing the data stored in the display interface in visual form.

The 16 lines of the address bus can contain 65,536 (2^{16}) different patterns of zeros and ones. The address bus lines may be attached to several devices such as RAMs, ROMs, and interfaces. To turn on or enable only the correct device, an address decoder samples the data on the address bus. The combinational logic of the address decoder activates the proper chip-select line, thus enabling the correct device. To simplify circuitry, not all 16 address lines go to the address decoder, memories, or interfaces. (22,p. 66-68)

The microcomputer selected for the purpose of experiment is MC68HC11A8 manufactured by Motorola. It is an advanced single-chip microcomputer (MCU) with highly sophisticated on-chip peripheral functions. New design techniques are used to achieve a nominal bus speed of two megahertz. In addition, the fully static design allows operation at frequencies down to dc, further reducing its low power consumption. Some of the hardware features of this microcomputer are:

- . 8K Bytes of ROM
- . 512 Bytes of EEPROM
- . 256 Bytes of RAM
- . Enhanced 16-bit timer system
- . An 8-bit pulse accumulator circuit
- . A enhanced NRZ serial communication interface (SCI)
- . A serial peripheral interface
- . Eight channel, 8-bit analog-to Digital Converter

- . Real Time Interrupt Circuit
- . Computer Operating Properly (COP) watchdog system

A block diagram of the MC68HC11A8 is shown in Fig. 9. (1,p. 1-1, 1-2)

4.2 Associated Software

Microprocessor-based systems contain all the essential ingredients found in any computer-based system, but the relative emphasis on each of these ingredients is often considerably different. (19, p. 18) Fig. 10 illustrates some typical microprocessor-based systems. Some of these systems such as the camera exposure control and the automobile fuel injection system represent extensive special purpose microprocessor design effort due to special packaging or power-consumption problems, but the rest could be handled by standard off-the-shelf microcomputers. In the latter case, the design problem thus principally boils down to the selection of a microcomputer system, design of a system interface, and design of programs for the system control. The first two steps just mentioned are described as the hardware design, while the latter step is that of software design. (4,p. 18)

Software refers to programs and the programming system used to control the operation of the computer system. (21,p. 4-1)

The physical units of a microcomputer shown in the boxes in Fig. 9 are referred to as hardware. To be useful, the program memory must tell the CPU what to do. Preparing the list of instructions is called programming. The list of instructions is a program and is stored either temporarily or permanently in the program memory. These programs manipulate information, called data. Software is a general term to cover all programs. (22, p. 1)

The program list is initially generated by the human programmer and entered into the computer memory. The basic process is shown in Fig. 11.

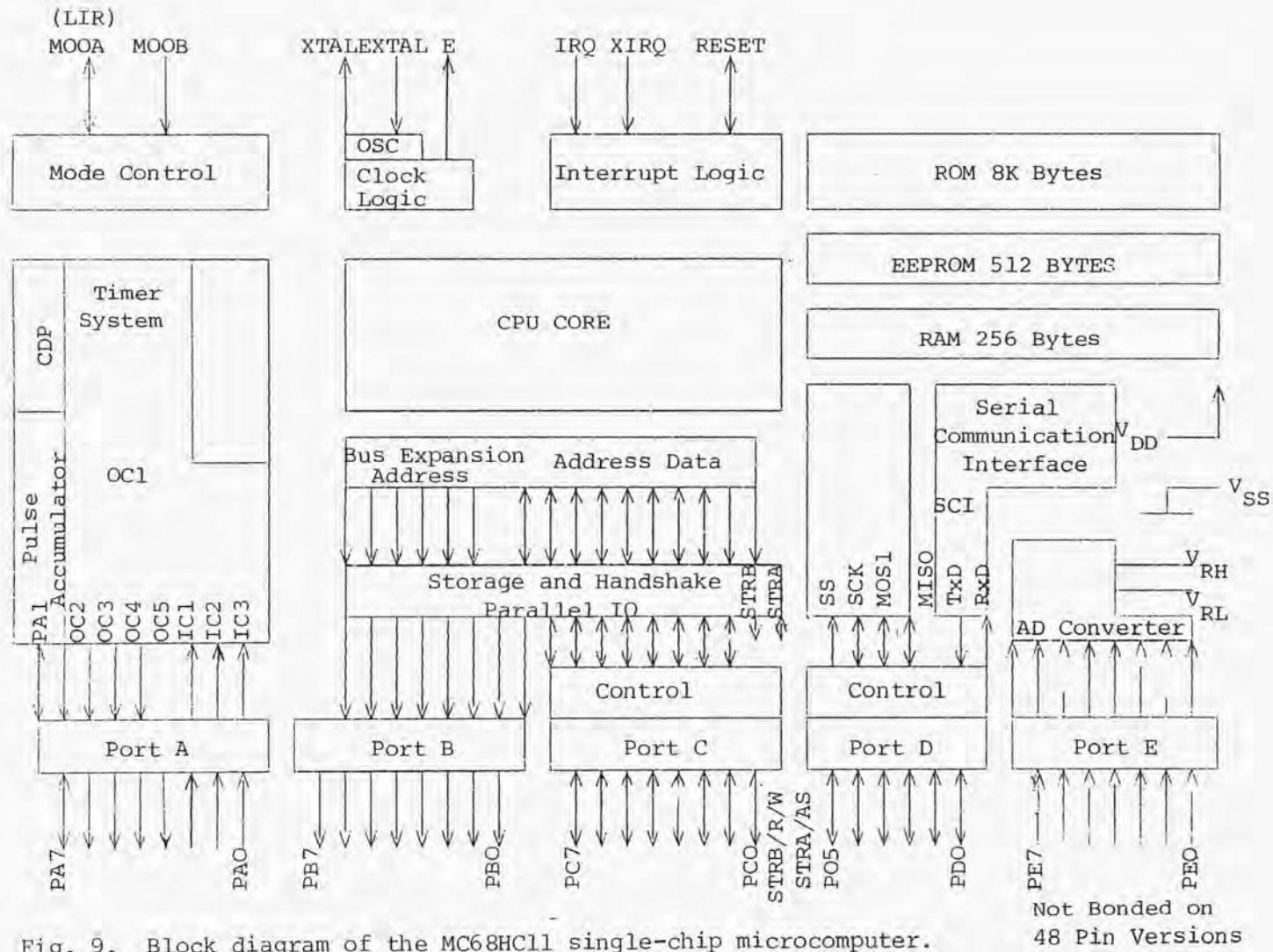
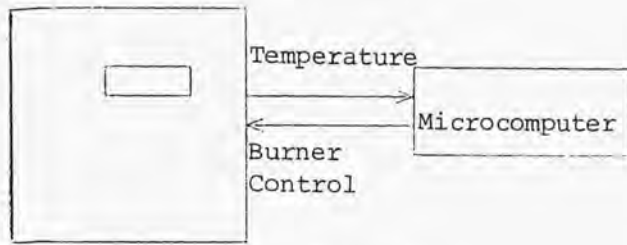
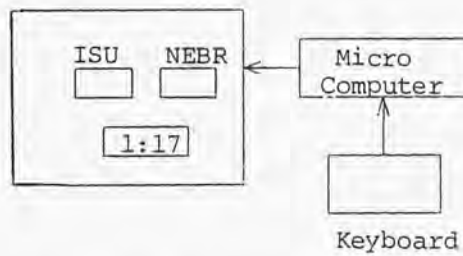


Fig. 9. Block diagram of the MC68HC11 single-chip microcomputer.

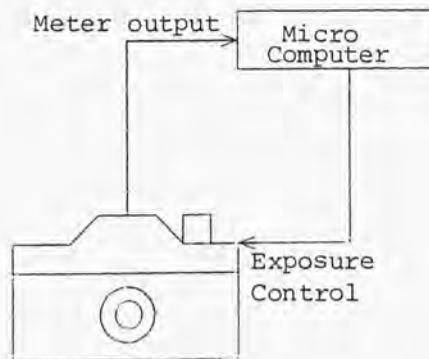
Not Bonded on
48 Pin Versions



(a) Electric oven control



(c) Athletic scoreboard



(e) Automatic camera

Fig. 10. Typical microcomputer-based products.

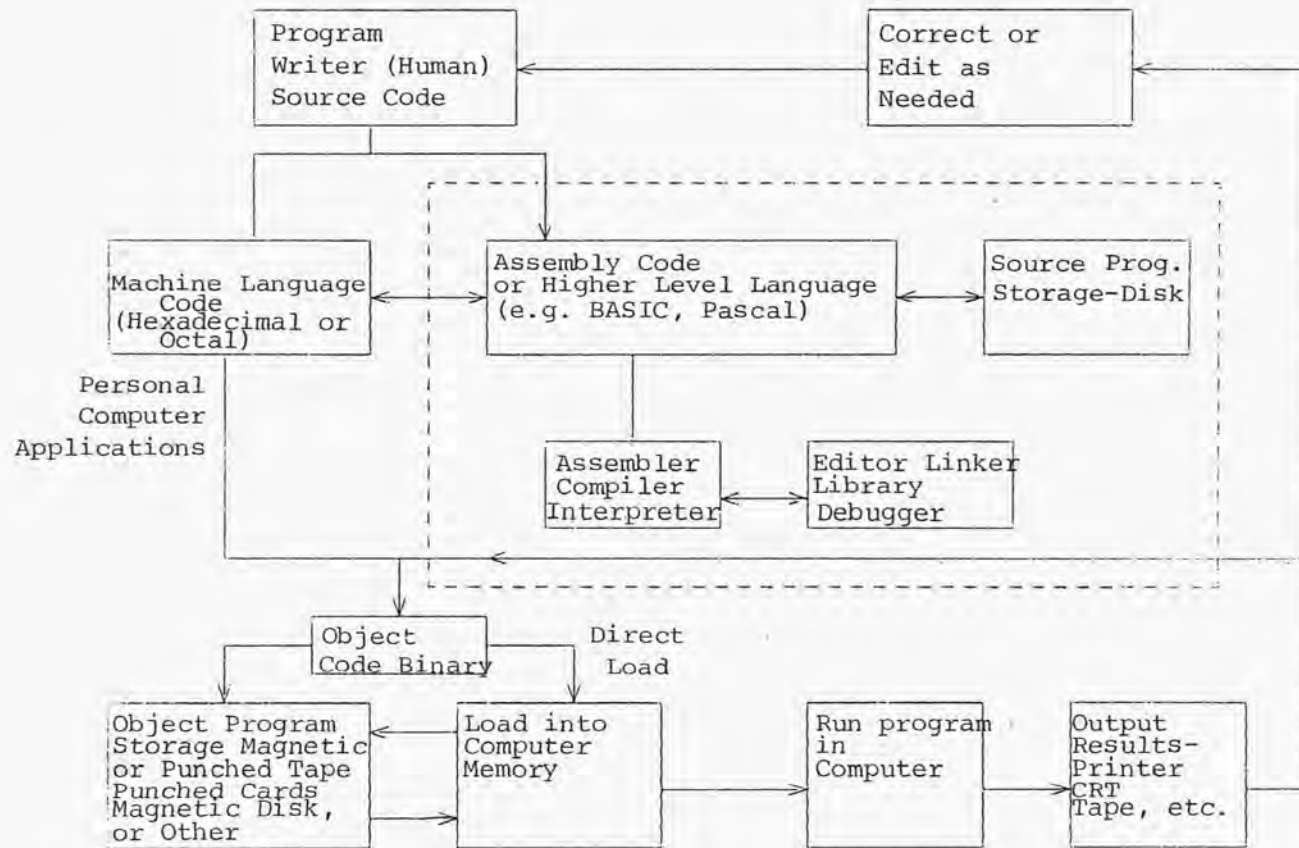


Fig. 11. The computer process.

There are two methods by which the program can enter the computer. Since the cpu only understands binary code, the human programmer must first prepare the program known as the source code. If the source code is prepared in binary (or the equivalent hexadecimal or octal, which is immediately convertible to binary), this program also becomes the binary object code which is understood by the computer. But machine language is very difficult for humans to remember and use without mistakes. It is much easier to write a program using an assembly language which features a structure such as memories, or code names resembling natural language, that helps the programmer to recall their meaning.

The alternative path to object code shown in Fig. 11, is through assembly or higher level languages, using the blocks shown enclosed in the dashed box. It is also possible, as shown, for the programmer to code in assembly language and then hand-convert to hexadecimal/object code, since there is a one-to-one relationship between them. The more professional way, which is invariably chosen by developers of industrial instruments or consumer produce using microprocessors, is to write source codes in assembly or a higher level language, and automatically (using a computer) translate this into a binary object code. This involves other programs (for the translating computer), known as assemblers for assembly language, or compilers or interpreters for higher level language. These programs are of considerable complexity and require a good deal of memory and operating time, but this operation need not be conducted on the microcomputer for which the program is designed. (3,p. 33-35)

The instruction which is a statement that specifies an operation and the values or location of its operands (5,p. G-2) generally divided into two parts: the Opcode (operation code), which tells the computer what to do, and the operand, which is a piece of data or information that the computer processes according to the Opcode. (23,p. 67)

Some of the software highlights of MC68HC11A8 single-chip microcomputer are:

- . Enhanced M6800/M6801 instruction set
- . 16 x 16 integer and fractional divide features
- . Bit manipulation
- . WAIT mode
- . Stop mode

Before making any attempt to write a program it is beneficial as well as efficient for the programmer to draw a flow chart for the program. Flow charts are a graphic way of describing the operation of a program. They are composed of different types of blocks interconnected with lines. A rectangular block describes each action the program takes. A diamond-shaped block is used for each decision, such as testing the value of a variable. An oval marks the beginning of the flow chart, with the name of the program placed inside it. An oval can also be used to mark the end of the flow chart. Three principle types of flow charting symbols are shown in Fig. 12. (19, p. 29)

The flow chart for a microprocessor-based interactive control system used for the special engine is shown in Fig. 13.

4.3 Software Specifications of MC68HC11

The MC68HC11 single-chip microcomputer unit (MCU) utilizes a four-page Opcode map, which increases the instruction set capacity. Page 1 of the map contains all of the M6801 MCU Opcodes in original locations, as well as several instructions relating strictly to the M68HC11 MCU. Three new Opcodes on page one serve as switchers to the other map pages. When the M68HC11 MCU is ready to execute an instruction, page 1 is searched for an appropriate Opcode. If found, the M68HC11 MCU executes the operation immediately as if there were no paging scheme. However, if the Opcode is located on map pages 2, 3, or 4, the M68HC11 MCU reads one of the Opcode switches on the map page 1 that directs the M68HC11 MCU to the applicable page. Pulling instructions from the map

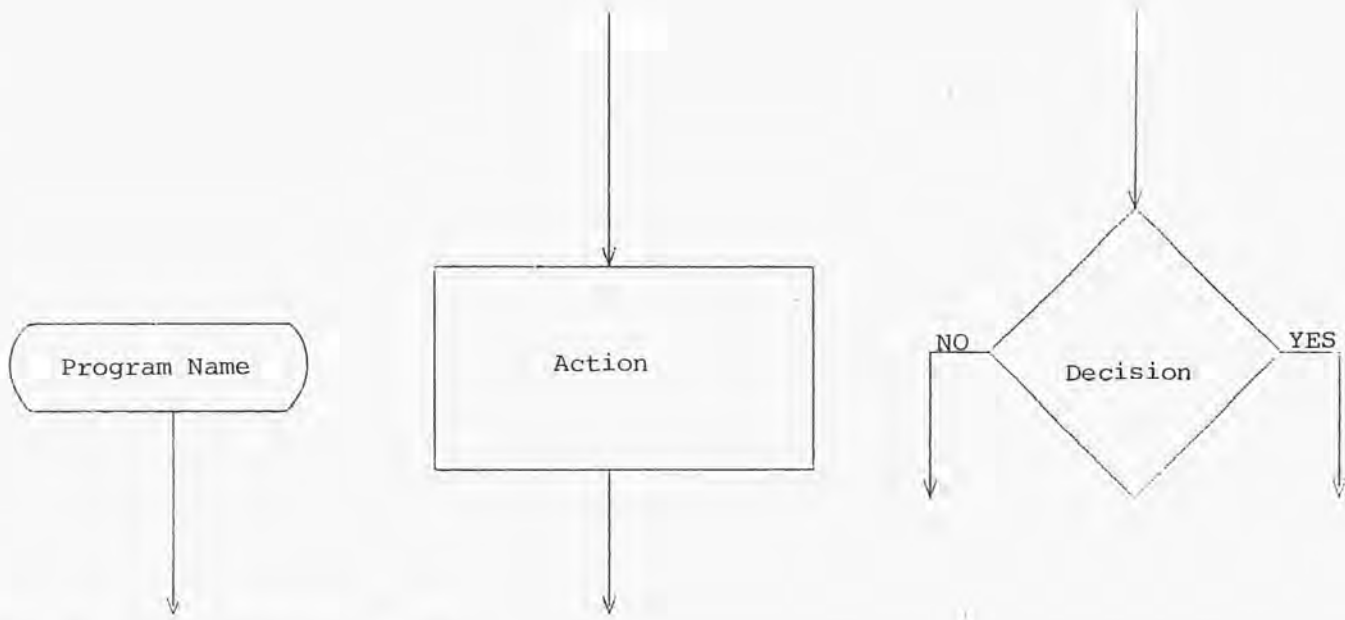


Fig. 12. Flowcharting Symbols

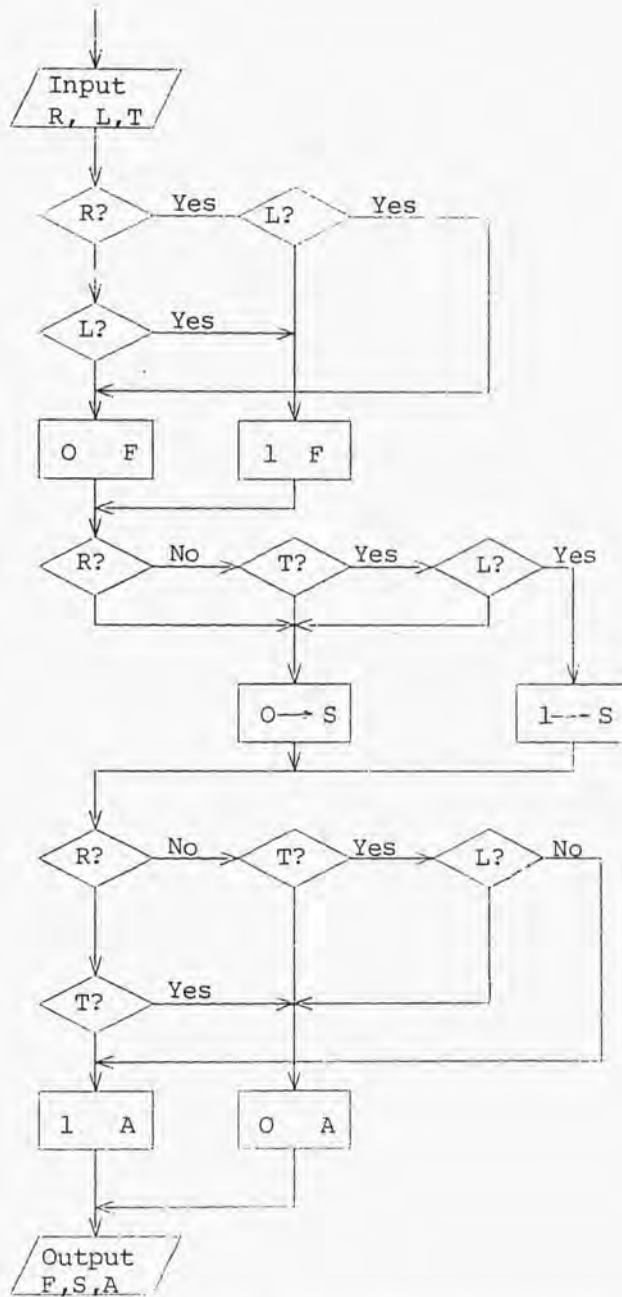


Fig. 13. Flowchart for a microprocessor-based system used for the automobile engine control system.

on pages 2 through 4 result in an extra instruction byte (prebyte) and an additional execution cycle.

Four bit-manipulation instructions-bit set (BSET), bit clear (BCLR), branch if bit set (BRSET), and branch if clear (BRCLR) - are used in conjunction with the M68HC11 MCU Opcode map. The bit-manipulation instruction contains a mask-byte operand used to indicate which bit or bits should be used by the instruction. (15,p. 1-1) The necessary information for programming purposes regarding M68HC11 can be found in Appendix 1.

Instead of going through several stages to convert the input signal from analog to digital and output signal from digital to analog in continuous control, it is more economical to build the system to operate in discrete mode. That's why discrete control in the form of ON/OFF has found many uses in industrial applications. In the next section it will be found that discrete (two-state) control offers an attractive solution for the design of microprocessor-based control systems.

4.4 The Microprocessor in a Control Loop

Fig. 14 shows a schematic diagram of a control loop containing a microprocessor. The dotted line encloses the components that would normally be located together and referred to as the microcomputer. It is intended to treat the practical aspects of specifying and interconnecting the elements in such a control loop to ensure their compatible operation. For simplicity, only one of the input variables is shown and interactions are omitted.

In the stage of input circuits design it should be kept in mind that in a typical control scheme most measurements will be made by analog transducers. Therefore, proper attention must be paid to the associated analog circuitry in some applications. Some of the questions that need to be considered at this stage are the following:

Do the signals need to be amplified, buffered or isolated before being transmitted?

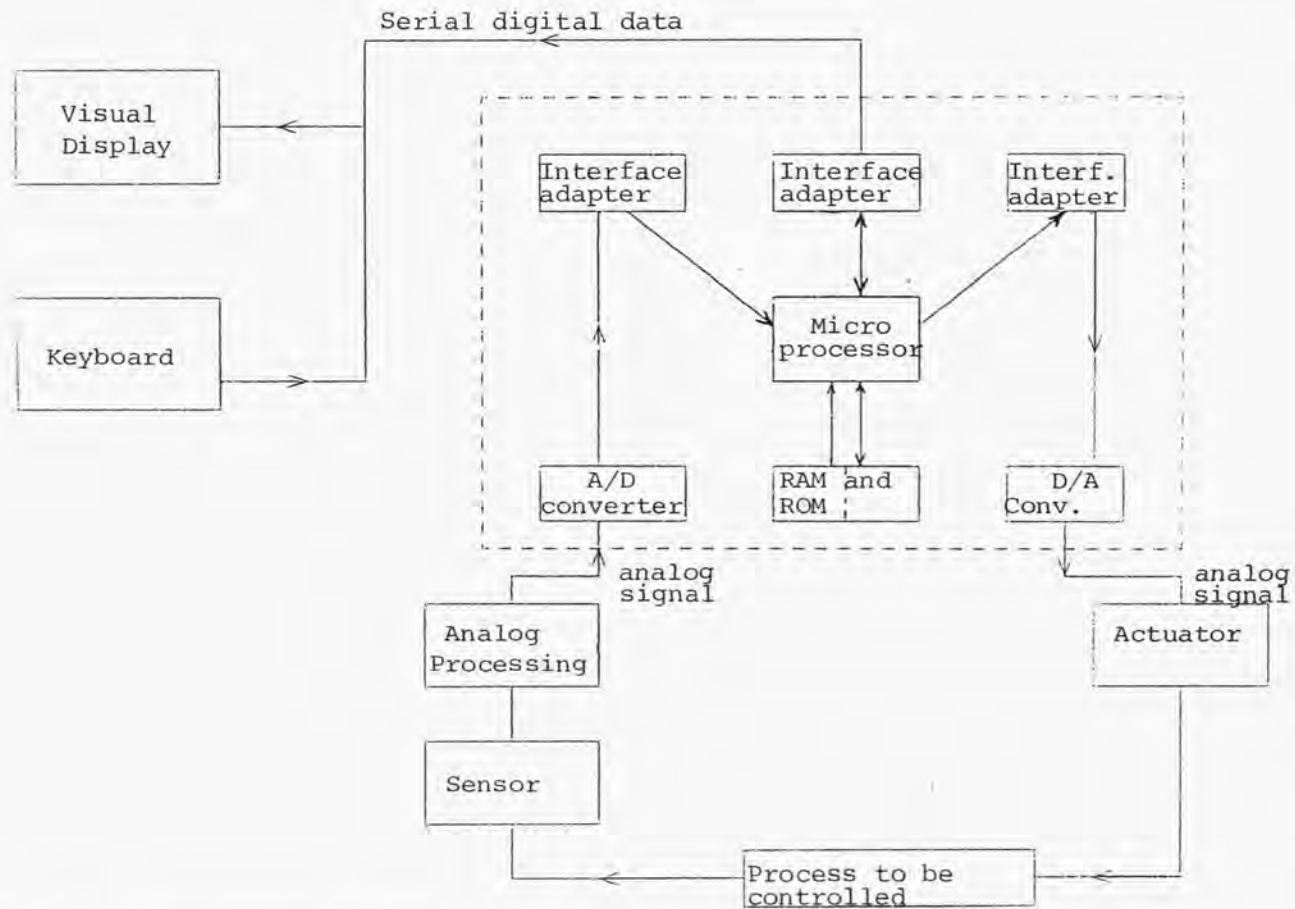


Fig. 14. A continuous control loop containing a microprocessor.

What measures, such as special grounding, shielding or analog filtering are needed to reduce interference?

Do the transducers need local excitation, open-circuit detection or some other application dependent consideration?

It is clear that the most difficult situation will occur when low-amplitude, wide-band width signals are to be transmitted over long distance through a noisy environment, with a high accuracy requirement. Some typical measurement inputs in an industrial control scheme may include, typically, fifty or more information signals arising from a variety of different devices at different geographical locations. Signal levels may vary widely and transmission distances of several hundred meters through electrically noisy environments are common. Bringing reliably to the the computer the information from this wide range of devices is an important operation. (13,p. 197-198) The selection of measurement devices are also important due to variation of types and level of signals arising. Where a sensor is located near to the computer, it is often possible to undertake special computation in a purpose-built application-oriented input.

Low-level analog signals need to be robust enough for transmission to the computer. Conversion to current signals is the most straight forward approach. A current signal can be sent over several kilometers since it is not affected by voltage drop and is more immune to noise than a voltage signal. If many analog signals arise remotely near to one point, they may be multiplexed, A/D converted and then transmitted serially along a pair of wires to the computer.

The computer is also required to detect the status (on or off) of switches, such as limit switches or auto-manual change over switches. Somewhat confusingly, signals arising from simple on-off switches tend to be called digital inputs in the commercial literature. Such signals can be considered as Boolean variables--they are input directly to the computer.

In small-scale applications, such as in the control of laboratory furnaces, it is usual for actuator to be the most expensive element in the control loop. Such systems, therefore, need to be specified with careful consideration of the actuators if a cost-effective design is to be produced. Actuator selection is very application-specific and it, therefore, cannot be explained here further.

Most processes that are called upon to control in the industrial applications operate in continuous time. This implies that an analog world must be interfaced to/from the digital computer through which the process is influenced.

The control aspect of A/D and D/A conversion can be understood by knowing the characteristics of these devices in so far as these affect the control loops into which they are connected.

Considering first analog-to-digital (A/D) conversion with the assumption that the signal $f(t)$ is to be discretized. At time T the signal $f(t)$ is connected to A/D converter.

Two questions arise:

How long does the conversion take?

How accurate is the conversion?

Considering digital-to-analog (D/A) conversion, three questions naturally arise.

These are:

How long does the conversion take?

How accurate is the initial conversion?

Is the output of the D/A converter subject to significant drift between the sampling intervals?

First D/A conversion will be considered, since every A/D converter necessarily contains a D/A converter.

4.4.1 Digital-to-Analog Conversion (D/A)

A digital-to-analog converter operates as shown in Fig. 15. A parallel digital word is converted by a logic and switching network into an equivalent resistance from which an analog voltage is derived. The final amplifier shown in the diagram prevents electrical loading and provides appropriate impedance conversion.

The settling time of a D/A converter is determined largely by the characteristics of the buffering amplifier. In some cases, the amplifier is omitted and then the settling time depends on the characteristics of the output circuit.

The output of a D/A converter may contain unwanted transients, sometimes called glitches, due to imperfectly matched switches. For instance, if the digital word 0111111 is being converted and that the word then increased by one unit to 1000000. Ideally the converter output should be as shown in Fig. 16. However, in practice the switches that control the resistor network may not be perfectly synchronized. If the six switches that represent the six least significant bits open before the switch that represents the seventh digit has closed, then the voltage from the converter contains a major glitch as shown in Fig. 17.

It should be noticed that major glitches occur only when there are major changes in the binary code. The change from 1000000 to 1000001, for instance, does not generate a glitch. The simplest way to remove glitches is probably to follow the D/A converter by a sample-and-hold device.

For analog output arrangements figures 18 and 19 shows two alternative configurations by which a multiple analog outputs may be produced. The first alternative (Fig. 18), in which each channel has its own D/A converter, is faster to respond and less prone to drift than the system of Fig. 19.

When a group of analog outputs needs to be located some distance, perhaps several kilometers, from the control computer, the configuration of Fig. 20 may offer a cost-

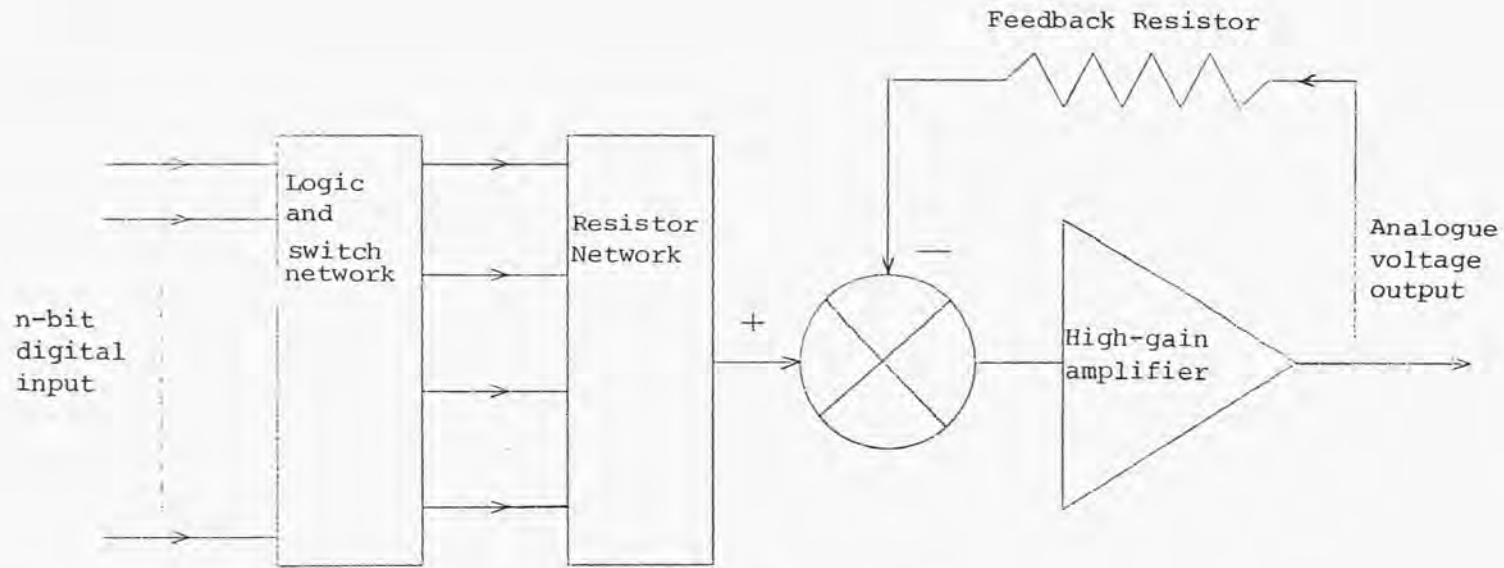


Fig. 15. Outline of the operation of a D/A converter.

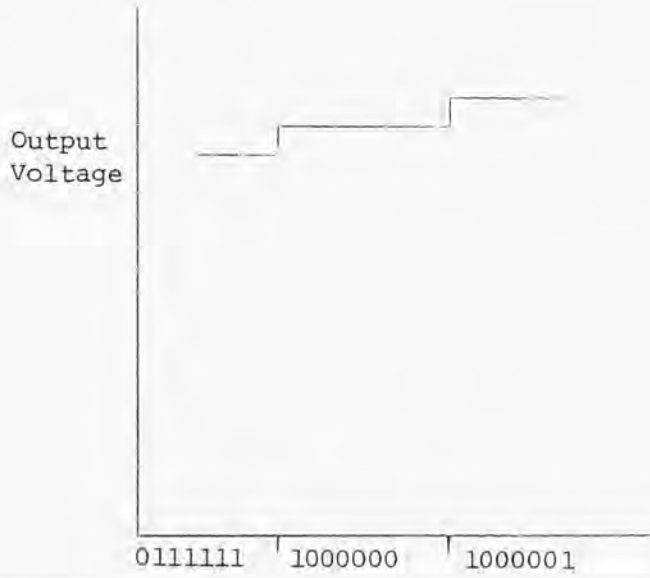


Fig. 16. The Input-Output Behavior of an Ideal D/A Converter.

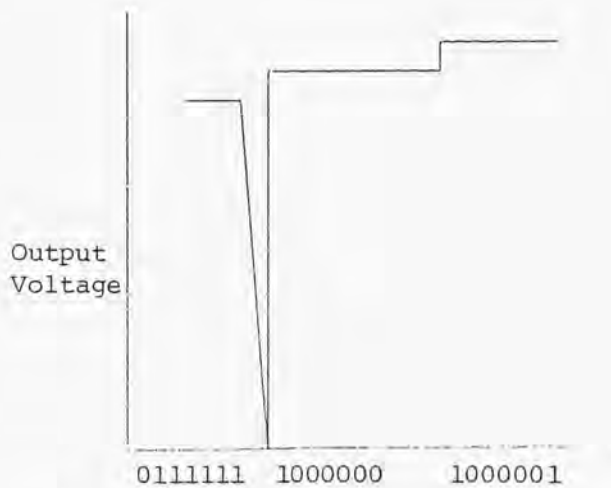


Fig. 17. The Input-Output Behavior of a non-ideal D/A Converter.

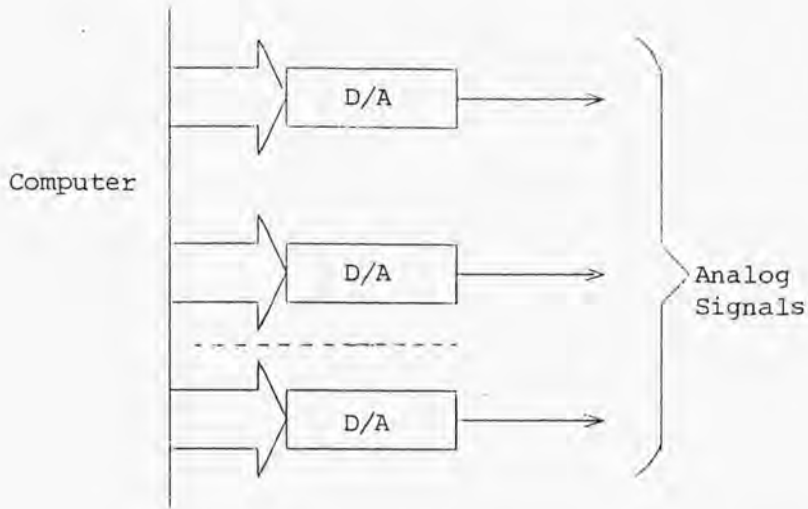


Fig. 18. Analog Output - One D/A Converter Per Channel.

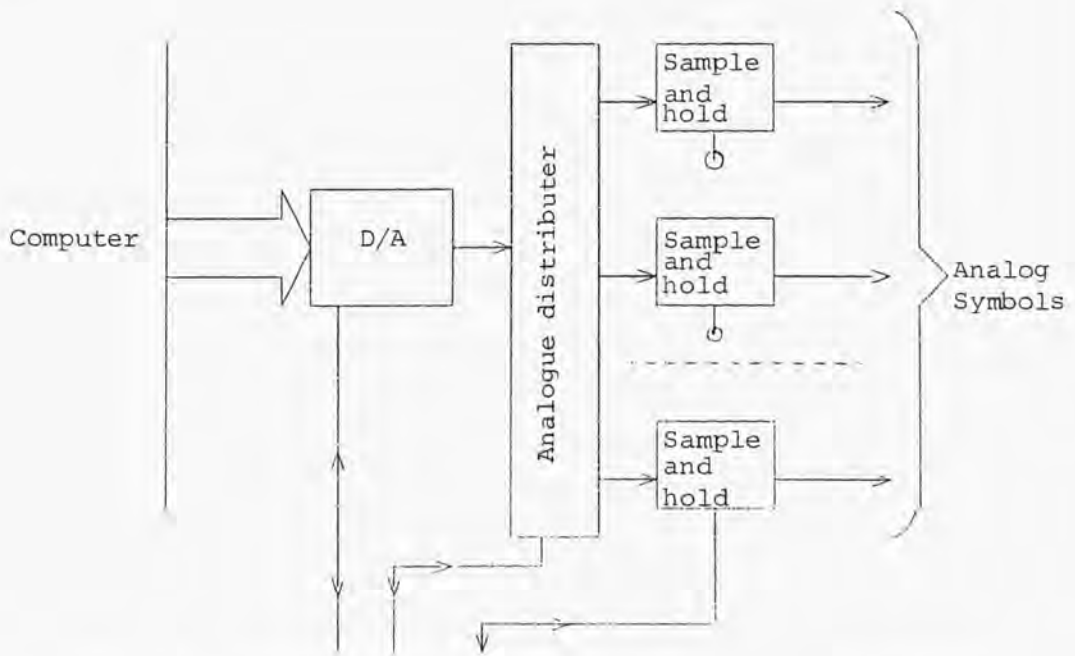


Fig. 19. Analog Output - One Sample and Hold Per Channel.

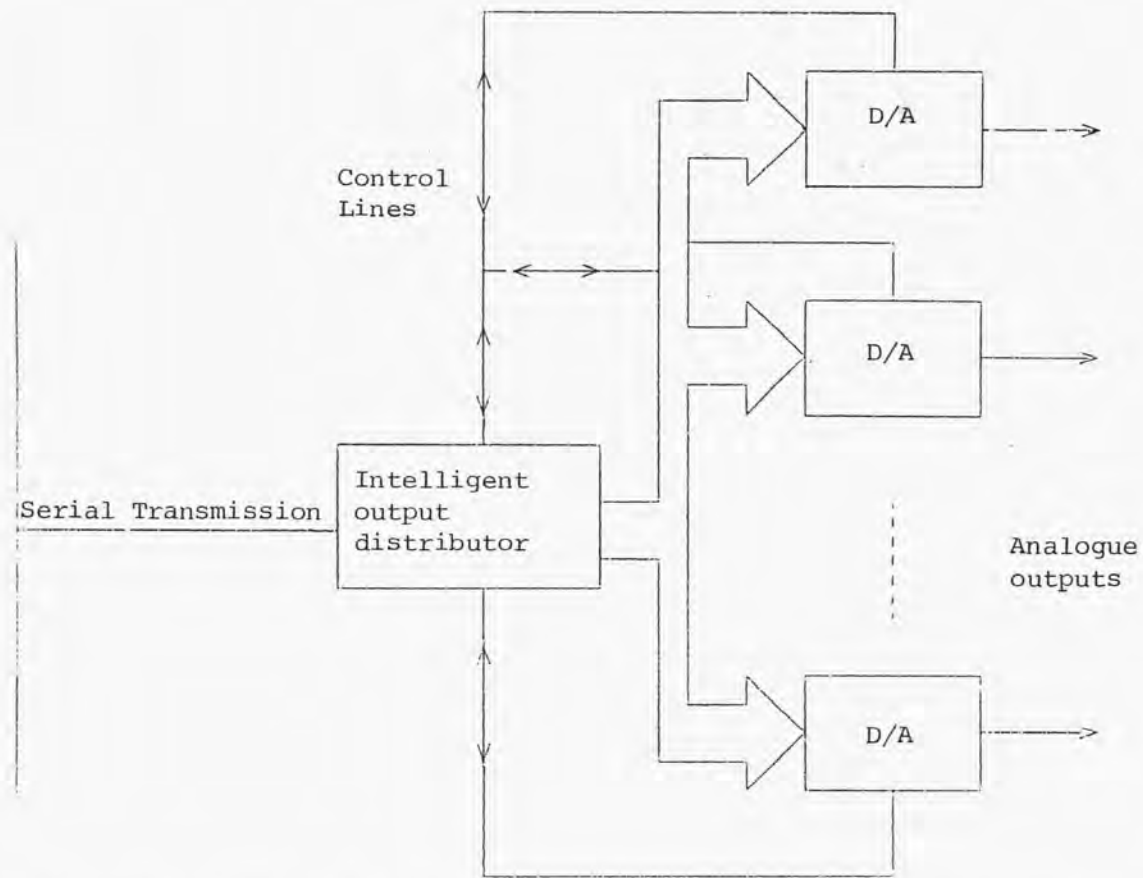


Fig. 20. Analogue Output Using Serial Transmission to a Group of Actuators.

effective solution. The system depends on an auto serial/parallel distributor that can drive a number of D/A converters (13,p. 202-206)

In some cases the digital signal appearing on the output of the latch is sufficient in itself for the control function. This is particularly true when on-off types of control functions are involved. In these cases, no digital-to-analog conversion is required. The more interesting problem in the use of computers for control is when the output is required to be an analog signal. In this case it is necessary to perform a conversion of the digital signal on the output of the latch to a proportional analog signal. The basic principle of digital-to-analog conversion (D/A) is that the digital data word is considered to define percentage or fraction of some reference signal. The fractional amount is determined from the original input signal by D/A converter. The actual output signal may be a current or voltage, but it is usually a voltage. In Fig. 21, the operation of the D/A converter is shown symbolically as producing an output voltage from the reference input based upon the value of the digital input. In equation form this can be written

$$V_{out} = \alpha V_{ref} \quad (1) \quad (4-1)$$

Where V_{out} = D/A converter output voltage

V_{ref} = D/A converter reference voltage

α = a fraction (<1) determined by the digital input signal.

The relationship between the fraction α and the digital signal is defined by considering the binary number of the digital data to be a fractional number. Thus, if an 8-bit digital output from the computer is 10110101_2 , this is considered to be 0.10110101_2 , with the decimal point to the left of the most significant bit (MSB). In this case, α is defined by

$$\alpha = b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \dots + b_n 2^{-n} \quad (4-2)$$

Where $b_1 b_2 \dots b_n$ = the binary number with b_1 the MSB.

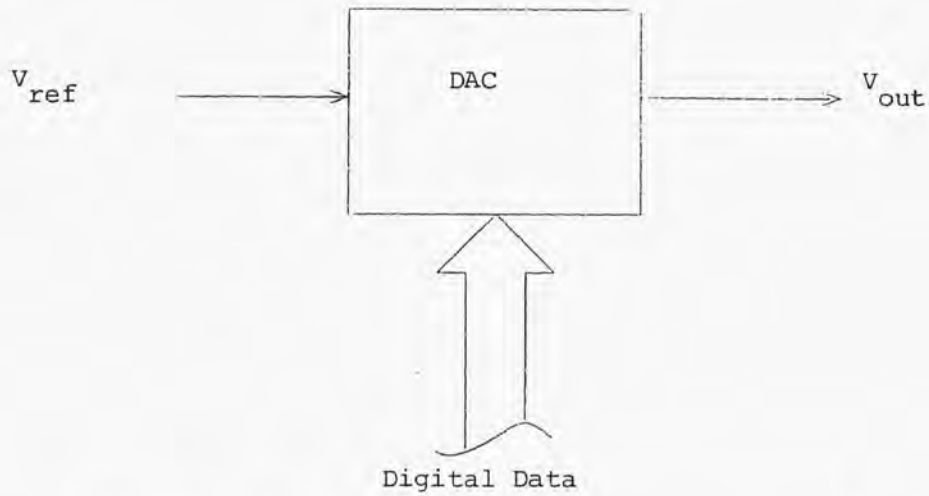


Fig. 21. The digital-to-analog converter (D/A).

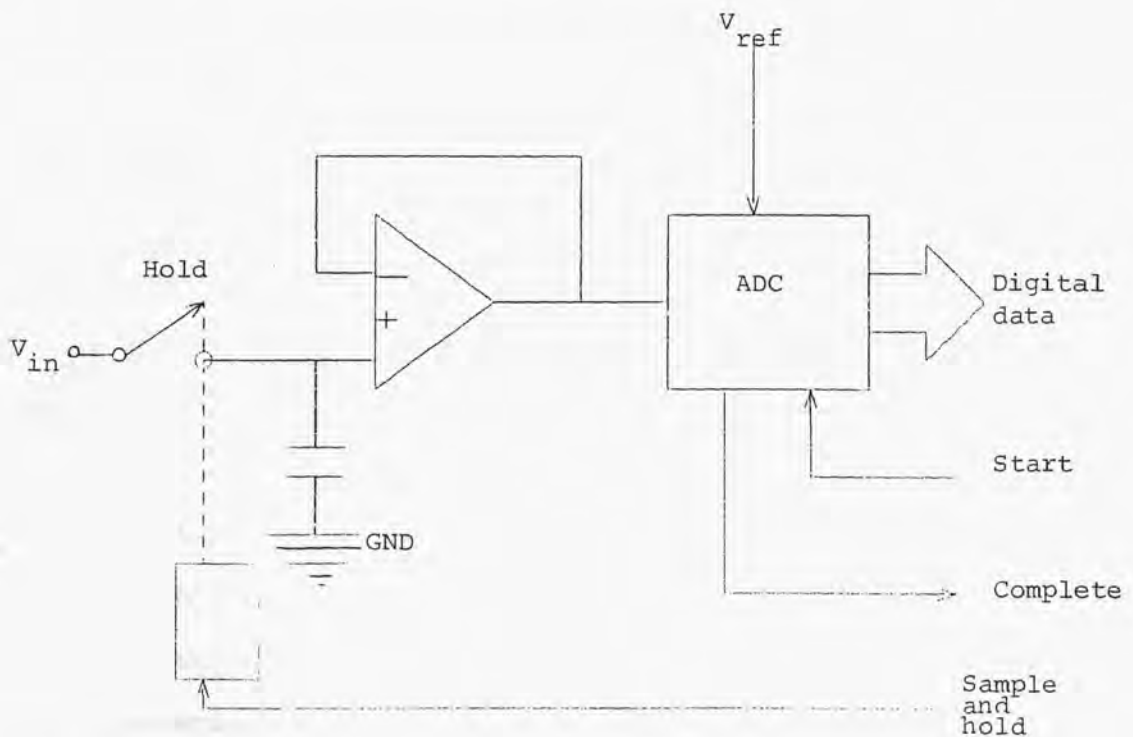


Fig. 22. A sample and hold as well as A/D converter for a data input system.

Of course, the above equation can be generalized to any number of bits in the data bus of the computer. Basically, the D/A converter simply calculates the value of V_{out} using the above equation and multiplies this times a reference voltage. To determine the output step size, if the input digital data are composed of 8 bits there are 256 possible states or the values of this number from 00H to FFH. Thus there will be 256 steps in the output voltage as determined by Eqs. (4-1) and (4-2). The size of each step is simply the reference divided into 256 values. This is called the resolution of the output voltage:

$$V_{out} = \frac{V_{ref}}{256} \quad (4-3)$$

For the general case of a digital signal of n bits, the resolution equation can be generalized to

$$V_{out} = V_{ref} 2^{-n} \quad (4-4)$$

Where V_{out} = step size of output voltage. The step size is very important because it indicates the fitness by which the output voltage can be varied. If very delicate and smooth control is desired, the step size must be very small.

Another important point regarding the use of D/A converters is that the maximum output voltage is not equal to the reference. The reason for this is that V_{out} will always be less than 1, even with the maximum digital input of all 1's.

Relationship Between Input and Output. It is not difficult to calculate the analog output from Eq. (4-2) when the digital input is known. The reverse problem, of finding the digital input that produces a specific output, is somewhat more complicated. Part of the problem is that, since the output jumps in increments of the step size, it is only possible to find the digital input that gives an output closest to that desired. This is done by finding the fraction and then converting this to the closest binary number with the specified number of bits. (9, p. 78-80)

4.4.2 Analog-to-Digital Conversion (A/D)

When a computer control system involves continuous variation of an analog variable over a range, the value of this variable must be converted into a proportional digital signal for input to the computer. This is the reverse of the problem of digital-to-analog conversion considered previously. It turns out that there are more difficulties associated with the analog-to-digital conversion, however, which makes their use in the control systems a little more complicated. The basic idea is to consider the analog data to be a "number" and to convert this into the equivalent binary number. The difficulty is at once obvious; the binary number can only have a finite number of bits, such as the common 8-bit microprocessor-based computers, and therefore can only represent a limited range of numbers. In fact, for 8 bits one can only represent 256 counting states (including zero). So it can be seen easily that there will be a loss in knowledge of the variable value in going from continuous analog information to finite bits of digital information. Anyway, as the D/A converter, it turns out to be the easiest to treat the analog data as a fraction of some reference. If voltage is taken to be the analog medium, input voltage will be considered some fraction of a reference voltage, V_{ref} . This means that the input voltage will have to be less than this reference.

What most A/D converters do is to find a fractional number, given by the binary output, that is the closest smaller fraction of the analog input voltage. In equation form,

$$V_{in} > \alpha V_{ref} \quad (4-5)$$

Where V_{in} = analog input voltage

V_{ref} = analog reference voltage

$$\alpha = b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \dots + b_n 2^{-n} \quad (4-6)$$

Equation (4-6) assumes an n-bit bit word results from the conversion. The inequality of Eq. (4-5) means that the voltage on the right side of the equation will always be less than

the input voltage, but never by more than the step-size voltage represented by LSB of the digital signal. Thus the uncertainty in this ideal case is never greater than

$$V = V_{ref}2^{-n} \quad (4-7)$$

Relationship Between Input and Output. The actual relationship between the input and output can be deduced by procedures like that used for the D/A converter. If digital output is known, and the reference, then limits can be placed upon the possible values of the analog input voltage. The limit is just that represented by the step-size voltage given by Eq. (4-7). When the input analog voltage is known, and the reference, and the binary output is desired, a calculation is performed like that for D/A conversion. The fractional ratio of input voltage to reference is first calculated. Then this is converted to a binary by the process of successive multiplication by 2. (9,p.89-90)

Conversion Time. One of the most important characteristics of A/D converters is that a finite amount of time is required for the device to produce a digital output from the input analog voltage. The length of time required for the A/D converter to calculate the binary output of an analog input varies over a large range, depending upon the type of conversion process employed. One of the most common processes is called the successive approximation A/D converter. This device will typically convert 8 bits in 30 to 50 μ s. Another type, commonly used for digital voltmeter, is called the dual-slope A/D converter and may take up to 1000 μ s for a conversion. The flash converters are among the fastest, since an 8-bit conversion may be completed in only a few nanoseconds, but this A/D converter suffers from other disadvantages that limit its usefulness.

The finite conversion time of A/D converters has several important consequences when the A/D converter is used in data-acquisition systems. The following paragraphs describe factors in the application of A/D converters that result from the finite conversion time.

1. State Convert Command. Since the A/D converter takes a finite length of time to determine the binary output of an analog input, the binary output does not represent the input at every instant of time. In fact, most A/D converters do not even calculate the binary output until receiving a command, in the form of a digital signal input, to start the conversion process. Thus the computer or external equipment must generate a command to the A/D converter to start the conversion process when the computer needs to input the data. This is often called the START CONVERT (SC) command.

2. Conversion Complete Signal. The length of time required to perform a conversion is not constant, even for a given A/D converter. The time is dependent on the frequency of an internal A/D converter clock. For this reason the A/D converter generates a digital output signal that notifies the computer or other external equipment when the conversion process is complete. This is a signal that the computer can input the binary output of the A/D converter. This is often called END OF CONVERT (EOC) or CONVERSION COMPLETE (CC) signal.

3. Analog Voltage. Since a finite length of time is required for A/D converter to compute the binary output, it stands to reason that input voltage must remain constant during this interval. The A/D converter refers to the value of input voltage during the conversion process. Therefore, if this voltage were changing, the conversion process would become confused and the output would be in error. Thus either the change in the input voltage must be very slow compared to conversion time or a system must be used to "hold" the voltage value at the moment a conversion is started by a convert start signal.

The most important consequences of the conversion time is its impact on the process of analog data input to a computer. In general, a four-step sequence must occur:

1. The computer issues a command to the A/D converter to start conversion (SC).
2. The computer goes into a wait mode while the conversion process is taking place.

3. The A/D convertor sends a conversion complete (CC) signal to the computer when the binary output has been determined and placed on the A/D converter binary output lines.

4. The computer reads the A/D converter binary output into the data bus.

Sample and Hold. In those cases when the input voltage changes at a rate not slow compared to the conversion time, it will be necessary to capture and "hold" an input value at the moment of a sample of analog voltage is to be converted. This is accomplished by a sample-and-hold circuit constructed using op-amps. The basic principle of such a circuit is shown in Fig. 22. The switch is a solid-state device, usually an FET, which is turned on by a digital input signal. In the on state the circuit is in the sample mode, and the changing input voltage will appear across the capacitor, C. The voltage-follower op-amp is selected to have very high impedance. When the digital input signal opens the switch, the circuit enters the hold mode. Whatever voltage was on the capacitor at the instant the switch was opened will now remain, regardless of subsequent changes of input voltage. The capacitor voltage will not change, even when "measured" by the A/D converter, since the high input impedance of the voltage follower prevents discharge of the stored voltage. Fig. 23 illustrates the time sequence of successive sampling and holding of a changing analog voltage. In fact, the actual binary signal input by the computer will be samples of the analog voltage at intervals determined by the time from one hold to the next hold. The fact that the computer has only periodic samples of process variables will have important consequences on control.

The ability of the capacitor voltage to track fast changes in the input voltage in the sample mode is determined by the source resistance, R_s , of the circuit providing V_{in} to the sample-and-hold circuit. The time constant $R_s C$ must be as small as possible. This is often assured by using a voltage follower on the input before the switch. The low output

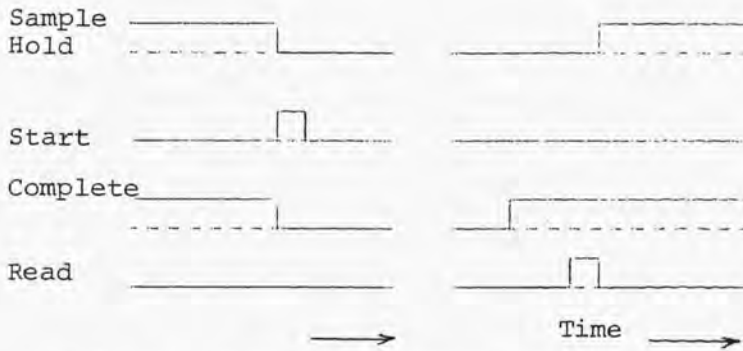


Fig. 23. A timing sequence for a data input process.

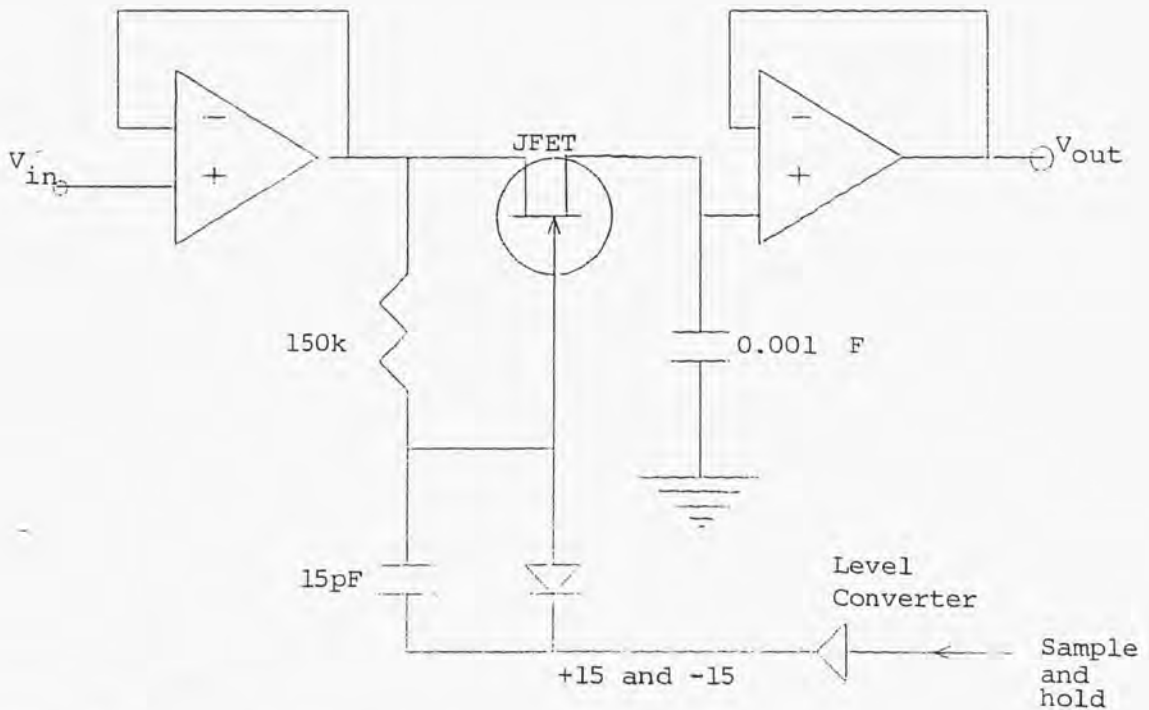


Fig. 24. A typical sample-and-hold circuit.

resistance of the follower and low "on" resistance of the switch provide for fast tracking of input voltage changes. A typical sample-and-hold circuit is shown in Fig. 24.

Use of the sample-and-hold circuit introduces the need for another command in the data-acquisition process. Now the computer must issue a hold command in addition to the START CONVERT command.

Input-Level Adjustment. The voltage generated by measurement of some process variable has a level and range dependent on the transducer and signal conditioning of the measurement process. The A/D converter will perform conversion on the basis of a voltage varying between 0 and V_{ref} . To obtain compatibility between the measurement and A/D converter, it is often necessary to use amplifiers, attenuators, and voltage bias circuit between the measurement system and the A/D converter. These circuit typically use standard op-amp approaches. It is very important to maintain traceability throughout such conditioning between the signal levels and ranges and the process variable. (9,p. 89-95)

4.4.3 Input Multiplexing

A multiplexer is a device for scanning across a number of analog signals and time-sharing them sequentially into a single analog output channel.

The switching is usually performed by JFET or CMOS transistors although mechanical read relays may still be preferred for some applications. The speed of a multiplexer depend on:

(a) the speed of the switch (typical switching times for JFET and Reed relays are 2×10^{-7} s and 10^{-3} s respectively)

(b) the settling time of the circuit fed by the multiplexer.

If the time constant of the circuit is τ seconds and A/D converter has n bits; then a time T_s must be allowed to elapse before the multiplexer output is A/D converted, where

$$T_s > \tau \ln(2^n - 1) \quad (4-8)$$

Each time the multiplexer switches, a transient occurs in the signal that is passed on to be A/D converted. Satisfaction of the given inequality guarantees that the transient has died away to a magnitude that cannot cause an error in the digital conversion, even in the worst case, in which the multiplexer switching is between signals at the opposite end of the conversion range. A typical value for τ might be $\tau = 10^{-6}$ s. This leads to a necessary waiting time for at least 5.5×10^{-6} s for 8-bit or 11×10^{-6} s for 16-bit, working.

The choice of a multiplexer for a particular application involves the familiar compromise between speed and accuracy--if both high speed and high accuracy are needed, the required device will be relatively expensive. So three alternative approaches should be considered. The flying capacitor method that has been found adequate for many industrial applications; the analog multiplexer that is required for the most exacting application; and digital multiplexing. The choice between these approaches can only be made by preparing comparative cost and performance budgets for envisaged application. (13,p. 211-214)

4.4.4 Signal Processing Cycle

The sampling is usually performed periodically with sampling time. To buy a multiplexer which is constructed together with an effective range selector and an analog/digital (A/D) converter. The digitized input data is sent to the central processor unit. There, the output data are calculated using programmed algorithms. If an analog signal is required for the actuator, the output data emerge through a D/A converter followed by a hold device. Fig. 25 shows a simplified block diagram.

The samplers of the input and output signal do not operate synchronously, but are displaced by an internal T_R . This interval results from the A/D conversion and the data processing within the central processing unit. Since this interval is usually small in comparison with the time constants of the actuators, processes and sensors, it can often be neglected. Synchronous sampling at the process computer input and output can therefore be assumed. Also the quantization of the signal is small for computers with a word length

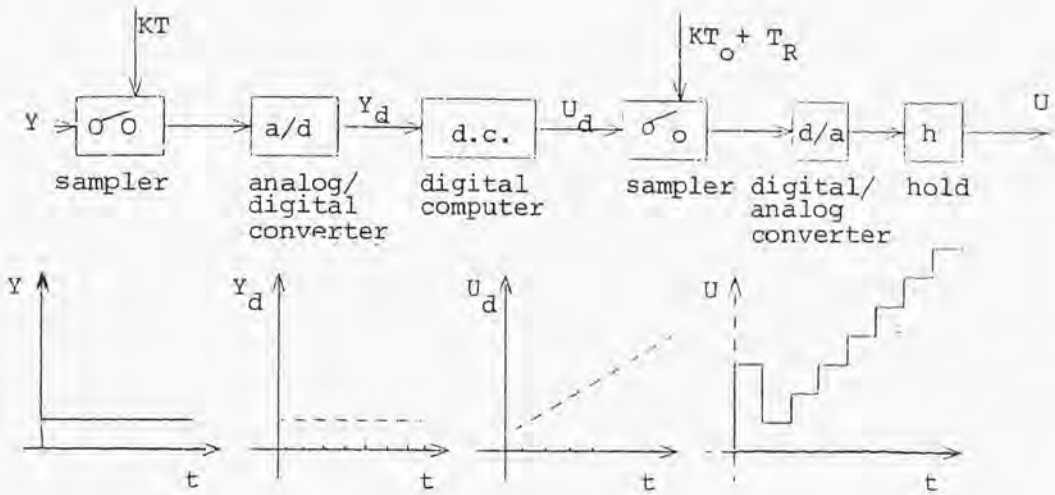


Fig. 25. The process computer as sampled-data controller.

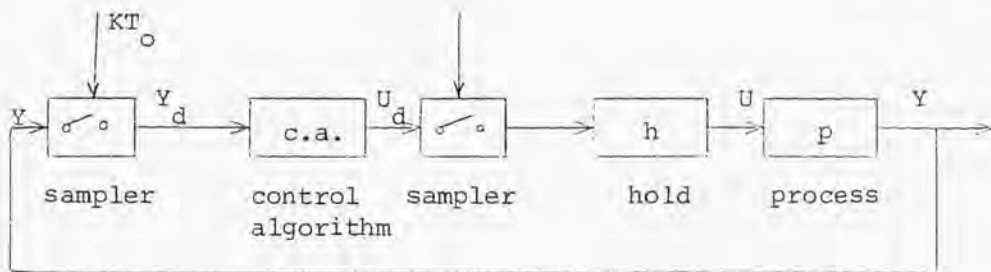


Fig. 26. Control loop with a computer as a sampled-data controller.

of 16 bits and more and A/D converters with at least 10 bits so that the signal amplitudes initially can be regarded as continuous.

These simplifications lead to the block diagram of Fig. 26, which shows a control loop with a process computer as a sampled-data controller. The samplers now operate synchronously and generate time-discrete signals. The manipulated variable U is calculated by a control algorithm using the control variable Y and the reference value W as inputs. Such sampled-data control loops do not only exist in connection with process computers. Sampled data also occurs when:

- measured variables are only present at definite instants
- multiplexing of expensive equipment (cables, channels) (8,p. 10-11)

4.4.5 Digital Interfacing

Digital interfacing is concerned with the technology digital data transfer between devices. The chief data transfers that are needed in control application are:

- (a) transfer of process measurement data from an A/D converter to a microprocessor-based system;
- (b) transfer of actuator commands from a microprocessor-based system to a D/A converter;
- (c) transfer of data between a microprocessor-based system and peripheral devices, such as keyboard, visual display unit, tape drives, printers, graph plotters and other computing devices such larger supervisory computers.

The data transfers are usually achieved by sequential (serial) transfer when the distances are large and by parallel transfers when devices are close together and rapid transfer is required. In part (a) and (b) are usually achieved by parallel transfer, serial transmission being reserved for special cases involving relatively long distance. The peripherals in part (c) are serviced by a mixture of serial and parallel transfers.

1. **Serial Interfacing.** Since a microcomputer configuration operates internally by parallel data transfer, it is necessary to use a serial-to-parallel device to interface a serial line to the system. Such a serial-to-parallel converter may consist of a register that is filled, one bit at a time, at the rate dictated by the system clock, by incoming serial data. When the register is full, it is connected to the system data bus. Serial data transfer is facilitated by the use of special serial/parallel chips that contain the logic necessary for organizing the operation. A common device is the ACIA (Asynchronous Communications Interface Adaptor), sometimes referred to as a UART (Universal Asynchronous Receiver and Transmit) device. A typical ACIA has two serial connections for input and output respectively. It has logic connections with the remote data source/sender and the address bus of the computer and it connects via a buffer register to the computer data bus as shown in Fig. 27.

2. **Parallel Interfacing.** Most of the parallel interfacing required in simple control applications is achieved through the use of PIA (Peripheral Interface Adaptor) chips. A PIA is programmable in so far as manipulation of particular bits in its control register. Alters the operating configuration. A PIA is the natural interfacing device to interpose between an A/D or D/A converter and a microprocessor system. A typical configuration is shown in Fig. 28. Much of the interfacing effort is devoted to proper connection of PIA device which are directly in the control loop.

Because of the importance of the PIA chip, it is intended to outline how the connection to an A/D converter is done. Fig. 29 shows a PIA chip with control register CR and data register DR. Its connection to the microprocessor is through the address bus, the data bus and an interrupt line. Its connection to the A/D converter is through a parallel port and two control lines.

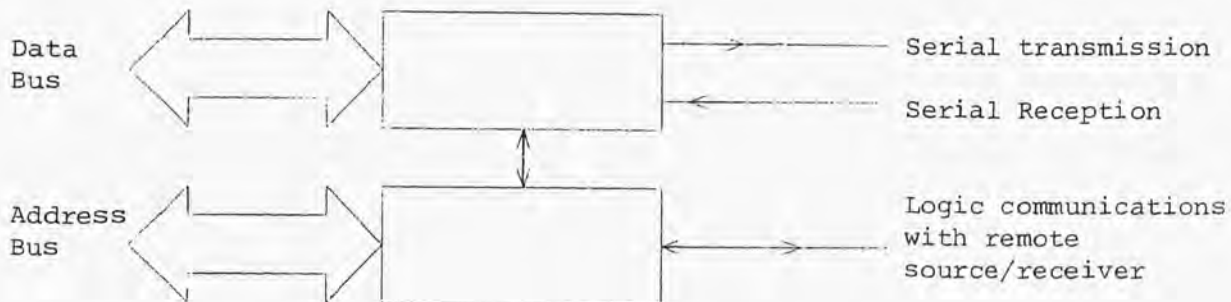


Fig. 27. Outline of an asynchronous communications interface adapter (ACIA).

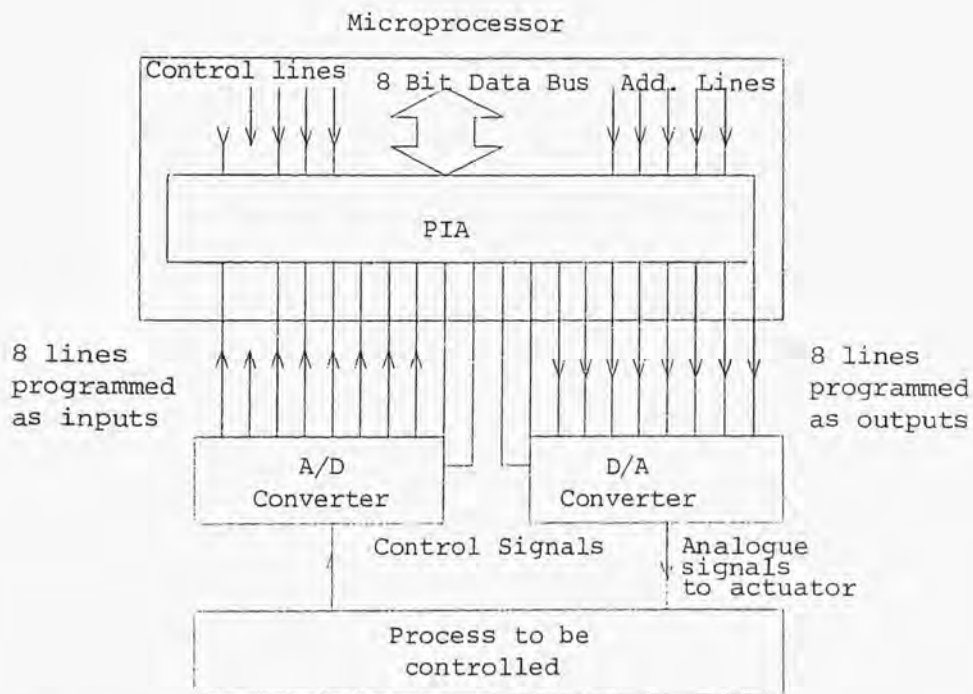


Fig. 28. A microprocessor in a basic control loop.

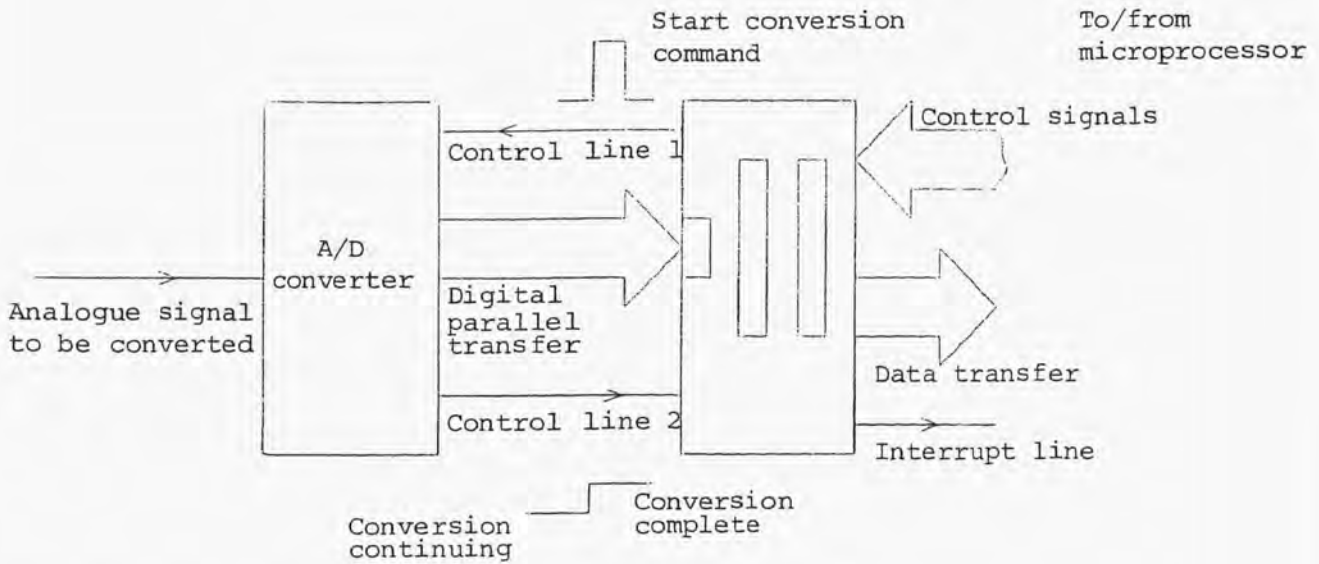


Fig. 29. Control of an A/D converter by a PIA chip.

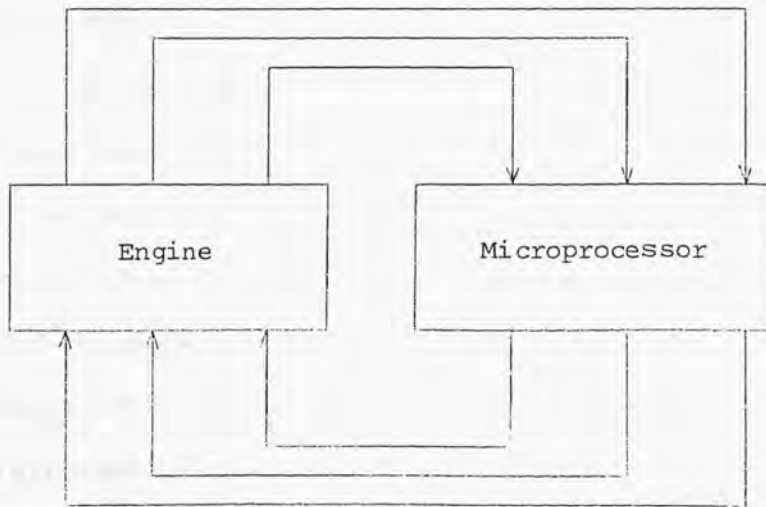


Fig. 30. Essential Feature of Engine Control.

A command from the microprocessor along the address bus fixes the configuration of the PIA so that, in particular, the parallel port is designed as an input port. The operation of the arrangement is then as follows:

(a) The PIA sends a "start-conversion" signal on control line, to the A/D converter.

(b) When conversion is complete, the A/D converter sends a "conversion completed signal along control line 2 to the PIA.

(c) PIA reads the data from the converter into its data registers, set a particular bit in its control register and sends an interrupt signal to the microprocessors.

(d) The microprocessor checks bits in the control registers of all PIA's to see which has raised the interrupt. It then transfers the data from the PIA and clears the bit in the control register.

(e) When the time is reached for a new input signal to be obtained the microprocessor initiates the procedure again. (13,p. 219-222)

4.5 Microprocessor-Based Implementation of Multivariable Interactive Discrete Control System

Many discrete-state systems are composed of variables that interact and for which the output states are dependent on the states of several input variables. A multivariable discrete two-state system with interaction is one for which a control output is determined by the states of a number of inputs. If the inputs are all expressed in digital format (ON and OFF), these state-dependent systems are the same as combination logic systems.

Generally, Boolean equations can be written by which the outputs are determined from the inputs. These equations can be solved by hardware combination logic circuits or by software in a computer. There are number of advantages to using the computer for solving these equations. Since software changes can be easily made to accommodate new designs, it is not necessary to attempt simplification of the equations, and many such equations can be handled by one computer.

If a discrete-state system has a set of n two-state input variables and a set of m two-state control outputs, a set of Boolean equations can be written for the control:

$$\begin{aligned}
 C_1 &= F_1(V_1, V_2, \dots, V_n) \\
 C_2 &= F_2(V_1, V_2, \dots, V_n) \\
 &\cdot \quad \cdot \\
 &\cdot \quad \cdot \\
 &\cdot \quad \cdot \\
 &\cdot \quad \cdot \\
 C_m &= F_m(V_1, V_2, \dots, V_n)
 \end{aligned}
 \tag{4-9}$$

Where $C_1 \dots C_m = m$ Boolean control outputs

$V_1 \dots V_n = n$ Boolean inputs

$F_1 \dots F_m = m$ functions relating inputs and outputs

The functions will consist of Boolean equations involving the input variables and their inverses along with AND and OR operations. The control problem reduces to finding ways to implement the equations of software. (9,p. 145-146)

4.6 Microprocessor-Based Implementation of the Special Engine Control System

Even though there are different detailed approaches to engine control employed recently in automotive industries all approaches require certain variables to be measured at a given time intervals. These measurements are used to decide, through computations, actions to be taken to control the engine.

The engine control system which is used as a model consists of three variables that are sensed and three actions controlled, as illustrated in Fig. 30. The values of the three input quantities are entered into a set of equations stored in the microprocessor. The result of mathematical calculations determines the optimum fuel feed (F), air feed (A), and spark

advance (S) mode. The three variables that are being measured are: speed in revolution per minute (R), temperature of engine (T), and load (L).

The center of the control is the microprocessor and its affiliated components.

Sensors collect the required data. Actuators respond to the control commands.

This microprocessor-based control system for a special engine control starts with a primary sensing element (sensor or transducer) that senses a condition, state, or value of a process variable and produces an output that reflects a condition. In the final stage of control referred to as actuators; a switch or contact may be opened or closed; a valve may be fully opened or closed; an electromagnetic device such as solenoid valve may be energized or de-energized; and a motor may be started or stopped.

The selected problem is a special engine control system having multivariable interactive discrete two-state input variables of rpm (R), temperature (T) and load (L). The two-state outputs are fuel feed (F), air feed (A), and spark advance (S). It is desired that the outputs to be high under the following conditions:

Fuel feed: When the rpm is low and the load is high, or when the rpm is high and load is low.

Air feed: When the temperature is high and the rpm and load are low, or when the temperature is low and the rpm is high.

Spark advance: When the temperature is high and the rpm is low and the load is high.

4.6.1 Algorithm: The algorithm is the set of equations and/or sequence of operations that solve some problem. In this case, the Boolean equations can be referred as the algorithm for the engine control system. Therefore, before attempting to design the software for this particular application, it is better to express the desired outputs in the form of Boolean equations which are:

$$\begin{aligned}
 F &= \bar{R} \cdot L + R \cdot \bar{L} \\
 A &= T \cdot \bar{R} \cdot \bar{L} + \bar{T} \cdot R \\
 S &= T \cdot \bar{R} \cdot L
 \end{aligned}
 \tag{4-10}$$

Where \cdot is AND (a logic notation)

$+$ is OR (a logic notation)

$\bar{}$ on the top of any input variables means NAND (a logic notation) showing the low or off condition.

Now the flow chart should be constructed for the aforesaid equations. The flow chart for this multivariable interactive discrete control system for the special engine already discussed is shown in Fig. 13. (9,p. 146-147)

4.6.2 I/O Truth Table

A common method used to tabulate all the possible combinations of input and output levels for a given Boolean equation is called truth table. Table 2 shows the truth table constructed for the special engine based on all the possible combinations of input and output. R, L, and T are the inputs and F, A, and S are the outputs. Zero means low (off condition) and 1 means high (ON condition).

4.6.3 Development of the Program in Assembly

Assembly language permits us to develop software using a mnemonic for each instruction instead of the 1s and 0s which the microprocessor understands. It also permits us to represent addresses in RAM and ROM with address labels. The process of assembly consists largely of translating the mnemonics and address labels of the assembly language source program into the object code of the microcomputer. (16,p. 352)

Assembly language is not one specific language, but a class of languages. Each microprocessor has its own machine language and therefore its own assembly language. The following is the program developed in assembly language for the model, the special

INPUT			OUTPUT		
D	L	T	F	A	S
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	0	1
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	1	0
1	1	1	0	0	0

Fig. 2. Truth table for the special engine control system which used as a model.

engine, which involves the three interactive discrete input variables R, L and T. The outputs are F, A, and S. This program uses assembly language based on MC68HC11 Motorola single-chip microcomputer. Only one port is used for I/O purposes. The address of the 1, and 2 are used to input T, L, and R respectively. Bit 3, 4, and 5 are used to output S, A, and F. Bit 6 and 7 are not used. The following program is run, and tested with simulated inputs.

The outcome of this program meets the desired outputs for the special engine control system.

<u>Label</u>	<u>Address</u>	<u>Instruction</u>	<u>Comments</u>
START	E000	CLRA	Clear Accumulator A
Test 1	E001	LDAA#4	Load A (input R)
	E003	ANDA \$1000	AND A with port A
	E006	BNE \$E00A	If Result = 0, Jump Test 2
	E008	BEQ \$E014	If Result = 0, Jump Test 3
Test 2	E00A	CLRA	Clear Accumulator A
	E00B	LDAA#2	Load A (input L)
	E00D	ANDA \$1000	AND A with port A
	E010	BNE \$E0A	If Result = 0, Then OFF1
	E012	BEQ \$E01E	If Result = 0, Then ON1
Test 3	E014	CLRA	Clear Accumulator A
	E015	LDAA #2	Load A (input L)
	E017	ANDA \$1000	AND A with port A
	E01A	BNE \$E01E	If Result = 0, Then ON1
	E01C	BEQ \$E02A	If Result = 0, Then OFF1
ON1	E01E	CLRA	Clear Accumulator A
	E01F	LDAA \$1000	Load A with Content of Memory Location \$1000
OFF1	E022	ORAA #20	OR A with 20H
	E024	STAA \$1000	Store Content of A in \$1000
	E027	JMP \$E036	Clear Accumulator A
	E02A	CLRA	
	E02B	LDAA \$1000	Load A with Location \$1000

	E02E	ANDA #DF	AND A with DFH
	E030	STAA \$1000	Store Content of A in \$1000
	E033	JMP \$E036	
Test 4	E036	CLRA	Clear Accumulator A
	E037	LDAA #4	Load A (input R)
	E039	ANDA \$1000	AND A with port A
	E03C	BNE \$E060	If Result = 0, Then OFF2
	E03E	BEQ \$E040	If Result = 0, Jump Test 5
Test 5	E040	CLRA	Clear Accumulator A
	E041	LDAA #1	Load A (input T)
	E043	ANDA \$1000	AND A with port A
	E046	BNE \$E04A	If Result = 0, Jump Test 6
	E048	BEQ \$E060	If Result = 0, Then OFF 2
Test 6	E04A	CLRA	Clear Accumulator A
	E04B	LDAA #2	Load A (input L)
	E04D	ANDA \$1000	AND A with port A
	E050	BNE \$E054	If Result = 0, then ON 2
	E052	BEQ \$E060	If Result = 0, Then OFF 2
ON2	E054	CLRA	Clear Accumulator A
	E055	LDAA \$1000	Load A with Content of Memory Location \$1000
	E058	ORAA # 8	OR A with 8H
	E05A	STAA \$1000	Store Content of A in \$1000
	E05D	JMP \$E06C	
OFF2	E060	CLRA	Clear Accumulator A
	E061	LDAA \$1000	Load A with Content of \$1000

	E064	ANDA #F7	AND A with F7H
	E066	STAA \$1000	Store Content of A in \$1000
	E069	JMP \$E06C	
Next 1	E06C	CLRA	Clear Accumulator A
	E06D	LDAA #4	Load A (input R)
	E06F	ANDA \$1000	AND A with port A
	E072	BEQ \$E076	If Result = 0, Jump Test 7
	E074	BNE \$E08A	If Result = 0, Jump Test 9
Test 7	E076	CLRA	Clear Accumulator A
	E077	LDAA #1	Load A (input T)
	E079	ANDA \$1000	AND A with port A
	E07C	BNE \$E080	If Result = 0, Jump Test 8
	E07E	BEQ \$E0A0	If Result = 0, Then OFF 3
Test 8	E080	CLRA	Clear Accumulator A
	E081	LDAA #2	Load A (input L)
	E083	ANDA \$1000	AND A with port A
	E086	BEQ \$E094	If Result = 0, Then Jump Test 8
	E088	BNE \$E0A0	If Result = 0, Then OFF3
Test 9	E08A	CLRA	Clear Accumulator A
	E08B	LDAA #1	Load A (input T)
	E08D	ANDA \$1000	AND A with port A
	E090	BNE \$E0A0	If Result = 0, Then OFF3
	E092	BEQ \$E094	If Result = 0, Then ON3
ON3	E094	CLRA	Clear Accumulator A
	E095	LDAA \$1000	Load A with Content of \$1000
	E098	ORAA #10	OR A with 10H

	E09A	STAA \$1000	Store A in Location \$1000
	E09D	JMP \$E0AC	
OFF3	E0A0	CLRA	Clear Accumulator A
	E0A1	LDAA \$1000	Load A with Content of \$1000
	E0A4	ANDA #EF	AND A with EFH
	E0A6	STAA \$1000	Store A in Location \$1000
	E0A9	JMP \$E0AC	Jump Start
Next 2	EOAC	JMP \$E000	
	E0AF	STOP	STOP

CHAPTER 5

RESULTS, CONCLUSIONS, AND RECOMMENDATIONS

5.1 Results

A technique for analysis and design of the multivariable interactive discrete control system was determined. Based on the technique, a microprocessor-based system was developed for an automobile engine model. Next, the software was developed for the model based on the truth table given in Table 2 and all possible conditions of the input were simulated. The following results were obtained during the simulation of the model for all possible input conditions.

(a) When the input R, L, and T were set low (grounded), all the outputs were low. This outcome met the output state for the first possible input condition shown in Table 2. For the first possible input condition when all the inputs floated, no change in the output was observed.

(b) The input T was set high (5 Vdc) and the other two inputs R and L were kept low (grounded). It was observed that the output A was high and the other two outputs were low. The outcome met the output state for the second possible input condition.

For the second possible input condition when the input T was high but the input R and L were floating, the output F and S reached to 3.5 volts and gradually dropped to zero. However, when the input T was disconnected after the input R and L floated, the output A stayed high (5.02 Vdc).

(c) The input L was set high (5 Vdc) and the other two inputs R₁ and T were kept low (grounded). It was observed that only the output F was high and the other two outputs were low. The outcome met the output state for the third possible input condition.

For the third possible input condition the input R and T were floated while the input L was high and the output A was low. It was observed that the output F and S were 2.62

volt and 2.10 volt respectively. However, when the input L was floated, all the output became low.

(d) When the input R was kept low (grounded), the other two inputs L and T were set high (5 Vdc). It was observed that only the outputs F and S were high. The outcome met the output state for the fourth possible input condition.

For the fourth possible input condition when R was floated while the input L and T were high, the output F and S remained unchanged. When the input L and T were floated after being high while the input R is low, it took 10 seconds for the outputs F and S to drop to zero.

(e) The input R was set high (5 Vdc) and the other two inputs L and T were kept low (grounded). It was observed that the output F and A were high but the output S was low. The outcome met the output state for the fifth possible input condition.

For the fifth possible input condition when the input R was floated after being high while the inputs L and T were grounded, the output A and F remained high. Next both inputs L and T were floated but no change in the state of either one of the outputs was observed.

(f) The input L was set low but the other two inputs were kept high (5 Vdc). It was observed that only the output F was high and the other two outputs were low. The outcome met the output state for the sixth possible input condition.

For the sixth possible input condition when the input R was floated after being high, the output F remained high. When the input T was floated, the output F and A stayed high for two seconds and then dropped to zero.

(g) Both inputs R and L were set high but the input T was kept low. It was observed that only the output A was high. The outcome met the output state for the seventh possible input condition.

For the seventh possible input condition when the input R and L were high and the input T was floated, the outputs remained unchanged. Even when the input R and L were floated, no change in the state of inputs was noticed.

(h) All the inputs R, L, and T were set high (5 Vdc). It was found that all the outputs were low. The outcome met the output state for the eighth possible input condition.

For the eighth possible input condition the inputs R, L, and T were floated one after the other but no change in the state of outputs was observed. All the outputs remained in low state.

(i) It was found that leaving any one of the inputs in the floating state would cause the state of the outputs to be unstable and unpredictable.

(j) A false triggering occurred during the simulation of the model due to the state up of an electric compressor which was in use nearby while the inputs floated. This proved that the state of the outputs could be unpredictable if any of them are floated.

(k) By changing the state of an input for any possible input condition not only changed the state of one output but also affected the state of the other outputs as well. This proved that the system was interactive.

(l) It was found that the I/O function could easily be altered by modifying the program for other tasks. This proved the flexibility of the microprocessor-based system.

(m) The software as well as the hardware functioned properly. No major problem occurred during the simulation of the model.

5.2 Conclusions

There are three major purposes for this study. The objectives and associated major findings are outlined below.

1. Identify a technique for analysis and design of multivariable interactive control systems which have interaction bonds.

Section 3.2 reveals that in the design of multivariable control systems obtaining an adequate process model is crucial. The relative gain array (RGA) method of Bristol is very useful for multivariable process involving interactions.

Section 3.3 identified canonical structures as the most important approach for determining the input/output behavior of the multivariable process.

Section 3.5 explains that the state representation of multivariable systems has several advantages over the transfer matrix notation. These techniques would be applicable if the system is intended to operate in continuous mode. However, since the automobile engine model has a discrete nature, setting up Boolean equations is found to be the most effective technique for solving the multivariable discrete (two state) process with interaction bonds.

2. Apply the technique to an industrial discrete control system model involving multivariable interaction.

Section 4.6.1 presents the algorithm for an automobile engine control system model having three input measured variables and three output controlled variables with interaction between the input and output variables. Based on the algorithm, an I/O truth table is developed for the model in section 4.6.2.

3. Develop a microprocessor-based system to solve the industrial discrete control system model.

Section 4.6.3 presents the software developed for the model based on the MC68HC11 Motorola single-chip microcomputer Opcode Maps. The program was debugged and run. The inputs were simulated with a regulated dc power supply based on the model algorithm given in Table 2 both in sequence and randomly. The output states were monitored on a DVM.

The output states matched expectations listed in Table 2. To design and implement a microprocessor-based control system, the following steps should be taken into consideration.

To design and implement a microprocessor-based control system, the following steps should be taken into consideration.

(a) Analysis of the control problem.

The nature of the control problem needs to be identified. It should be determined whether the control system can be operated in continuous or discrete mode.

(b) Formulation of a potential approach to solve the problem. Proper techniques should be employed to solve the problem. For instance, if a multivariable interaction control process has a continuous nature, then state space approach can be considered as one of the alternatives. If the process is discrete, ON/OFF control approach can be an alternative for implementation of an efficient and inexpensive system.

(c) Implementation of Algorithms (control laws) which simply defines the relationship between input and output variables written in the form of equations.

(d) Selection of a microprocessor-based system. An appropriate microprocessor-based system should be chosen based on required speed and memory capacity to perform a particular task.

(e) Development of software. It is very important to draw a flow chart for any process before attempting to write the program. For some common industrial applications, software packages are available at a reasonable price.

(f) Testing and modifying if necessary.

(g) System interfacing to the real world for the process control.

5.3 Recommendations

Based on this study, the following recommendations are made.

(a) Model the effect of noise disturbances on the digital system.

- (b) Expand the study to include continuous signals, A/D and D/A converters.
- (c) Investigate the state space approach as an alternative to the Bristol array approach.
- (d) Research other models of microprocessor systems.
- (e) Verify the results of the study with actual systems instead of models.

Bibliography

BIBLIOGRAPHY

1. Advance Information on HCMOS Single-Chip Microcomputer, Motorola, 1985.
2. Bateson, Robert, Introduction to Control System Technology, Merrill Publication Co., Ohio, 1980.
3. Bibbero, Robert J.; Stern, David M., Microprocessor Systems Interfacing and Applications, John Wiley and Sons, 1982.
4. Camp, R. C.; Smay, T. A.; Triska, C. J. Microprocessor Systems Engineering, Matrix Publishers, Oregon, 1979.
5. Cannon, Don L.; Luccke G., Understanding Microprocessors, Texas Instruments, Inc., 1986.
6. Considine, Douglas M., Process Instruments and Controls, McGraw-Hill, New York, 1982.
7. Harrison, H. L.; Bollinger, J. C., Automatic Controls, Intext Educational Publishers, New York, 1969.
8. Isermann, Rolf, Digital Control Systems, Springer-Verlag, NY, 1981.
9. Johnson, Curtis D., Microprocessor-Based Process Control, Prentice-Hall, New Jersey, 1984.
10. Kucera, Vladimir, Discrete Linear Control, John Wiley and Sons, Prague, 1979.
11. Kuo, Benjamin C., Automatic Control Systems, Prentice-Hall, N.J., 1982.
12. Lee, Edwin, "Design and Document Microprocessor Systems for Easy Maintenance", Automatic Control Conference, 1977.
13. Leigh, J. R., Applied Digital Control, Prentice-Hall, N.J., 1985.
14. Maloney, Timothy J., Industrial Solid-State Electronics, Prentice-Hall, N.J., 1986.
15. M68HC11 Programmer's Reference Manual, Motorola, 1986.
16. Peatman, John B., Microcomputer-Based Design, McGraw-Hill, N.Y., 1977.
17. Radke, F. "Identification and System Parameter Estimation", Vol. 2, 1977.
18. Singh, Madan G., Elloy, Jean-Pierre, Mezencev, R., Munro, Neil, Applied Industrial Controls, Pergamon Press, N.Y., 1980.
19. Slater, M.; Bronson, B., Practical Microprocessors, Hewlett-Packard Co., 1979.
20. Tao, T. F., Yehoshua, D. Bar, Martinez, R., "Applications of Microprocessors in Control Problems", Automatic Control Conference, 1977.

21. The Staff of Buck Engineering Co., Inc., Microprocessor Concepts and Applications, Lab-Volt, 1983.
22. Tokheim, Roger L., Microprocessor Fundamentals, McGraw-Hill, N.Y., 1983.
23. Wist, A. O.; Meiksin, Z. H., Electronic Design of Microprocessor-Based Instruments and Control Systems, Prentice-Hall, N.J., 1986.

LITERATURE SEARCH

Books

- Alexandridis, Nikitas A., Microprocessor System Design Concept, Computer Science Press, MD, 1984.
- Arnold, James T., Simplified Digital Automation with Microprocessors, Academic Press, 1979.
- Aumiaux, Michel, Microprocessor Systems, Wiley, NY, 1982.
- Auslander, David M., Introducing Systems and Controls, McGraw-Hill, 1974.
- Bibbero, Robert T., Microprocessor Systems, Wiley, NY, 1982.
- Bibbero, Robert J., Microprocessors in Industrial Control, ISA, 1982.
- Bishop, Albert B., Introduction to Discrete Linear Controls, Academic Press, 1975.
- Cahill, S. J., Digital and Microprocessor Engineering, Halsted Press, NY, 1982.
- Chesmond, C. J., Control System Technology, Edward Arnold Publication, London, 1984.
- Considine, Douglas M., Process Instruments and Controls Handbook, McGraw-Hill, 1974.
- Davis, Thomas W., Experimentation with Microprocessor Applications, Reston Publishing Co., VA, 1981.
- D'Azzo, John J., Linear Control System Analysis and Design, McGraw-Hill, NY, 1981.
- Dransfield, Petter, Engineering Systems and Automatic Control, Prentice-Hall, NJ, 1968.
- Elbert, Theodore F., Estimation and Control of Systems, Van Nostrand Reinhold Co., NY, 1984.
- Friedland, Bernard, Control System Design, McGraw-Hill, NY, 1986.
- Gapal, M., Modern Control Theory, Wiley, NY, 1984.
- Garner, K. C., Introduction to Control System Performance Measurements, Pergamon Press, Oxford, 1968.
- Groover, Mikell P., Automation, Production Systems, and Computer-Aided Manufacturing, Prentice-Hall, NJ, 1980.
- Holland, R. C., Microcomputers for Process Control, Pergamon Press, Oxford, 1983.

- Hsu, Jay C., Modern Control Principles and Applications, McGraw-Hill, NY, 1968.
- Isermann, Rolf, Digital Control Systems, Springer-Verlay, Berlin, 1981.
- Jacobites, O. L. R., Introduction to Control Theory, Clarendon Press, Oxford, 1974.
- Karl, J. Astrom, Computer Controlled Systems, Prentice-Hall, NJ, 1984.
- Kucera, Vladimir, Discrete Linear Control, Wiley, NY, 1976.
- Kuo, Bengamin C., Automatic Control Systems, Prentice-Hall, NJ, 1982.
- Lago-Gladwyn Vaile, Control System Theory, Ronald Press, NY, 1962.
- Leigh, J. R., Applied Digital Control, Prentice-Hall, U.K., 1985.
- Lytel, Allan, Digital Computers in Automation, Photofact, 1966.
- Manifold, George O., Automatic Control for Power and Process, McGraw-Hill, NY, 1964.
- Mishkin, Eli Braun, Ludwing J. R., Adaptive Control Systems, McGraw-Hill, 1961.
- Mohler, Ronald R., Bilinear Control Processes, Academic Press, NY, 1973.
- Motorola Semiconductor Product Inc., Microprocessor Applications Manual, McGraw-Hill, NY, 1975.
- Netushil, A., Theory of Automatic Control, Mir Publishers, Moscow, 1978.
- Ogata, Katsuhiko, State Space Analysis of Control Systems, Prentice-Hall, NJ, 1967.
- Phillips, Charles L., Digital Control System Analysis and Design, Prentice-Hall, NJ, 1984.
- Popov, E. P., The Dynamics of Automatic Control Systems, Addison-Wesley Publishing Co., 1962.
- Savant, C. J., Control System Design, McGraw-Hill, NY, 1964.
- Schmitt, Neil M., Understanding Automation Systems, Texas Instruments Publishing, 1984.
- Sensicle, Allan, Introduction to Control Theory for Engineers, Blackie and Sons Press, London, 1968.
- Singh, Madan G., Applied Industrial Control, Pergamon Press, NY, 1980.
- Smardzewski, Richard R., Microprocessor Programming and Applications for Scientists and Engineers, Elsevier Publishing Co., NY, 1984.

- Steckhahn, A. D., Industrial Applications for Microprocessors, Reston Publishing Co., VA, 1982.
- Stout, David F., Microprocessor Applications Handbook, McGraw-Hill, NY, 1982.
- Thomas, John B., Zadeh, Lotfi A., Introduction to Statistical Dynamics of Automatic Control System, Dover Publishing Co., NY, 1960.
- Tredennick, Nick, Microprocessor Logic Design, Digital Press, MA, 1987.

Journals and Conferences

- Bruijn, P. M., Focop: An In-Line Control Package Written in FORTH, Mini and Microcomputers and Their Applications. Proceedings of the ISMM International Symposium, Sant Feliu de Guizols, Spain, 1985.
- Bruijn, P. M., An In-Line Control Package Written in Forth, Microcomputer Applications, USA, Vol. 5, No. 2, 1986.
- Changqiao, L., Corke, P. I., Jamieson, I. D., Anderson, J. H., Simulation and Real-Time Control of Some Dynamical Systems Using Mini-and Microcomputers, Second Conference on Control Engineering 1982, Newcastle, Australia, 1982.
- Cutler, C. R., Dynamic Matrix Control of Imbalanced Systems, ISA Trans., vol. 21, no. 1, 1982.
- Cutler, C. R., Perry, R. T., Real Time Optimization with Multivariable Control is Required to Maximize Profits, Comput & Chem. Eng., vol. 7, no. 5, 1983.
- Evans, F. J., Ioannou, J., Structural Analysis of Decentralized Control Systems, IEE Colloquium on 'The Use of Personal Computers in Control Systems Analysis', London, England, May 23, 1986.
- Frederick, D. K., Draft, R. P., Sadeghi, T., Computer-Aided Control System Analysis and Design Using Interactive Computer Graphics, IEEE Control Syst. Mag., vol. 2, no. 4, December 1982.
- Furuta, K., Hatakeyama, S., Kominami, H., Structural Identification and Software Package for Linear Multivariable Systems, Automatica, vol. 17, no. 5, September 1981.
- Gonzalez de Santos, P., A Software Package for Computer Aided Design of Multivariable Control Systems, Software for Computer Control 1982, Proceedings of the Third IFAC/IFIP Symposium, IFAC, IFIP, Madrid, Spain, 1983.
- Gossman, G. I., Buncombe, A., The Application of A Microprocessor-Based Multivariable Controller to a Gold Milling Circuit, Automation in Mining, Mineral and Metal Processing 1983, Proceedings of the 4th IFAC Symposium IFAC, Helsinki, Finland, 1984.

- Hae-Young-Jung, Won-Kyoo-Lee, Modified Derivative Decoupling Control of Nonlinear Interactive Systems, Control Science and Technology for the Progress of Society, Proceedings of the Eighth Triennial World Congress of the International Federation of Automatic Control, Kyoto, Japan, 1982.
- Hulbert, D. G., Braae, M., Multivariable Control of A Milling Circuit at East Driefontein Gold Mine, Nat. Inst. Metall., Randburg, S. Africa, 1981.
- Jamsa, S. L., Melama H., Penttinen, J., Design and Experimental Evaluation of a Grinding Circuit Control system, Automation in Mining, Mineral and Metal Processing 1983, Proceedings of the 4th IFAC Symposium, IFAC, Helsinki, Finland, August 22-25, 1983.
- Jones, A. H., CAD/CAT of Digital PID Controllers for Multivariable Plants, IEE Colloquium on 'New Developments and Applications of CAD Packages to Control System Design', IEE, London, England, May 15, 1985.
- Kotta, U., On-Line Eigenvector Algorithms for the Identification of Dynamic Systems, Identification and System Parameter Estimation 1982, Proceedings of the Sixth IFAC Symposium, IFAC, Washington, DC, 1983.
- Konor, A. F., Mahesh, J. K., Computer-Aided Engineering of Large-Scale Process Control Systems, On-Line Process Simulation Techniques in Industrial Control Eleventh Annual Advanced Control Conference, Purdue University, 1985.
- Lehtinen, B., Geysler, L. C., AESOP A Computer-Aided Design Program for Linear Multivariable Control Systems, Proceedings of the 1982 American Control Conference, Arlington, VA, 1982.
- MacFarlane, A. G. J., Hung, Y. S., Gains, Phases and Angles (Multivariable Feedback Systems), Bridge Between Control Science and Technology, Proceedings of the Ninth Triennial World Congress of IFAC, Budapest, Hungary, 1985.
- Maciejowski, J. M., Jeanes, S. E., The Cambridge Linear Analysis and Design Programs-CLADP, Proceedings of the 1982 American Control Conference, Arlington, VA, 1982.
- Mensah, S., "Potential Benefits of a CAD Package for Designing Multivariable Control Systems," Canadian Nuclear Society/American Nuclear Society International Conference on Numerical Methods in Nuclear Engineering, Montreal, Que., Canada, 1983.
- Mensah, S., Frketich, G., Mvpack: A Computer-Aided Design Tool for Multivariable Control Systems, Atomic Energy Canada Ltd., Chalk River, Ont., 1985.
- Owens, D. H., Chotai, A., Robust Controller Design for Linear Dynamic Systems Using Approximate Models, IEE Proc. D., vol. 130, no. 2, March 1983.
- Polak, E., Interactive Software for Computer-Aided-Design of Control Systems via Optimization, Proceedings of the 20th IEEE Conference on Decision and Control Including the Symposium on Adaptive Processes, IEEE, San Diego, CA, 1981.

APPENDIX 1

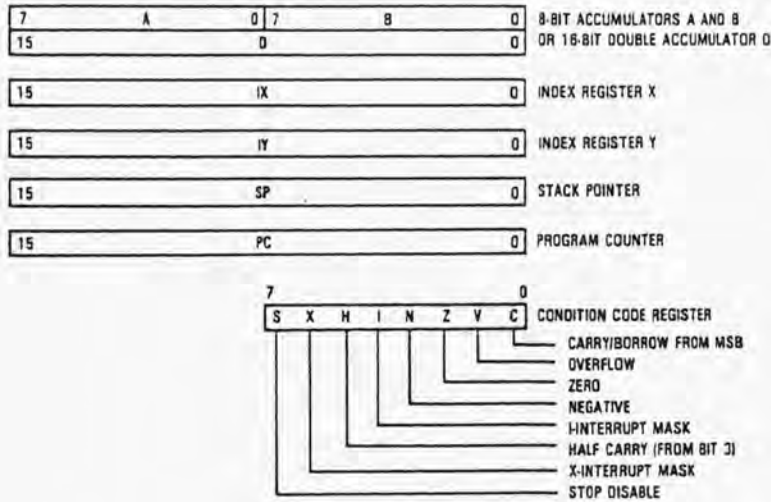
Software Specification for M68HC11 Motorola
Single-Chip Microcomputer

- Polak, E., Optimization-Based Computer-Aided-Design of Control systems, Proceedings of the 1981 Joint Automatic Control Conference, AIChE, ASME, IEEE, ISA, Charlottesville, VA, 1981.
- Polak, E., Siegel, P., Wu, T., Nye, W. T., and Mayne, D. Q., Delight, Mimo: An Interactive, Optimization-Based Multivariable Control System Design Package, IEEE Control Syst. Mag., vol. 2, no. 4, December 1982.
- Rauch, H. E., Automated Synthesis of Control Systems: A Design Approach, Applications of Nonlinear Programming to Optimization and Control, Proceedings of the 4th IFAC Workshop, IFAC, San Francisco, CA, 1984.
- Roberts, G. N., Winch, K. J., Real-Time Digital Simulation of A Gas Turbine Marine Propulsion Plant, Proceedings of the 1985 Summer Computer Simulation Conference, SCS, Chicago, IL, 1985.
- Sadeghi, T., Wozny, M. J., Computer Aided Multivariable Control System Design Package, Proceedings of the 1982 American Control Conference, Arlington, VA, 1982.
- Schafer, R. M., Computer Requirements for Computer Aided Multivariable Control System Design, Proceedings of the Second Annual Workshop on Interactive Computing: CAD/CAM: Electrical Engineering Education, IEEE, CAD/CAM Consortium, Washington, DC, 1983.
- Schafer, R. M., Sain, M. K., Cardiac Approach to system Dominance with Application to Turbofan Engine Models, Conference Record of the Thirteenth Asilomar Conference on Circuits, Systems and Computers, Naval Postgraduate School, Monterey University, Santa Clara, IEEE, 1979.
- Seraji, H., Design of Digital Two-and Three-Term Controllers for Discrete-Time Multivariable Systems, Int. J. Control, vol. 38, no. 4, October 1983.
- Stein, G., Pratt, S., Multivariable Design Tools, AGARD Lecture Series, Multivariable Analysis and Design Techniques, Ankara, Turkey, September 1981.
- van Alste, J. A., Schoute, A. L., Vaartjes, S. R., and Boom, H. B. K., Interactive Control of Isolated Heart Experiments, Computers in Cardiology, Ninth Meeting of Computers in Cardiology, IEEE, Seattle, WA, 1983.
- van der Weiden, A. J. J., Bosgra, O. H., The Analysis of System Properties Relevant for Multivariable Control System Design, J. A., vol 23, no. 1, 1982.
- Wright, S. M., Microsim, A Control System Simulation and Analysis Package for the IBM PC, IEE Colloquium on 'The Use of Personal Computers in Control Systems Analysis', London, England, May 23, 1986.

SECTION 2 REGISTER DESCRIPTIONS

2.1 INTRODUCTION

This section describes the M68HC11 MCU registers that are available to programmers. In addition to being able to execute all M6800 and M6801 MCU instructions, the M68HC11 MCU uses a four-page opcode map to allow execution of 91 new opcodes. Seven registers, described in the following paragraphs, are shown in Figure 2-1. Figure 2-2 illustrates the interrupt stacking order.



1-473

Figure 2-1. Programming Model

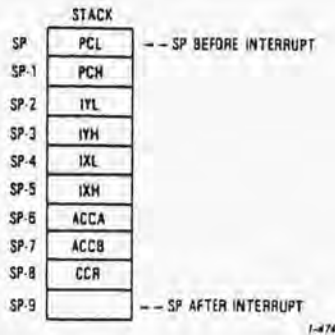


Figure 2-2. Interrupt Stacking Order

2.2 ACCUMULATORS A AND B

Accumulator A and accumulator B are general purpose 8-bit registers used to hold operands and results of arithmetic calculations or data manipulations. These accumulators can be concatenated into a single 16-bit accumulator called the D accumulator.

2.3 INDEX REGISTER X (IX)

The 16-bit index register X is used for indexed mode addressing. It provides a 16-bit indexing value which is added to an 8-bit offset provided in an instruction to create an effective address. The index register X can also be used as a counter or as a temporary storage register.

2.4 INDEX REGISTER Y (IY)

The 16-bit index register Y is also used for indexed mode addressing similar to the index register X; however, all instructions using the index register Y require an extra machine code byte and an extra cycle of execution time since the instructions are two byte opcodes.

2.5 STACK POINTER (SP)

The stack pointer is a 16-bit register that contains the address of the next free location on the stack. The stack is configured as a sequence of last-in-first-out read/write registers which allow important data to be stored during interrupts and subroutine calls. Each time a new byte is added to the stack (push), the stack pointer is decremented; whereas, each time a byte is removed from the stack (pull) the stack pointer is incremented.

2.6 PROGRAM COUNTER (PC)

The program counter is a 16-bit register that contains the address of the next instruction to be executed.

2.7 CONDITION CODE REGISTER (CCR)

The condition code register is an 8-bit register in which each bit signifies the results of the instruction just executed. Each bit can be individually tested by a program and a specific action can be taken as a result of the test. Each condition code register bit is described in the following paragraphs.

2.7.1 Carry/Borrow (C)

The carry/borrow bit is set if there was a carry or borrow out of the arithmetic logic unit (ALU) during the last arithmetic operation. The C bit is also affected during the shift and rotate instructions.

2.7.2 Overflow (V)

The overflow bit is set if there was an arithmetic overflow as a result of the operation; otherwise, the V bit is cleared.

2.7.3 Zero (Z)

The zero bit is set if the result of the last arithmetic, logic, or data manipulation operation was zero; otherwise, the Z bit is cleared.

2.7.4 Negative (N)

The negative bit is set if the result of the last arithmetic, logic, or data manipulation operation was negative; otherwise, the N bit is cleared.

2.7.5 I Interrupt Mask (I)

The interrupt mask bit is set either by hardware or program instruction to disable (mask) all maskable interrupt sources (both external and internal).

2.7.6 Half Carry (H)

The half carry bit is set to a logic one when a carry occurs between bits 3 and 4 of the arithmetic logic unit during an ADD, ABA, or ADC instruction; otherwise, the H bit is cleared.

2.7.7 X Interrupt Mask (X)

The X interrupt mask bit is set only by hardware (RESET or XIRQ acknowledge), and it is cleared only by program instruction (TAP or RTI).

2.7.8 Stop Disable (S)

The stop disable bit is set to disable the STOP instruction, and cleared to enable the STOP instruction. The S bit is program controlled. The STOP instruction is treated as no operation (NOP) if the S bit is set.



Table 3-1. Opcode vs Instruction Cross Reference

Opcode	Operands	Instruction	ADDR Mode	Cycle
00		TEST	INH	—
01		NOP	INH	2
02		IDIV	INH	41
03		FDIV	INH	41
04		LSRD	INH	3
05		ASLD/LSLD	INH	3
06		TAP	INH	2
07		TPA	INH	2
08		INX	INH	3
09		DEX	INH	3
0A		CLV	INH	2
0B		SEV	INH	2
0C		CLC	INH	2
0D		SEC	INH	2
0E		CLI	INH	2
0F		SEI	INH	2
10		SBA	INH	2
11		CBA	INH	2
12	dd mm rr	BRSET (opr) (msk) (rel)	DIR	6
13	dd mm rr	BRCLR (opr) (msk) (rel)	DIR	6
14	dd mm	BSET (opr) (msk)	DIR	6
15	dd mm	BCLR (opr) (msk)	DIR	6
16		TAB	INH	2
17		TBA	INH	2
18		(Page 2 Switch)		
19		DAA	INH	2
1A		(Page 3 Switch)		
1B		ABA	INH	2
1C	ff mm	BSET (opr) (msk)	IND,X	7
1D	ff mm	BCLR (opr) (msk)	IND,X	7
1E	ff mm rr	BRSET (opr) (msk) (rel)	IND,X	7
1F	ff mm rr	BRCLR (opr) (msk) (rel)	IND,X	7
20	rr	BRA (rel)	REL	3
21	rr	BRN (rel)	REL	3
22	rr	BHI (rel)	REL	3
23	rr	BLS (rel)	REL	3
24	rr	BCC/BHS (rel)	REL	3
25	rr	BCS/BLO (rel)	REL	3
26	rr	BNE (rel)	REL	3
27	rr	BEQ (rel)	REL	3

Opcode	Operands	Instruction	ADDR Mode	Cycle
28	rr	BVC (rel)	REL	3
29	rr	BVS (rel)	REL	3
2A	rr	BPL (rel)	REL	3
2B	rr	BMI (rel)	REL	3
2C	rr	BGE (rel)	REL	3
2D	rr	BLT (rel)	REL	3
2E	rr	BGT (rel)	REL	3
2F	rr	BLE (rel)	REL	3
30		TSX	INH	3
31		INS	INH	3
32		PULA	INH	4
33		PULB	INH	4
34		DES	INH	3
35		TXS	INH	3
36		PSHA	INH	3
37		PSHB	INH	3
38		PULX	INH	5
39		RTS	INH	5
3A		ABX	INH	3
3B		RTI	INH	12
3C		PSHX	INH	4
3D		MUL	INH	10
3E		WAI	INH	14
3F		SWI	INH	14
40		NEGA	INH	2
43		COMA	INH	2
44		LSRA	INH	2
46		RORA	INH	2
47		ASRA	INH	2
48		ASLA/LSLA	INH	2
49		ROLA	INH	2
4A		DECA	INH	2
4C		INCA	INH	2
4D		TSTA	INH	2
4F		CLRA	INH	2
50		NEGB	INH	2
53		COMB	INH	2
54		LSRB	INH	2
56		RORB	INH	2
57		ASRB/ASLB	INH	2
58		LSLB	INH	2
59		ROLB	INH	2
5A		DECB	INH	2
5C		INCB	INH	2
5D		TSTB	INH	2
5F		CLRB	INH	2
60	ff	NEG (opr)	IND,X	6
63	ff	COM (opr)	IND,X	6
64	ff	LSR (opr)	IND,X	6
66	ff	ROR (opr)	IND,X	6
67	ff	ASR (opr)	IND,X	6

Table 3-1. Opcode vs Instruction Cross Reference (Continued)

Opcode	Operands	Instruction	ADDR Mode	Cycle
68	ff	ASL/LSL	IND,X	6
69	ff	ROL (opr)	IND,X	6
6A	ff	DEC (opr)	IND,X	6
6C	ff	INC (opr)	IND,X	6
6D	ff	TST (opr)	IND,X	6
6E	ff	JMP (opr)	IND,X	3
6F	ff	CLR (opr)	IND,X	6
70	hh ll	NEG (opr)	EXT	6
73	hh ll	COM (opr)	EXT	6
74	hh ll	LSR (opr)	EXT	6
76	hh ll	ROR (opr)	EXT	6
77	hh ll	ASR (opr)	EXT	6
78	hh ll	ASL/LSL (opr)	EXT	6
79	hh ll	ROL (opr)	EXT	6
7A	hh ll	DEC (opr)	EXT	6
7C	hh ll	INC (opr)	EXT	6
7D	hh ll	TST (opr)	EXT	6
7E	hh ll	JMP (opr)	EXT	3
7F	hh ll	CLR (opr)	EXT	6
80	ii	SUBA (opr)	IMM	2
81	ii	CMPA (opr)	IMM	2
82	ii	SBCA (opr)	IMM	2
83	jj kk	SUBD (opr)	IMM	4
84	ii	ANDA (opr)	IMM	2
85	ii	BITA (opr)	IMM	2
86	ii	LDAA (opr)	IMM	2
88	ii	EORA (opr)	IMM	2
89	ii	ADCA (opr)	IMM	2
8A	ii	ORAA (opr)	IMM	2
8B	ii	ADDA (opr)	IMM	2
8C	jj kk	CPX (opr)	IMM	4
8D	rr	BSR (rel)	REL	6
8E	jj kk	LDS (opr)	IMM	3
8F		XGDX	INH	3
90	dd	SUBA (opr)	DIR	3
91	dd	CMPA (opr)	DIR	3
92	dd	SBCA (opr)	DIR	3
93	dd	SUBD (opr)	DIR	5
94	dd	ANDA (opr)	DIR	3
95	dd	BITA (opr)	DIR	3
96	dd	LDAA (opr)	DIR	3
97	dd	STAA (opr)	DIR	3
98	dd	EORA (opr)	DIR	3
99	dd	ADCA (opr)	DIR	3
9A	dd	ORAA (opr)	DIR	3
9B	dd	ADDA (opr)	DIR	3
9C	dd	CPX (opr)	DIR	5
9D	dd	JSR (opr)	DIR	5
9E	dd	LDS (opr)	DIR	4
9F	dd	STS (opr)	DIR	4
A0	ff	SUBA (opr)	IND,X	4

Opcode	Operands	Instruction	ADDR Mode	Cycle
A1	ff	CMPA (opr)	IND,X	4
A2	ff	SBCA (opr)	IND,X	4
A3	ff	SUBD (opr)	IND,X	6
A4	ff	ANDA (opr)	IND,X	4
A5	ff	BITA (opr)	IND,X	4
A6	ff	LDAA (opr)	IND,X	4
A7	ff	STAA (opr)	IND,X	4
A8	ff	EORA (opr)	IND,X	4
A9	ff	ADCA (opr)	IND,X	4
AA	ff	ORAA (opr)	IND,X	4
AB	ff	ADDA (opr)	IND,X	4
AC	ff	CPX (opr)	IND,X	6
AD	ff	JSR (opr)	IND,X	6
AE	ff	LDS (opr)	IND,X	5
AF	ff	STS (opr)	IND,X	5
B0	hh ll	SUBA (opr)	EXT	4
B1	hh ll	CMPA (opr)	EXT	4
B2	hh ll	SBCA (opr)	EXT	4
B3	hh ll	SUBD (opr)	EXT	6
B4	hh ll	ANDA (opr)	EXT	4
B5	hh ll	BITA (opr)	EXT	4
B6	hh ll	LDAA (opr)	EXT	4
B7	hh ll	STAA (opr)	EXT	4
B8	hh ll	EORA (opr)	EXT	4
B9	hh ll	ADCA (opr)	EXT	4
BA	hh ll	ORAA (opr)	EXT	4
BB	hh ll	ADDA (opr)	EXT	4
BC	hh ll	CPX (opr)	EXT	6
BD	hh ll	JSR (opr)	EXT	6
BE	ii	LDS (opr)	EXT	5
BF	ii	STS (opr)	EXT	5
C0	ii	SUBB (opr)	IMM	2
C1	hh ll	CMPB (opr)	IMM	2
C2	hh ll	SBCB (opr)	IMM	2
C3	jj kk	ADDI (opr)	IMM	4
C4	ii	ANDB (opr)	IMM	2
C5	ii	BITB (opr)	IMM	2
C6	ii	LDAB (opr)	IMM	2
C8	ii	EORB (opr)	IMM	2
C9	ii	ADCB (opr)	IMM	2
CA	ii	ORAB (opr)	IMM	2
CB	ii	ADDB (opr)	IMM	2
CC	jj kk	LDD (opr)	IMM	3
CD		(Page 4 Switch)		
CE	jj kk	LDX (opr)	IMM	3
CF		STOP	INH	2
D0	dd	SUBB (opr)	DIR	3
D1	dd	CMPB (opr)	DIR	3
D2	dd	SBCB (opr)	DIR	3
D3	dd	ADDI (opr)	DIR	5
D4	dd	ANDB (opr)	DIR	3



Table 3-1. Opcode vs Instruction Cross Reference (Continued)

Opcode	Operands	Instruction	ADDR Mode	Cycle
D5	dd	BITB (opr)	DIR	3
D6	dd	LDAB (opr)	DIR	3
D7	dd	STAB (opr)	DIR	3
D8	dd	EORB (opr)	DIR	3
D9	dd	ADCB (opr)	DIR	3
DA	dd	ORAB (opr)	DIR	3
DB	dd	ADDB (opr)	DIR	3
DC	dd	LDD (opr)	DIR	4
DD	dd	STD (opr)	DIR	4
DE	dd	LDX (opr)	DIR	4
DF	dd	STX (opr)	DIR	4
E0	ff	SUBB (opr)	IND,X	4
E1	ff	CMPB (opr)	IND,X	4
E2	ff	SBCB (opr)	IND,X	4
E3	ff	ADDD (opr)	IND,X	6
E4	ff	ANDB (opr)	IND,X	4
E5	ff	BITB (opr)	IND,X	4
E6	ff	LDAB (opr)	IND,X	4
E7	ff	STAB (opr)	IND,X	4
E8	ff	EORB (opr)	IND,X	4
E9	ff	ADCB (opr)	IND,X	4
EA	ff	ORAB (opr)	IND,X	4
EB	ff	ADDB (opr)	IND,X	4
EC	ff	LDD (opr)	IND,X	5
ED	ff	STD (opr)	IND,X	5
EE	ff	LDX (opr)	IND,X	5
EF	ff	STX (opr)	IND,X	5
F0	hh ll	SUBB (opr)	EXT	4
F1	hh ll	CMPB (opr)	EXT	4
F2	hh ll	SBCB (opr)	EXT	4
F3	hh ll	ADDD (opr)	EXT	6
F4	hh ll	ANDB (opr)	EXT	4
F5	hh ll	BITB (opr)	EXT	4
F6	hh ll	LDAB (opr)	EXT	4
F7	hh ll	STAB (opr)	EXT	4
F8	hh ll	EORB (opr)	EXT	4
F9	hh ll	ADCB (opr)	EXT	4
FA	hh ll	ORAB (opr)	EXT	4
FB	hh ll	ADDB (opr)	EXT	4
FC	hh ll	LDD (opr)	EXT	5
FD	hh ll	STD (opr)	EXT	5
FE	hh ll	LDX (opr)	EXT	5
FF	hh ll	STX (opr)	EXT	5
18 08		INY	INH	4
18 09		DEY	INH	4
18 1C	ff mm	BSET (opr) (msk)	IND,Y	8
18 1D	ff mm	BCLR (opr) (msk)	IND,Y	8
18 1E	ff mm rr	BRSET (opr) (msk) (rel)	IND,Y	8

Opcode	Operands	Instruction	ADDR Mode	Cycle
18 1F	ff mm rr	BRCLR (opr) (msk) (rel)	IND,Y	8
18 30		TSY	INH	4
18 35		TYS	INH	4
18 38	ff	PULY	INH	6
18 3A		ABY	INH	4
18 3C		PSHY	INH	5
18 60	ff	NEG (opr)	IND,Y	7
18 63	ff	COM (opr)	IND,Y	7
18 64	ff	LSR (opr)	IND,Y	7
18 66	ff	ROR (opr)	IND,Y	7
18 67	ff	ASR (opr)	IND,Y	7
18 68	ff	ASL/LSL (opr)	IND,Y	7
18 69	ff	ROL (opr)	IND,Y	7
18 6A	ff	DEC (opr)	IND,Y	7
18 6C	ff	INC (opr)	IND,Y	7
18 6D	ff	TST (opr)	IND,Y	7
18 6E	ff	JMP (opr)	IND,Y	4
18 6F	ff	CLR (opr)	IND,Y	7
18 3C	ij kk	CPY (opr)	IMM	5
18 3F		XGDY	INH	4
18 9C	dd	CPY (opr)	DIR	6
18 A0	ff	SUBA (opr)	IND,Y	5
18 A1	ff	CMPA (opr)	IND,Y	5
18 A2	ff	SBCA (opr)	IND,Y	5
18 A3	ff	SUBD (opr)	IND,Y	7
18 A4	ff	ANDA (opr)	IND,Y	5
18 A5	ff	BITA (opr)	IND,Y	5
18 A6	ff	LDAA (opr)	IND,Y	5
18 A7	ff	STAA (opr)	IND,Y	5
18 A8	ff	EORA (opr)	IND,Y	5
18 A9	ff	ADCA (opr)	IND,Y	5
18 AA	ff	ORAA (opr)	IND,Y	5
18 AB	ff	ADDA (opr)	IND,Y	5
18 AC	ff	CPY (opr)	IND,Y	7
18 AD	ff	JSR (opr)	IND,Y	7
18 AE	ff	LDS (opr)	IND,Y	6
18 AF	ff	STS (opr)	IND,Y	6
18 BC	hh ll	CPY (opr)	EXT	7
18 CE	ij kk	LDY (opr)	IMM	4
18 DE	dd	LDY (opr)	DIR	5
18 DF	dd	STY (opr)	DIR	5
18 E0	ff	SUBB (opr)	IND,Y	5
18 E1	ff	CMPB (opr)	IND,Y	5
18 E2	ff	SBCB (opr)	IND,Y	5
18 E3	ff	ADDD (opr)	IND,Y	5
18 E4	ff	ANDB (opr)	IND,Y	5
18 E5	ff	BITB (opr)	IND,Y	5
18 E6	ff	LDAB (opr)	IND,Y	5
18 E7	ff	STAB (opr)	IND,Y	5

Table 3-1. Opcode vs Instruction Cross Reference (Concluded)

Opcode	Operands	Instruction	ADDR Mode	Cycle
18 E8	ff	EORB (opr)	IND,Y	5
18 E9	ff	ADCB (opr)	IND,Y	5
18 EA	ff	ORAB (opr)	IND,Y	5
18 EB	ff	ADDB (opr)	IND,Y	5
18 EC	ff	LDD (opr)	IND,Y	6
18 ED	ff	STD (opr)	IND,Y	6
18 EE	ff	LDY (opr)	IND,Y	6
18 EF	ff	STY (opr)	IND,Y	6
18 FE	hh ll	LDY (opr)	EXT	6
18 FF	hh ll	STY (opr)	EXT	6
1A 83	ij kk	CPD (opr)	IMM	5

Opcode	Operands	Instruction	ADDR Mode	Cycle
1A 93	dd	CPD (opr)	DIR	6
1A A3	ff	CPD (opr)	IND,X	7
1A AC	ff	CPY (opr)	IND,X	7
1A B3	hh ll	CPD (opr)	EXT	7
1A EE	ff	LDY (opr)	IND,X	6
1A EF	ff	STY (opr)	IND,X	6
CD A3	ff	CPD (opr)	IND,Y	7
CD AC	ff	CPX (opr)	IND,Y	7
CD EE	ff	LDX (opr)	IND,Y	6
CD EF	ff	STX (opr)	IND,Y	6

NOTES:

Cycle:

- * = Infinity or until reset occurs
- ** = 12 cycles are used beginning with the opcode fetch. A wait state is entered which remains in effect for an integer number of MPU E-clock cycles (n) until an interrupt is recognized. Finally, two additional cycles are used to fetch the appropriate interrupt vector (total = 14 + n).

Operand(s):

- dd = 8-bit direct address \$0000 - \$00FF. (High byte assumed to be \$00.)
- ff = 8-bit positive offset \$00 (0) to \$FF (255) added to index.
- hh = High order byte of 16-bit extended address.
- ll = One byte of immediate data.
- ij = High order byte of 16-bit immediate data.
- kk = Low order byte of 16-bit immediate data.
- ll = Low order byte of 16-bit extended address.
- mm = 8-bit mask (set bits to be affected).
- rr = Signed relative offset \$80 (-128) to \$7F (+127).
Offset relative to the address following the machine code offset byte.



Table 3-2. Instructions vs. Addressing Mode Cross Reference

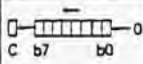
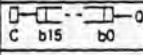
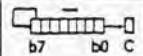
Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		Bytes	Cyc	Condition Codes							
				Opcode	Operand(s)			S	X	H	I	N	Z	V	C
ABA	Add Accumulators	$A + B \rightarrow A$	INH	1B		1	2	-	-	↑	-	↑	↑	↑	↑
ABX	Add B to X	$IX + 00:B \rightarrow IX$	INH	3A		1	3	-	-	-	-	-	-	-	-
ABY	Add B to Y	$IY + 00:B \rightarrow IY$	INH	18 3A		2	4	-	-	-	-	-	-	-	-
ADCA (opr)	Add with Carry to A	$A + M + C \rightarrow A$	A IMM A DIR A EXT A IND,X A IND,Y	89 99 89 A9 18 A9	ii dd hh II ff ff	2 2 3 2 3	2 3 4 4 5	-	-	↑	-	↑	↑	↑	↑
ADCB (opr)	Add with Carry to B	$B + M + C \rightarrow B$	B IMM B DIR B EXT B IND,X B IND,Y	C9 D9 F9 E9 18 E9	ii dd hh II ff ff	2 2 3 2 3	2 3 4 4 5	-	-	↑	-	↑	↑	↑	↑
ADDA (opr)	Add Memory to A	$A + M \rightarrow A$	A IMM A DIR A EXT A IND,X A IND,Y	88 98 88 A8 18 A8	ii dd hh II ff ff	2 2 3 2 3	2 3 4 4 5	-	-	↑	-	↑	↑	↑	↑
ADD8 (opr)	Add Memory to B	$B + M \rightarrow B$	B IMM B DIR B EXT B IND,X B IND,Y	C8 D8 F8 E8 18 E8	ii dd hh II ff ff	2 2 3 2 3	2 3 4 4 5	-	-	↑	-	↑	↑	↑	↑
ADDD (opr)	Add 16-Bit to D	$D + M:M + 1 \rightarrow D$	IMM DIR EXT IND,X IND,Y	C3 D3 F3 E3 18 E3	ii kk dd hh II ff ff	3 2 3 2 3	4 5 6 6 7	-	-	-	-	↑	↑	↑	↑
ANDA (opr)	AND A with Memory	$A * M \rightarrow A$	A IMM A DIR A EXT A IND,X A IND,Y	84 94 84 A4 18 A4	ii dd hh II ff ff	2 2 3 2 3	2 3 4 4 5	-	-	-	-	↑	↑	0	-
ANDB (opr)	AND B with Memory	$B * M \rightarrow B$	B IMM B DIR B EXT B IND,X B IND,Y	C4 D4 F4 E4 18 E4	ii dd hh II ff ff	2 2 3 2 3	2 3 4 4 5	-	-	-	-	↑	↑	0	-
ASL (opr)	Arithmetic Shift Left		EXT IND,X IND,Y	78 68 18 68	hh II ff ff	3 2 3	6 6 7	-	-	-	-	↑	↑	↑	↑
ASLA			A INH	48		1	2								
ASLB			B INH	58		1	2								
ASLD	Arithmetic Shift Left Double		INH	05		1	3	-	-	-	-	↑	↑	↑	↑
ASR (opr)	Arithmetic Shift Right		EXT IND,X IND,Y	77 67 18 67	hh II ff ff	3 2 3	6 6 7	-	-	-	-	↑	↑	↑	↑
ASRA			A INH	47		1	2								
ASRB			B INH	57		1	2								
BCC (rel)	Branch if Carry Clear	$\neg C = 0$	REL	24	rr	2	3	-	-	-	-	-	-	-	-
BCLR (opr) (msk)	Clear Bit(s)	$M * (\overline{mm}) \rightarrow M$	DIR IND,X IND,Y	15 10 18 1D	dd mm ff mm ff mm	3 3 4	6 7 8	-	-	-	-	↑	↑	0	-
BCS (rel)	Branch if Carry Set	$\neg C = 1$	REL	25	rr	2	3	-	-	-	-	-	-	-	-
BEQ (rel)	Branch if = Zero	$\neg Z = 1$	REL	27	rr	2	3	-	-	-	-	-	-	-	-
BGE (rel)	Branch if \geq Zero	$\neg N \oplus V = 0$	REL	2C	rr	2	3	-	-	-	-	-	-	-	-
BGT (rel)	Branch if > Zero	$\neg Z + (N \oplus V) = 0$	REL	2E	rr	2	3	-	-	-	-	-	-	-	-
BHI (rel)	Branch if Higher	$\neg C + Z = 0$	REL	22	rr	2	3	-	-	-	-	-	-	-	-

Table 3-2. Instructions vs. Address Mode Cross Reference (Continued)

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		Bytes	Cycles	Condition Codes								
				Opcode	Operand(s)			S	X	H	I	N	Z	V	C	
BHS (rel)	Branch if Higher or Same	? C = 0	REL	24	rr	2	3	-	-	-	-	-	-	-	-	-
BITA (opr)	Bit(s) Test A with Memory	A * M	A IMM	85	ii	2	2	-	-	-	-	1	1	0	-	
			A DIR	95	dd	2	3	-	-	-	-	-	-	-	-	
			A EXT	85	hh ll	3	4	-	-	-	-	-	-	-	-	
			A IND,X	A5	ff	2	4	-	-	-	-	-	-	-	-	
			A IND,Y	18 A5	ff	3	5	-	-	-	-	-	-	-	-	
BITB (opr)	Bit(s) Test B with Memory	B * M	B IMM	C5	ii	2	2	-	-	-	-	1	1	0	-	
			B DIR	D5	dd	2	3	-	-	-	-	-	-	-	-	
			B EXT	F5	hh ll	3	4	-	-	-	-	-	-	-	-	
			B IND,X	E5	ff	2	4	-	-	-	-	-	-	-	-	
			B IND,Y	18 E5	ff	3	5	-	-	-	-	-	-	-	-	
BLE (rel)	Branch if ≤ Zero	? Z + (N ⊕ V) = 1	REL	2F	rr	2	3	-	-	-	-	-	-	-	-	
BLO (rel)	Branch if Lower	? C = 1	REL	25	rr	2	3	-	-	-	-	-	-	-	-	
BLS (rel)	Branch if Lower or Same	? C + Z = 1	REL	23	rr	2	3	-	-	-	-	-	-	-	-	
BLT (rel)	Branch if < Zero	? N ⊕ V = 1	REL	2D	rr	2	3	-	-	-	-	-	-	-	-	
BMI (rel)	Branch if Minus	? N = 1	REL	28	rr	2	3	-	-	-	-	-	-	-	-	
BNE (rel)	Branch if Not = Zero	? Z = 0	REL	26	rr	2	3	-	-	-	-	-	-	-	-	
BPL (rel)	Branch if Plus	? N = 0	REL	2A	rr	2	3	-	-	-	-	-	-	-	-	
BRA (rel)	Branch Always	? 1 = 1	REL	20	rr	2	3	-	-	-	-	-	-	-	-	
BRCLR(opr) (msk) (rel)	Branch if Bit(s) Clear	? M * mm = 0	DIR	13	dd mm rr	4	6	-	-	-	-	-	-	-	-	
			IND,X	1F	ff mm rr	4	7	-	-	-	-	-	-	-		
			IND,Y	18 1F	ff mm rr	5	8	-	-	-	-	-	-	-		
BRN (rel)	Branch Never	? 1 = 0	REL	21	rr	2	3	-	-	-	-	-	-	-	-	
BRSET(opr) (msk) (rel)	Branch if Bit(s) Set	? (M) * mm = 0	DIR	12	dd mm rr	4	6	-	-	-	-	-	-	-	-	
			IND,X	1E	ff mm rr	4	7	-	-	-	-	-	-	-		
			IND,Y	18 1E	ff mm rr	5	8	-	-	-	-	-	-	-		
BSET(opr) (msk)	Set Bit(s)	M + mm → M	DIR	14	dd mm	3	6	-	-	-	-	1	1	0	-	
			IND,X	1C	ff mm	3	7	-	-	-	-	-	-	-	-	
			IND,Y	18 1C	ff mm	4	8	-	-	-	-	-	-	-	-	
BSR (rel)	Branch to Subroutine	See Special Ops	REL	8D	rr	2	6	-	-	-	-	-	-	-	-	
BVC (rel)	Branch if Overflow Clear	? V = 0	REL	28	rr	2	3	-	-	-	-	-	-	-	-	
BVS (rel)	Branch if Overflow Set	? V = 1	REL	29	rr	2	3	-	-	-	-	-	-	-	-	
CBA	Compare A to B	A - B	INH	11		1	2	-	-	-	-	1	1	1	1	
CLC	Clear Carry Bit	0 → C	INH	0C		1	2	-	-	-	-	-	-	-	0	
CLI	Clear Interrupt Mask	0 → I	INH	0E		1	2	-	-	-	0	-	-	-	-	
CLR (opr)	Clear Memory Byte	0 → M	EXT	7F	hh ll	3	6	-	-	-	-	0	1	0	0	
			IND,X	6F	ff	2	6	-	-	-	-	-	-	-	-	
			IND,Y	18 6F	ff	3	7	-	-	-	-	-	-	-	-	
CLRA	Clear Accumulator A	0 → A	A INH	4F		1	2	-	-	-	-	0	1	0	0	
CLRB	Clear Accumulator B	0 → B	B INH	5F		1	2	-	-	-	-	0	1	0	0	
CLV	Clear Overflow Flag	0 → V	INH	0A		1	2	-	-	-	-	-	-	-	0	
CMPA (opr)	Compare A to Memory	A - M	A IMM	81	ii	2	2	-	-	-	-	1	1	1	1	
			A DIR	91	dd	2	3	-	-	-	-	-	-	-	-	
			A EXT	B1	hh ll	3	4	-	-	-	-	-	-	-	-	
			A IND,X	A1	ff	2	4	-	-	-	-	-	-	-	-	
			A IND,Y	18 A1	ff	3	5	-	-	-	-	-	-	-	-	
CMPB (opr)	Compare B to Memory	B - M	B IMM	C1	ii	2	2	-	-	-	-	1	1	1	1	
			B DIR	D1	dd	2	3	-	-	-	-	-	-	-	-	
			B EXT	F1	hh ll	3	4	-	-	-	-	-	-	-	-	
			B IND,X	E1	ff	2	4	-	-	-	-	-	-	-	-	
			B IND,Y	18 E1	ff	3	5	-	-	-	-	-	-	-	-	
COM (opr)	1's Complement Memory Byte	\$FF - M → M	EXT	73	hh ll	3	6	-	-	-	-	1	1	0	1	
			IND,X	63	ff	2	6	-	-	-	-	-	-	-	-	
			IND,Y	18 63	ff	3	7	-	-	-	-	-	-	-	-	
COMA	1's Complement A	\$FF - A → A	A INH	43		1	2	-	-	-	-	1	1	0	1	
COMB	1's Complement B	\$FF - B → B	B INH	53		1	2	-	-	-	-	1	1	0	1	



Table 3-2. Instructions vs. Addressing Mode Cross Reference (Continued)

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		Bytes	Cycle	Condition Codes							
				Opcode	Operand(s)			S	X	H	I	N	Z	V	C
CPD (opr)	Compare D to Memory 16-Bit	D ← M:M - 1	IMM	1A 83	jj kk	4	5	-	-	-	-				
			DIR	1A 93	dd	3	6								
			EXT	1A B3	hh ll	4	7								
			IND,X	1A A3	ff	3	7								
IND,Y	CD A3	ff	3	7											
CPX (opr)	Compare X to Memory 16-Bit	IX ← M:M - 1	IMM	8C	jj kk	3	4	-	-	-	-				
			DIR	9C	dd	2	5								
			EXT	8C	hh ll	3	6								
			IND,X	AC	ff	2	6								
			IND,Y	CD AC	ff	3	7								
CPY (opr)	Compare Y to Memory 16-Bit	IY ← M:M + 1	IMM	18 8C	jj kk	4	5	-	-	-	-				
			DIR	18 9C	dd	3	6								
			EXT	18 BC	hh ll	4	7								
			IND,X	1A AC	ff	3	7								
			IND,Y	18 AC	ff	3	7								
DAA	Decimal Adjust A	Adjust Sum to BCD	INH	19		1	2	-	-	-	-				
DEC (opr)	Decrement Memory Byte	M - 1 ← M	EXT	7A	hh ll	3	6	-	-	-	-				-
			IND,X	6A	ff	2	6								
			IND,Y	18 6A	ff	3	7								
DECA	Decrement Accumulator A	A - 1 ← A	A INH	4A		1	2	-	-	-	-				-
DECB	Decrement Accumulator B	B - 1 ← B	B INH	5A		1	2	-	-	-	-				-
DES	Decrement Stack Pointer	SP - 1 ← SP	INH	3A		1	3	-	-	-	-	-	-	-	-
DEX	Decrement Index Register X	IX - 1 ← IX	INH	09		1	3	-	-	-	-	-	-	-	-
DEY	Decrement Index Register Y	IY - 1 ← IY	INH	18 09		2	4	-	-	-	-	-	-	-	-
EORA (opr)	Exclusive OR A with Memory	A ⊕ M ← A	A IMM	88	ii	2	2	-	-	-	-			0	-
			A DIR	98	dd	2	3								
			A EXT	88	hh ll	3	4								
			A IND,X	A8	ff	2	4								
			A IND,Y	18 A8	ff	3	5								
EORB (opr)	Exclusive OR B with Memory	B ⊕ M ← B	B IMM	C8	ii	2	2	-	-	-	-			0	-
			B DIR	D8	dd	2	3								
			B EXT	F8	hh ll	3	4								
			B IND,X	E8	ff	2	4								
			B IND,Y	18 E8	ff	3	5								
FDIV	Fractional Divide 16 by 16	D/IX ← IX; r ← D	INH	03		1	41	-	-	-	-				
IDIV	Integer Divide 16 by 16	D/IX ← IX; r ← D	INH	02		1	41	-	-	-	-		0		
INC (opr)	Increment Memory Byte	M + 1 ← M	EXT	7C	hh ll	3	6	-	-	-	-				-
			IND,X	6C	ff	2	6								
			IND,Y	18 6C	ff	3	7								
INCA	Increment Accumulator A	A + 1 ← A	A INH	4C		1	2	-	-	-	-				-
INCB	Increment Accumulator B	B + 1 ← B	B INH	5C		1	2	-	-	-	-				-
INS	Increment Stack Pointer	SP + 1 ← SP	INH	31		1	3	-	-	-	-	-	-	-	-
INX	Increment Index Register X	IX + 1 ← IX	INH	08		1	3	-	-	-	-	-	-	-	-
INY	Increment Index Register Y	IY + 1 ← IY	INH	18 08		2	4	-	-	-	-	-	-	-	-
JMP (opr)	Jump	See Special Ops	EXT	7E	hh ll	3	3	-	-	-	-	-	-	-	-
			IND,X	6E	ff	2	3								
			IND,Y	18 6E	ff	3	4								
JSR (opr)	Jump to Subroutine	See Special Ops	DIR	9D	dd	2	5	-	-	-	-	-	-	-	-
			EXT	BD	hh ll	3	6								
			IND,X	AD	ff	2	6								
			IND,Y	18 AD	ff	3	7								
LDAA (opr)	Load Accumulator A	M ← A	A IMM	86	ii	2	2	-	-	-	-			0	-
			A DIR	96	dd	2	3								
			A EXT	86	hh ll	3	4								
			A IND,X	A6	ff	2	4								
			A IND,Y	18 A6	ff	3	5								

Table 3-2. Instructions vs. Addressing Mode Cross Reference (Continued)

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		Bytes	Cycle	Condition Codes							
				Opcode	Operands			S	X	H	I	N	Z	V	C
LDAB (opr)	Load Accumulator B	$M \rightarrow B$	B IMM	C6	ii	2	2	-	-	-	-	↑	↑	0	-
			B DIR	D6	dd	2	3								
			B EXT	F6	hh ll	3	4								
			B IND,X	E6	ff	2	4								
			B IND,Y	18 E6	ff	3	5								
LDD (opr)	Load Double Accumulator D	$M \rightarrow A, M+1 \rightarrow B$	IMM	CC	jj kk	3	3	-	-	-	-	↑	↑	0	-
			DIR	DC	dd	2	4								
			EXT	FC	hh ll	3	5								
			IND,X	EC	ff	2	5								
			IND,Y	18 EC	ff	3	6								
LDS (opr)	Load Stack Pointer	$M: M+1 \rightarrow SP$	IMM	8E	jj kk	3	3	-	-	-	-	↑	↑	0	-
			DIR	9E	dd	2	4								
			EXT	BE	hh ll	3	5								
			IND,X	AE	ff	2	5								
			IND,Y	18 AE	ff	3	6								
LDX (opr)	Load Index Register X	$M: M+1 \rightarrow IX$	IMM	CE	jj kk	3	3	-	-	-	-	↑	↑	0	-
			DIR	DE	dd	2	4								
			EXT	FE	hh ll	3	5								
			IND,X	EE	ff	2	5								
			IND,Y	CD EE	ff	3	6								
LDY (opr)	Load Index Register Y	$M: M+1 \rightarrow IY$	IMM	18 CE	jj kk	4	4	-	-	-	-	↑	↑	0	-
			DIR	18 DE	dd	3	5								
			EXT	18 FE	hh ll	4	6								
			IND,X	1A EE	ff	3	6								
			IND,Y	18 EE	ff	3	6								
LSL (opr)	Logical Shift Left		EXT	78	hh ll	3	6	-	-	-	-	↑	↑	↑	↑
			IND,X	68	ff	2	6								
			IND,Y	18 68	ff	3	7								
			A INH	48		1	2								
			B INH	58		1	2								
LSLA															
LSLB															
LSLD	Logical Shift Left Double		INH	05		1	3	-	-	-	-	↑	↑	↑	↑
LSR (opr)	Logical Shift Right		EXT	74	hh ll	3	6	-	-	-	-	0	↑	↑	↑
			IND,X	64	ff	2	6								
			IND,Y	18 64	ff	3	7								
			A INH	44		1	2								
			B INH	54		1	2								
LSRA															
LSRB															
LSRD	Logical Shift Right Double		INH	04		1	3	-	-	-	-	0	↑	↑	↑
MUL	Multiply B by B	$A \times B \rightarrow D$	INH	3D		1	10	-	-	-	-	-	-	-	↑
NEG (opr)	2's Complement Memory Byte	$0 - M \rightarrow M$	EXT	70	hh ll	3	6	-	-	-	-	↑	↑	↑	↑
			IND,X	60	ff	2	6								
			IND,Y	18 60	ff	3	7								
NEGA	2's Complement A	$0 - A \rightarrow A$	A INH	40		1	2	-	-	-	-	↑	↑	↑	↑
NEGB	2's Complement B	$0 - B \rightarrow B$	B INH	50		1	2	-	-	-	-	↑	↑	↑	↑
NOP	No Operation	No Operation	INH	01		1	2	-	-	-	-	-	-	-	-
ORAA (opr)	OR Accumulator A (Inclusive)	$A + M \rightarrow A$	A IMM	8A	ii	2	2	-	-	-	-	↑	↑	0	-
			A DIR	9A	dd	2	3								
			A EXT	BA	hh ll	3	4								
			A IND,X	AA	ff	2	4								
			A IND,Y	18 AA	ff	3	5								
ORAB (opr)	OR Accumulator B (Inclusive)	$B + M \rightarrow B$	B IMM	CA	ii	2	2	-	-	-	-	↑	↑	0	-
			B DIR	DA	dd	2	3								
			B EXT	FA	hh ll	3	4								
			B IND,X	EA	ff	2	4								
			B IND,Y	18 EA	ff	3	5								
PSHA	Push A onto Stack	$A \rightarrow Stk, SP = SP - 1$	A INH	35		1	3	-	-	-	-	-	-	-	-

Table 3-2. Instructions vs. Addressing Mode Cross Reference (Continued)

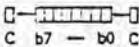
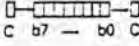
Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		Bytes	Cycle	Condition Codes											
				Opcode	Operand(s)			S	X	H	I	N	Z	V	C				
PSHB	Push B onto Stack	$B \leftarrow Stk, SP = SP - 1$	B INH	37		1	3	-	-	-	-	-	-	-	-	-	-		
PSHX	Push X onto Stack (Lo First)	$IX \leftarrow Stk, SP = SP - 2$	INH	3C		1	4	-	-	-	-	-	-	-	-	-	-		
PSHY	Push Y onto Stack (Lo First)	$IY \leftarrow Stk, SP = SP - 2$	INH	18 3C		2	5	-	-	-	-	-	-	-	-	-	-		
PULA	Pull A from Stack	$SP = SP + 1, A \leftarrow Stk$	A INH	32		1	4	-	-	-	-	-	-	-	-	-	-		
PULB	Pull B from Stack	$SP = SP + 1, B \leftarrow Stk$	B INH	33		1	4	-	-	-	-	-	-	-	-	-	-		
PULX	Pull X from Stack (Hi First)	$SP = SP + 2, IX \leftarrow Stk$	INH	38		1	5	-	-	-	-	-	-	-	-	-	-		
PULY	Pull Y from Stack (Hi First)	$SP = SP + 2, IY \leftarrow Stk$	INH	18 38		2	6	-	-	-	-	-	-	-	-	-	-		
ROL (opr)	Rotate Left 		EXT	79	hh ll	3	6	-	-	-	-	-	-	-	-	-	-		
			IND, X	69	ff	2	6	-	-	-	-	-	-	-	-	-	-	-	
			IND, Y	18 69	ff	3	7	-	-	-	-	-	-	-	-	-	-	-	-
ROLA			A INH	49		1	2	-	-	-	-	-	-	-	-	-	-	-	-
ROLB	B INH		59		1	2	-	-	-	-	-	-	-	-	-	-	-	-	
ROR (opr)	Rotate Right 		EXT	76	hh ll	3	6	-	-	-	-	-	-	-	-	-	-		
			IND, X	66	ff	2	6	-	-	-	-	-	-	-	-	-	-	-	
			IND, Y	18 66	ff	3	7	-	-	-	-	-	-	-	-	-	-	-	-
RORA			A INH	46		1	2	-	-	-	-	-	-	-	-	-	-	-	-
RORB	B INH		56		1	2	-	-	-	-	-	-	-	-	-	-	-	-	
RTI	Return from Interrupt	See Special Ops	INH	38		1	12												
RTS	Return from Subroutine	See Special Ops	INH	39		1	5	-	-	-	-	-	-	-	-	-	-		
SBA	Subtract B from A	$A - B \leftarrow A$	INH	10		1	2	-	-	-	-	-	-	-	-	-	-		
SBCA (opr)	Subtract with Carry from A	$A - M - C \leftarrow A$	A IMM	82	ii	2	2	-	-	-	-	-	-	-	-	-	-		
			A DIR	92	dd	2	3	-	-	-	-	-	-	-	-	-	-	-	
			A EXT	B2	hh ll	3	4	-	-	-	-	-	-	-	-	-	-	-	
			A IND, X	A2	ff	2	4	-	-	-	-	-	-	-	-	-	-	-	-
	A IND, Y		18 A2	ff	3	5	-	-	-	-	-	-	-	-	-	-	-	-	
SBCB (opr)	Subtract with Carry from B	$B - M - C \leftarrow B$	B IMM	C2	ii	2	2	-	-	-	-	-	-	-	-	-	-		
			B DIR	D2	dd	2	3	-	-	-	-	-	-	-	-	-	-	-	
			B EXT	F2	hh ll	3	4	-	-	-	-	-	-	-	-	-	-	-	
			B IND, X	E2	ff	2	4	-	-	-	-	-	-	-	-	-	-	-	-
	B IND, Y		18 E2	ff	3	5	-	-	-	-	-	-	-	-	-	-	-	-	
SEC	Set Carry	$1 \leftarrow C$	INH	0D		1	2	-	-	-	-	-	-	-	-	-	-		
SEI	Set Interrupt Mask	$1 \leftarrow I$	INH	0F		1	2	-	-	-	1	-	-	-	-	-	-		
SEV	Set Overflow Flag	$1 \leftarrow V$	INH	0B		1	2	-	-	-	-	-	-	-	-	-	-		
STAA (opr)	Store Accumulator A	$A \leftarrow M$	A DIR	97	dd	2	3	-	-	-	-	-	-	-	-	-	-		
			A EXT	B7	hh ll	3	4	-	-	-	-	-	-	-	-	-	-	-	
			A IND, X	A7	ff	2	4	-	-	-	-	-	-	-	-	-	-	-	
			A IND, Y		18 A7	ff	3	5	-	-	-	-	-	-	-	-	-	-	-
STAB (opr)	Store Accumulator B	$B \leftarrow M$	B DIR	D7	dd	2	3	-	-	-	-	-	-	-	-	-	-		
			B EXT	F7	hh ll	3	4	-	-	-	-	-	-	-	-	-	-	-	
			B IND, X	E7	ff	2	4	-	-	-	-	-	-	-	-	-	-	-	
			B IND, Y		18 E7	ff	3	5	-	-	-	-	-	-	-	-	-	-	-
STD (opr)	Store Accumulator D	$A \leftarrow M, B \leftarrow M + 1$	DIR	D0	dd	2	4	-	-	-	-	-	-	-	-	-	-		
			EXT	F0	hh ll	3	5	-	-	-	-	-	-	-	-	-	-	-	
			IND, X	E0	ff	2	5	-	-	-	-	-	-	-	-	-	-	-	
			IND, Y		18 E0	ff	3	6	-	-	-	-	-	-	-	-	-	-	-
STOP	Stop Internal Clocks		INH	CF		1	2	-	-	-	-	-	-	-	-	-	-		
STS (opr)	Store Stack Pointer	$SP \leftarrow M: M - 1$	DIR	9F	dd	2	4	-	-	-	-	-	-	-	-	-	-		
			EXT	BF	hh ll	3	5	-	-	-	-	-	-	-	-	-	-	-	
			IND, X	AF	ff	2	5	-	-	-	-	-	-	-	-	-	-	-	
			IND, Y		18 AF	ff	3	6	-	-	-	-	-	-	-	-	-	-	-
STX (opr)	Store Index Register X	$IX \leftarrow M: M + 1$	DIR	DF	dd	2	4	-	-	-	-	-	-	-	-	-	-		
			EXT	FF	hh ll	3	5	-	-	-	-	-	-	-	-	-	-	-	
			IND, X	EF	ff	2	5	-	-	-	-	-	-	-	-	-	-	-	
			IND, Y		CD EF	ff	3	6	-	-	-	-	-	-	-	-	-	-	-
STY (opr)	Store Index Register Y	$IY \leftarrow M: M - 1$	DIR	18 DF	dd	3	5	-	-	-	-	-	-	-	-	-	-		
			EXT	18 FF	hh ll	4	6	-	-	-	-	-	-	-	-	-	-	-	
			IND, X	1A EF	ff	3	6	-	-	-	-	-	-	-	-	-	-	-	
			IND, Y		18 EF	ff	3	6	-	-	-	-	-	-	-	-	-	-	-

Table 3-2. Instructions vs. Addressing Mode Cross Reference (Concluded)

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		Bytes	Cycle	Condition Codes							
				Opcode	Operand(s)			S	X	H	I	N	Z	V	C
SUBA (opr)	Subtract Memory from A	$A - M \rightarrow A$	A IMM	80	ii	2	2	-	-	-	-				
			A DIR	90	dd	2	3	-	-	-	-				
			A EXT	80	hh ll	3	4	-	-	-	-				
			A IND,X	A0	ff	2	4	-	-	-	-				
			A IND,Y	18 A0	ff	3	5	-	-	-	-				
SUBB (opr)	Subtract Memory from B	$B - M \rightarrow B$	B IMM	C0	ii	2	2	-	-	-	-				
			B DIR	D0	dd	2	3	-	-	-	-				
			B EXT	F0	hh ll	3	4	-	-	-	-				
			B IND,X	E0	ff	2	4	-	-	-	-				
			B IND,Y	18 E0	ff	3	5	-	-	-	-				
SUBD (opr)	Subtract Memory from D	$D - M:M - 1 \rightarrow D$	IMM	83	jj kk	3	4	-	-	-	-				
			DIR	93	dd	2	5	-	-	-	-				
			EXT	83	hh ll	3	6	-	-	-	-				
			IND,X	A3	ff	2	6	-	-	-	-				
			IND,Y	18 A3	ff	3	7	-	-	-	-				
SWI	Software Interrupt	See Special Ops	INH	3F		1	14	-	-	-		-	-	-	
TAB	Transfer A to B	$A \rightarrow B$	INH	16		1	2	-	-	-			0	-	
TAP	Transfer A to CC Register	$A \rightarrow CCR$	INH	06		1	2								
TBA	Transfer B to A	$B \rightarrow A$	INH	17		1	2	-	-	-			0	-	
TEST	TEST (Only in Test Modes)	Address Bus Counts	INH	00		1	*	-	-	-	-	-	-	-	
TPA	Transfer CC Register to A	$CCR \rightarrow A$	INH	07		1	2	-	-	-	-	-	-	-	
TST (opr)	Test for Zero or Minus	$M - 0$	EXT	7D	hh ll	3	6	-	-	-	-			0	0
			IND,X	6D	ff	2	6	-	-	-	-			0	0
			IND,Y	18 6D	ff	3	7	-	-	-	-			0	0
TSTA		$A - 0$	A INH	4D		1	2	-	-	-			0	0	
TSTB		$B - 0$	B INH	5D		1	2	-	-	-			0	0	
TSX	Transfer Stack Pointer to X	$SP - 1 \rightarrow IX$	INH	30		1	3	-	-	-	-	-	-	-	
TSY	Transfer Stack Pointer to Y	$SP - 1 \rightarrow IY$	INH	18 30		2	4	-	-	-	-	-	-	-	
TXS	Transfer X to Stack Pointer	$IX - 1 \rightarrow SP$	INH	35		1	3	-	-	-	-	-	-	-	
TYS	Transfer Y to Stack Pointer	$IY - 1 \rightarrow SP$	INH	18 35		2	4	-	-	-	-	-	-	-	
WAI	Wait for Interrupt	Stack Regs & WAIT	INH	3E		2	**	-	-	-	-	-	-	-	
XGDY	Exchange D with X	$IX \rightarrow D, D \rightarrow IX$	INH	8F		1	3	-	-	-	-	-	-	-	
XGDY	Exchange D with Y	$IY \rightarrow D, D \rightarrow IY$	INH	18 8F		2	4	-	-	-	-	-	-	-	

NOTES:

Cycle:

* = Infinity or until reset occurs

** = 12 cycles are used beginning with the opcode fetch. A wait state is entered which remains in effect for an integer number of MPU E-clock cycles (n) until an interrupt is recognized. Finally, two additional cycles are used to fetch the appropriate interrupt vector (total = 14 + n).

Operand(s):

dd = 8-bit direct address \$0000 - \$00FF. (High byte assumed to be \$00.)

ff = 8-bit positive offset \$00 (0) to \$FF (255) added to index.

hh = High order byte of 16-bit extended address.

ii = One byte of immediate data.

jj = High order byte of 16-bit immediate data.

kk = Low order byte of 16-bit immediate data.

ll = Low order byte of 16-bit extended address.

mm = 8-bit mask (set bits to be affected).

rr = Signed relative offset \$80 (-128) to \$7F (+127).

Offset relative to the address following the machine code offset byte.

Condition Codes:

- Bit not changed.

0 Always cleared (logic 0).

1 Always set (logic 1).

| Bit cleared or set depending on operation.

| Bit may be cleared, cannot become set.

SECTION 3 ADDRESSING MODES

3.1 INTRODUCTION

This section describes the M68HC11 MCU addressing modes. Six addressing modes can be used to reference memory; they include: immediate, direct, extended, indexed (with either of two 16-bit index registers and an 8-bit offset), inherent, and relative. Some instructions require an additional byte before the opcode to accommodate a multi-page opcode map; this byte is called a prebyte.

Each of the addressing modes (except inherent) results in an internally generated double byte value referred to as the effective address. This is the resultant value of a statement operand field and is the value that appears on the address bus during the memory reference cycle. The addressing mode is an implicit part of every M68HC11 MCU opcode.

Bit manipulation instructions actually employ two or three addressing modes during execution but are classified by the addressing mode used to access the primary operand. All bit manipulation instructions use immediate address mode to fetch a bit mask and branch versions use relative address mode to determine a branch destination.

The following paragraphs provide a description of each addressing mode and the prebyte instruction. In these descriptions the term effective address is used to indicate the memory address from which the argument is fetched or stored, or from which execution is to proceed.

Also included, after the addressing mode and prebyte instruction descriptions, are opcode map page illustrations and cross-reference tables pertaining to opcodes vs instructions and instructions vs addressing modes. These opcode map illustrations and tables are used for quick cross-referencing purposes during machine code/assembly language programming and debugging operations.

3.2 IMMEDIATE ADDRESSING

In the immediate addressing mode, the actual argument is contained in the byte(s) immediately following the instruction, where the number of bytes matches the size of the register. These are two, three, or four (if prebyte is required) byte instructions.

Machine code byte(s) that follow the operation code are the value of the statement operand field rather than the address of a value. The effective address of the instruction in this case is specified by the character # sign and implicitly points to the byte following the opcode. The immediate value is limited to either one or two bytes depending on the size of the register included in the statement. Examples of several statements which use the immediate addressing mode are shown as follows. Symbols and expressions used in these statements are defined immediately after the examples.

Machine Code		Label	Operation	Operand	Comments
86	16		LDAA	#22	#22-ACCA
C8	34		EORB	#\$34	XOR (\$34,ACCB)
81	24		CMPA	##%100100	CMPA#\$24
		CAT	EQU	7	CAT SAME AS 7
86	07		LDAA	#CAT	7-ACCA
CC	12	34	LDD	#\$1234	
CC	00	07	LDD	#7	7-ACCA:ACCB
86	12		LDAA	#@22	OCTAL
86	41		LDAA	#'A	ASCII
CE	10	00	LDX	#TABLE	ADDR (TABLE)-X

Examine the above machine code and observe the value of each statement operand field appears in byte(s) immediately following the opcode. Note that the operand field for immediate addressing begins with the character # sign. The character # sign is used by the assembler to detect the immediate mode of addressing.

A variety of symbols and expressions can be used following the character # sign. Character prefixes used in the above example are defined as follows:

Prefix	Definition
None	Decimal
\$	Hexadecimal
@	Octal
%	Binary
'	Single ASCII Character

In the last statement of the above example, the immediate bytes consist of the value of the symbol TABLE. The value of any symbol is equal to its address except when used in the label field of an equate (EQU) statement. The value of a symbol that appears in the label field of an EQU directive is defined by the value in the operand field of the statement.

3.3 DIRECT AND EXTENDED ADDRESSING

Direct addressing allows the user to access \$0000 through \$00FF using two byte instructions and execution time is reduced by eliminating the additional memory access. In most applications, this 256-byte area is reserved for frequently referenced data. In the M68HC11 MCU, software can configure the memory map so that internal RAM, and/or internal registers, or external memory space can occupy these addresses.

In the direct addressing mode, the least significant byte of the effective address (operand) is contained in a single byte following the opcode and the most significant byte is assumed to be \$00. The length of most instructions using the direct addressing mode is two bytes: one for the opcode and one for the least significant byte of the effective address.

In the extended addressing mode, the effective address of the instruction appears explicitly in the two bytes following the opcode. Therefore, the length of most instructions using the extended addressing mode is three bytes: one for the opcode and two for the effective address. The second and third bytes (following the opcode) contain the absolute address of the operand. These

are three or four (if prebyte is required) byte instructions: one or two for the opcode, and two for the effective address. Instructions from the second, third, and fourth opcode map pages require a page select prebyte prior to the opcode byte.

Thus, the direct and extended addressing modes differ in two respects: (1) the memory range that can be accessed and (2) the length of the instruction. Using direct addressing, an instruction can reference memory only within the range \$0000-\$00FF, whereas in the extended addressing mode the entire memory space can be accessed.

There are some instructions that provide an extended addressing mode but not a direct mode. These instructions are members of a group called "read-modify-write" instructions (opcodes \$40-\$75 on all opcode pages except JMP and TST) which operate directly on memory, M, and have the following form:

<operation> M-M

The INC, DEC, CLR, and COM instructions are members of this group and each has an extended addressing mode but no direct mode. The following examples show the direct and extended addressing modes.

<u>Machine Code</u>	<u>Label</u>	<u>Operation</u>	<u>Operand</u>	<u>Comments</u>
B3 00 12		SUBD	CAT	FWD REF TO CAT
	CAT	EQU	\$12	DEFINE CAT=\$12
93 12		SUBD	CAT	BKWD REF TO CAT
7F 00 12		CLR	CAT	EXTENDED ONLY



In the above sequence, the first reference to the CAT symbol was a forward reference and the assembler selected the extended addressing mode. The second reference was a backward reference which enabled the assembler to know the symbol value when processing the statement, and the assembler selected the direct addressing mode. The last reference to CAT is also a backward reference to a symbol in the direct area, and the extended addressing mode was selected because the particular instruction does not have a direct addressing mode. Some assemblers allow the direct or extended addressing modes to be forced even when other conditions would suggest the other mode.

3.4 INDEXED ADDRESSING

In the indexed addressing mode, either the X or Y index register is used in calculating the effective address. In this case, the effective address is variable and depends on two factors:

- the current contents of the X or Y index register being used, and
- the 8-bit unsigned offset contained in the instruction.

This addressing mode allows referencing any memory location in the 64K byte address space. These are usually two or three (if prebyte is required) byte instructions, the opcode plus the 8-bit offset.

In microprocessor-based systems, instructions usually reside in read only memory (ROM). Therefore, the offset in the instruction should be considered a static value determined at assembly time rather than during program execution. The use of dynamic single byte offset is

facilitated with the use of the add ACCB to index register (ABX) instruction. More complex address calculations are aided by the 16-bit arithmetic capability of the 16-bit D accumulator and the exchange D with X (XGDY) and exchange D with Y (XGDY) instructions.

If no offset is specified or desired, the instruction will contain \$00 in the offset byte. The offset is an unsigned single byte value that when added to the current value in the index register yields the effective address of the operand leaving the index register unchanged. Because the offset byte is unsigned, a negative offset cannot be specified.

Examples of the indexed addressing mode are shown in the following statements where EA indicates effective address.

<u>Machine Code</u>	<u>Label</u>	<u>Operation</u>	<u>Operand</u>	<u>Comments</u>
E3 00		ADDD	X	EA = (X)
E3 00		ADDD	,X	EA = (X)
E3 00		ADDD	0,X	EA = (X)
E3 04		ADDD	4,X	EA = (X) + 4
	CAT	EQU	7	DEFINE CAT = 7
E3 07		ADDD	CAT,X	EA = (X) + 7
E3 22		ADDD	\$22,X	EA = (X) + \$22
E3 22		ADDD	CAT*8/2 + 6,X	EA = (X) + (CAT*8/2 + 6)

3.5 INHERENT ADDRESSING

In the inherent addressing mode, all of the information to execute the instruction is contained in the opcode. The operands (if any) are registers and no memory reference is required. These are usually one or two byte instructions.

Many M68HC11 MCU instructions do not require an operand because the effective address is inherent within the instruction. For instance, the ABA instruction causes the CPU to add the contents of accumulators A and B and place the result in accumulator A. The instruction INCB causes the contents of accumulator B to be incremented by one. Similarly, the INX instruction causes the index register X to be incremented by one. These three inherent instruction examples, shown in the following statements, do not require an operand and require only a single machine code byte.

<u>Machine Code</u>	<u>Label</u>	<u>Operation</u>	<u>Operand</u>	<u>Comments</u>
1B		ABA		A + B → A
5C		INCB		B + 1 → B
08		INX		X + 1 → X

3.6 RELATIVE ADDRESSING

The relative addressing mode is used for branch instructions. If the branch condition is true, the contents of the 8-bit signed byte following the opcode (offset) is added to the contents of the program counter to form the effective branch address; otherwise, control proceeds to the next instruction. These are usually two byte instructions.

In both the direct and extended addressing modes, the address contained in the operand byte(s) is an absolute numerical address. The relative addressing mode is used only for branch instructions and specifies a location relative to the current value to the program counter. The program counter will always point to the next statement while the addition is being performed. A zero offset byte will result in a no branch instruction regardless of the test involved.

Branch instructions, other than the branching versions of bit manipulation instructions, generate two machine code bytes: one for the opcode and one for the relative offset. Because it is desirable to branch in either direction, the offset byte is a signed two's complement offset with a range of -128 to +127 bytes. The effective branch range must be computed with respect to the address of the next instruction. For branch instructions that consist of two bytes, the next instruction is at PC + 2. If the branch destination address is defined as R, the range is computed as follows:

$$(PC + 2) - 128 \leq R \leq (PC + 2) + 127$$

or

$$PC - 126 \leq R \leq PC + 129$$

The above result indicates that the destination of the branch instruction must be within -126 to +129 memory locations of the first byte of the branch instruction. If it is desired to transfer control beyond this range, then the JMP or JSR instruction must be used. Examples of the relative addressing mode are shown in the following statements.



<u>Machine Code</u>	<u>Label</u>	<u>Operation</u>	<u>Operand</u>	<u>Comments</u>
24 08		BCC	LBCC	L-O-N-G BCC
20 00	THERE	BRA	WHERE	FORWARD BRANCH
22 FC	WHERE	BHI	THERE	BACKWARD BRANCH
27 FE	HANG	BEQ	HANG	BRANCH TO SELF
27 FE		BEQ	*	*MEANS "HERE"
7E 10 00	LBCC	JMP	\$1000	
8D F7		BSR	HANG	

The following are examples of simple, signed, unsigned conditional, and bit manipulation branches.

— SIMPLE BRANCHES —

<u>Mnemonic</u>	<u>Opcode</u>	<u>Cycles</u>
BRA	20	3
BRN	21	3
BSR	8D	7

— SIMPLE CONDITIONAL BRANCHES —

<u>Test</u>	<u>True Opcode</u>	<u>False Opcode</u>
N=1	BMI 2B	BPL 2A
Z=1	BEQ 27	BNE 26
V=1	BVS 29	BVC 28
C=1	BCS 25	BCC 24

— SIGNED CONDITIONAL BRANCHES —

<u>Test</u>	<u>True Opcode</u>	<u>False Opcode</u>
r>m	BGT 2E	BLE 2F
r≥m	BGE 2C	BLT 2D
r=m	BEQ 27	BNE 26
r≤m	BLE 2F	BGT 2E
r<m	BLT 2D	BGE 2C

— UNSIGNED CONDITIONAL BRANCHES —

<u>Test</u>	<u>True Opcode</u>	<u>False Opcode</u>
r>m	BHI 22	BLS 23
r≥m	BHS/BCC 24	BLO/BCS 25
r=m	BEQ 27	BNE 26
r≤m	BLS 23	BHI 22
r<m	BLO/BCS 25	BHS/BCC 24

— BIT MANIPULATION BRANCHES —

BRCLR — Branch if all selected bits are clear
 (opcode) (operand addr) (mask) (rel offset)
 M*mm = 0? M = operand in memory; mm = mask

BRSET — Branch if all selected bits are set
 (opcode) (operand addr) (rel offset)
 (M)*mm = 0? M = operand in memory; mm = mask

3.7 PREBYTE

In order to expand the number of instructions used in the MC68HC11 MCU, a prebyte instruction has been added to certain instructions. The instructions affected are usually associated with the Y index register. Instructions which do not require a prebyte reside in the opcode map page 1. Instructions requiring a prebyte reside in the opcode map pages 2 through 4. The opcode map prebyte assignment is \$18 for page 2, \$1A for page 3, and \$CD for page 4. Figures 3-1 through 3-4 illustrate opcode map page 1 through 4, respectively.

The opcode map pages illustrate the instruction set vs opcode relationships and can be used during logic analyzer debugging operations. From a binary logic analyzer trace, machine code bytes can be reverse assembled to yield assembly language mnemonics to aid in the debugging operation. First a machine code byte is broken into four bit halves. The higher order half identifies a column in the opcode map and the low order half then identifies the line within that column where the assembly language mnemonic can be read.

Table 3-1 provides the opcode vs instruction cross-reference listing which is useful for machine code reverse assembly. Some users will find this table easier to use than the opcode map pages. In addition to showing the assembly language mnemonic and addressing mode, this table also lists operand construction details and gives the total number of E cycles required to execute the instruction. Table 3-1 is organized by opcode, operands, instruction, number of cycles, and addressing mode.

Table 3-2 provides the instruction vs addressing mode cross-reference listing which is useful for hand assembly of machine code or as a condensed summary of important instruction set details. For hand assembly the user would write out a program using source instruction mnemonics and notations. Then each mnemonic would be looked up in Table 3-2 to translate the mnemonic into the appropriate opcode taking into account the desired addressing mode. Table 3-2 is organized by instruction (source form), operation, Boolean expression, addressing mode for operand, machine coding (opcode and operand), number of bytes, number of cycles, and condition code register bit states.



										ACCA				ACCB				
		INH	INH	REL	INH	ACCA	ACCB	IND,X	EXT	IMM	DIR	IND,X	EXT	IMM	DIR	IND,X	EXT	
MSB		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
LSB		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0	TEST*	SBA	BRA	TSX	NEG				SUB								0
0001	1	NOP	CBA	BRN	INS					CMP								1
0010	2	IDIV	BRSET	BH	PULA					SBC								2
0011	3	FDIV	BRCLR	BLS	PULB	COM				SUB0				A000				3
0100	4	LSRD	BSET	BCC	DES	LSR				AND								4
0101	5	ASLD	BCLR	BCS	TXS					BIT								5
0110	6	TAP	TAB	BNE	PSHA	ROR				LDA								6
0111	7	TPA	TBA	BEO	PSHB	ASR				STA		STA				7		
1000	8	INX	PAGE2	BVC	PULX	ASL				EOR								8
1001	9	DEX	DAA	BVS	RTS	ROL				ADC								9
1010	A	CLV	PAGE3	BPL	ABX	DEC				ORA								A
1011	B	SEV	ABA	BMI	RTI					ADD								B
1100	C	CLC	BSET	BGE	PSHX	INC				CPX				LOO				C
1101	D	SEC	BCLR	BLT	MUL	TST				BSR	JSR		PAGE4	STD			D	
1110	E	CLI	BRSET	BGT	WAI	JMP				LDS				LDX				E
1111	F	SEI	BRCLR	BLE	SWI	CLR				XGOX	STS		STOP	STX			F	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

* Test instruction executable only in test mode.

Figure 3-1. Opcode Map Page 1

										ACCA				ACCB				
		INH			INH			IND,Y		IMM	DIR	IND,Y	EXT	IMM	DIR	IND,Y	EXT	
MSB		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
LSB		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0				TSY			NEG		SUB				SUB				0
0001	1									CMP				CMP				1
0010	2									SBC				SBC				2
0011	3							COM		SUB0				A000				3
0100	4							LSR		AND				AND				4
0101	5				TYS					BIT				BIT				5
0110	6							ROR		LDA				LDA				6
0111	7							ASR		STA				STA				7
1000	8	INY			PULY			ASL		EOR				EOR				8
1001	9	OEY						ROL		ADC				ADC				9
1010	A				ABY			DEC		ORA				ORA				A
1011	B									ADD				ADD				B
1100	C		BSET		PSHY			INC		CPY				LDD				C
1101	D		BCLR					TST		JSR				STD				D
1110	E		BRSET					JMP		LDS				LDY				E
1111	F		BRCLR					CLR		XGOY	STS		STY				F	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Figure 3-2. Opcode Map Page 2 (18xx)

										ACCA				ACCB			
										IMM	DIR	IND.X	EXT	IND.X			
MSB	LSB	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0																
0001	1																0
0010	2																1
0011	3										CPD						2
0100	4																3
0101	5																4
0110	6																5
0111	7																6
1000	8																7
1001	9																8
1010	A																9
1011	B																A
1100	C										CPY						B
1101	D																C
1110	E															LDY	D
1111	F															STY	E
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Figure 3-3. Opcode Map Page 3 (1Axx)

										ACCA				ACCB			
												IND.Y		IND.X			
MSB	LSB	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0																
0001	1																0
0010	2																1
0011	3										CPD						2
0100	4																3
0101	5																4
0110	6																5
0111	7																6
1000	8																7
1001	9																8
1010	A																9
1011	B																A
1100	C										CPY						B
1101	D																C
1110	E															LDX	D
1111	F															STX	E
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Figure 3-4. Opcode Map Page 4 (CDxx)