



**VARIAN 620
TRAINING MANUAL**



varian data machines/a varian subsidiary

© 1972



varian data machines

98 A 9902 504

JANUARY 1973

This manual is intended for training purposes only. For detailed information, refer to the applicable document for the system you are using.



CONTENTS

TABLE OF CONTENTS

CHAPTER I COMPUTER FUNDAMENTALS

Section 1	IntroductionI-1
1.1	General DescriptionI-1
1.2	Computer ConceptsI-2
1.2.1	Central Processing Unit (CPU)I-2
1.2.2	MemoryI-3
1.2.3	Input UnitI-3
1.2.4	Output UnitI-3
Section 2	Numerical AnalysisI-4
2.1	IntroductionI-4
2.2	Square Root ExtractionI-4

CHAPTER II PROGRAMMING

Section 1	IntroductionII-1
1.1	Communication With a ComputerII-1
1.2	MnemonicsII-1
1.3	Numerical CodeII-1
1.4	Instruction SetII-2
1.5	Preparing the ProgramII-2
Section 2	Flow-ChartingII-3
2.1	IntroductionII-3
2.2	NotationII-6
2.3	SymbolsII-7
2.3.1	FunctionsII-7
2.3.2	DecisionsII-8
2.3.3	Input/OutputII-9
2.3.4	Start and StopII-10
2.3.5	Fixed ConnectorsII-10



CONTENTS

2.4	A Simple Flow Chart	II-11
2.4.1	Substitution	II-12
2.4.2	Subroutines	II-13
2.4.3	Assertions	II-14
2.5	A More Complicated Flow Chart	II-14
Section 3	Machine Language Preparation	II-17
3.1	Introduction	II-17
3.2	Instruction Repertoire	II-17
3.2.1	Instruction Types	II-18
3.2.2	Addresses	II-18
3.2.3	Codes	II-18
3.3	Sample Programs	II-19
3.4	Machine Language	II-30
3.5	Looping	II-30
3.6	Indexing	II-30
3.6.1	Specifying the Index Register	II-33
3.6.2	An Example of Indexing	II-33
3.6.3	Address Modification by Indexing	II-36
3.7	Subroutines	II-36
3.8	Coding	II-37
Section 4	Programming in Assembly Language	II-39
4.1	DAS Assembler	II-39
4.1.1	DAS 4KA	II-39
4.1.2	DAS 8KA	II-40
4.1.3	DAS MR	II-40
4.1.4	Stand-Alone MR	II-40
4.2	DAS Source Language	II-40
4.3	Statements	II-41
4.3.1	Statement Format	II-41
4.3.2	Label Field	II-41
4.3.3	Operation Field	II-41
4.3.4	Variable Field	II-42
4.3.5	Comment Field	II-42
4.3.6	Comment Statements	II-42
4.3.7	Blank Statements	II-42
4.4	Programming in Symbolic Assembly Language	II-43



CONTENTS

CHAPTER III COMPUTER OPERATION

Section 1	Word Formats	III-1
1.1	Introduction.....	III-1
1.2	Single-Word Instructions.....	III-2
1.2.1	Addressable.....	III-2
1.2.2	Nonaddressable.....	III-3
1.3	Two-Word Instructions.....	III-4
1.3.1	Jump, Jump and Mark, and Execute Instructions.....	III-4
1.3.2	Memory In/Out Instructions.....	III-5
1.3.3	Immediate Instructions.....	III-5
1.4	MACRO-Instructions.....	III-6
1.5	Instruction List.....	III-6
Section 2	Paper Tape Formats	III-8
2.1	Source Tape Format.....	III-8
2.2	Bootstrap Format.....	III-8
2.3	Binary Object (Program Object) Format.....	III-8
2.4	MOS Relocatable Object Format.....	III-12
Section 3	Operating Sequences for 620/i, 620/L	III-14
3.1	Access Operand in Memory.....	III-14
3.2	Store Operand in Memory.....	III-17
3.3	Indirect Operand Access.....	III-17
Section 4	Computer Failure	III-19
4.1	Errors.....	III-19
4.2	Mistakes.....	III-20
4.3	Malfunctions.....	III-21
4.3.1	Diagnostic Routines for Corrective Maintenance.....	III-21
4.3.2	Diagnostic Routines for Preventive Maintenance.....	III-21

CHAPTER IV 620 COMPUTER SYSTEMS

Section 1	620/i and 620/L Systems	IV-1
1.1	Introduction.....	IV-1



CONTENTS

1.2	Switches and Indicators.....	IV-6
1.2.1	Displays.....	IV-6
1.2.2	Controls.....	IV-8
1.3	Manual Operation.....	IV-10
1.3.1	Power Control.....	IV-10
1.3.2	Manual Program Entry and Execution.....	IV-11
1.3.3	Instruction Repeat.....	IV-11
1.3.4	SENSE Switches.....	IV-12
1.4	Organization.....	IV-12
1.4.1	Memory.....	IV-12
1.4.2	Control.....	IV-14
1.4.3	Arithmetic/Logic.....	IV-14
1.4.4	Input/Output.....	IV-15
1.4.5	Bus Structure.....	IV-15
1.5	Timing.....	IV-16
1.5.1	Clocks.....	IV-16
1.5.2	Clock Modifiers.....	IV-18
1.5.3	Sequence Control.....	IV-19
1.6	Information Transfer.....	IV-21
1.6.1	P Register to Memory.....	IV-21
1.6.2	Memory to U Register.....	IV-21
1.6.3	U Register to Memory.....	IV-21
1.6.4	Memory to R Register.....	IV-22
1.6.5	Adder to Operation Registers.....	IV-22
1.6.6	Operation Registers to Memory.....	IV-22
1.6.7	Memory to Operation Registers.....	IV-22
1.6.8	Input to Memory.....	IV-22
1.6.9	Output from Memory.....	IV-23
1.6.10	Input to Operation Registers.....	IV-23
1.6.11	Output from Operation Registers.....	IV-23
1.6.12	Operation Register to Operation Register.....	IV-23
1.7	Decoding.....	IV-23
1.7.1	Operation Code Decoding.....	IV-24
1.7.2	M Field Decoding.....	IV-24
Section 2	620/f, 620/f-100 System.....	IV-30
2.1	Introduction.....	IV-30



CONTENTS

2.2	Switches and Indicators	IV-37
2.2.1	Power Switch	IV-37
2.2.2	STEP/RUN Switch and STEP and RUN Indicators	IV-39
2.2.3	BOOTSTRAP Switch	IV-39
2.2.4	START Switch	IV-40
2.2.5	REGISTER Switches	IV-40
2.2.6	Register Entry Switches and Display Indicators	IV-40
2.2.7	LOAD Switch	IV-41
2.2.8	REPEAT Switch	IV-42
2.2.9	SENSE Switches	IV-42
2.2.10	INT (Interrupt) Switch	IV-42
2.2.11	RESET Switch	IV-43
2.2.12	OVFL (Overflow) Indicator	IV-43
2.2.13	ALARM Indicator	IV-43
2.3	Manual Operation	IV-43
2.3.1	Loading Into Sequential Memory Addresses	IV-43
2.3.2	Displaying From Sequential Memory Addresses	IV-44
2.3.3	Manual Execution of Stored Programs	IV-44
2.3.4	Manual Repetition of Instructions	IV-45
2.4	Organization	IV-45
2.4.1	Control Section	IV-45
2.4.2	Decoding Section	IV-45
2.4.3	Arithmetic Unit	IV-45
2.4.4	Operation Registers	IV-47
2.4.5	Auxiliary Registers	IV-47
2.4.6	Data Switch Section	IV-48
2.4.7	Register Entry Switches/Display Indicators	IV-48
2.4.8	Shift-and-Rotate Circuit	IV-48
2.4.9	Internal Buses	IV-48
2.5	Timing	IV-49
2.5.1	Clocks	IV-49
2.5.2	Clock Modifiers	IV-51
2.5.3	Sequence Control	IV-52
2.6	Information Transfer	IV-52
2.6.1	P Register to Memory	IV-53
2.6.2	Memory to I Register	IV-53
2.6.3	I Register to Memory	IV-53



CONTENTS

2.6.4	Memory to R Register	IV-55
2.6.5	Arithmetic Unit to Operation Registers	IV-55
2.6.6	Operation Registers to Memory	IV-55
2.6.7	Memory to Operation Registers	IV-56
2.6.8	Input to Memory	IV-56
2.6.9	Output from Memory	IV-56
2.6.10	Input to Operation Registers	IV-56
2.6.11	Output from Operation Registers	IV-56
2.6.12	Operation Register to Operation Register	IV-56
Section 3	620/L-100 Systems	IV-57
3.1	Introduction	IV-57
3.2	System Operation	IV-62
3.3	Manual Operations	IV-66
3.4	Central Processing Unit	IV-69

CHAPTER V LOGIC DESCRIPTIONS



CONTENTS

LIST OF ILLUSTRATIONS

I-1	Typical Computer System.....	I-2
II-1	Typical Flow Chart.....	II-4
II-2	Flow Chart Symbols.....	II-5
II-3	Flow Chart for $T = AX^2 + BX \quad X \sin 0$	II-15
II-4	Flow Chart for a Positive and Negative Number Count.....	II-25
II-5	Count of Positive Numbers.....	II-31
II-6	Loop Program.....	II-32
II-7	Add a Table of Three Numbers.....	II-34
II-8	Subroutines	II-38
II-9	Example I, Coding Form	II-44
II-10	Example I, Assembly Listing.....	II-46
II-11	Example J, Coding Form.....	II-48
II-12	Example J, Assembly Listing.....	II-51
II-13	Example K, Coding Form	II-55
II-14	Example K, Assembly Listing.....	II-56
II-15	Example L, Coding Form.....	II-58
II-16	Example L, Assembly Listing.....	II-59
III-1	Formats for Data Words and Indirect Addresses	III-1
III-2	Single-Word Instruction Format.....	III-2
III-3	Single-Word Nonaddressable Instructions	III-3
III-4	Two-Word Instruction Format.....	III-4
III-5	Immediate Instruction Format.....	III-5
III-6	MACRO-Command Format	III-6
III-7	620 Series Instruction List	III-7
III-8	Source Tape Format.....	III-9
III-9	Bootstrap Format.....	III-10
III-10	Binary Object Format.....	III-11
III-11	MOS Relocatable Object Format.....	III-13
III-12	Operand Access from Memory Sequence.....	III-15
III-13	Operand Storage in Memory Sequence.....	III-16
III-14	Indirect Operand Access Sequence.....	III-18



CONTENTS

IV-1	620/i Outline.....	IV-6
IV-2	620/i Control Console.....	IV-7
IV-3	620/L Control Console.....	IV-7
IV-4	620/i Organization.....	IV-13
IV-5	Basic Timing Clocks.....	IV-17
IV-6	Example of a Modified Clock Sequence.....	IV-10
IV-7	Data 620/L Organization.....	IV-21
IV-8	620/f Computer Control Panel.....	IV-38
IV-9	620/f Computer Functional Organization.....	IV-46
IV-10	Basic Timing Clocks.....	IV-50
IV-11	Example of a Modified Clock Sequence.....	IV-53
IV-12	620/f Organization.....	IV-54
IV-13	Varian 620/L-100 Mainframe.....	IV-61
IV-14	Varian 620/L-100 Control Panel.....	IV-63
IV-15	Varian 620/L-100 Computer Organization.....	IV-70
IV-16	Basic Clock Waveforms.....	IV-79
IV-17	Example of a Modified Clock Sequence.....	IV-81
IV-18	Accessing an Operand in Memory.....	IV-82
IV-19	Storing an Operand in Memory.....	IV-83
IV-20	Accessing an Operand Indirectly.....	IV-85
V-1	Quadruple 2-Input NAND Gate (SN7400N, 7400PC, N7400A).....	V-2
V-2	Quadruple 2-Input Positive NOR Gate (SN7402N, MC7402P, 7402PC).....	V-2
V-3	Quadruple 2-Input Positive NAND Gate (Open Collector) (SN7403N).....	V-3
V-4	Hex Inverters (SN7404N, MC7404P, N7404A, 7404PC).....	V-3
V-5	Hex Inverter with Open-Collector Circuit (SN7405J, MC7405L).....	V-4
V-6	Triple 3-Input Positive NAND Gate (SN7410N, 7410PC, N7410A).....	V-4
V-7	Dual 4-Input Positive NAND Gate (SN7420N, MC7420, 7420PC).....	V-5
V-8	Dual 4-Input Positive NAND Buffer (SN7440N, MC7440L, 7440DC).....	V-6
V-9	Quadruple 2-Input Positive NAND Gate (SN74H00N, MC3000P).....	V-7
V-10	Quadruple 2-Input Positive NAND Gate with Open Collector (SN74H01N, MC3004P).....	V-7
V-11	Hex Inverter (SN74H04N, MC30018).....	V-8
V-12	Hex Inverter with Open-Collector Output (SN74H05N).....	V-8
V-13	Triple 3-Input Positive NAND Gate (SN74H10N).....	V-9
V-14	Triple 3-Input Positive AND Gate (SN74H11N).....	V-9
V-15	Dual 4-Input Positive NAND Gate (SN74H20N, MC3010).....	V-10



CONTENTS

V-16	Dual 4-Input Positive AND Gate (SN74H21N, MC3011).....	V-11
V-17	Dual 4-Input Positive NAND Gate (SN74H22N).....	V-12
V-18	8-Input Positive NAND Gate (SN74H30N).....	V-12
V-19	Dual 4-Input Positive NAND Buffer (SN74H40N, MC3024P).....	V-13
V-20	Dual 2-Wide 2-Input AND-OR-Invert Gates (SN74H50 and 51N, MC3020 and 3023).....	V-14
V-21	Expandable 2-2-2-3-Input AND-OR Gate (SN74H52N, MC3031P).....	V-15
V-22	Expandable 2-2-2-3-Input AND-OR Invert Gate (SN74H53N, MC3032).....	V-16
V-23	3-2-2-3-Input AND-OR Expander (SN74H62N, MC3018P).....	V-17
V-24	J-K Master-Slave Flip-Flop (SN7472N).....	V-18
V-25	J-K Master-Slave Flip-Flop (SN74H72N).....	V-19
V-26	Dual J-K Master-Slave Flip-Flops (SN7473 and 74107N).....	V-20
V-27	Dual J-K Master-Slave Flip-Flops (SN74H73N).....	V-21
V-28	Dual D-Type Edge-Triggered Flip-Flop (SN7474N).....	V-22
V-29	Dual D-Type Edge-Triggered Flip-Flop (SN74H74N).....	V-23
V-30	4-Bit Binary Counter (SN7493N).....	V-24
V-31	4-Bit Right-Shift Left-Shift Register (SN7495N).....	V-25
V-32	Dual J-K Edge-Triggered Flip-Flop (SN74H108N).....	V-26
V-33	Synchronous 4-Bit Up/Down Counter (SN74193J).....	V-27
V-34	Data Selector/Multiplexor (SN74150N).....	V-28
V-35	Arithmetic Logic Unit/Function Generator (SN74181N).....	V-29
V-36	Look-Ahead Carry Generator (SN74182N).....	V-30
V-37	Gated Full Adder (SN7480N).....	V-31
V-38	256-Bit Read-Only Memory (SN7488N).....	V-33
V-39	High-Speed Buffer Memory/Register File (SN74170N).....	V-35
V-40	Quadruple Bistable Latch (SN7475N).....	V-37
V-41	Quadruple 2-Input NAND Gate (SN15846N).....	V-38
V-42	Triple 3-Input NAND Gate (SN15862N).....	V-38
V-43	Dual 4-Input NAND Power Gate (SN6006N).....	V-39
V-44	Pulse-Triggered Binary (SN15850N).....	V-40
V-45	Monostable Multivibrator (SN15851N).....	V-41
V-46	Retriggerable Monostable Multivibrator (Fairchild U6A960159X).....	V-42
V-47	Quadruple 2-Input AND Gate (MC3001P).....	V-42
V-48	Quadruple 2-Input NOR Gate (MC3002P).....	V-43
V-49	Dual Sense Amps.....	V-43
V-50	Binary to Octal Converter (MC4006P).....	V-44
V-51	3-Line to 8-Line Decoders.....	V-45



CONTENTS

LIST OF TABLES

IV-1	620/i and 620/L Specifications.....	IV-2
IV-2	Controls and Indicators.....	IV-8
IV-3	Basic Timing Clock.....	IV-18
IV-4	Instruction Storage in U Register.....	IV-26
IV-5	Operation Code Classes.....	IV-27
IV-6	Operation Code Sets.....	IV-27
IV-7	Operation Code Groups.....	IV-28
IV-8	M Field Decoding.....	IV-28
IV-9	620/f Specifications.....	IV-31
IV-10	Basic Timing Clocks.....	IV-51
IV-11	620/L-100 Specifications.....	IV-58
IV-12	Bootstrap Loader Routines.....	IV-67
IV-13	Varian 620/L-100 System Clocks.....	IV-78



CHAPTER I

COMPUTER FUNDAMENTALS



varian data machines



CHAPTER I COMPUTER FUNDAMENTALS

SECTION I INTRODUCTION

1.1 GENERAL DESCRIPTION

Digital computing devices are not new. The first machine to employ some of the principles of modern computers was invented by Charles Babbage of England in 1822. His device, which he called the analytical engine, was a steam-driven assembly of gears that automatically computed and printed tables. The analytical engine incorporated three elements used in computers today:

- a. Storage (memory for holding information)
- b. A mill (arithmetic unit), including the machinery for making decisions, to work on the information
- c. A control to govern the mill automatically and call for the next piece of information in sequence when required

About the same time, George Boole, an English mathematician, was laying the foundations of logical algebra. Boolean algebra is the cornerstone of computer logic circuit design.

The era of the modern computer began in 1937 with the Mark 1, an automatic sequence-controlled calculator. Present electronic computers retain the same operational principles as earlier machines. The great advances in computer technology have been in the fields of circuit design and new components.

The digital computer is not a brain, but merely a machine that must be given precise instructions on what and how to perform. What the human lacks in lightning speed and unerring memory, the computer has in abundance. What the computer lacks in the ability to reason, analyze, deduce, organize, and plan, man can supply. The computer is insensitive to human emotion. When the button is pressed, the computer goes all the way, given correct instructions, power, and data input. But, for all its merits, the computer is useless if humans have not analyzed and prepared the problem for the machine and told the computer exactly what it must do and when, in a language understandable to the computer. The computer does not need human inspiration, but it must have the explicit direction and control that can come only from the human brain. Man's role in solving this problem is to program the computer.



CHAPTER I COMPUTER FUNDAMENTALS

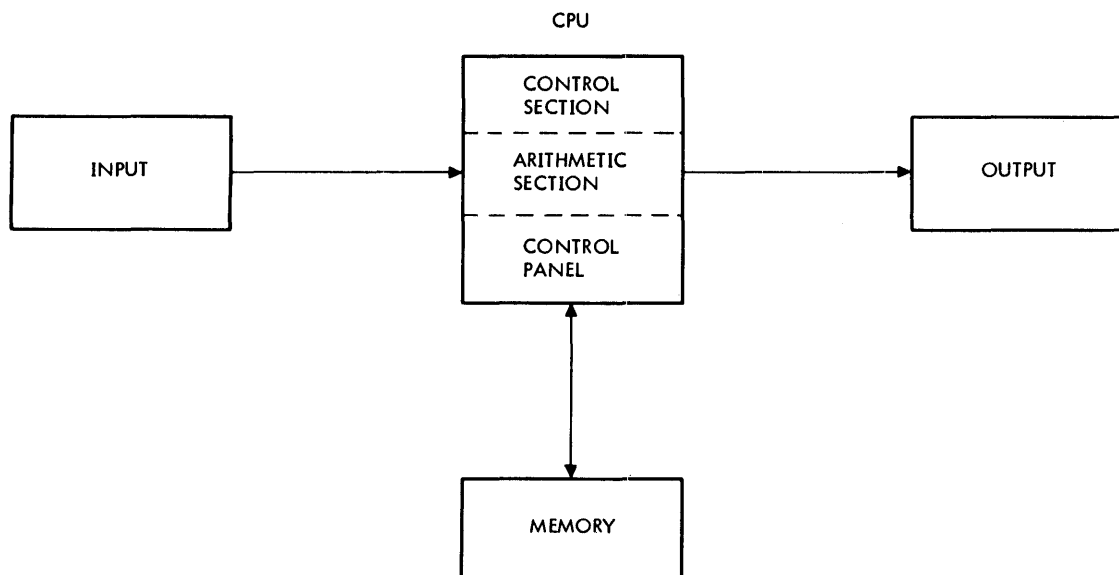
1.2 COMPUTER CONCEPTS

A digital computer system can be divided into four basic sections as shown in the figure below. The computer proper, called the central processing unit (CPU), has three subsections: the control section, the arithmetic section, and the control panel. The primary unit for the storage of information is called the memory. The input unit provides information and instructions to the computer. The output unit gives the user the processed data or information (answer).

1.2.1 Central Processing Unit (CPU)

The control section coordinates computer operations. It directs data transfers and controls the manipulation of the data. The control section also interprets and executes the instructions and information read from memory or received from the input unit.

The arithmetic section performs calculations using basic arithmetic operations. It also manipulates data under the supervision of the control section. The arithmetic section usually contains registers (accumulators) that hold the data and the results of the calculations and manipulations and logic circuitry that enables the data in the registers to be combined with information transferred from memory or input devices.



VTII-0908

Figure I-1. Typical Computer System



CHAPTER I COMPUTER FUNDAMENTALS

The control panel gives the user direct access to and control over CPU operations and memory. Switches and indicators on the CPU permit examination or alteration of the contents of memory or determination of the current status of the CPU and the program operating in it. The control panel and a keyboard input device (teletypewriter) are often grouped together under the term console.

1.2.2 Memory

The memory is a storage device for instructions and data. It is termed permanent storage because its contents remain unchanged unless alterations are specifically requested by the program or user. Since all information processed by the computer system passes through memory, memory is considered the heart of any data-processing system.

1.2.3 Input Unit

The input unit receives instructions and data from input devices, e.g., punched card readers, teletypewriter keyboards, magnetic tape or disc devices, etc. The input unit translates the information received from these devices into a form that memory can accept and store.

1.2.4 Output Unit

The output unit translates finished, processed data (answers) from the CPU into a form that can be accepted by output devices, e.g., card punches, line printers, magnetic tape or disc devices, etc., and transmits the translated data to these devices.

Note that some peripheral devices, e.g., magnetic tape or disc devices, can function both as input and output devices.



CHAPTER I COMPUTER FUNDAMENTALS

SECTION 2 NUMERICAL ANALYSIS

2.1 INTRODUCTION

The numerical analysis of a problem demands mathematical skill and ingenuity from the digital computer user. Scientific and engineering problems are not expressed in terms that can be directly handled by the computer. Roots, vectors, trigonometric functions, differential equations, and similar mathematical expressions and operations must be reduced to (or expressed as) a series of arithmetical operations if the computer is to solve the problem.

Specialists in the field of numerical analysis seldom actually use a digital computer; their major concern is to provide the computer user with techniques, algorithms, routines, and other mathematical assistance to permit computer applications. Computer users, however, must have an elementary knowledge of numerical analysis.

2.2 SQUARE ROOT EXTRACTION

Extracting the square root of a quantity is a basic mathematical operation that is often useful. The way in which this operation is executed by a digital computer is presented as an example of an algorithm provided by the science of numerical analysis. The operation is known as Newton's algorithm for square roots or the square root algorithm.

To begin, let X represent any number and let $Y = X$; i.e., Y is approximately equal to \sqrt{X} or an approximation of \sqrt{X} . Y can represent \sqrt{X} with sufficient precision for the required purpose. Y will be equal to 0 only if X is equal to 0. For any other value of X , Y is equal to or less than X ($Y \leq X$). For example, if $X = 25$, then Y can initially have any value between 0 and 25.

The formula for the square root algorithm is

$$Y_{i+1} = \frac{1}{2} \left(Y_i + \frac{X}{Y_i} \right)$$

where i = the order of the approximation; i.e., the number of times the equation has been solved for Y . The following example shows how the square root of 25 is obtained when 24 is selected as the first approximation.



CHAPTER I COMPUTER FUNDAMENTALS

$$\begin{aligned} Y_1 &= 1/2(24 + \frac{25}{24}) \\ &= 1/2(24 + 1.0417) \end{aligned}$$

$$Y_1 = 12.5208$$

$$\begin{aligned} Y_2 &= 1/2(12.52 + \frac{25}{12.52}) \\ &= 1/2(12.52 + 1.9968) \end{aligned}$$

$$Y_2 = 7.2584$$

$$\begin{aligned} Y_3 &= 1/2(7.26 + \frac{25}{7.26}) \\ &= 1/2(7.26 + 3.4435) \end{aligned}$$

$$Y_3 = 5.3517$$

$$\begin{aligned} Y_4 &= 1/2(5.35 + \frac{25}{5.35}) \\ &= 1/2(5.35 + 4.6729) \end{aligned}$$

$$Y_4 = 5.0114$$

$$\begin{aligned} Y_5 &= 1/2(5.01 + \frac{25}{5.01}) \\ &= 1/2(5.01 + 4.99) \end{aligned}$$

$$Y_5 = 5$$

The fifth approximation in this example gave the exact value of Y , but it is possible that an additional step (or one less) would be required if no roundoffs were made during computation. Frequently, an exact root cannot be found as square roots are often irrational numbers.

The value first assigned to Y does not affect the precision with which \sqrt{X} can be found. Y represents only the number of approximations to be performed.

The theory of square root extraction by this method states that:

a. When any number is divided by its square root, the quotient is the square root.
 $25/5 = 5$

b. When a square root is added to a square root and the sum divided by 2, the quotient is the square root.
 $1/2(5 + 5) = 5$



CHAPTER I COMPUTER FUNDAMENTALS

c. Therefore

$$Y = 1/2(5 + \frac{25}{5})$$

$$= 1/2(10)$$

$$Y = 5$$

An equivalent and alternate form of the square root algorithm formula is

$$Y_{i+1} = Y_i + \frac{1/2(X - Y_i)}{Y_i}$$

The following is an example of this formula using the same values as the first example. The results of the first three approximations show the equivalency of the two formulas; the fourth and fifth approximations are omitted.

$$Y_1 = 24 + \frac{1/2(25 - 24)}{24}$$

$$= 1/2(1.0417 + 24)$$

$$= 24 + 11.4791$$

$$Y_1 = 12.5209$$

$$Y_2 = 12.51 + \frac{1/2(25 - 12.52)}{12.25}$$

$$= 12.52 + 1/2(1.9968 + 12.52)$$

$$= 12.52 + 5.2616$$

$$Y_2 = 7.2584$$

$$Y_3 = 7.26 + \frac{1/2(25 - 7.26)}{7.26}$$

$$= 7.26 + 1/2(3.4435 + 7.26)$$

$$Y_3 = 5.3517$$

In conventional usage of the subscript, $i = 0$ is the first approximate root as illustrated below. A formula for successive approximations (iterations) is also shown.



CHAPTER I COMPUTER FUNDAMENTALS

First Approximation

$$\begin{aligned} Y_{i+1} &= 1/2(Y_i + \frac{X}{Y_i}) \\ Y_1 &= 1/2(Y_0 + \frac{X}{Y_0}) \\ Y_{i+1} &= Y_i + 1/2(\frac{X}{Y_i} - Y_i) \end{aligned}$$

Second Approximation

$$\begin{aligned} Y_{i+2} &= 1/2(Y_i + \frac{X}{Y_i}) \\ Y_2 &= 1/2(Y_1 + \frac{X}{Y_1}) \\ Y_{i+2} &= Y_{i+1} + 1/2(\frac{X}{Y_{i+1}} - Y_{i+1}) \\ Y_2 &= Y_1 + 1/2(\frac{X}{Y_1} - Y_1) \end{aligned}$$

The square root algorithm is particularly applicable for use with digital computers due to the following advantages:

- a. The computer is required to execute only the arithmetic operations of addition, subtraction, and multiplication or division on the data supplied.
- b. The programmer need only approximate the square root, and the closeness of this first approximation will not affect the precision of the final result.

The precision of the final result is dependent upon the number of iterations performed. The first approximation determines the number of iterations required to accomplish this final precision. As illustrated below, it does not matter whether this first approximation is larger or smaller than the correct root.

Let $X = 36$, and $Y_0 = 4$

$$\begin{aligned} Y_1 &= Y_0 + 1/2(\frac{X}{Y_0} - Y_0) & Y_2 &= 6.5 + 1/2(\frac{36}{6.5} - 6.5) \\ &= 4 + 1/2(\frac{36}{4} - 4) & &= 6.5 + 1/2(5.5385 - 6.5) \\ &= 4 + 1/2(9 - 4) & &= 6.5 + 1/2(-0.9615) \\ &= 4 + 2.5 & &= 6.5 - 0.48 \\ Y_1 &= 6.5 & Y_2 &= 6.02 \end{aligned}$$

The square root algorithm is an example of many such algorithms and similar techniques developed by the science of numerical analysis for use in digital computer problem solving.



varian data machines



CHAPTER II

PROGRAMMING



varian data machines



CHAPTER II PROGRAMMING

SECTION 1 INTRODUCTION

1.1 COMMUNICATION WITH A COMPUTER

Any computer accepts and executes a certain number of instructions (commands). Such expressions can be plain English words or phrases, e.g., CLEAR AND ADD, MULTIPLY, STORE WORD, etc. However, because of the length of such expressions and the frequency of their use in computer programs, abbreviated mnemonic forms are often substituted.

1.2 MNEMONICS

A mnemonic code is usually composed of a two- or three-letter group representing a specific computer instruction. Each of the letters in the group corresponds to the first (or key) letter in the complete word or phrase. For example, the instruction CLEAR AND ADD can have the mnemonic CAD.

Mnemonics simplify the writing of computer programs. However, because the computer responds only to numerical input, neither the mnemonic nor the full written instruction is readily understandable to the computer. Each instruction, therefore, has a corresponding numerical equivalent or numerical code.

1.3 NUMERICAL CODES

The numerical code exists in the computer as a combination of the binary digits 0 and 1. For simple machines, such codes can be expressed in full binary form. However, for most computers the codes will be condensed to octal (base 8) or hexadecimal (base 16) numbers. The use of octal or hexadecimal numbers allows the code groups to be expressed in fewer digits than required by binary notation.

Octal notation is widely used because of the ease with which octal to binary conversions can be performed. Thus, data fed into the computer in octal form can be readily converted by the computer to binary for storage in memory. If the computer is not equipped to perform this conversion, it can be done mentally by the programmer as he loads the data.



CHAPTER II PROGRAMMING

1.4 INSTRUCTION SET

The list of instructions or codes that a computer accepts and executes is called the instruction set (repertoire) for that computer. The binary codes for these instructions comprise the machine language for that computer, the only language the computer understands.

1.5 PREPARING THE PROBLEM

The writing of a routine for the solution of a particular problem by a specific computer requires the formulation of a computer message in its machine language. The message writing procedure begins with the numerical analysis of the problem and ends with the writing of the routine in machine language. One or more people may be involved, each utilizing his individual area of interest to accomplish a specific part of the total task.

- a. The numerical analyst analyzes the problem and finds the best mathematical approach for the particular application.
- b. The programmer decides:
 - (1) Operations to be performed and order of performance.
 - (2) Addresses for the required number of instructions, and any address modifications necessary.
 - (3) Addresses for the required number of data words.

The programmer can use mnemonics for the written routine rather than specify the actual addresses to be used.

- c. The coder prepares the final routine in numerical code with all addresses designated.

When the entire program is coded in machine language, it is ready to be loaded into the machine.



CHAPTER II PROGRAMMING

SECTION 2 FLOW-CHARTING

2.1 INTRODUCTION

In preparing a problem for the computer, the user must first clearly define the problem and a method of solving it. The computer can only follow instructions and cannot devise its own methods of problem-solving. The definition of a simple problem and the method of solving it may be obvious enough that the program can be coded directly as a list of instructions in mnemonic form. However, if a problem is this simple, use of a computer is probably unnecessary.

In typical computer problems from the business and scientific worlds, solutions require many steps and decisions. As an aid to programming the solutions to such problems, the flow chart is invaluable. The flow chart illustrated in figure II-1 is a schematic diagram of the logical steps required to solve the problem. The chart consists of a series of connected geometric figures, each denoting a step in the solving of the problem. Each geometric shape has a particular logical significance, e.g., rectangles indicate computer processing steps, diamonds indicate decisions (refer to figure II-2). The geometric forms are annotated with explanations and connected with lines and arrows showing the sequence and direction of process flow.

Since the flow chart is a tool for analysis and clarification, it generally shows only enough information to ensure:

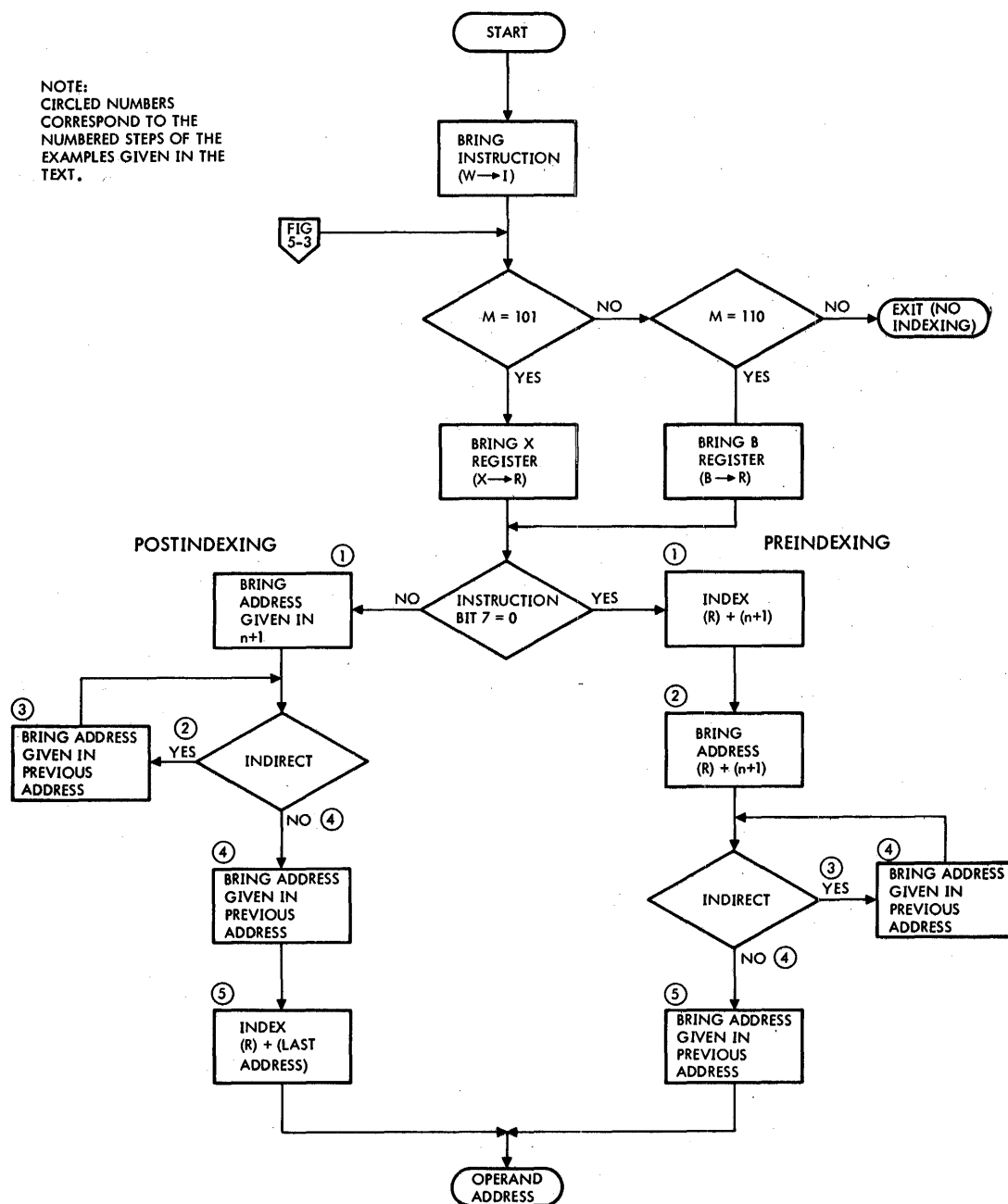
- a. Correct compilation of the instructions in the program
- b. Proper planning for and allocation of memory space

Flow-charting is sometimes considered an art rather than a science because it is unlikely that two programmers will produce identical flow charts for the solution of the same problem. Given the various ways of approaching any problem, it is not necessary that such charts be identical. Even the number of functions considered to be a single step for inclusion in one geometric figure in the chart will vary. Since flow charts are tools, the way in which this tool is used depends on the programmer's training, experience, ability, preferences, and even personality. These differences do not reduce the importance of the flow chart or the necessity of mastering flow-charting techniques. A good flow chart gives maximum assistance in compiling a program and allocating memory space. Flow-charting is thus an art with a sound basis in scientific methods.



CHAPTER II PROGRAMMING

NOTE:
CIRCLED NUMBERS
CORRESPOND TO THE
NUMBERED STEPS OF THE
EXAMPLES GIVEN IN THE
TEXT.



VT13-0243

Figure II-1. Typical Flow Chart

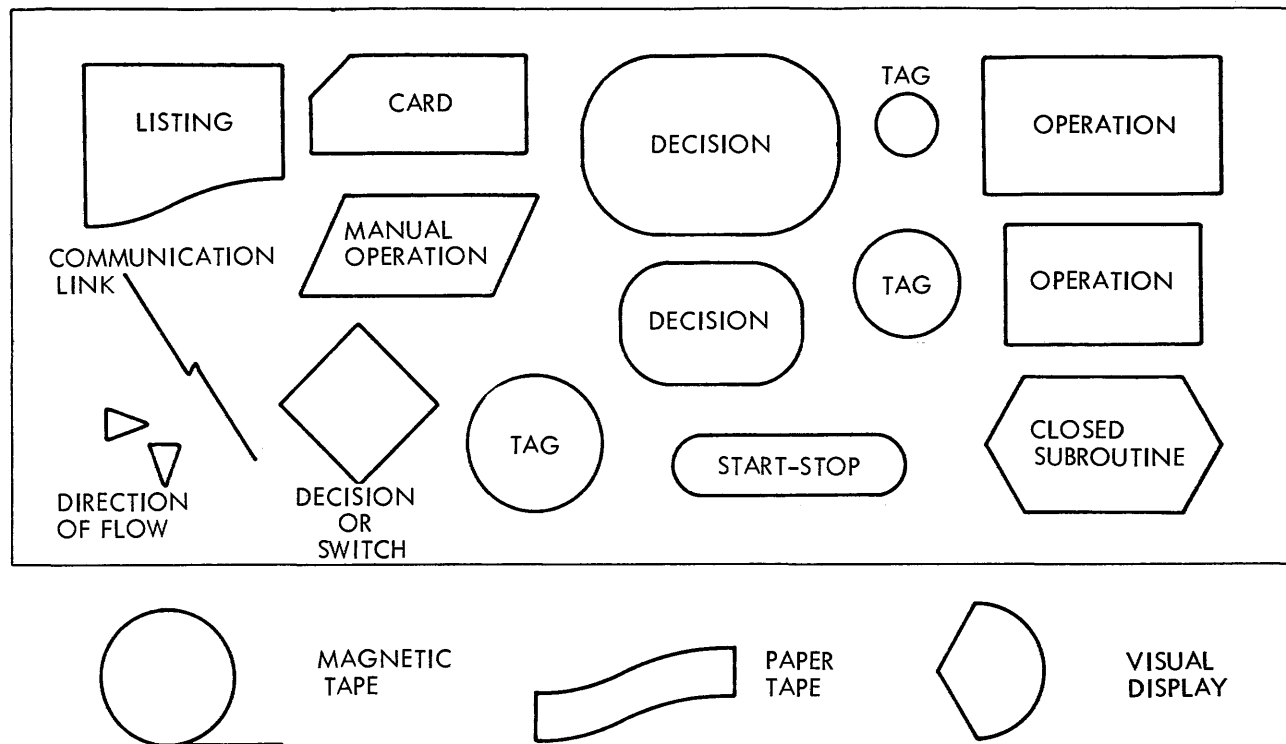


Figure II-2. Flow Chart Symbols

VTII-0240



CHAPTER II PROGRAMMING

The completed flow chart should be examined to see that a program based on it will actually allow the computer to solve the problem. Alternate methods that might require less computer running time and/or memory space should be considered. When these considerations have been taken into account and the final flow chart approved, the programmer is ready to start writing the program based on the chart.

2.2 NOTATION

The following is a list of common notations used to signify the writing of computer programs.

Notation	Meaning
()	The content of; (A) indicates the contents of the A register.
A1	A register bit 1
A1-5	A register bits 1 through 5
Asn	A register sign bit
→	Indicates the transfer of data; (A)→(B) means the contents of the A register are transferred to the contents of the B register.
	The magnitude of; X indicates the magnitude of X
<	Smaller than; (A) < (B) indicates that the contents of the A register are smaller than the contents of the B register.
>	Greater than; (A) > (B) indicates that the contents of the A register are greater than the contents of the B register



CHAPTER II PROGRAMMING

2.3 SYMBOLS

2.3.1 Functions

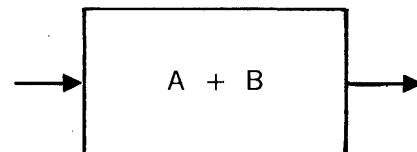
The solution of a problem requires the execution of several functional steps. In the context of flow-charting, a functional step is one that requires the execution of an arithmetical or logical operation on the data being processed or the transfer of data within the computer. A function box in the flow chart indicates a functional step or operation as illustrated below.

The number of functions in any function box is at the discretion of the programmer. In the preliminary part of the analysis, two or more functional operations may appear in a single box if the operations are related or sequential, such as those shown in figure II-3. As the analysis progresses and the flow chart becomes more complete and detailed, complexity permits only one functional operation per box.

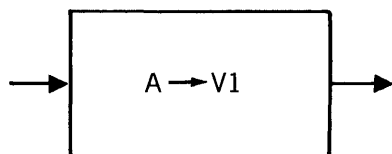
Not annotated



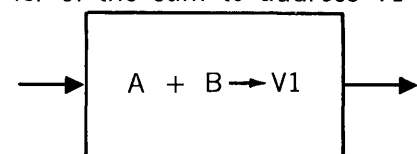
Annotated to show the addition
of word A to word B



Annotated to show the transfer
of word A to address V1



Annotated to show the addition
of words A and B and the transfer
of the sum to address V1



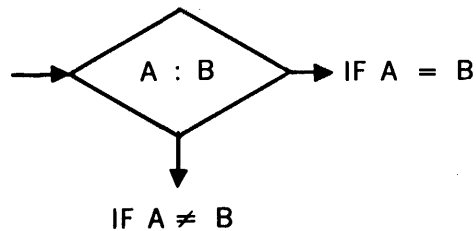


CHAPTER II PROGRAMMING

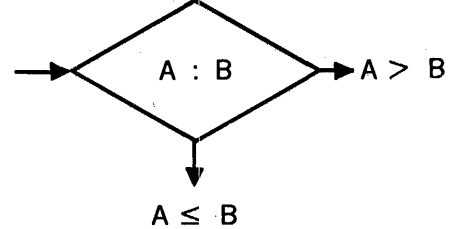
2.3.2 Decisions

A decision box is used to show a step which requires the computer to make a decision. Computer decisions are two-valued (yes or no or true or false); therefore, the decision box has one input path and two output paths (see below). For a particular operation, only one of the output paths is taken dependent upon the decision.

Annotated to show a decision based on an equality comparison of values of words A and B



Annotated to show a decision based on the comparative values of words A and B





CHAPTER II PROGRAMMING

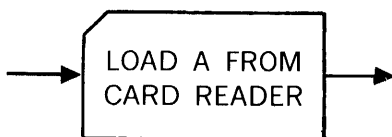
2.3.3 Input/Output

Whenever problem solution requires that information be received via the input equipment, or transmitted via the output equipment, an input/output box is used in the flow chart as illustrated.

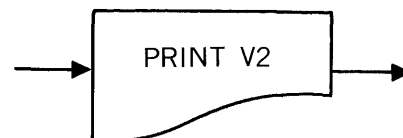
The input/output box is annotated to indicate:

- The operation to be performed by the use of words such as LOAD, STORE, PRINT, DISPLAY, etc.
- Data involved as stated in the literal term appearing in the equation (a , x , b^2), a memory address, or an auxiliary store address.
- The input/output equipment involved. The equipment may be implied by the operation (PRINT implies a teletypewriter), or explicitly stated.

Annotated for the loading of
word A into memory via the
card reader



Annotated for the printing of
the word at address V2

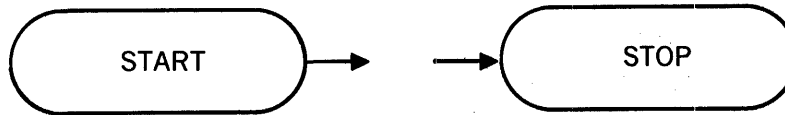




CHAPTER II PROGRAMMING

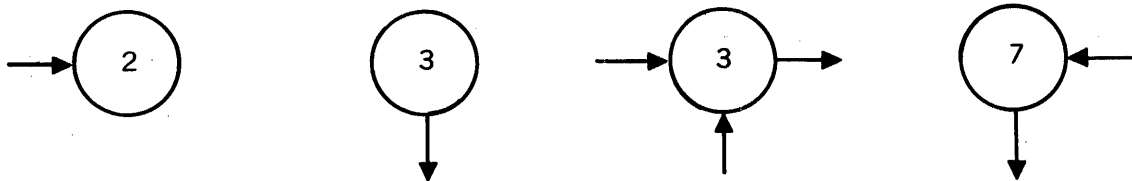
2.3.4 Start and Stop

Start and stop boxes designate the beginning and the end of a flow chart as illustrated below.



2.3.5 Fixed Connectors

A flow chart appears as a sequence of interconnected boxes arranged in either columns or rows. It is normally not possible to present an entire flow chart on one page. Therefore, a system of fixed connector circles is used to relate the separate sheets of the flow chart. These circles are numbered; all fixed connectors with the same number refer to the same location on the chart. A fixed connector may have one or more input/output paths as illustrated.

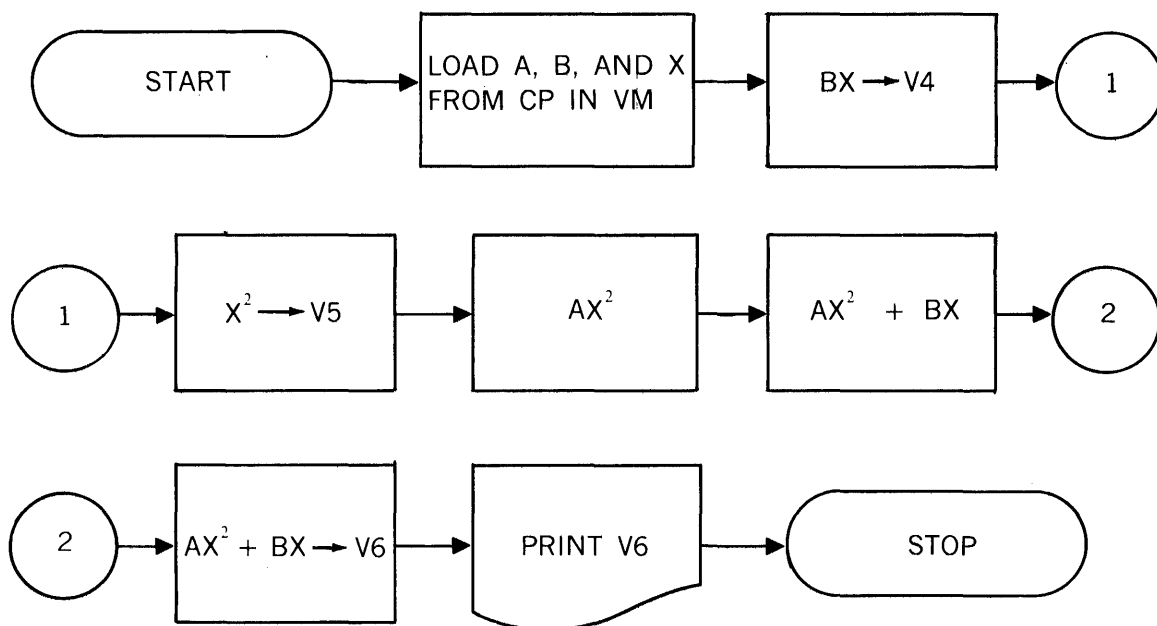


CHAPTER II
PROGRAMMING

2.4 A SIMPLE FLOW CHART

Using only the symbols presented so far, it is possible to formulate a simple flow chart to solve the equation:

$$T = AX^2 + BX$$





CHAPTER II PROGRAMMING

In the preceding flow chart:

- a. FROM CP IN VM indicates that data is input from the control panel and stored in variable memory. Variable memory is the area of memory reserved for the storage of data with variable values, as opposed to data representing mathematical constants with fixed values.
- b. $BX \rightarrow V4$ indicates that B is to be multiplied by X and the product stored in variable memory address V4. V4 is not the actual address 4, but is relative to other addresses in the flow chart (all relative addresses are replaced with specific memory addresses when the chart is coded).

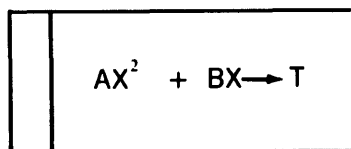
This flow chart is only one of many that could be constructed for this particular problem. The number of boxes used and the amount of information given in each box is at the discretion of the programmer. For instance, the loading of each of the three terms (A, B, and X) could be shown by a separate input box, and each of these boxes could be annotated to show the storage address in variable memory.

The general rule is to show only the information necessary to compile the list of instructions and to ensure proper allocation of memory addresses.

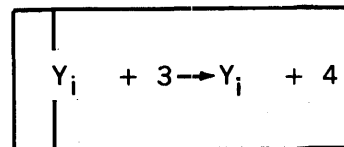
2.4.1 Substitution

The preceding flow chart does not explicitly show when the value of T is computed but implicates it in the function box $AX^2 + BX$ before fixed connector box 2. At this point, a substitution box can be used to indicate that a computation has been performed and the result can be substituted to simplify the notations that follow.

Annotated for point of
computation and substitution



Annotated to change the value of
a subscript





CHAPTER II PROGRAMMING

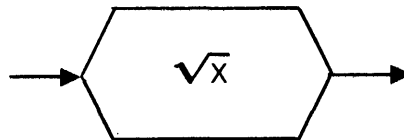
The substitution box can also be used to:

- a. Advance or change the value of a subscript (see above).
- b. Provide a term or quantity that is equivalent to one that is not available (i.e., the cosine of 60° that is available in memory for the sine 60° that is not).
- c. Modify an address to effectively substitute one address for another.

2.4.2 Subroutines

Often the solution of a problem requires the solution of one or more subordinate or auxiliary problems. If these secondary problems appear repeatedly, either within the same problem or in different problems, a separate routine is written for each problem and used whenever that problem appears. These previously prepared routines for secondary problems are called subroutines.

A flow chart is prepared for a subroutine in the same manner as larger routines except that the START and STOP boxes are annotated identically to identify the subroutine. A subroutine computation box is used to indicate where the subroutine is to appear on the larger flow chart; the subroutine flow chart itself does not appear.

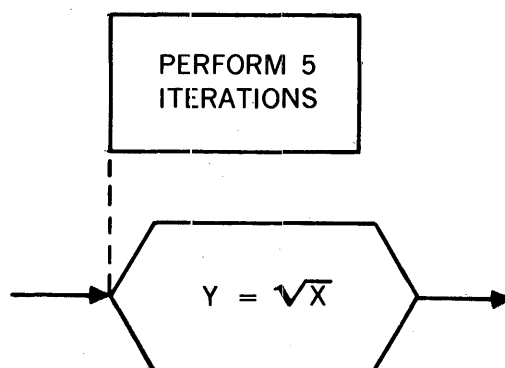




CHAPTER II PROGRAMMING

2.4.3 Assertions

An assertion box is used to present additional information or explanatory notes. It is appended outside the path of the flow chart to indicate where the information is applicable.



2.5 A MORE COMPLICATED FLOW CHART

Figure II-3 is a flow chart to solve the problem:

$$T = AX^2 + BX + \sqrt{X} + \sin 0$$

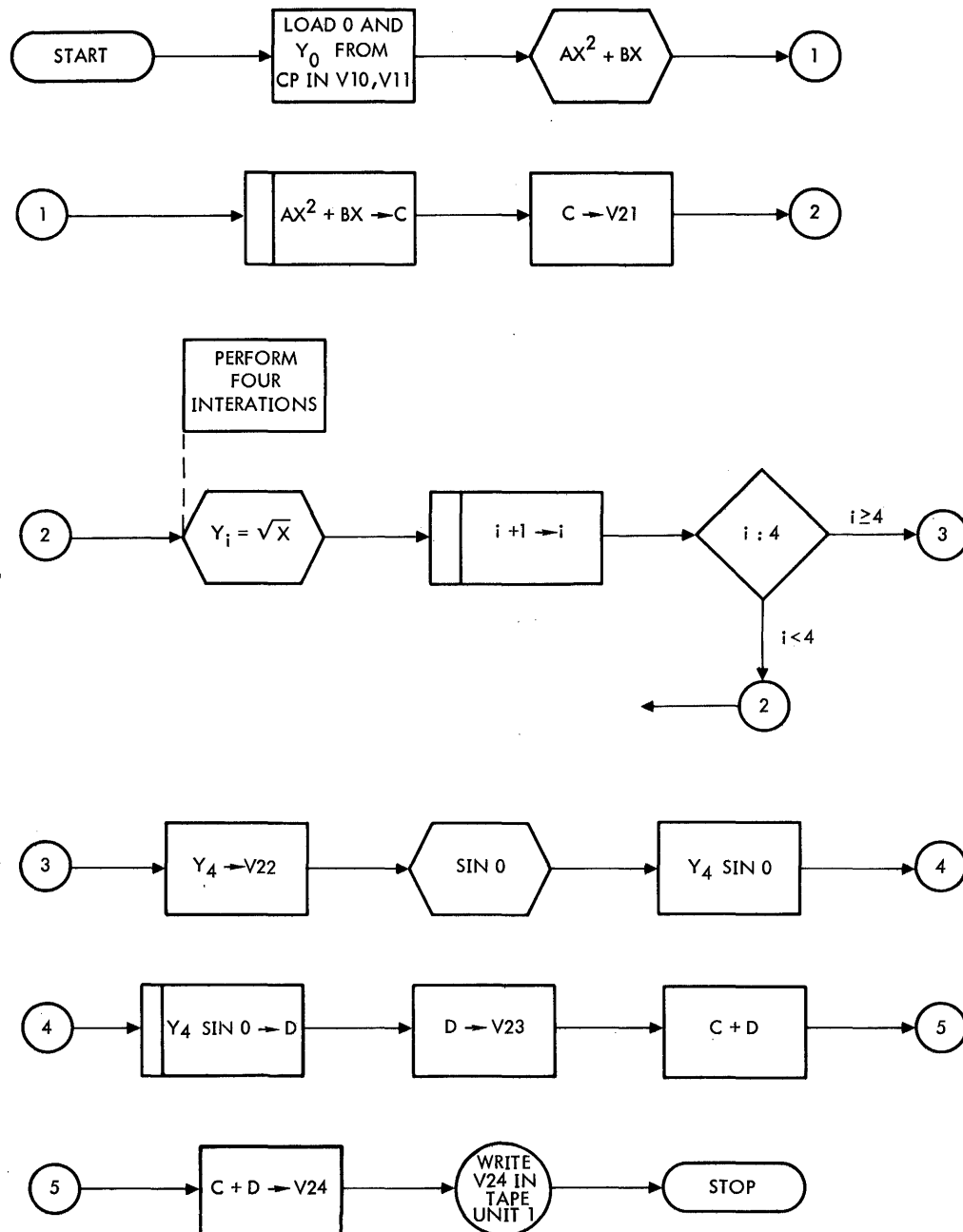
Subroutines are available for $AX^2 + BX$, \sqrt{X} , and $\sin 0$. It is assumed that the correct values of all terms are stored at a known memory address (any address except 0). A value for Y in the square root algorithm must be assigned and stored in memory.

Between fixed connectors 2 and 3, the value of 1 is added to the subscript value of Y each time a square root iteration is performed. This value is then compared with a value in the decision box to determine when the required number of iterations (in this case, four) have been performed so that the computation can proceed to the next step. During the square root computations a closed loop exists in the flow chart between the bottom of the decision box and the left end of the square root subroutine computation box (fixed connector 2). Such a loop is known as a program loop.

Since a decision box always has two output paths, the appearance of a decision box in a flow chart will always produce a program loop or a program branch; i.e., if one of the



CHAPTER II PROGRAMMING



VTH-1163

Figure II-3. Flow Chart for $T = AX^2 + BX \sqrt{X} \sin 0$



CHAPTER II PROGRAMMING

output paths does not loop back to a previous point in the flow chart, then the decision must be to branch out on one or the other of the two possible output paths. Thus, a branch determines which of two possible methods (or program paths), are to be used for the remainder of the problem solution.



CHAPTER II PROGRAMMING

SECTION 3 MACHINE LANGUAGE PREPARATION

3.1 INTRODUCTION

After a problem has been analyzed and a method of solution determined by the construction of a flow chart, the routine (or program) is prepared. The programmer first verifies that:

- a. The flow chart utilizes a method that will actually solve the problem.
- b. This method is presented in its simplest form.
- c. There is not a more efficient method.

3.2 INSTRUCTION REPERTOIRE

A program is written for a specific computer in response to the language used and the repertoire provided. The number of instructions in a repertoire varies. A large repertoire does not necessarily mean that the program will be more complex, only that more instructions are available for the programmer's use. A thorough knowledge of the specific computer's repertoire is necessary to fully utilize the instructions provided.

To write a program, the programmer must know:

- a. The name and mnemonic of each instruction in the repertoire.
- b. The result obtained through the use of each instruction.
- c. The conditions governing the use of a particular instruction.
- d. The type and format of the computer's instruction and data words.
- e. The types of addressing available, and the use of addresses with the various instructions.



CHAPTER II PROGRAMMING

3.2.1 Instruction Types

All computer instructions are classified as arithmetic or processing instructions, transfer of information instructions, or transfer of control instructions.

Certain instructions are included in two classifications; i.e., instructions that transfer data to the arithmetic unit and perform an operation there. Examples are the ADD, SUBTRACT, MULTIPLY, and DIVIDE instructions. This type of instruction always requires the use of an address in the address part of the instruction word to specify the location of the data word.

SHIFT LEFT and SHIFT RIGHT are examples of instructions that are only arithmetic or processing instructions. This type of instruction does not require an address as no data are transferred.

Transfer of information instructions such as LOAD and STORE generally require the use of an address in the instruction word to indicate the point of origin or destination of the word being transferred. An exception is an instruction such as TRANSFER where the origin and destination are given.

There are many transfer of control instructions, including all the JUMP instructions. These instructions contain an address in the address part of the instruction word to indicate the location of the instruction to which the program jumps. This is the address of another instruction not a data word.

3.2.2 Addresses

A program can use either the actual address or a relative address. If the problem is fairly simple, the actual address is used throughout. In more complex problems, relative addresses are used until memory requirements are established and actual addresses can replace the relative addresses. Actual addresses are expressed in octal or hexadecimal notation; relative addresses can use any convenient format.

3.2.3 Codes

In the program, instructions are written as mnemonic code references. Each mnemonic in turn references a numeric code which is the final input to the computer and constitutes the machine language. Mnemonic codes and the associated numeric codes for each instruction are provided by the manufacturer of the specific computer. After the program is completed, the mnemonics are coded into the numeric equivalents.



CHAPTER II PROGRAMMING

3.3 SAMPLE PROGRAMS

The following programs are presented to illustrate a computer repertoire. A specific computer repertoire can comprise more or fewer instructions than used in these programs.

A digital computer is normally used only for complex or lengthy problems as operating expense does not make it practical for use in simple problems. The examples that follow are presented as illustrations of flow charts and repertoires; they are too simple to be typical computer problems. For additional information about these problems, refer to the applicable system reference manual and to the instruction repertoire (figure III-7).



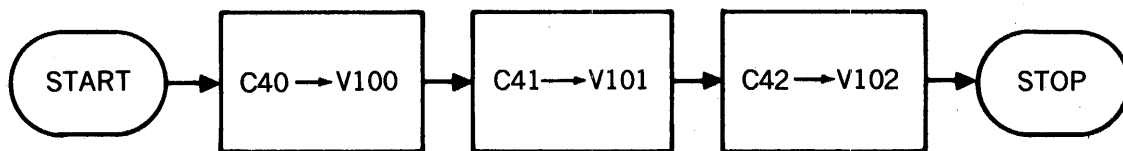
CHAPTER II PROGRAMMING

PROBLEM 1

Given: Three constants stored in constant memory locations:
C40, C41, and C42.

Problem: To store these constants in variable memory locations:
V100, V101, and V102, respectively.

Flow chart



**CHAPTER II
PROGRAMMING****Repertoire**

Step No.	Operation	Address	Remarks
000	LDA	C40	This instruction clears the accumulator to 0 and loads the contents of C40 into the accumulator.
001	STA	V100	This instruction stores the contents of the accumulator in V100.
002	LDA	C41	Same operations as step 000, for the contents of C41.
003	STA	V101	Same as step 001, stored in V101.
004	LDA	C42	Same as step 000, for C42.
005	STA	V102	Same as step 001, stored in V102.
006	HLT		Computer stops. End of program.

NOTE: The step number column is used at the programmers's option to keep track of the program and provide a means of reference to a particular operation in the program sequence.



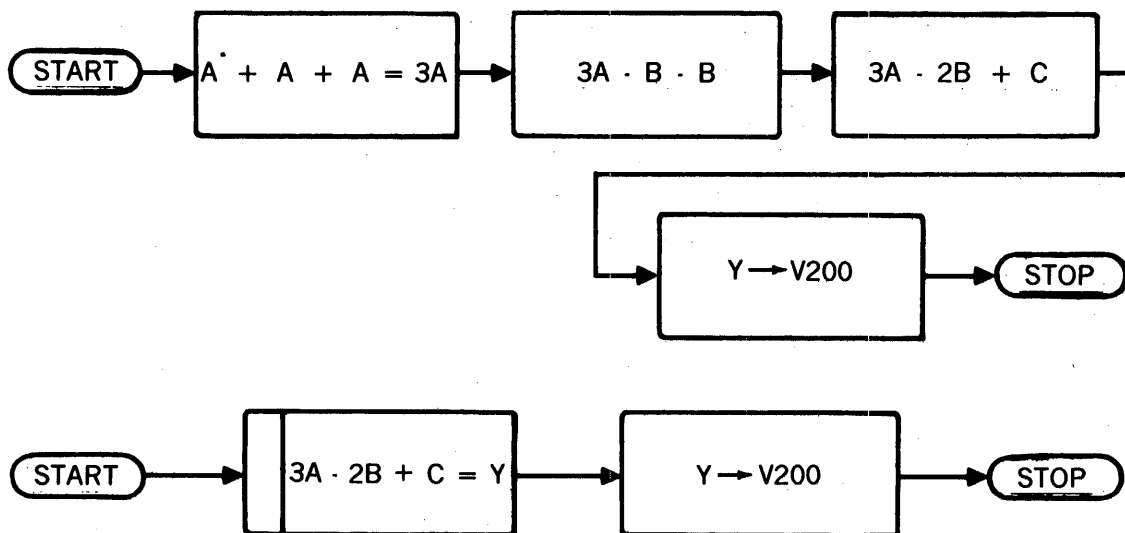
CHAPTER II PROGRAMMING

PROBLEM 2

Given: A located in C40, B in C41, and C in C42.

Problem: Write a program to solve for Y and store Y in V200:
 $Y = 3A - 2B + C$.

Flow charts



Two flow charts are shown; both present the same solution to the problem. The first flow chart is more detailed and breaks the operations down into three computation boxes, plus a transfer box. The second flow chart places all the computations into one substitution box.



CHAPTER II PROGRAMMING

Repertoire

Step No.	Operation	Address	Remarks
000	LDA	C40	Accumulator register (AR) is cleared. Contents of C40 loaded into AR.
001	ADD	C40	$A + A = 2A$ in AR.
002	ADD	C40	$A + 2A = 3A$ in AR.
003	SUB	C41	$3A - B$ in AR.
004	SUB	C41	$3A - 2B$ in AR.
005	ADD	C42	$3A - 2B + C$ in AR.
006	STA	V200	$Y = 3A - 2B + C$ to V200.
007	HLT		Computer stops. End of program.



CHAPTER II PROGRAMMING

PROBLEM 3

Problem 3 illustrates the use of program tags. Program tags aid the programmer in jumping to an unknown program address. They are annotated with one to four alphanumeric characters, one of which must be a letter. X, X3, X123, XYZ, ABCD, XXX, and PB12 are examples of program tag notation.

Given: Five numbers are stored in C0 through C4; C10 contains 0; C11 contains 1.

Problem: Write a program to place the count of negative numbers in V100, the count of positive numbers in V101.

To solve the problem, write a program that:

- a. Clears addresses V100 and V101 to 0.
- b. Transfers each number to the accumulator to determine if it is positive or negative.
- c. Stores a one in V101 if the number is positive, and a one in V100 if it is negative.

Figure II-4 illustrates a flow chart that causes the computer to step sequentially through the program if all the numbers are positive. If C0 is a positive number, the program proceeds normally to store a one in V101. If C0 is negative, however, the computer must jump out of the normal sequence to store a one in V100. Therefore, provision is made in the program to jump to some other step in the case of a negative number. Since it is not known when the program is written where a negative number will occur, program tags are used. This will allow the programmer to go back and assign a program step number to the program tag.

The reference boxes in the flow chart are annotated to correspond to the step number in the repertoire that is associated with the program tag closest to that box.

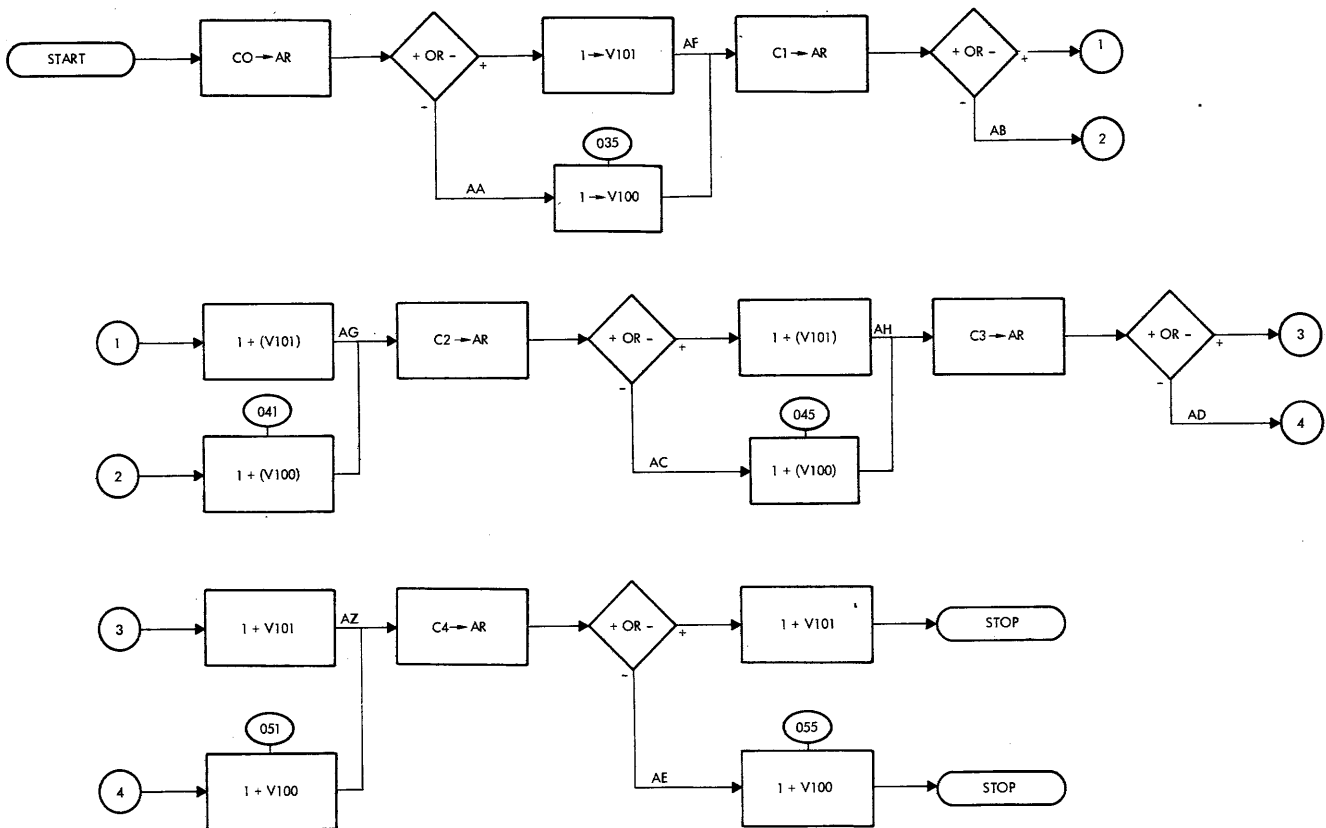


Figure II-4. Flow Chart for a Positive and Negative Number Count

VTII-1164

CHAPTER II
PROGRAMMING

Repertoire for Problem 3

Tag	Step No.	Operation	Address	Remarks
	000	LDA	C10	0 in AR.
	001	STA	V100	0 in V100.
	002	STA	V101	0 in V101.
	003	LDA	C0	Contents of C0 in AR.
	004	JAN	AA (035)	If C0 is negative, jump to AA (we learn later in the program that AA is step 035 and add the step number at that time).
	005	LDA	C11	1 in AR if C0 is positive.
	006	ADD	V101	Contents of V101 (0) to contents of AR (1).
	007	STA	V101	Contents of AR to V101.
AF	010	LDA	C1	Contents of C1 to AR.
	011	JAN	AB (041)	Jumps if C1 is negative.
	012	LDA	C11	1 in AR if C1 is positive.
	013	ADD	V101	Add contents of V101 to AR (1).
	014	STA	V101	Contents of AR to V101.
AG	015	LDA	C2	Contents of C2 to AR.
	016	JAN	AC (045)	Jumps if C2 is negative.
	017	LDA	C11	1 in AR if C2 is positive.



CHAPTER II PROGRAMMING

Repertoire for Problem 3 (continued)

Tag	Step No.	Operation	Address	Remarks
	020	ADD	V101	Add contents of V101 to AR (1).
	021	STA	V101	Contents of AR to V101.
AH	022	LDA	C3	Contents of C3 to AR.
	023	JAN	AD (051)	Jumps if C3 is negative.
	024	LDA	C11	1 in AR if C3 is positive.
	025	ADD	V101	Add contents of V101 to AR (1).
	026	STA	V101	Contents of AR to V101.
AI	027	LDA	C4	Contents of C4 to AR.
	030	JAN	AE (055)	Jumps if C4 is negative.
	031	LDA	C11	1 in AR if C4 is positive.
	032	ADD	V101	Add contents of V101 to AR (1).
	033	STA	V101	Contents of AR to V101.
	034	HLT		Stops here if C4 is positive.
AA	035	LDA	C11	1 in AR if C0 is negative.
	036	ADD	V100	Contents of V100 (0) to contents of AR (1).
	037	STA	V100	Contents of AR to V100.
	040	JMP	AF (010)	Jumps back to bring in C1.
AB	041	LDA	C11	1 in AR if C1 is negative.
	042	ADD	V100	Contents of V100 added to

**CHAPTER II
PROGRAMMING****Repertoire for Problem 3 (continued)**

Tag	Step No.	Operation	Address	Remarks
				contents of AR (1).
	043	STA	V100	Contents of AR to V100.
	044	JMP	AG (015)	Jumps back to bring in C2.
AC	045	LDA	C11	1 in AR if C2 is negative.
	046	ADD	V100	Add contents of V100 to AR (1).
	047	STA	V100	Contents of AR to V100.
	050	JMP	AH (022)	Jumps back to bring in C3.
AD	051	LDA	C11	1 in AR if C2 is negative.
	052	ADD	V100	Add contents of V100 to AR (1).
	053	STA	V100	Contents of AR to V100.
	054	JMP	AI (027)	Jumps back to bring in C4.
AE	055	LDA	C11	1 in AR if C4 is negative.
	056	ADD	V100	Add contents of V100 to AR (1).
	057	STA	V100	Contents of AR to V100.
	060	HLT		Stops here if C4 is negative.

To review this program:

- a. In Step 003, the first number (C0) is transferred to the accumulator. A count will be stored in either of two locations, depending upon whether this number is positive or negative.



CHAPTER II PROGRAMMING

- b. In Step 004, the program states that if the number is negative, it will be dealt with during a later portion of the program. The step number is not available yet, so the programmer tags the point as AA and puts a reference box above the function box on the flow chart.
- c. Continue the program as though C0 and all the other words are positive, but a decision box and alternate branch are charted following each function box annotated to show the transfer of a word in the AR.
- d. At Step 034, a stop operation is programmed. To be complete, the program must now store a count of the negative numbers.
- e. At the first negative number, the computer was directed to jump to AA.
 - (1) The programmer can now assign a program step number to AA; this is Step 035.
 - (2) He goes back to Step 004 and indicates that the address of tag AA is Step 035.
 - (3) He must also complete a cross reference to indicate that Step 035 is AA by placing AA in the column to the left of the step number.
- f. Assuming that C0 was negative, a one must be stored in V100; this is Step 037.
 - (1) The computer must return to examine the second number at Step 010.
 - (2) The programmer programs an Unconditional Jump (JMP) to tag AF.
- g. Looking back to where the second number (C1) was brought into the accumulator, he finds that this is Step 010. He tags Step 010 with an AF.
- h. He continues with this procedure until he has stored a count of all negative numbers in V100 and then programs a HLT operation at Step 060.



CHAPTER II PROGRAMMING

3.4 MACHINE LANGUAGE

Figures II-5 and II-6 apply flow diagrams to problems involving decisions. The problem is to make a flow diagram and write the machine language for a program that will count the number of positive numbers contained in a group of four core memory words. The count is kept in location 600.

Machine language programs are written in the language that a specific computer understands. In the following programs, the code column numbers are the machine language; the mnemonics are nothing more than a memory aid. Refer to the table in figure II-5 for an example of a program using machine language.

3.5 LOOPING

Problems programmed for computer solutions have some repetitious aspects; a repetitious process drawn on a flow chart appears as a loop. A single flow chart can have several loops, and loops can be nested within one another.

Looping indicates a return to an earlier operation thus avoiding a sequence of almost identical operation boxes. When the return is made, the computation will involve either a new data item or a new estimate of a computed quantity. The operation will remain the same, however. Looping is used in solving the problem illustrated in the flow chart in figure II-6.

3.6 INDEXING

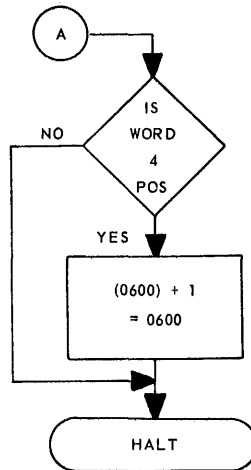
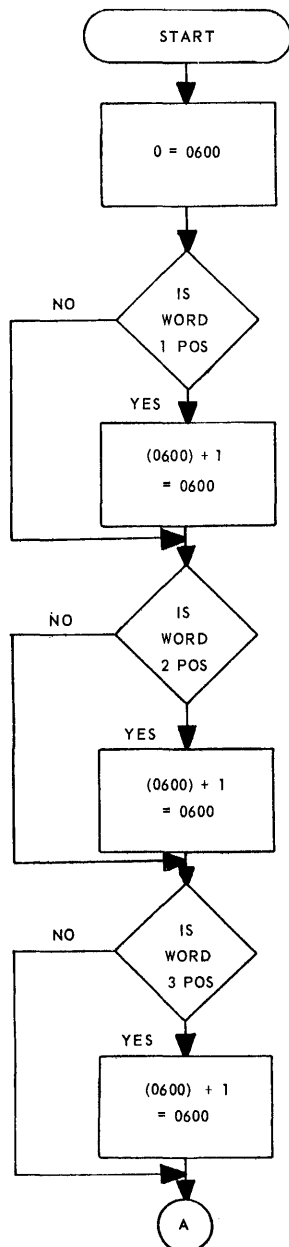
Index registers reference a sequence of memory addresses and are used in nonloop situations to permit rapid access to tables in memory and to enable branches beyond range limits. Thus, indexing permits a number of operations to be executed rapidly.

In previous examples involving memory storage and various mathematical processes, subscripts were generally assigned to the variables. Operations to be performed were indicated in terms of these subscripts. For example, a_i represented the i value of a series of numbers a . To indicate that, after a_i was processed a_{i+1} was to be processed, operation on the subscript was required: $i + 1 = i$. This means that 1 is to be added to the subscript, which generates the address to the next number.

To process a_i as indicated, it is necessary to modify the appropriate instructions. They can be modified by treating them as data and adding 1 to the operation and addresses once each loop cycle. Three operations on the index register are required:



CHAPTER II PROGRAMMING

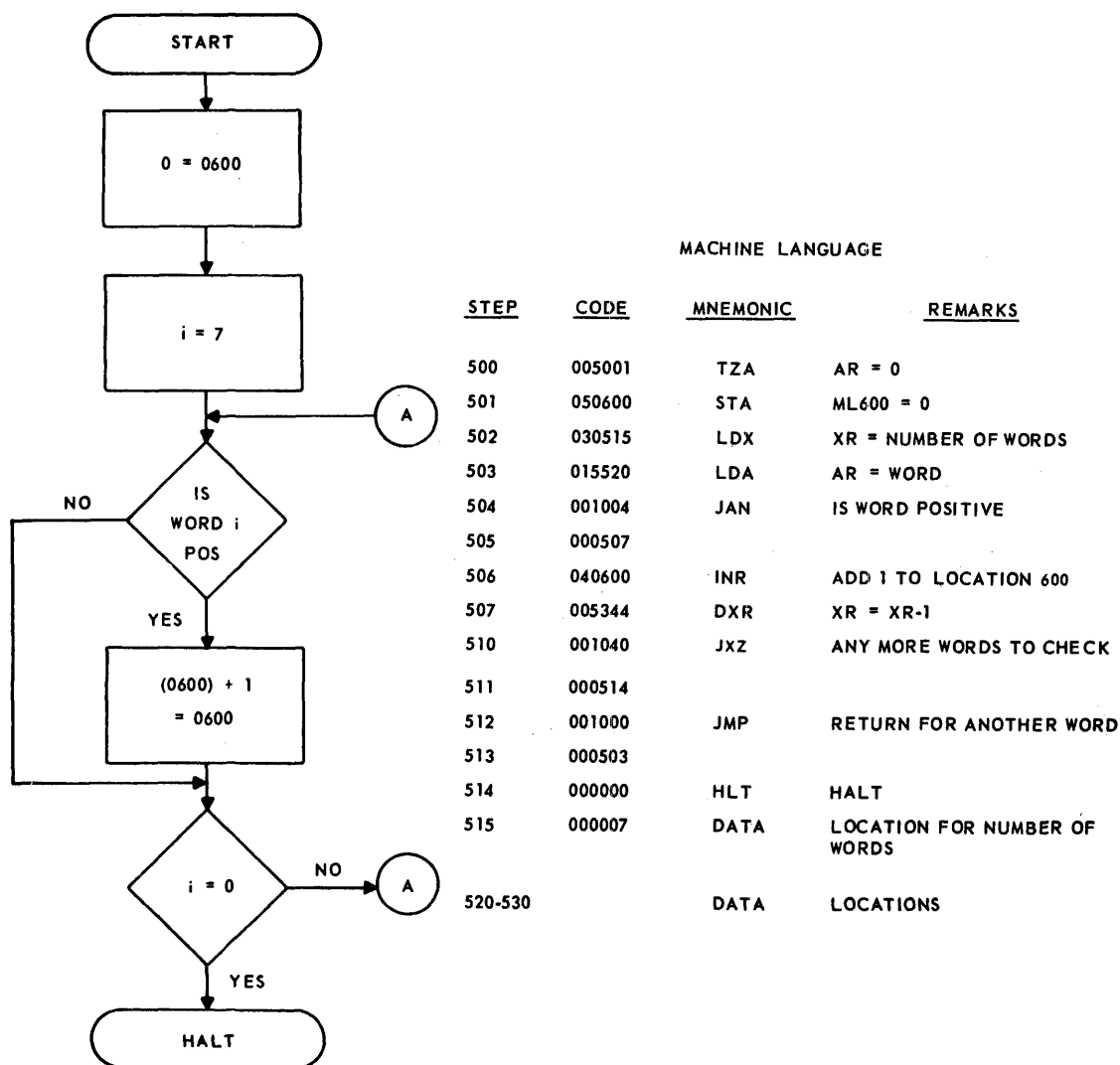


MACHINE LANGUAGE

STEP	CODE	MNEMONIC	REMARKS
500	005001	TZA	AR-0
501	050600	STA	AR ML 600
502	010530	LDA	AR = WORD 1
503	001004	JAN	IS WORD NEGATIVE
504	000506		
505	040600	INR	ADD 1 TO LOCATION 600
506	010531	LDA	AR = WORD 2
507	001004	JAN	IS WORD 2 NEGATIVE
510	000512		
511	040600	INR	ADD 1 TO LOCATION 600
512	010532	LDA	AR = WORD 3
513	001004	JAN	IS WORD 3 NEGATIVE
514	000516		
515	040600	INR	ADD 1 TO LOCATION 600
516	010533	LDA	AR = WORD 4
517	001004	JAN	IS WORD 4 NEGATIVE
520	000522		
521	040600	INR	ADD 1 TO LOCATION 600
522	000000	HLT	HALT
530-533		DATA	LOCATIONS

VTII-1165

Figure II-5. Count of Positive Numbers

CHAPTER II
PROGRAMMING

VTII-1166

Figure II-6. Loop Program



CHAPTER II PROGRAMMING

- a. Setting the index register
 $1 = i$
- b. Increasing or decreasing the index register
 $i + 1 = i$ or $i - 1 = i$
- c. Testing the value of the index register
 $(i - n) = 0$

The 620 computer systems include index registers for performing indexing operations. The contents of these registers are used to automatically modify the operand address of instructions. Index registers are designated by number within the computer. If an instruction makes reference to an index register by its number, the contents of its operand address are modified by the contents of that register.

3.6.1 Specifying the Index Register

A specified index register is referred to as a tag; a tag is indicated in machine language by placing its numerical designator in the M field (bits 9 through 11) of the machine code. The first index register (the X register) is designated by a 5 in the M field; the second (B register) by a 6. The instruction counter, which is a form of address modification, is designated by a 4 in the M field.

05	4	100	=	P counter
05	5	100	=	X register
05	6	100	=	B register

In an instruction with no tag (0 in the M field), the address of the word that is processed is simply the operand address. In an instruction with a tag, however, the address of the processed word is given by the sum of the operand address and the contents of the modifier. This address modification is automatic and temporary; the instruction does not change, but the affect is as though it were changed during the execution of the instruction. For example: let $XR = 100$. The instruction 12 5 000 adds the address $000 + 100$ to form the effective address so that the contents of location 100 will be added to the accumulator.

3.6.2 An Example of Indexing

Given: A table three registers in length, starting at location 0550. The sum is to be placed in register 0600.



CHAPTER II PROGRAMMING

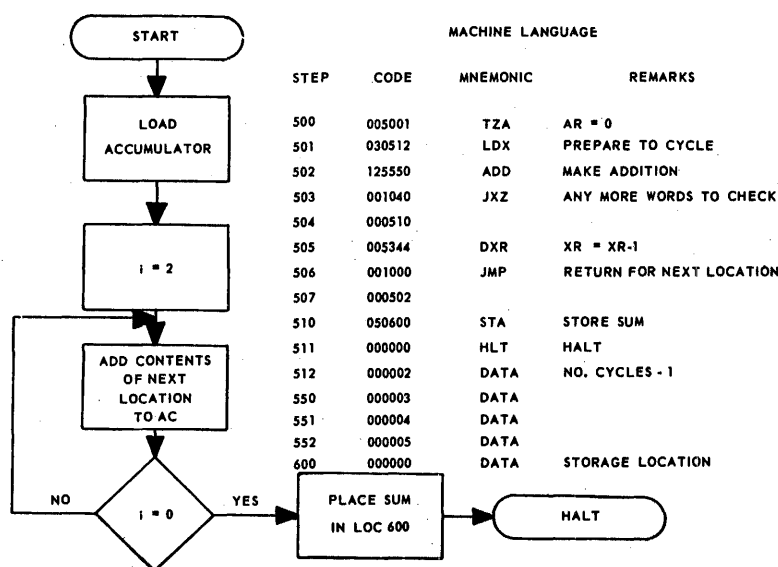
Problem: Write a program to compute the sum of the contents of the table.

The table looks like this

0550	3
0551	4
0552	5

Refer to figure II-7 as an aid in following the sequence of events described below.

- a. TZA 005001 clears the accumulator and LDX 030512 sets the first index register. The current count of the index register is cleared to 0 and the value 2 placed in it; thus, the index appears in binary as: 0000000000000010. The



VTII-1167

Figure II-7. Add a Table of Three Numbers



CHAPTER II PROGRAMMING

index register is set to 2 to cause the program to perform three passes of the loop for the three core locations in the table.

- b. The instruction ADD 125550 at core location 0502 is executed next. During the interpretation of this instruction, the computer forms the effective address to reference core for data. The contents of the address field (0550) are added to the contents of the specified index register (the first index register with a current count of 2). Therefore, the effective address is $0550 + 2$, or 0552. The result of the operation of the instruction at 0502 thus causes the contents of core location 0552 to be added to the contents of the accumulator (0 at the start of the program).
- c. If the current count of the specified index register is 0, transfer control to the instruction whose location is given in the address field of the jump instruction, JXZ. If the current count is not 0, do not jump; continue with the next instruction in sequence, DXR.
- d. The DXR instruction subtracts 1 from the current count of the index register. The current count is 2; therefore, $2 - 1 = 1$. The program goes to the next instruction JMP to 0502 which is a loop instruction to ADD 125550. The effective address for this instruction is now $0550 + 1$ or 0551; therefore, the contents of location 0551 are added to the accumulator.
- e. The process is repeated. The JXZ test does not indicate a jump; the DXR instruction at 0505 decreases the index register count by 1. The program then loops again to ADD 125550 and the resulting effective address of $0550 + 0$ adds the contents of location 0550 to the contents of the accumulator.
- f. This time the JXZ test is positive as the count of the index register is 0. The program then jumps to location 0510 and places the total in the accumulator register into location 0600.
- g. The program halts at location 0511.



CHAPTER II PROGRAMMING

3.6.3 Address Modification by Indexing

Indirect addressing is used primarily in address modification. For example, consider a situation where five instructions exist in a program, each having DATA 1 in its address field. If, for a second pass, we needed to add 1 to the address field of each of these five instructions, five or more instructions would be needed to perform the modification. With indirect addressing, each of the five original instructions could indirectly address DATA 1 and thus refer to a single register at the end of the program to obtain the effective address. Then, if modification is needed, the single register at the end of the program can be modified.

The 620 computer systems have multilevel indirect addressing. In zero-level addressing, the operand is located in the address field. In first-level or direct addressing, the address of this operand is located in the address field; the operand is one level removed from the instruction. If an operand is two levels removed, the addressing is second-level or indirect; the address of the operand (the indirect address) is located in a word whose address is in the instruction.

To indicate that the operand in an instruction is two levels removed, a special mark (called a tag or a flag) is required in the instruction. The 620 computer systems use a 7 in the operation field of the instruction word to indicate indirect addressing. The following is an example of indirect addressing. For example, the machine language word 010100 means that the contents of 100 is to be loaded into the accumulator. If this instruction is flagged as 017100, the accumulator is to be loaded with the contents of the location specified in 100; 100 is the operand address of the instruction.

3.7 SUBROUTINES

The programmer can create subroutines when mathematical routines are repeated in several places within one program. The subroutine can be entered from each point in the program where it is needed; the program returns to that point when the subroutine has been executed. The inputs to a subroutine, therefore, are called entrance parameters; the outputs from a subroutine are exit parameters. These parameters are sometimes contained in a location immediately following the branch to the main program. The subroutine branch is referred to as a call.

Subroutines are considered a single operation or instruction which operates on given data. The operation: Sum of the squares of (A), (B), (C), and (D) can be written as: SUMSQ, A/B/C/D/E. This instruction operates on four numbers and stores the sum of their squares, thus A, B, C, and D are the inputs and E is the address of the answer.



CHAPTER II PROGRAMMING

Figure II-8 illustrates the control path from a main program to a subroutine at three different locations.

3.8 CODING

The foregoing examples show routines that consist of a list of mnemonic code groups with an associated list of relative addresses. The coder converts each relative address into an instruction word so that the routine becomes a list of instruction words with an accompanying list of data words. Each of the words in these lists also has an associated address specifying where that word is to be stored in memory.

The numeric code equivalent of a mnemonic code group becomes the operation part of the instruction word. An actual address is substituted for the relative address, and this becomes the address part of the instruction word. In some cases for the repertoire used in the examples, two addresses are used in the instruction word. For example, when a word is transferred from a specified address in the input equipment to a specified address in the internal memory, the instruction word must have a format that accommodates both addresses. However, when either or both of the address parts of an instruction word are not used, the coder must fill all the bit positions in the unused parts of the word with zeros or an address might be inadvertently specified.

The instruction words as prepared by the coder can be written in either binary or octal notation, depending upon the computer requirements. In any case, the final list of instruction words as prepared on paper by the coder must be entirely in numerical expressions. Although the work of the coder does not require a high order of ingenuity or mathematical ability, it does require painstakingly close attention to detail. If even one digit is in error in the final routine, it is quite likely that the solution, if any, will be unusable.

CHAPTER II
PROGRAMMING

LOCATION

100
101
102
103
104

BODY OF
MAIN
PROGRAM

105
106

, JMPM , XSQT

107
110
111
112
113

BODY OF
MAIN
PROGRAM

114
115

, JMPM , XSQT

116
117
120
121
122

BODY OF
MAIN
PROGRAM

123
124

, JMPM , XSQT

125
126
127
130
131

BODY OF
MAIN
PROGRAM

132 XSQT, ENTR, 107-116-125

133
134
135
136
137

BODY OF
SUB-
ROUTINE

140 , JMP* , XSQT

BECAUSE ANY JMPM INSTRUCTION
SAVES THE NEXT CORE LOCATION
IN LOCATION 132₈ AND THEN
JUMPS TO THAT CELL +1, IT IS
POSSIBLE TO ENTER AND RETURN
WITHOUT LOSING TRACK OF
WHERE WE ARE.

VTII-1168

Figure II-8. Subroutines



CHAPTER II PROGRAMMING

SECTION 4 PROGRAMMING IN ASSEMBLY LANGUAGE

4.1 DAS ASSEMBLER

The 620 assembler (DAS) permits instructions, addresses, address modifiers and constants to be specified in a straightforward manner.

- a. Instruction mnemonics such as STB (Store B Register) are used in place of numeric instruction codes.
- b. Addresses can be referenced by labels rather than absolute locations.
- c. Constants can be defined without conversion to binary or octal notation.
- d. Comments can be added either between symbolic statements or with the statement itself to document the program.

Several versions of the DAS are available for the 620 computer systems: DAS 4KA, DAS 8KA, DAS MR (for use with MOS), and stand-alone MR.

4.1.1 DAS 4KA

There are two sections to the DAS 4KA assembler. The first section (I/O) allows the user to specify the type of I/O devices to be used. The second section is the assembler to be used with 4K of memory and up. DAS 4KA recognizes 620/f mnemonics.



CHAPTER II PROGRAMMING

4.1.2 DAS 8KA

There are two sections to the DAS 8KA assembler. The first section (I/O) allows the user to specify the type of I/O devices to be used. The second section is the assembler to be used with 4K of memory and up. DAS 8KA recognizes 620/f mnemonics.

4.1.3 DAS MR

The DAS MR assembler is designed to be used with the Varian Master Operating System (MOS). This enables the user to assign different I/O devices before calling the assembler. DAS MR recognizes all instructions used with the various 620 computer systems, plus several new ones including micro-programming.

4.1.4 Stand-Alone MR

Stand-Alone MR functions in the same manner as DAS MR but it can operate without the supervision of MOS thus allowing more memory for user programs.

4.2 DAS SOURCE LANGUAGE

DAS translates symbolic instructions (the source program) into binary computer code (the object program). Except for certain pseudoinstructions, each symbolic source statement will generate one or two computer words.

Computer codes generated by DAS fall into two categories, instructions and data. A source statement consists of several parts, or fields. Each source statement can contain a combination of these fields depending on the requirements of the instruction or pseudoinstruction being processed. The fields are: the label, instruction, variable and remarks fields.



CHAPTER II PROGRAMMING

4.3 STATEMENTS

4.3.1 Statement Format

A symbolic source statement has four fields: label, operation, variable, and comment. Each field is variable in length and terminated by one or more blank characters. The label, instruction, and variable fields can also be separated by commas. The label field must begin in the first character position and other fields can begin in any remaining character position; this is described as free-form. However, for convenience and uniformity of the assembly listing output, it is suggested that the beginning of each field appear in the same character position throughout an assembly.

4.3.2 Label Field

A symbolic source statement can be associated with a symbolic name or label which allows the statement to be referenced from other statements within the program. A label field is usually optional. If used, the label field must begin in the first (left-most) character position, and is terminated by a blank character or comma.

4.3.3 Operation Field

The operation field begins in the first nonblank character position following the label field, if used. If the label field is not used, the operation field begins in the first nonblank character position after the first character position. The operation field is terminated by a blank character or comma. If the operation field is absent or not definable, the statement is in error and two No Operation (NOP) instructions are generated in the object program.



CHAPTER II PROGRAMMING

4.3.4 Variable Field

The variable field, if used, must begin within the eight nonblank character positions following the operation field. If more than eight blank character positions occur after the operation field, the variable field is considered void and a value of zero (absolute) is assumed. Also, if an invalid term is encountered in the variable field, a value of zero (absolute) is assumed. The variable field contains subfields separated by commas.

4.3.5 Comment Field

The comment field is optional and is used as a documentation convenience. The contents of this field are output on the assembly listing, but otherwise have no affect upon the assembly process. The comment field begins in the first nonblank character position following the variable field, or the operation field if the variable field is absent.

4.3.6 Comment Statements

A statement with an asterisk (*) character in the first character position is entirely commentary and its contents have no affect upon the assembly process. However, the statement is output in the assembly listing.

4.3.7 Blank Statements

A statement comprising blank characters from the first character position to character position 72 is processed as a comment statement.



CHAPTER II PROGRAMMING

4.4 PROGRAMMING IN SYMBOLIC ASSEMBLY LANGUAGE

Figures II-9 through II-16 provide examples of symbolic assembly language programs. Each example is in two parts: the program as coded on a DAS coding form, and the program as it appears on the assembled listing. Refer to the comment field for a functional explanation of each instruction.



PROGRAMMER		DAS CODING FORM		PAGE 1 OF 2		varian data machines a varian subsidiary	
PROGRAM		THREE NUMBERS PROGRAM		IDENTIFICATION			
LABEL	OPERATION	VARIABLE AND COMMENT FIELD					
* GIVEN THREE NUMBERS A,B,C. COMPARE AND STORE THE LARGER							
* OF THE THREE NUMBERS IN LOCATION LRGR.							
* A=ABLE							
* B=BAKE							
* C=CAND							
*							
BEGN	ORG	0500	STARTING ADDRESS OF PROG				
	LDA	ABLE	A = AR				
	SUB	BAKE	A - B = AR				
	JAN	CKCB	AR = NEG, B IS LARG THAN A				
	LDA	ABLE	A = AR				
	SUB	CAND	A - C = AR				
	JAN	STRC	AR = NEG, C IS LARGEST				
	LDA	ABLE	AR = POS, A IS LARGEST				
STLG	STA	LRGR	LARGEST TO STORE				
	HLT	7	FLAGGED HALT				
CKCB	LDA	BAKE	B = AR				
	SUB	CAND	B - C = AR				
	JAN	STRC	AR = NEG, C IS LARGEST				
	LDA	BAKE	AR = POS, B IS LARGEST				
	JMP	STLG	JUMP TO STORE LARGEST				
STRC	LDA	CAND	C = AR				

VT12-0360

Figure II-9. Example I, Coding Form



CHAPTER II
PROGRAMMING

PROGRAMMER		DATE		PROGRAM		PAGE 2 of 2		varian data machines a varian subsidiary	
LABEL	OPERATION	VARIABLE AND COMMENT FIELD		IDENTIFICATION					
	JMP	STLG	JUMP TO STORE LARGEST						
* DATA LOCATIONS									
ABLE	DATA	1	A = 1						
BAKE	DATA	2	B = 2						
CAND	DATA	3	C = 3						
LRGR	BSS	1	STORAGE FOR LARGEST						
	END	BEGN	ADDR OF FIRST EXECUTABLE INSTRUCTION AT RUN TIME						

Figure II-9. Example I, Coding Form (continued)

VT12-0361



CHAPTER II PROGRAMMING

PAGE 000001

```

*EXAMPLE I                                THREE NUMBERS PROGRAM
*
* GIVEN THREE NUMBERS A,B,C, COMPARE AND STORE THE LARGER
* OF THE THREE NUMBERS IN LOCATION LRGR,
*
* A=ABLE
* B=BAKE
* C=CAND
*
000500                                ,ORG      ,0500      STARTING ADDRESS OF PROG
000500 010525 BEGN ,LDA ,ABLE      A = AR
000501 140526      ,SUB ,BAKE      A = B = AR
000502 001004      ,JAN ,CKCB      AR = NEG, B IS LARG THAN A
000503 000513 R      ,LDA ,ABLE      A = AR
000504 010525      ,SUB ,CAND      A = C = AR
000505 140527      ,JAN ,STRC      AR = NEG, C IS LARGEST
000506 001004      ,LDA ,ABLE      AR = POS, A IS LARGEST
000507 000522 R      ,STA ,LRGR     LARGEST TO STORE
000510 010525      ,HLT ,7         FLAGGED HALT
000511 050530 STLG ,STA ,LRGR      B = AR
000512 000007      ,LDA ,BAKE      B = C = AR
000513 010526 CKCB ,SUB ,CAND      AR = NEG, C IS LARGEST
000514 140527      ,JAN ,STRC      AR = POS, B IS LARGEST
000515 001004      ,LDA ,BAKE      JUMP TO STORE LARGEST
000516 000522 R      ,JMP ,STLG
000517 010526      ,LDA ,CAND      C = AR
000520 001000      ,JMP ,STLG      JUMP TO STORE LARGEST
000521 000511 R STRC ,LDA ,CAND
000522 010527      ,JMP ,STLG
000523 001000
000524 000511 R
*
* DATA LOCATIONS
*
000525 000001 ABLE ,DATA ,1      A = 1
000526 000002 BAKE ,DATA ,2      B = 2
000527 000003 CAND ,DATA ,3      C = 3
000530      LRGR ,RSS ,1         STORAGE FOR LARGEST
                                000500 R ,END ,BEGN      ADDR OF FIRST EXECUTABLE

```

LITERALS

POINTERS

VTII-1169

Figure II-10. Example I, Assembly Listing



CHAPTER II PROGRAMMING

SYMBOLS

1 000530 R LRGR

PAGE 000002

1 000527 R CAND

1 000526 R BAKE

1 000525 R ABLE

1 000522 R STRC

1 000513 R CKCB

1 000511 R STLG

1 000500 R REGN

VTH-1170

Figure II-10. Example I, Assembly Listing (*continued*)

CHAPTER II
PROGRAMMING

PROGRAMMER		PROGRAM		PAGE	varian data machines a varian subsidiary	
EXAMPLE J		SQUARE ROOT PROGRAM		1	3	
LABEL	OPERATION	VARIABLE AND COMMENT FIELD				IDENTIFICATION
1	2	3	4	5	6	7
*		THIS IS A ROUTINE TO CALL THE SQUARE ROOT (XSQT) SUBROUTINE.				
*		ERROR RETURN FOR SQUARE ROOT OF NEGATIVE NUMBERS IS IN CALL				
*		+2 (n+2). NORMAL RETURN FROM SQUARE ROOT IS AT CALL + 3 (n+3).				
*		THIS ROUTINE IS DESIGNED TO TAKE THE SQUARE ROOT				
*		OF 40 OCTAL NUMBERS AND STORE THE ANSWER IN 40 OCTAL LOC.				
*						
	ORG	0500	STARTING ADDRESS			
	LDXI	037	XR = COUNT - 1			
NEXT	LDB	LOC, 1	BR = (LOC + XR)			
	CALL	XSQT, 0777	SUBR CALL WITH ERROR RETURN			
	STB	SQRT, 1	NORMAL RETURN STORE RESULT			
*						
*		NOTE THAT THE DATA IS RETRIEVED AND STORED FROM				
*		BOTTOM TO TOP				
*						
	JXZ	HALT	XR = 0 END OF ROUTINE			
	DXR		INDEX - 1 = INDEX			
	JMP	NEXT	RETURN FOR NEXT NUMBER			
HALT	HLT		NORMAL HALT			
LOC	DATA	25, 30, 36, 050, -1, 100, 0100, 0, 4, 200				
	DATA	1000, 0700, -40, 50, 60, 70, 80, 90, 110, 120				
	DATA	0, 02000, 2, 9, 3000, 03000, 15, 17, 130, 0140				
	DATA	0204, 300, 310, 320, 330, 340, 350, 400, 500, -10				

VT12-0362

Figure II-11. Example J, Coding Form

PROGRAMMER		DATE		PROGRAM		PAGE		varian data machines	
						2 OF 3		a varian subsidiary	
* LABEL		OPERATION		VARIABLE AND COMMENT FIELD		IDENTIFICATION			
1	2	3	4	5	6	7	8	9	10
SQRT	BSS	040							
RESERVE 40 OCTAL LOCATIONS									
* INTEGER SQUARE ROOT SUBROUTINE CALCULATED BY THE APPROXIMATION									
* $\frac{1}{2} (X_i + \frac{A}{X_i}) = X_{i+1}$ (DO NOT KEY PUNCH)									
* ENTER WITH NUMBER FOR SQUARE ROOT IN THE B REGISTER. THE									
* X REGISTER IS SAVED AND REPLACED ON EXIT. ERROR RETURN FOR									
* SQUARE ROOT OF NEGATIVE NUMBERS AT n+2 FROM CALL.									
* NORMAL RETURN AT n+3 FROM CALL WITH SQUARE ROOT OF NUMBER									
* IN THE B REGISTER									
XSQT	ENTR								
PLACE WHERE RETURN ADDR IS SAVED									
	JBZ	EXIT+1							
SQ RT. OF 0=0									
	TBA								
NUMBER = BR = AR									
	JAN*	XSQT							
ERROR RETURN TO N+2									
	STB	NMBR							
SAVE NUMBER									
	STB	APRX							
NUMBER = 1ST APPROXIMATION									
	STX	SAVE							
SAVE XR									
	LDXI	7							
INITIALIZE XR FOR APPR.									
AGN	TZA								
ZERO AR FOR DIVIDE									
	LDB	NMBR							
NUMBER = BR									
	DIY	APRX							
NUMBER / APPROXIMATION									
	TBA								
A/X = BR = AR									

VTI2-0363

Figure II-11. Example J, Coding Form (continued)





CHAPTER II
PROGRAMMING

DAS CODING FORM															PAGE 3 OF 3		VDM varian data machines a varian subsidiary		
PROGRAMMER					DATE					PROGRAM					IDENTIFICATION				
* LABEL					OPERATION					VARIABLE AND COMMENT FIELD									
1					2					3					4				
6					7					8					9				
12					13					14					15				
16					17					18					19				
20					21					22					23				
24					25					26					27				
28					29					30					31				
32					33					34					35				
36					37					38					39				
40					41					42					43				
44					45					46					47				
48					49					50					51				
52					53					54					55				
56					57					58					59				
60					61					62					63				
64					65					66					67				
68					69					70					71				
72					73					74					75				
76					77					78					79				
80					81					82					83				
84					85					86					87				
88					89					90					91				
92					93					94					95				
96					97					98					99				
100					101					102					103				
104					105					106					107				
108					109					110					111				
112					113					114					115				
116					117					118					119				
120					121					122					123				
124					125					126					127				
128					129					130					131				
132					133					134					135				
136					137					138					139				
140					141					142					143				
144					145					146					147				
148					149					150					151				
152					153					154					155				
156					157					158					159				
160					161					162					163				
164					165					166					167				
168					169					170					171				
172					173					174					175				
176					177					178					179				
180					181					182					183				
184					185					186					187				
188					189					190					191				
192					193					194					195				
196					197					198					199				
200					201					202					203				
204					205					206					207				
208					209					210					211				
212					213					214					215				
216					217					218					219				
220					221					222					223				
224					225					226					227				
228					229					230					231				
232					233					234					235				
236					237					238					239				
240					241					242					243				
244					245					246					247				
248					249					250					251				
252					253					254					255				
256					257					258					259				
260					261					262					263				
264					265					266					267				
268					269					270					271				
272					273					274					275				
276					277					278					279				
280					281					282					283				
284					285					286					287				
288					289					290					291				
292					293					294					295				
296					297					298					299				
300					301					302					303				
304					305					306					307				
308					309					310					311				
312					313					314					315				
316					317					318					319				
320					321					322					323				
324					325					326					327				
328					329					330					331				
332					333					334					335				
336					337					338					339				
340					341					342					343				
344					345					346					347				
348					349					350					351				
352					353					354					355				
356					357					358					359				
360					361					362					363				
364					365					366					367				
368					369					370					371				
372					373					374					375				
376					377					378					379				
380					381					382					383				
384					385					386					387				
388					389					390					391				
392					393					394					395				
396					397					398					399				
400					401					402					403				
404					405					406					407				
408					409					410					411				
412					413					414					415				
416					417					418					419				
420					421					422					423				
424					425					426					427				
428					429					430					431				
432					433					434					435				
436					437					438					439				
440					441					442					443				
444					445					446					447				
448					449					450					451				
452					453					454					455				
456					457					458					459				
460					461					462					463				
464					465					466					467				
468					469					470					471				
472					473					474					475				
476					477					478					479				
480					481					482					483				
484					485					486					487				
488					489					490					491				
492					493					494					495				
496					497					498					499				
500					501					502					503				
504					505					506					507				
508					509					510					511				
512					513					514					515				
516					517					518					519				
520					521					522					523				
524					525					526					527				
528					529					530					531				
532					533					534					535				
536					537					538					539				
540					541					542					543				
544					545					546					547				
548					549					550					551				
552					553					554					555				
556					557					558					559				
560					561					562					563				
564					565					566					567				
568					569					570					571				
572					573					574					575				
576					577					578					579				
580					581					582					583				
584					585					586					587				
588					589					590					591				
592					593					594					595				
596					597					598					599				
600					601					602					603				
604					605					606					607				
608					609					610					611				
612					613					614					615				
616					617					618					619				
620					621					622					623				
624					625					626					627				
628					629					630					631				
632					633					634					635				
636					637					638					639				
640					641					642					643				
644					645					646					647				
648					649					650					651				
652					653					654					655				
656					657					658					659				
660					661					662					663				
664					665					666					667				
668					669					670					671				
672					673					674					675				
676					677					678					679				
680					681					682					683				
684					685					686					687				
688					689					690					691				
692					693					694					695				
696					697					698					699				
700					701					702					703				
704					705					706					707				
708					709					710					711				
712					713					714					715				
716					717					718					719				
720					721					722					723				
724					725					726					727				
728					729					730					731				
732					733					734					735				
736					737					738					739				
740					741					742					743				
744					745					746					747				
748					749					750					751				
752					753					754					755				
756					757					758					759				
760					761					762					763				
764					765					766					767				
768					769					770					771				
772					773					774					775				
776					777					778					779				
780					781					782					783				
784					785					786					787				
788					789					790					791				
792					793					794					795				
796					797					798					799				
800					801					802					803				
804					805					806					807				
808					809					810					811				
812					813					814					815				
816					817					818					819				
820					821					822					823				
824					825					826					827				
828					829					830					831				
832					833					834					835				
836					837					838					839				
840					841					842					843				
844					845					846					847				
848					849					850					851				
852					853					854					855				
856					857					858					859				
860					861					862					863				
864					865					866					867				
868					869					870					871				
872					873					874					875				
876					877					878					879				
880					881					882					883				
884					885					886					887				
888					889					890					891				
892					893					894					895				
896					897					898					899				
900					901					902					903				
904					905					906					907				
908					909					910					911				
912					913					914					915				
916					917					918					919				
920					921					922					923				
924					925					926					927				
928					929					930					931				
932					933					934					935				
936					937					938					939				
940					941					942					943				
944					945					946					947				
948					949					950					951				
952					953					954					955				
956					957					958					959				
960					961					962					963				
964					965					966					967				
968					969					970					971				
972					973					974					975				
976					977					978					979				
980					981					982					983				
984					985					986					987				
988					989					990					991				
992					993					994					995				
996					997					998					999				
1000					1001					1002					1003				
1004					1005					1006					1007				
1008					1009					1010					1011				
1012					1013					1014					1015				
1016					1017					1018									



CHAPTER II PROGRAMMING

PAGE 000001

```

*EXAMPLE J                                SQUARE ROOT PROGRAM
*
* THIS A ROUTINE TO CALL THE SQUARE ROOT (XSQT) SUBROUTINE.
* ERROR RETURN FOR SQUARE ROOT OF NEGATIVE NUMBERS IS IN CALL
* +2 (N+2) NORMAL RETURN FROM SQUARE ROOT IS AT CALL + 3 (N+3)
* THIS ROUTINE IS DESIGNED TO TAKE THE SQUARE ROOT
* OF 40 OCTAL NUMBERS AND STORE THE ANSWER IN 40 OCTAL LOC,
*
000500      ,ORG      ,0500      STARTING ADDRESS
000500 006030      ,LDXI     ,037      XR = COUNT = 1
000501 000037
000502 025515      NEXT ,LDB     ,LOC,1      BR = (LOC + XR)
000503 002000      ,CALL     ,XSQT,0777      SUBR CALL WITH ERROR RETURN
000504 000626 R
000505 000777
000506 065566      ,STB      ,SQRT,1      NORMAL RETURN STORE RESULT
*
* NOTE THAT THE DATA IS RETRIEVED AND STORED FROM
* BOTTOM TO TOP
*
000507 001040      ,JXZ      ,HALT      XR = 0 END OF ROUTINE
000510 000514 R
000511 005344      ,DXR      ,
000512 001000      ,JMP      ,NEXT      INDEX = 1 = INDEX
000513 000502 R      RETURN FOR NEXT NUMBER
000514 000000      HALT ,HLT      ,
000515 000031      LOC ,DATA     ,25,30,36,050,-1,100,01,00,0,4,200
000516 000036
000517 000044
000520 000050
000521 177777
000522 000144
000523 000001
000524 000000
000525 000000
000526 000004
000527 000310
000530 001750      ,DATA     ,1000,0700,-40,50,60,70,80,90,110,120
000531 000700
000532 177730
000533 000062
000534 000074

```

VTH-1171

Figure II-12. Example J, Assembly Listing

CHAPTER II
PROGRAMMING

PAGE 000002

```

000535 000106
000536 000120
000537 000132
000540 000136
000541 000170
000542 000000
000543 002000
000544 000002
000545 000011
000546 005670
000547 003000
000550 000017
000551 000021
000552 000202
000553 000001
000554 000204
000555 000454
000556 000466
000557 000500
000560 000512
000561 000524
000562 000536
000563 000620
000564 000764
000565 177766
000566

```

```

,DATA ,0,02000,2,9,3000,03000,15,17,130,01 40

```

```

,DATA ,0204,300,310,320,330,340,350,400,500,-10

```

```

SQRT ,RSS ,040 RESERVE 40 OCTAL LOCATIONS
*
* INTEGER SQUARE ROOT SUBROUTINE CALCULATED BY THE APPROXIMATION
*
*  $1/2 (X + \frac{A}{X_1}) = X_1 + 1$ 
*
* ENTER WITH NUMBER FOR SQUARE ROOT IN THE B REGISTER. THE
* X REGISTER IS SAVED AND REPLACED ON EXIT. ERROR RETURN FOR
* SQUARE ROOT OF NEGATIVE NUMBERS AT N+2 FROM CALL.
* NORMAL RETURN AT N+3 FROM CALL WITH SQUARE ROOT OF NUMBER
* IN THE B REGISTER
*
XSQT ,ENTR , PLACE WHERE RETURN ADDR IS SAVED
,JBZ ,EXIT+1 SQ RT. OF 0=0
, TBA , NUMBER = BR = AR
,JAN* ,XSQT ERROR RETURN TO N+2

```

```

000626 000000
000627 001020
000630 000657 R
000631 005021
000632 001004

```

VTII-1172

Figure II-12. Example J, Assembly Listing (continued)



CHAPTER II PROGRAMMING

PAGE 000003

000633	100626 R				
000634	060662		,STB	,NMBR	SAVE NUMBER
000635	060663		,STB	,APRX	NUMBER = 1ST APPROXIMATION
000636	070664		,STX	,SAVE	SAVE XR
000637	006030		,LDXI	,7	INITIALIZE XR FOR APPR.
000640	000007				
000641	005001	AGN	,TZA	,	ZERO AR FOR DIVIDE
000642	020662		,LDB	,NMBR	NUMBER = BR
000643	170663		,DIV	,APRX	NUMBER / APPROXIMATION
000644	005021		,TBA	,	A/X = BR = AR
000645	120663		,ADD	,APRX	A/X+X = AR
000646	005012		,TAB	,	A/X+X = AR = BR
000647	004101		,ASRB	,1	(A/X+X)1/2 = BR
000650	060663		,STB	,APRX	NEXT APPROXIMATION
000651	005344		,DXR	,	XR = 1 = XR
000652	001040		,JXZ	,EXIT	SQ RT. = BR
000653	000656 R				
000654	001000		,JMP	,AGN	COMPLETE APPROXIMATION
000655	000641 R				
000656	030664	EXIT	,LDX	,SAVE	RESTORE XR
000657	040626		,INR	,XSQT	UPDATE ENTRY TO N+2
000660	001000		,RETU*	,XSQT	GO BACK TO MAIN PROGRAM
000661	100626 R				
000662		NMBR	,RSS	,1	
000663		APRX	,RSS	,1	
000664		SAVE	,RSS	,1	
	000000		,END	,	NO EXECUTION ADDRESS

LITERALS

POINTERS

SYMBOLS

1	000664 R	SAVE
1	000663 R	APRX
1	000662 R	NMBR
1	000656 R	EXIT
1	000641 R	AGN
1	000626 R	XSQT
1	000566 R	SQRT

VIII-1173

Figure II-12. Example J, Assembly Listing (continued)



CHAPTER II PROGRAMMING

PAGE 000004

```
1 000515 R LQC
1 000514 R HALT
1 000502 R NEXT
```

VTII-1174

Figure II-12. Example J, Assembly Listing (continued)

PROGRAMMER		DATE		PROGRAM		PAGE 1 OF 1	varian data machines a varian subsidiary
* EXAMPLE K				INSTRUCTION EXAMPLES			
LABEL	OPERATION	VARIABLE AND COMMENT FIELD		IDENTIFICATION			
1	2	3	4	5	6	7	8
	ORG	010000		STARTING LOCATION OF PROGRAM			
	ZERØ	7		ZERØ A, B, X REGISTERS			
	DECR	1		AR = -1			
	INCR	2		BR = +1			
	DAR			AR = AR-1			
	MERGE	034		INCLUSIVE ØR ØF A+B INTO XR			
	LDAI	NMBR		AR = NMBR			
	STA	LØC		AFTER ASSEMBLY THIS IS REL			
	JMP	CØNT					
NMBR	EQU	10		NMBR = 10 DECIMAL			
LØC	BSS	1		RESERVE 1 LOCATION			
CØNT	LDA	LØC		GENERATE INDIRECT PØINTER			
	STA*	LØC		ALSO GENERATE INDIRECT PØINTER			
	LDXI	050					
	LDAE	NMBR, 1		LDA CONTENTS ØF LOCATION 62 ØCTAL			
	LDB	=3		GENERATE LITERAL			
	SEN	0101, *+5		SENSE WRITE REGISTER ØF TTY READY			
	NØP						
	JMP	*-3		JUMP BACK IF NOT READY			
	ØME	Ø1, CHAR		ØUTPUT (CHAR) TO TTY			
	HLT	Ø777		FLAGGED HALT			
CHAR	DATA	'A'		ASCII FOR A			
	END						

VT12-0365

Figure II-13. Example K, Coding Form



CHAPTER II
PROGRAMMING

PAGE 000001

		*EXAMPLE K	INSTRUCTION EXAMPLES
010000		,ORG ,010000	STARTING LOCATION OF PROGRAM
010000	005007	,ZERO ,7	ZERO A,B,X REGISTERS
010001	005301	,DECR ,1	AR = -1
010002	005102	,INCR ,2	BR = +1
010003	005311	,DAR ,	AR = AR-1
010004	005034	,MERGE ,034	INCLUSIVE OR OF A+B INTO XR
010005	006010	,LOAI ,NMBR	AR = NMBR
010006	000012		
010007	054002	,STA ,LOC	AFTER ASSEMBLY THIS IS REL
010010	001000	,JMP ,CONT	
010011	010013 R		
	000012	NMBR ,EQU ,10	NMBR = 10 DECIMAL
010012		LOC ,BSS ,1	RESERVE 1 LOCATION
010013	017200 I	CONT ,LOA ,LOC	GENERATE INDIRECT POINTER
010014	057201 I	,STA* ,LOC	ALSO GENERATE INDIRECT POINTER
010015	006030	,LDXI ,050	
010016	000050		
010017	006015	,LDAE ,NMBR,1	LDA CONTENTS OF LOCATION 62 OCTAL
010020	000012		
010021	021000	,LDR ,#3	GENERAL LITERAL
010022	101101	,SEN ,0101,*+5	SENSE WRITE REGISTER OF TTY READY
010023	010027 R		
010024	005000	,NOP ,	
010025	001000	,JMP ,*-3	JUMP BACK IF NOT READY
010026	010022 R		
010027	103001	,OME ,01,CHAR	OUTPUT (CHAR) TO TTY
010030	010032 R		
010031	000777	,HLT ,0777	FLAGGED HALT
010032	000301	CHAR ,DATA ,'A'	ASCII FOR A
	000000	,END ,	
LITERALS			
001000	000003		
POINTERS			
000200	010012		
000201	110012		
SYMBOLS			
1	010032 R	CHAR	

VII-11'S

Figure II-14. Example K, Assembly Listing



CHAPTER II PROGRAMMING

PAGE 000002
1 010013 R CONT
1 010012 R LOC
1 000012 NMBR

VTII-II76

Figure II-14. Example K, Assembly Listing (continued)

CHAPTER II PROGRAMMING


PROGRAMMER		DAS CODING FORM		PAGE		 varian data machines a varian subsidiary	
PROGRAM		PROGRAM		PAGE		OF	
*EXAMPLE 1		EXAMPLE WITH ERRORS					
LABEL	OPERATION	VARIABLE AND COMMENT FIELD	IDENTIFICATION				
1	2	3	4	5	6	7	8
	ORG	015000					
	TZA	010	CANNOT HAVE A VAR. FIELD				
SEC	TZA						
	HLT	777	VARIABLE FIELD TOO LARGE				
	HLT	0777					
	LDA	ALFA,1	EXP 1 TOO LARGE				
	LDAE	ALFA,1					
	LDXI	ALFA					
SEC	LDA	0,1	DOUBLE DEFINITION				
	LDA	0,4	EXP 2 HAS TO BE A 1 OR 2				
	LDA	0,1					
	LDA	0,2					
	LDA	ALFA	CREATE A REL ADDRESS				
	LDAI	77777	VAR FIELD TOO LARGE				
	LDAI	077777					
	LDAI	32767					
	LDAI	-32768					
	JZZ	ALFA	ILLEGAL OPERATION CODE				
	JXZ	ALFA					
	JMP	BRA	BRA UNDEFINED				
	JMP	BRAU					
ALFA	DATA	5					
BRAU	DATA	014045					
STR?	BSS	1					

Figure II-15. Example L, Coding Form



CHAPTER II PROGRAMMING

PAGE .00001

		*EXAMPLE L		EXAMPLE WITH ERRORS
015000		,ORG	,015000	
015000	005011	,TZA	,010	CANNOT HAVE A VAR. FIELD
*SZ				
015001	005001	SEC ,TZA	,	
*DD				
015002	001411	,HLT	,777	VARIABLE FIELD TO LARGE
*SZ				
015003	000777	,HLT	,0777	
015004	015036	,LDA	,ALFA,1	EXP 1 TO LARGE
*AD				
015005	006015	,LDAE	,ALFA,1	
015006	015036 R			
015007	006030	,LDXI	,ALFA	
015010	015036 R			
015011	015000	SEC ,LDA	,0,1	DOUBLE DEFINITION
*DD				
015012	000004	,LDA	,0,4	EXP 2 HAS TO BE A 1 OR 2
*TF				
015013	015000	,LDA	,0,1	
015014	016000	,LDA	,0,2	
015015	014020	,LDA	,ALFA	CREATE A REL ADDRESS
015016	006010	,LDAI	,77777	VAR FIELD TO LARGE
*SZ				
015017	027721			
015020	006010	,LDAI	,077777	
015021	077777			
015022	006010	,LDAI	,32767	
015023	077777			
015024	006010	,LDAI	,-32768	
015025	100000			
		,JZZ	,ALFA	ILLEGAL OPERATION CODE
*NP				
015030	001040	,JXZ	,ALFA	
015031	015036 R			
015032	001000	,JMP	,BRA	BRA UNDEFINED
*SY				
015033	000000			
015034	001000	,JMP	,BRAV	
015035	015037 R			
015036	000005	ALFA ,DATA	,5	
015037	014045	BRAV ,DATA	,014045	

VIII-1177

Figure II-16. Example L, Assembly Listing



CHAPTER II PROGRAMMING

PAGE 000002

015040 000000 STR ,BSS ,1
 ,END ,

LITERALS

POINTERS

SYMBOLS

0	015040	R	STP
1	015037	R	BRAV
1	015036	R	ALFA
0	015001	R	SEC



CHAPTER III

COMPUTER OPERATION



varian data machines



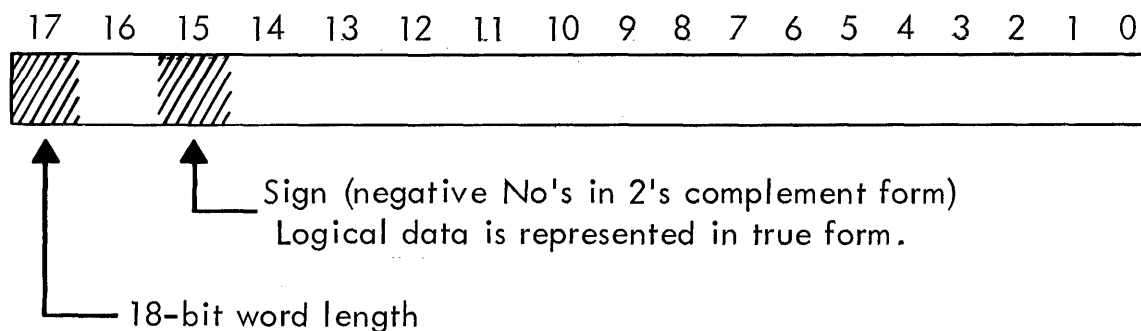
CHAPTER III COMPUTER OPERATION

SECTION 1 WORD FORMATS

1.1 INTRODUCTION

Word formats for the 620 series computers are divided into two categories: data words and instruction words. Each category has been optimized for the system environment. 620/f and 620/L systems are available only in 16-bit word lengths; the 622/i system has an 18-bit word length. The data format is extendable for 18-bit words with the sign bit in the high-order positions (refer to figure III-1).

There are four instruction word formats: single-word, double-word, generic, and macro-instruction.



INDIRECT ADDRESS FORMAT

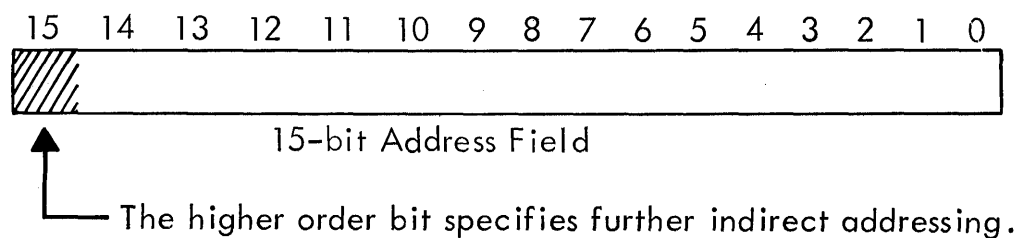


Figure III-1. Formats for Data Words and Indirect Addresses



CHAPTER III COMPUTER OPERATION

1.2 SINGLE-WORD INSTRUCTIONS

1.2.1 Addressing

There are 12 basic instructions and two optional instructions that have single-word memory reference formats (addressing). The single-word instruction is divided into three fields as shown in figure III-2. There are five addressing modes including: direct addressing to 2,048 words, relative to P with a delta range of 512, indexing with the X or B register, and indirect from the contents of the memory location addressed.

Single-word addressing instructions include: LDA, LDB, LDX, INR, ADD, SUB, MUL, STA, STB, STX, ERA, ORA, ANA, and DIV. All basic single-word instructions are executed in two cycles (except INR, MUL and DIV), including relative and indexed addressing modes. In addition, one cycle is added for each level of indirect addressing.

The single-word addressing instruction format is designed to enable the system user to write his programs with a minimum number of memory addresses and to execute these programs in minimum time. The format is uncomplicated and the fields divide into convenient octal groupings so that programs can be written and checked rapidly.

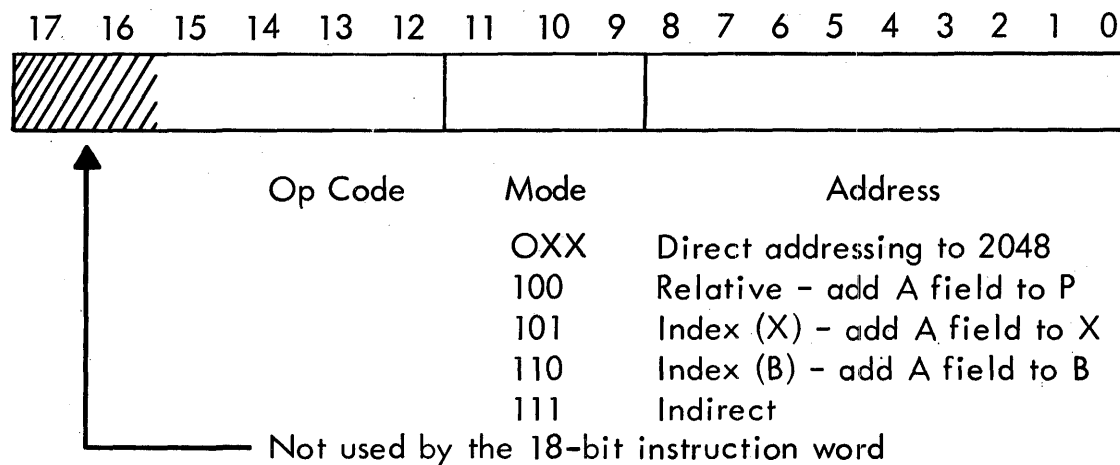


Figure III-2. Single-Word Instruction Format



CHAPTER III COMPUTER OPERATION

1.2.2 Nonaddressing

Twenty-six instructions are single-word nonaddressing. Each divides into three fields of class codes, operation codes, and definitions as illustrated in figure III-3.

These instructions perform arithmetic unit, control unit, and input/output functions. The operations are: halt, register change, shift (12), overflow (2), external control, and input/output for the A and B registers (11).

The shift instructions can shift up to 31 places. The sense and external function instructions can address up to 64 peripheral devices and define up to eight functions. The input and output commands can select A or B, A and B, and clear and input to A or B, A and B. The input/output instructions can address up to 64 devices. (The in-memory and out-memory instructions and the sense command are two-word instructions.)

The single-word nonaddressing instructions are octal-grouped for user convenience. They provide flexibility for input/output processing.

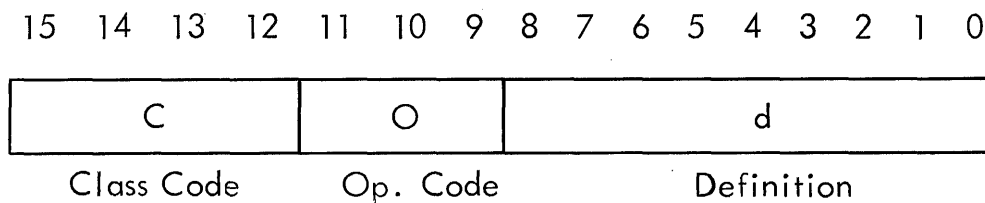


Figure III-3. Single-Word Nonaddressing Instructions



CHAPTER III COMPUTER OPERATION

1.3 TWO-WORD INSTRUCTIONS

There are two classes of two-word instructions and six types: jump, jump and mark, execute, immediate, in/out memory, and sense. The 620/f computer has the additional two-word instructions IJMP, JSR, SRE, and BT. The two-word instruction format is illustrated in figure III-4.

There are a total of 45 standard and over 16 optional two-word instructions. The efficiency and power of the two-word instructions becomes more and more apparent with use. They provide direct and random addressing and accessing of up to 32,768 words. In most cases, they permit a two memory location sequence of instruction to replace the usual three memory location sequence. The amount of memory conserved and time saved by these instructions depends upon the application, and ranges from 5 to 25 percent.

1.3.1 Jump, Jump and Mark, and Execute Instructions

The first word of the jump, jump and mark, and execute instructions contains three fields: the C field containing the class code, the O field containing the operation code, and the condition field specifying any combination of nine conditions. The conditions are: SS1, SS2, SS3, X = 0, B = 0, A = 0, A Negative, A Positive, and Overflow. On the 620/f, if bits 1 and 2 are on, the other bits specify not conditions.

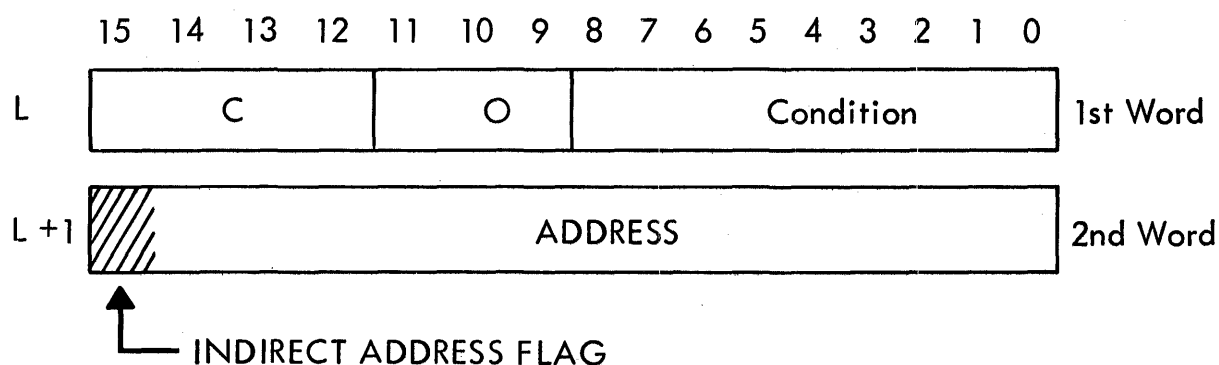


Figure III-4. Two-Word Instruction Format



CHAPTER III COMPUTER OPERATION

The second word contains the jump address, the jump and mark address, or the address of the instruction to be executed. If the specified conditions of the first word are met, the instruction is executed. If the conditions are not met, the second word is skipped and the P register is incremented.

1.3.2 Memory In/Out Instructions

The memory in/out instructions have a format similar to that of the instructions discussed in section 1.3.1. The condition field of the IME/OME instructions addresses the selected device; the second word contains the memory for the data. Indirect addressing is not permitted.

1.3.3 Immediate Instructions

The immediate instructions have a special two-word format as illustrated in figure III-5. There are 12 immediate instructions plus two that are optional; these are: LDAI, LDBI, LDXI, ADDI, SUBI, INRI, MULI, STAI, STBI, STXI, ERAI, ORAI, ANNAI, and DIVI.

Bits 3 through 6 define one of the immediate instructions listed above. These instructions provide literal addressing which contains the operand in the operand address field. They

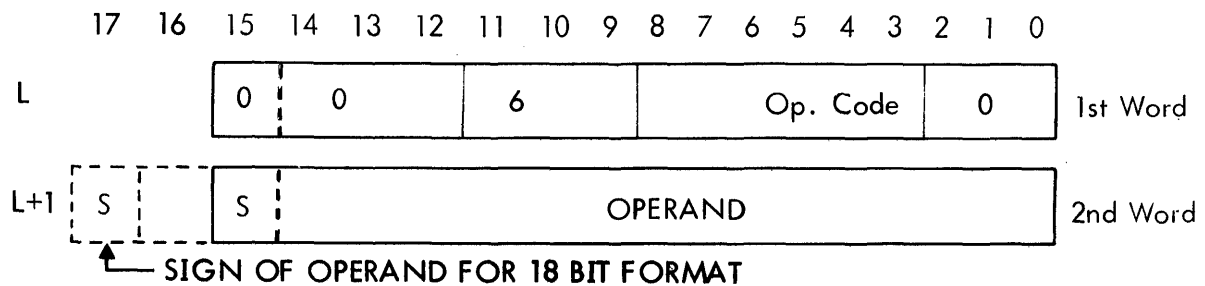


Figure III-5. Immediate Instruction Format



CHAPTER III COMPUTER OPERATION

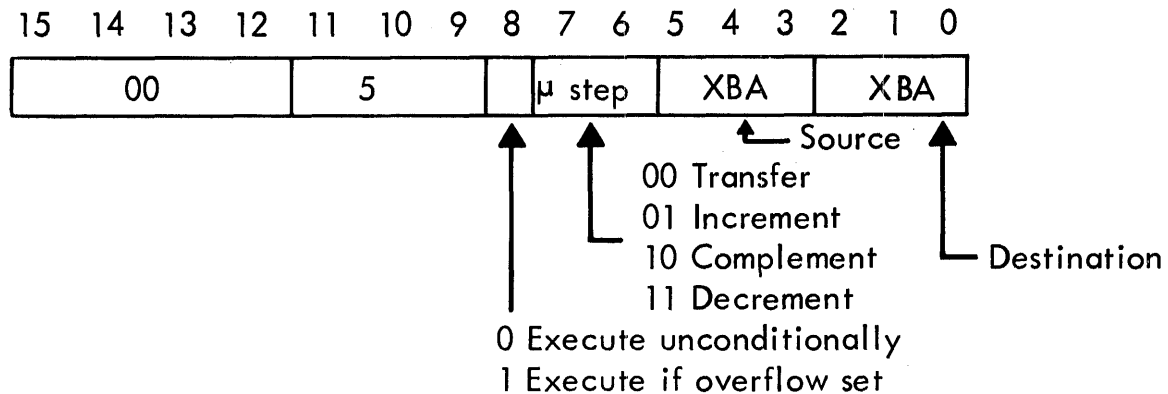


Figure III-6. Macro-Command Format

automatically increment the P counter after the execution, and the next instruction is obtained from $P + 2$.

1.4 MACRO-INSTRUCTIONS

A number of micro-steps are programmable into a macro-instruction with the single word Macro-Command. This command has over 128 useful combinations including those listed in the instruction set (section 1.5). The macro-command format is illustrated in figure III-6.

The X, B, and A Register contents can be logically ORed, cleared, transferred, set to a common value, complemented, NORed, incremented, decremented, and, if desired, perform the above conditionally on an overflow. Sequences of micro-commands can be used to perform additional logical functions customary in a system environment.

1.5 INSTRUCTION LIST

Figure III-7 provides a list of the instructions available with the 620 series computers. Certain instructions are notated for their specific application to the 620/f computer. In addition, all multiply/divide and extended addressing is optional in the 620/i and 620/L systems.



CHAPTER III COMPUTER OPERATION

Varian 620 and V73 Computer Systems							
Instn.	Octal Code	Instn.	Octal Code	Instn.	Octal Code	Instn.	Octal Code
ADD	120000	INRE	00604z	LASR	004500 ⁺ⁿ	STBI	006060
ADDE	00612z	INRI	006040	LDA	010000	STX	070000
ADDI	006120	IXR	005144	LDAE	00601z	STXE	00607z
ANA	150000	JAN	001004	LDAI	006010	STXI	006070
ANAE	00615z	JANM	002004	LDB	020000	SUB	140000
ANAI	006150	JANZ@	001016	LDBE	00602z	SUBE	00614z
AOFA	005511	JANZM@	002016	LDBI	006020	SUBI	006140
AOFB	005522	JAP	001002	LDX	030000	TAB	005012
AOFX	005544	JAPM	002002	LDXE	00603z	TAX	005014
ASLA	004200 ⁺ⁿ	JAZ	001010	LDXI	006030	TBA	005021
ASLB	004000 ⁺ⁿ	JAZM	002010	LLRL	004440 ⁺ⁿ	TBX	005024
ASRA	004300 ⁺ⁿ	JBNZ@	001026	LLSR	004540 ⁺ⁿ	TSA@	007402
ASRB	004100 ⁺ⁿ	JBNZM@	002026	LRLA	004240 ⁺ⁿ	TXA	005041
BT@	0064vw	JBZ	001020	LRLB	004040 ⁺ⁿ	TXB	005042
CIA	1025xx	JBZM	002020	LSRA	004340 ⁺ⁿ	TZA	005001
CIAB	1027xx	JMP	001000	LSRB	004140 ⁺ⁿ	TZB	005002
CIB	1026xx	JMPM	002000	MUL	160000	TZX	005004
CPA	005211	JOE	001001	MULE	00616z	XAN	003004
CPB	005222	JOEM	002001	MULI	006160	XANZ@	003016
CPX	005244	JOEN@	001007	NOP	005000	XAP	003002
DAR	005311	JOENM@	002007	OAR	1031xx	XAZ	003010
DBR	005322	JS1M	002100	OAB	1033xx	XBNZ@	003026
DIV	170000	JS2M	002200	OBR	1032xx	XBZ	003020
DIVE	00617z	JS3M	002400	OME	1030xx	XEC	003000
DIVI	006170	JSR@	00650x	ORA	110000	XOF	003001
DXR	005344	JSS1	001100	ORAE	00611z	XOFN@	003007
ERA	130000	JSS2	001200	ORAI	006110	XS1	003100
ERAE	00613z	JSS3	001400	ROF	007400	XS2	003200
ERAI	006130	JS1N@	001106	SEN	101xxx	XS3	003400
EXC	100xxx	JS2N@	001206	SOF	007401	XS1N@	003106
HLT	000xxx	JS3N@	001406	SOFA	005711	XS2N@	003206
IAR	005111	JS1NM@	002106	SOFB	005722	XS3N@	003406
IBR	005122	JS2NM@	002206	SOFX	005744	XXNZ@	003046
IJMP@	00670x	JS3NM@	002406	SRE@	0066yx	XXZ	003040
IME	1020xx	JXNZ@	001046	STA	050000	W=0-15	Y=1,2,4
INA	1021xx	JXNZM@	002046	STAE	00605z	X=0-7	Z=4-7
INAB	1023xx	JXZ	001040	STAI	006050	V=0-3	
INB	1022xx	JXZM	002040	STB	060000		
INR	040000	LASL	004400 ⁺ⁿ	STBE	00606z		

@ Instruction unique to 620/f, 620/f-100, and V73.

All MUL/DIV and Extended Addressing is optional with Varian 620/i.

VTII-1831

Figure III-7. 620 Series Instruction List



CHAPTER III COMPUTER OPERATION

SECTION 2 PAPER TAPE FORMATS

2.1 SOURCE TAPE FORMAT

Source tapes for the Varian computer systems are normally generated by using a Teletype in off-line mode. The Varian 33/35 ASR Teletype will always punch channel 8 as illustrated in figure III-8.

Source tapes for the 4KA, 8KA, and MR assemblers and the source editor program (EDIT) all use the same format. As developed by Varian, this format is a modified ASCII code of eight bits, using one frame of tape per character. Varian part numbers for source tapes are labeled with a T; for example, 92T0201-054A.

2.2 BOOTSTRAP FORMAT

The first part of the binary load/dump program and the executive routine require the bootstrap format (figure III-9). This format is loaded with a bootstrap routine, normally located in memory location 07756 through 07775.

Bit 8 is only used for leader and trailer. Bit 7 is always the logical complement of bit 6, and bits 6 through 1 contain two octal numbers. Three frames make up one 18-bit word (six octal characters). Bit 6 of the first frame is the most significant bit (MSB), and bit 1 of the third frame is the least significant bit (LSB) of the word. The first valid frame (first binary frame) is the first channel 7 punch. The feed hole is located between channels 3 and 4.

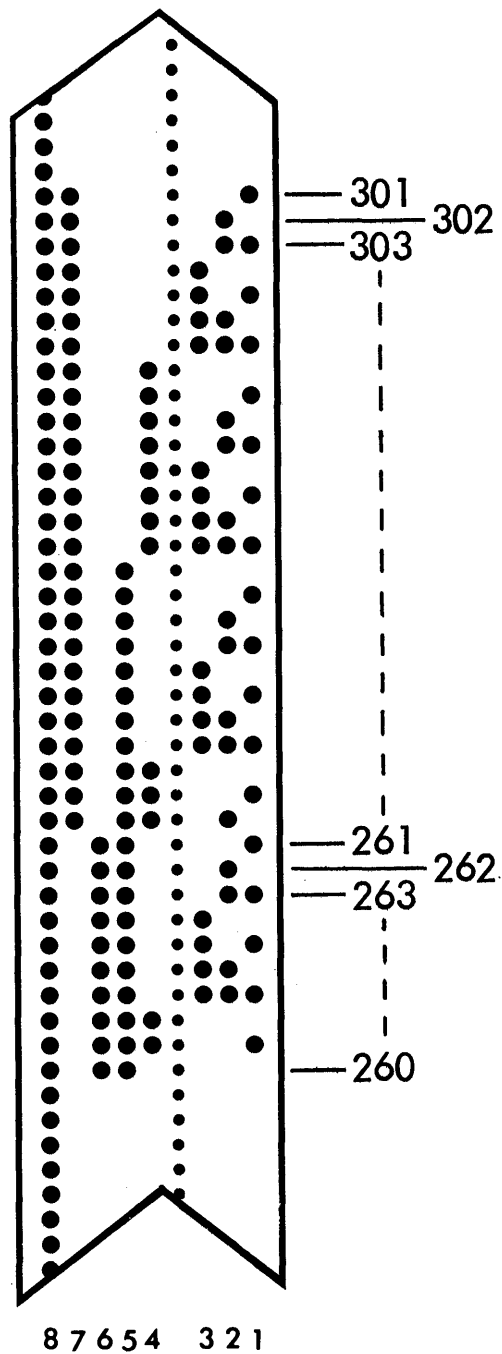
2.3 BINARY OBJECT (PROGRAM OBJECT) FORMAT

All tapes labeled object (except for stand-alone FORTRAN tapes) are in the binary object format. These tapes may be identified by the use of a U in the Varian part number; for example, 92U0107-001.

In this format, bit 8 is used only for leader or trailer, bit 7 is always the logical complement of bit 6 (except for record marks), and bits 6 through 1 contain data. Three frames comprise a word with bit 6 of the first frame as the most significant bit and bit 1 of the third frame as the least significant bit.



CHAPTER III
COMPUTER OPERATION

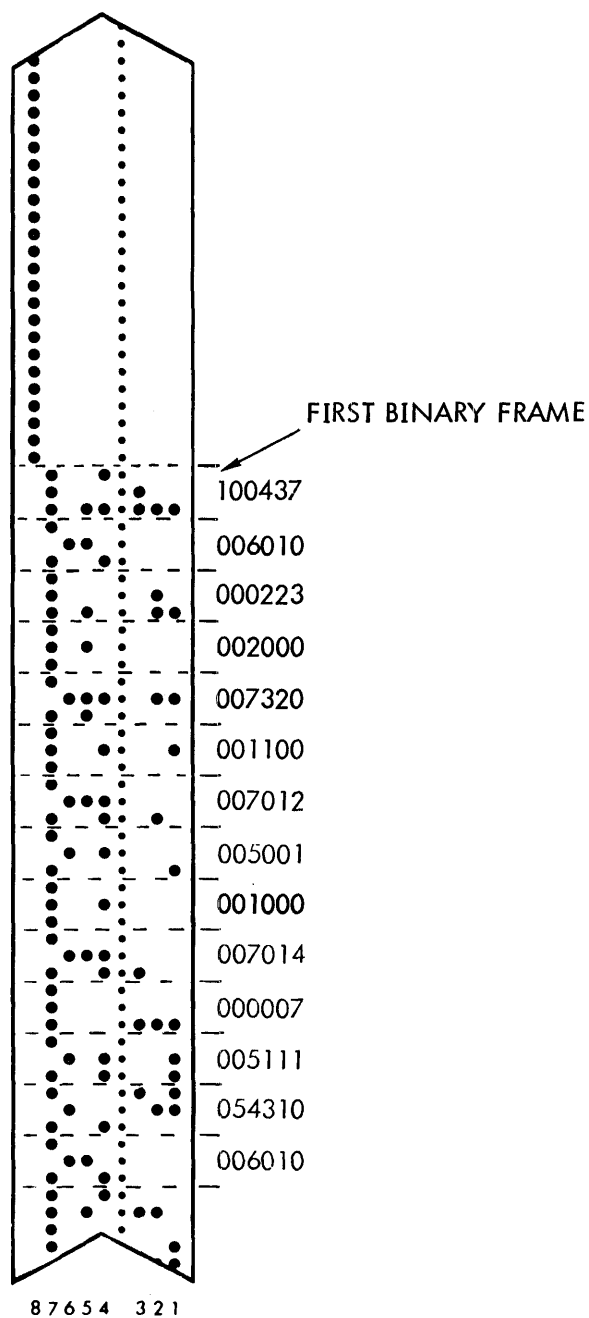


VTII-1187

Figure III-8. Source Tape Format



CHAPTER III COMPUTER OPERATION

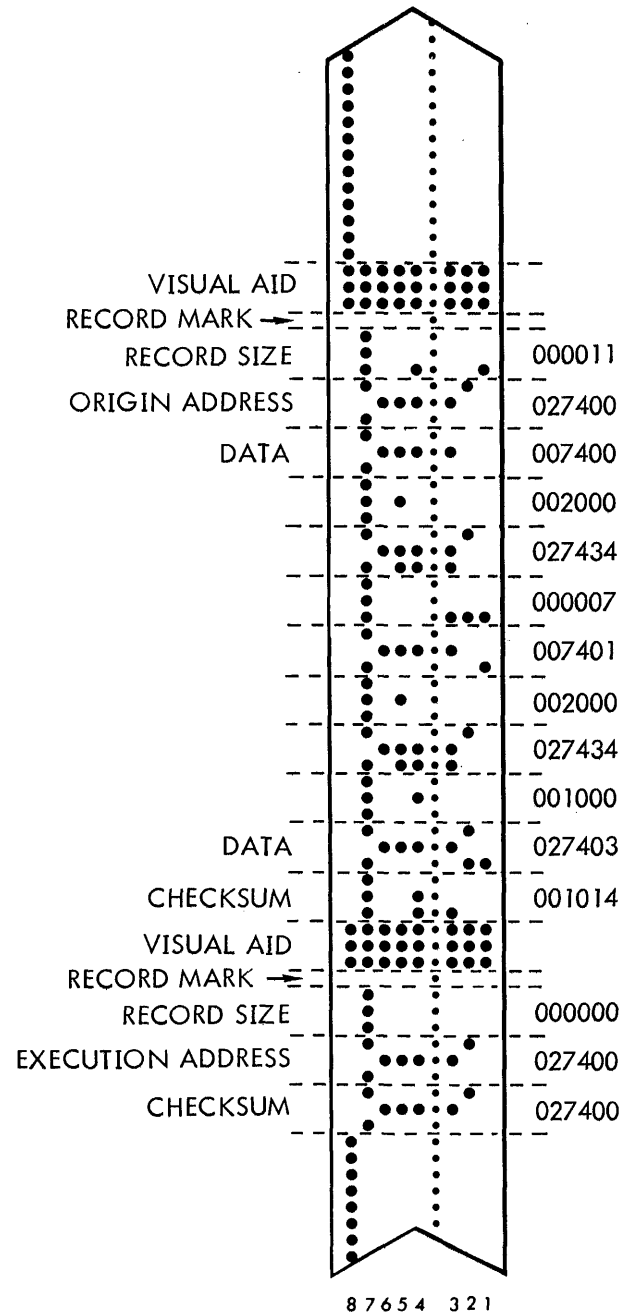


VIII-1188A

Figure III-9. Bootstrap Format



CHAPTER III COMPUTER OPERATION



VTII-1189

Figure III-10. Binary Object Format



CHAPTER III COMPUTER OPERATION

Figure III-10 locates the following information as it appears in the binary object format:

Record mark	indicates the start of a record to the binary loader
Record size	the number of data words in the record $0 < n < 62$ DAS tape $0 < n < 65$ BLD tape 0 indicates the end of a program
Origin address	the memory address where the binary loader is to put the data words
Checksum	exclusive-OR of all words in the record (except for the checksum word)
Execution address	run address in the case of load and execute

A program can contain many records.

2.4 MOS RELOCATABLE OBJECT FORMAT

All tapes produced by the MOS assembler or compiler will have this format (the assembler and the compiler produce identical object tapes). Refer to appendix B of the MOS manual (98 A 9952 090) for a detailed description of the relocatable MOS object format. Object tape produced by the MOS debug program is slightly different (refer to appendix A of the MOS manual).

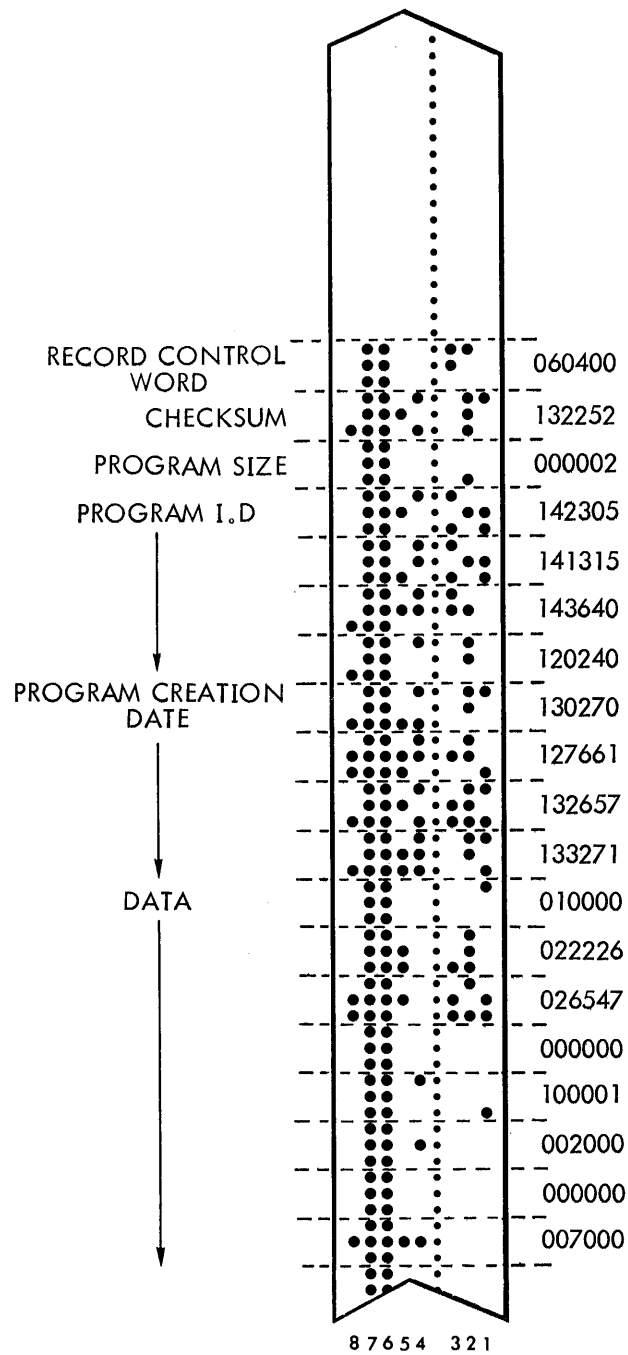
In this format, blank tape with feed holes is used for leader and trailer. Bits 7 and 6 are not part of the word; they are ignored. Bits 8, 5, 4, 3, 2, and 1 contain data. Three frames comprise a word with bit 8 of the first frame as the most significant bit and bit 1 of the third frame as the least significant bit.

Object records have a fixed length of 60 words; the records are separated by three blank frames known as the record mark. Program identification and the creation date are in packed-ASCII format.

Figure III-11 is an illustration of MOS relocatable object format.



CHAPTER III COMPUTER OPERATION



VTII-1190

Figure III-11. MOS Relocatable Object Format



CHAPTER III COMPUTER OPERATION

SECTION 3 OPERATING SEQUENCES FOR 620/i, 620/L

Three typical operating sequences are described in the following paragraphs. There are variations to these sequences, depending upon the particular instruction being executed; however, an understanding of these fundamental operations will enable the user to quickly understand the timing of each individual instruction sequence.

3.1 ACCESS OPERAND IN MEMORY

The simplest and most basic sequence is one in which a single-word, directly-addressed operand is read from memory. This is typical of the load, arithmetic (excluding multiply and divide), and logic type instructions.

The timing of the suboperations of this sequence is illustrated in figure III-12. At time 0, the instruction cycle (ICYX +) for the n th instruction is initiated. Note that the $n - 1$ instruction is being executed (IEPX +) while the current instruction (n) is being read from memory. At time 0.9, the instruction is transferred to the U register. During the instruction address phase (IAPX +), which occurs while the instruction just read is being restored to memory, the operand address is generated.

Since the operand is not indirectly addressed, the operand cycle (OCYX +) is initiated at time 1.8. After the operand has been read from memory and stored in the R register, the address of the next instruction, $n + 1$, is generated (normally by adding 1 to the P register) and transferred to the memory L register. This suboperation is performed while the operand is being restored in memory. The instruction cycle (ICYX +) for $n + 1$ is then initiated at time 3.6.

Note that the operation to be performed upon the operand now contained in the R register is executed during the instruction execution phase (IEPX +) of ICYX + for $n + 1$. This operation could be, for example, adding the operand value to contents of the A register and storing the result in A (ADD), or simply transferring the operand to one of the operation registers (LDA, LDB, or LDX).

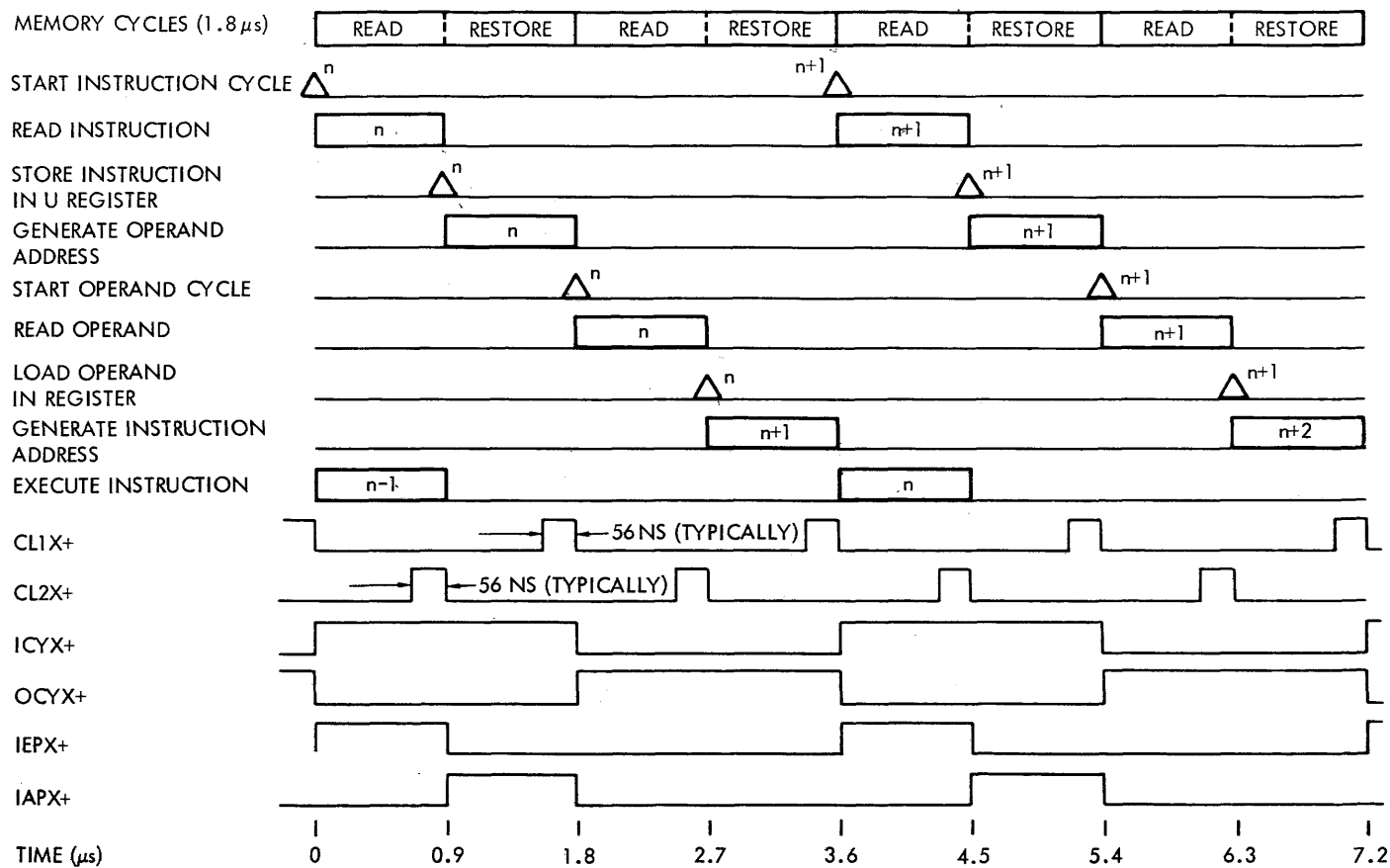
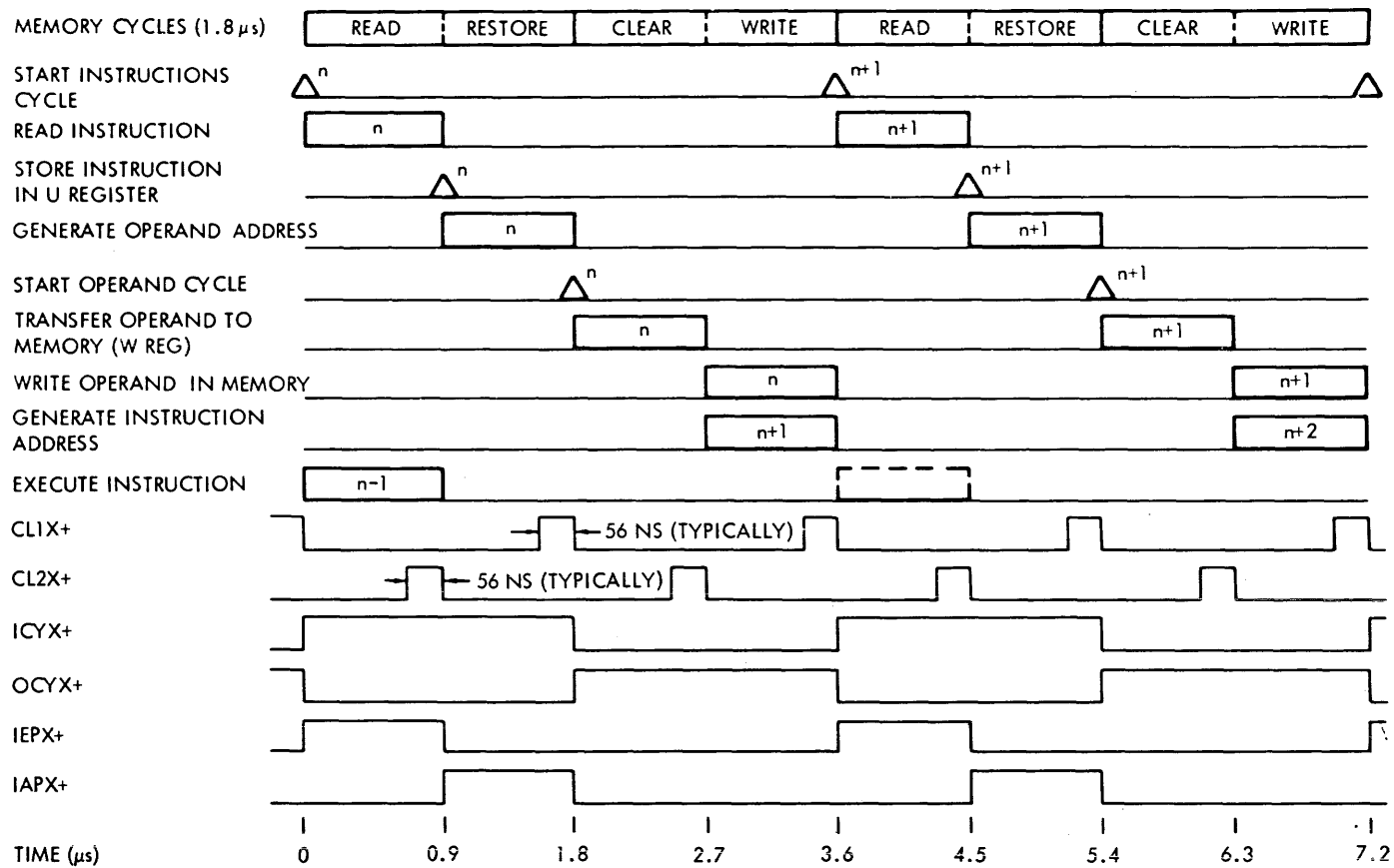


Figure III-12. Operand Access from Memory Sequence

VTII-1183

CHAPTER III
COMPUTER OPERATION

VTII-1184

Figure III-13. Operand Storage in Memory Sequence



CHAPTER III COMPUTER OPERATION

3.2 STORE OPERAND IN MEMORY

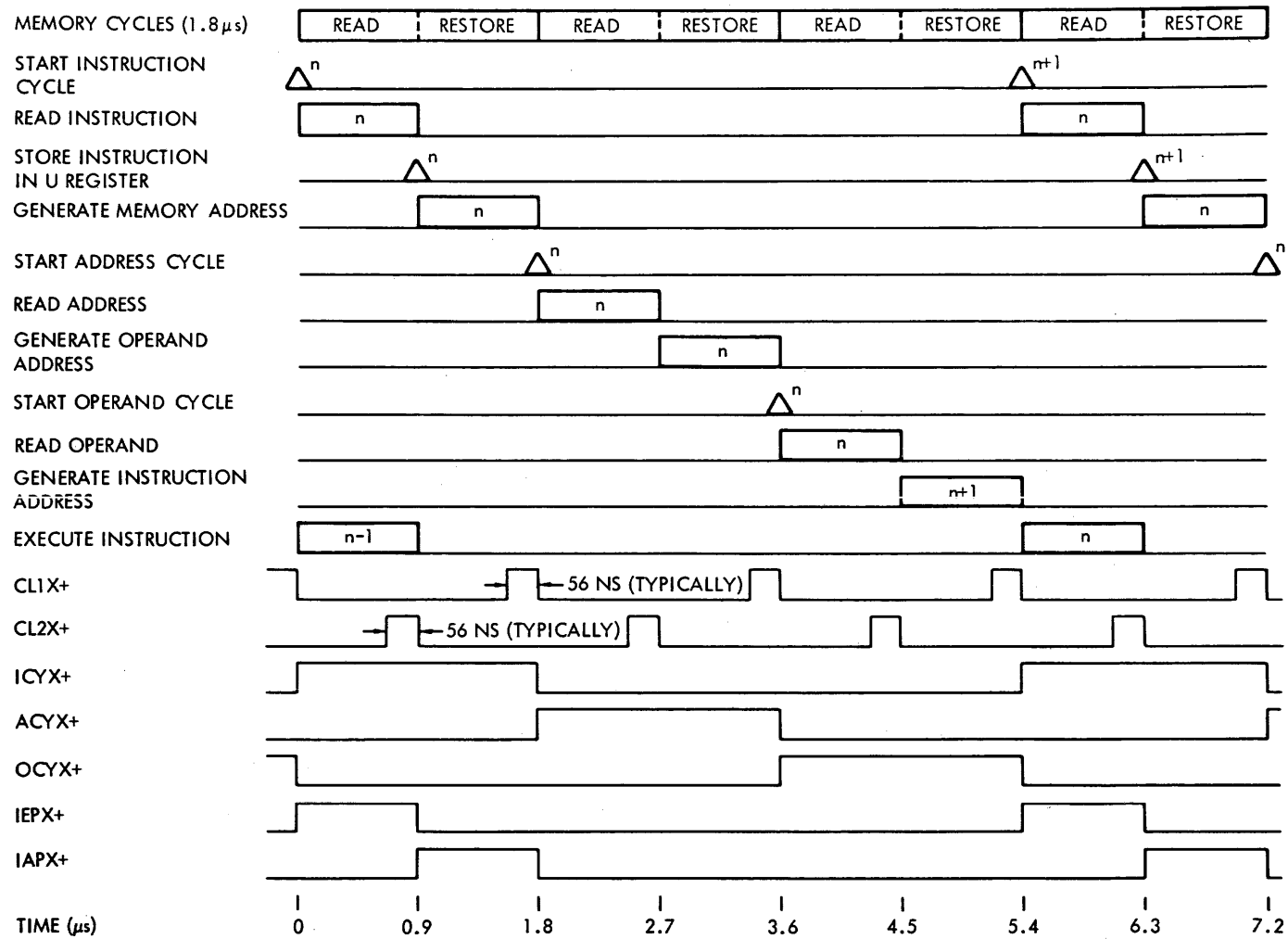
The sequence for storing an operand in memory (STA, STB, STX) is essentially identical to that for accessing an operand, except that the specified memory cell is cleared and the operand written into it. The sequence of suboperations is shown in figure III-13.

The n th instruction is accessed and the operand address generated during ICYX+ as before; execution of the $n - 1$ instruction occurs during IEPX+ of the n th cycle as indicated. However, during the operand cycle (OCYX+), the operand is transferred to memory while the referenced cell is being cleared. During the last half of the cycle, the operand is stored into the cleared cell. During this time, the address for the next instruction is generated. Note that there is no execution, as such, for this type of instruction (indicated by dashed lines) because the execution has already been accomplished in effect by the transfer and storage of the operand in memory.

3.3 INDIRECT OPERAND ACCESS

The third basic sequence involves indirectly accessing an operand in memory by a single-word instruction. In this case, an address cycle (ACYX+) is required to read the indirect address word from memory before performing the operand cycle (OCYX-).

The sequence of suboperations is illustrated in figure III-14. During the instruction cycle (ICYX+), the n th instruction is read from memory and stored in the U register as before. The previous instruction, $n - 1$, is executed during IEPX+. During the instruction address phase, IAPX+, the location of the (indirect) address word is generated. This address word is read from memory and stored in the R register as indicated in the timing diagram. For the case illustrated, the address word accessed contains the address of the operand (otherwise, another address cycle would be initiated to access a second address word, and so on). The operand address is transferred to the memory L Register during the last half of ACYX+ to locate the operand read out during the succeeding OCYX+. The generation of the address for instruction $n + 1$ and the execution of instruction n are then performed as in the simple operand access instructions (section 3.1).

CHAPTER III
COMPUTER OPERATION

VTII-1185

Figure III-14. Indirect Operand Access Sequence



CHAPTER III COMPUTER OPERATION

SECTION 4 COMPUTER FAILURE

When a computer fails to produce a result (stops before completing the routine), or when the result produced is obviously incorrect; it is because of an error, a mistake, or a malfunction.

- a. An *error* is a fault that can be attributed to the numerical analysis or the method chosen for solution of the problem: The computer is operating correctly and the routine is exactly what the programmer wants it to be, but the results are incorrect because the wrong methods or techniques of problem solving were chosen.
- b. A *mistake* is an inadvertent fault in the routine. The computer is doing exactly what it is being instructed to do, but the routine is not what the programmer thinks it is. A mistake can be made in writing the mnemonic routine, in coding the routine, in loading the routine into the computer, or all three. This entire process can be a lengthy one, and the opportunities for making mistakes are abundant.
- c. A *malfunction* is a fault in the computer itself; i.e., some electrical or mechanical fault in the computer causes it to stop or to produce incorrect results.

According to these definitions, a computer cannot make mistakes and a programmer cannot malfunction.

4.1 ERRORS

Errors probably account for the least amount of computer failures. This is because a programmer will not usually attempt to write a routine until he is convinced that he knows how to solve the problem. Also, the algorithms and other tools provided by the numerical analyst have been proven in use many times over and have been refined to mathematical perfection. Therefore, one should not look for errors until the possibility of a mistake or malfunction has been eliminated.



CHAPTER III COMPUTER OPERATION

4.2 MISTAKES

Mistakes account for the largest amount of computer failures. Many routines work incorrectly the first time they are put into the computer. The computer may stop when it encounters an instruction that is not in its repertoire; or, if it does succeed in completing the program, the result may be so large or so small that it is clearly unreasonable.

The process of locating and correcting the mistakes in a routine is known as debugging. It is a good idea to first ensure that the routine stored in the computer is the same as the one written down on paper. Therefore:

- a. If the routine was loaded into the computer manually through a control panel or some other punchkey or punchbutton system, it is advisable to load the program a second time to see if the second results are identical to the first.
- b. If the routine was loaded by magnetic or punched paper tape, it might be advisable to prepare the tape a second time. However, if the equipment for verification is available, load the prepared tape in the verification equipment and have a printed routine produced. Verification equipment is designed to read a tape and produce a typed or printed copy of the information on the tape. The printed routine and the original manuscript of the routine can be compared to see if any mistakes were made in transcribing the routine on the tape.

The specific actions and procedures followed in debugging a routine are determined by the characteristics and design features of the computer being used. Some computers have a display system which can be used to show the word stored in the accumulator (and possibly other registers) at any given time. Other computers have a step-by-step feature that uses the signal from a manually operated switch to cause the computer to execute one instruction and stop. If both of these features are present, the operator can observe the results produced by the execution of each step. This provides a very thorough, although time consuming, debugging method.

Regardless of the specific techniques used, the object of debugging a routine is to ensure that the routine is properly coded, that the routine and associated data words are correctly stored in memory, and that the routine does not exceed the computer's capability; e.g., additions do not cause a computer overflow. It is also necessary to determine that the proper instructions have been chosen.

When the programmer and computer operator are convinced that the routine is correct,



CHAPTER III COMPUTER OPERATION

that it has been accurately coded, and properly loaded into memory, and that the results (if obtained) are still wrong, it is time to check for a malfunction.

4.3 MALFUNCTIONS

In the case of a malfunction, corrective maintenance of the computer is indicated. The first step in most corrective maintenance procedures is the location and isolation of the fault. This is accomplished in digital computers through use of diagnostic or check routines.

4.3.1 Diagnostic Routines for Corrective Maintenance

Diagnostics are routines of proven quality and correctness that have been prepared for the specific purpose of determining the operating condition of one or more computer units or sections. A library of diagnostic routines is an essential part of the maintenance equipment for a computer.

The diagnostics for a given computer differ as to length and specific purpose. One routine can be designed to check the control section, another for the arithmetic unit, etc.; and several routines may be required to obtain an indication of the malfunction and its location. For example, a specific diagnostic can determine that a malfunction exists in a certain register, and other diagnostics can then be used to localize the trouble to a particular flip-flop in that register. When the maximum amount of information has been obtained by use of the diagnostic routines, the malfunction can be further isolated through use of the available electronic test equipment.

4.3.2 Diagnostic Routines for Preventive Maintenance

As well as being important to corrective maintenance, diagnostic routines are valuable for use in preventive maintenance. A well-designed diagnostic (check) routine used with reasonable frequency provides an excellent method of GO/NO-GO checking of the computer.

In addition to the use of check routines, a method known as marginal checking is frequently a part of the preventive maintenance procedures. The object of marginal checking is to determine and measure the amount of variation in the operating voltages (and perhaps frequencies) from their normal levels and values. These variations occur before a malfunction is caused; the operating conditions of the circuits to be checked are deteriorated in a controlled manner until a malfunction occurs.



CHAPTER III COMPUTER OPERATION

Marginal checking operates in the following manner:

- a. The value of the parameters being varied is read and recorded. This is done in a systematic and automatic manner, and a day-by-day record is kept of the values at which failure occurs.
- b. These daily values can be compared with previously determined values of the maximum margins or tolerances allowed to determine when a component has deteriorated to the point that it should be replaced.

Marginal checking is most effective in detecting gradually deteriorating components before the deterioration has become severe enough to cause a malfunction. It will not necessarily prevent abrupt failures, such as shorted elements or wiring, but it can be used as a means of diagnosis and fault location once a malfunction has occurred.



CHAPTER IV
620 COMPUTER SYSTEMS



varian data machines



CHAPTER IV 620 COMPUTER SYSTEMS

SECTION 1 620/i AND 620/L SYSTEMS

1.1 INTRODUCTION

The 620/i and 620/L computers are system-oriented, high-speed parallel binary computers. Modular design and extensive use of integrated circuits permit a compact package, occupying only 10.5 inches of rack space. With flexibility built-in, the 620/i and 620/L computers are ideally suited for use as general-purpose computers or as on-line system devices.

The 620/i and 620/L computers feature:

- 1.8-microsecond memory cycle
- 16- or 18-bit words
- Nine hardware registers
- Six addressing modes
- Over 100 basic instructions
- Memory sizes of 4,096 words minimum, 32,768 words maximum

The 620/i and 620/L systems are designed to be user-oriented. Input/output flexibility allows a wide selection of option facilities including: Direct Memory Access, Real-Time Clock, Power Failure/Restart, and Buffer Interlace Controller. These features combined with priority interrupts, external sense lines and external control lines enable the systems to meet every possible I/O requirement. Specifications for the 620/i and 620/L computers are listed in table IV-1.



CHAPTER IV

620 COMPUTER SYSTEMS

Table IV-1. 620/i and 620/L Specifications

Description	System-oriented, general-purpose digital computers, designed for on-line data system requirements, utilizing magnetic core memory, binary, parallel, single-address, with bus organization and micro-control	
Memory	Magnetic core, 16 bits (18 bits optional), 1.8 microseconds full cycle, 700 nanoseconds access time, 4,096 words minimum expandable to 32,768 words maximum	
Arithmetic	Parallel, binary, fixed-point, two's complement	
Word Length	16 bits standard, 18 bits optional	
Speed (Fetch and execute)	Add or subtract	3.6 microseconds
	Multiply (optional on 620/i)	18.0 microseconds, 16-bit
		19.8 microseconds, 18-bit
	Divide (optional)	18.0 to 25 microseconds, 16-bit
	Register change class	1.8 microseconds
	I/O - from A or B	3.6 microseconds
	Memory	5.4 microseconds
Operation Registers	A register: accumulator, input/output, 16/18 bits B register: double-length accumulator, input/output, index register, 16/18 bits X register: index register, 16/18 bits P register: program counter, 16/18 bits	
Buffer Registers	R register: operand register, 16/18 bits U register: instruction register, 16/18 bits S register: shift register, five bits, operates with the U register for executing shift instructions L register: memory address register, 16 bits W register: memory word register, 16/18 bits	



CHAPTER IV
620 COMPUTER SYSTEMS

Table IV-1. 620/i and 620/L Specifications (*continued*)

Control

Addressing Modes:

Direct addressing: to 2,048 words
Relative to P register: 512 words
Index with X register: hardware, does
not add to execution time
Index with B register: hardware, does
not add to execution time
Multilevel indirect addressing
Immediate
Extended addressing (optional)

Instruction Types:

Single-word, addressing
Double-word, addressing
Single-word, nonaddressing
Double-word, nonaddressing

Instructions:

Over 100 standard instructions as listed
below, plus more than 128 micro-
instructions

Load (three)
Store (three)
Arithmetic (five, two optional on 620/i)
Logical (three)
Jump (10)
Jump and mark (10)
Execute (10)
Immediate (14, two optional on 620/i)
Input/output (13)
Register change (26)
Logical shift (six)
Arithmetic shift (six)
Control (two)
Extended addressing (14 optional on 620/i)
Micro-instructions (over 128)

Micro-EXEC (Optional):

Facility and hardware to construct a



CHAPTER IV 620 COMPUTER SYSTEMS

Table IV-1. 620/i and 620/L Specifications (*continued*)

hardware program external to the 620/i
eliminates stored program memory accessing
by use of hardware program

Console:

Display and data entry switches for all
operation registers, three sense switches,
instruction repeat, single step, run, and
power on/off

Input/output

Processor input/output:

Programmed data transfer
Single word to/from memory
Single word to/from A and B registers
External control lines
External sense lines

Automatic data transfer

Direct memory access facility transfer with
rates over 200,000 words per second

Priority interrupts (optional):

Group enable/disable, individually
arm/disarm, single instruction interrupt
capability

Real-time clock (optional):

Adjustable time base: can be programmed
as multiple internal timer

Power failure/restart (optional):

Interrupts on power failure and auto-
matically restarts on power recovery

Physical

Dimensions:

Mainframe: 10-1/2 inches high, 19 inches
wide, 15 inches deep



CHAPTER IV
620 COMPUTER SYSTEMS

Table IV-1. 620/i and 620/L Specifications (continued)

Weight:

Mainframe: 35 pounds

Power:

3 amps 115V ac, 60 Hz (340 watts).
115 \pm 10V, 60 \pm 2 Hz. Power supplies
are regulated; additional regulation is
not required under normal commercial
power sources. Conversion for 50 Hz and
other voltages available at added cost.

Expansion:

Main processor contains provisions and
space for all internal options

Installation:

Mounts in standard 19-inch cabinet, no
air conditioning, sub-flooring, special
wiring, or site preparation required

Environments:

0 to 45 degrees C; 0 to 90 percent rela-
tive humidity

Mainframe

Integrated circuit, 8.8-MHz clock, logic

Logic and signals

levels 0V false, +5V true

Figure IV-1 presents an outline of the 620/i computer.



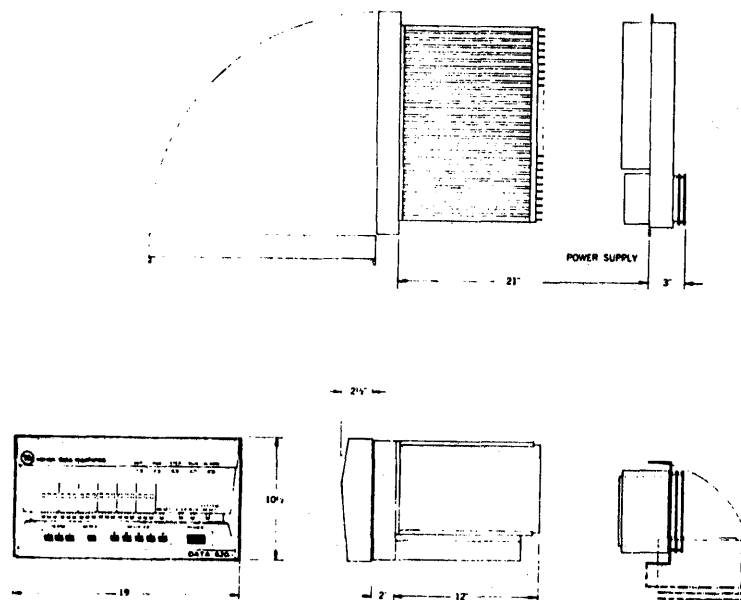
CHAPTER IV 620 COMPUTER SYSTEMS

1.2 SWITCHES AND INDICATORS

The 620/i and 620/L control consoles, as illustrated in figures IV-2 and IV-3, provide for operator communication with the computer. This communication is accomplished through use of the register displays and control switches.

1.2.1 Displays

The contents of all operation registers in the computer (including the instruction register), are displayed in binary-octal form when selected by the register display switches. Indicators and switches permit independent control over each bit in table IV-2. During normal operation (run mode) the display is active; however, the register entry and reset switches are deactivated to prevent accidental alternation of the register contents.

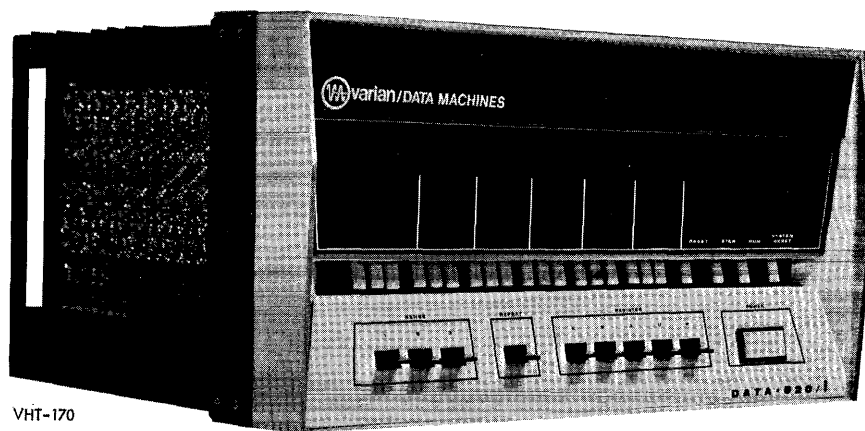


V711-0933

Figure IV-1. 620/i Outline

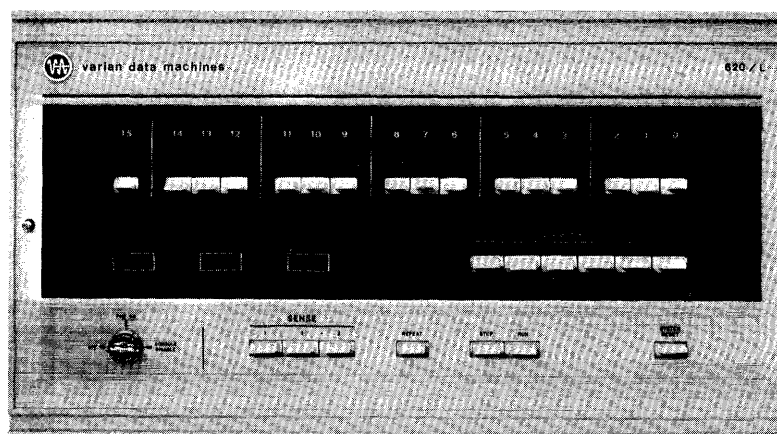


CHAPTER IV 620 COMPUTER SYSTEMS



VHT-170

Figure IV-2. 620/i Control Console



VHT-171

Figure IV-3. 620/L Control Console



CHAPTER IV

620 COMPUTER SYSTEMS

1.2.2 Controls

Control switches permit the operator to manually alter normal program operation. These switches, also described in table IV-2, provide considerable control flexibility for maintenance, troubleshooting, and program debugging. The sense switch controls are also useful in normal program operation to allow operator selection of particular program sequences to be executed.

Table IV-2. Controls and Indicators

Control or indicator	Function
A Register	Illuminated indicators display the contents of the A register upon selection of A from the console. Pressing a switch enters a one into the corresponding bit position; pressing RESET clears the register. The overflow light is set or reset by execution of the set overflow (or arithmetic condition) and reset overflow instructions, or by pressing SYSTEM RESET.
B Register	Illuminated indicators display the contents of the B register upon selection of B from the console. Pressing a switch enters a one into the corresponding bit position. Pressing RESET clears the entire register.
Instruction Register (U)	Illuminated indicators display the current instruction being held in the instruction (U) register during execution. Pressing a switch enters a one into the corresponding bit position, and pressing RESET clears the entire register.
Instruction Counter (P)	Illuminated indicators display the memory location of the next instruction to be executed when the computer halts. Upon selection of P from the console, pressing a switch enters a one into the corresponding bit position. Pressing RESET clears the entire register.



CHAPTER IV 620 COMPUTER SYSTEMS

Table IV-2. Controls and Indicators *(continued)*

Control or indicator	Function
Index Register(X)	Illuminated indicators display the contents of the index (X) register upon selection of X from the console. Pressing a switch enters a one into the corresponding bit position, and Pressing RESET clears the entire register.
SYSTEM RESET	Momentary-contact switch that permits manual reset following memory temperature overload condition, also used to initialize computer and peripheral equipment.
RUN	Momentary-contact switch that sets computer to normal operation mode. Indicator is off and operation is halted when the STEP switch is pressed or a program halt instruction is executed.
STEP	Momentary-contact switch that permits operation to be halted and the program executed one instruction at a time. Pressing this switch in the RUN mode stops operation, turns the RUN indicator lamp off, and turns STEP indicator lamp on. The instruction register display indicates the next instruction to be executed when STEP is pressed and the program counter indicates the location of the next instruction to be executed after the instruction in the instruction register is executed. Normal operation is started and STEP turned off when the RUN switch is pressed.
REPEAT	Toggle switch that permits the manual repetition of an instruction in the instruction register. Pressing STEP executes the instruction and advances the program counter; however, the contents of the instruction register are left unchanged. The switch on the control console is activated only when the STEP switch is on (operation halted).
SENSE SWITCHES 1, 2, and 3	Toggle switches permitting manual program control whenever the sense switch jump, jump-and-mark, or execute instructions (JSS1, JSS2, JSS3, JS1M,



CHAPTER IV 620 COMPUTER SYSTEMS

Table IV-2. Controls and Indicators (*continued*)

Control or indicator	Function
	JS2M, JS3M, XS1, XS2, and XS3) are executed only if the corresponding SENSE switch is set on.
POWER: 620/i	Alternate-action switch/indicator turns memory and logic power supplies on and off, also controls Teletype controller power. Switch indicator lamp lights when memory and computer power are both on; indicator is off when power is turned off.
620/L	Key-operated power switch controls the ac input to the 620 power supply: PWR OFF position, disables ac input to the power supply primary; in the PWR ON position, supplies ac power to the power supply primary to make the system fully operational; and, in the PWR ON DISABLE position supplies ac power to the power supply primary to make the computer operational; however, disables all control console switches except the power switch itself so that pressing any other switch at this time has no effect. The control panel and power supply indicator lights are functional when the POWER switch is in PWR ON or PWR ON DISABLE. The key can be removed from the power switch in any of the three positions. To turn off the computer, place the power switch in the PWR ON position, lift the STEP/RUN switch then turn the power switch to PWR OFF.

1.3 MANUAL OPERATIONS

Control console operation is simple and can be understood by reference to table IV-1 and figures IV-2 and IV-3. The following paragraphs describe typical operation sequences that illustrate normal computer use.

1.3.1 Power Control

Power to the computer, the memory, and control logic for the Teletype is turned on and off by the POWER switch: the 33 ASR Teletype has a separate power switch. Provision is also made for controlling power to other I/O device controllers from the control console switch. If the memory temperature sensor detects an overload condition, the ALARM indicator on the console illuminates; the memory should be disabled and power turned off. Power should not be restored until the temperature is returned to normal.



CHAPTER IV 620 COMPUTER SYSTEMS

1.3.2 Manual Program Entry and Execution

When the computer is halted (step mode), programs and data can be read from memory and entered into memory, and a prestored program manually executed. To load words into memory (either instructions or data), set the desired word in the A, B, or X register, set up the appropriate store instruction (STA, STB, STX) with the desired operand address in the instruction (U) register, and press STEP to execute the store operation.

To display the contents of any memory cell in the A, B, or X register display, set the appropriate load instruction (LDA, LDB, LDX) with the proper memory address in the instruction register, and press STEP to load the selected word into the register.

To manually execute a program stored in memory, set the starting location of the program in the program counter. When STEP is pressed, the instruction contained in the instruction register is executed, and the instruction of the selected location is transferred to the instruction register for execution when the switch is again pressed. Repeated operation of STEP will then step through the program one instruction at a time. All operations such as multilevel indirect addressing will be performed for each instruction each time STEP is pressed. Note that I/O instructions involving an asynchronous device that transfers data in a block (such as a magnetic tape unit or the Teletype) generally cannot be operated in a single-step mode.

Note

To select a register from the console, place the desired register switch in the UP position on the 620/i and in the DOWN position on the 620/L. Select only one register at a time.

1.3.3 Instruction Repeat

In the step mode, the instruction register contains the next instruction to be executed when STEP is pressed, and the program counter contains the location of the next instruction to be transferred to the instruction register after the current instruction is executed. In some cases, it is desirable to manually execute an instruction several times. When REPEAT is on, instruction register loading is inhibited even though the instruction counter is advanced each time STEP is pressed. This mode is particularly useful for loading words into sequential memory locations, and for displaying the contents of sequential memory cells.



CHAPTER IV 620 COMPUTER SYSTEMS

To load a group of sequential memory cells, set the appropriate store instruction (STA, STB, STX) in the instruction register with the relative address mode in the M field, the A field set to 0, and the base address in the program counter. Repeated operation of STEP will store the contents of A, B, or X into sequential memory locations. The word loaded on each step can be changed by entering the desired value into the operation register for each step.

To display the contents of a group of sequential memory cells, set the appropriate load instruction (LDA, LDB, LDX) in the instruction register, in the relative address mode, with the base address in the program counter, and the A field set to 0. The contents of the sequential location will be displayed in the selected operation register each time STEP is pressed.

1.3.4 SENSE Switches

The SENSE switches allow the operator to dynamically alter a program sequence in either run or step mode. The three SENSE switches provide a logical-AND function with bits 6 through 8 of the instruction word and, consequently, can be used for various logical branches as set up on the console.

1.4 ORGANIZATION

A block diagram of the 620/i computer is shown in figure IV-4. The computer is composed of four major sections: memory, control, arithmetic/logic, and input/output.

1.4.1 Memory

The basic memory module contains a minimum of 4,096 words; total memory capacity can be expanded in 4,096-word increments to a maximum of 32,768 words. As illustrated in figure IV-4, each memory module is connected with the same location (L) and word (W) register. The L register contains the location of the word to be accessed in memory during either a clear/write or read/restore cycle. The W register receives words read from memory during a read/restore cycle and receives words from the central bus (C bus) during a clear/write cycle. The W register is 16 or 18 bits long. Outputs from W register are gated onto the W bus through line drivers by appropriate timing signals.



CHAPTER IV
620 COMPUTER SYSTEMS

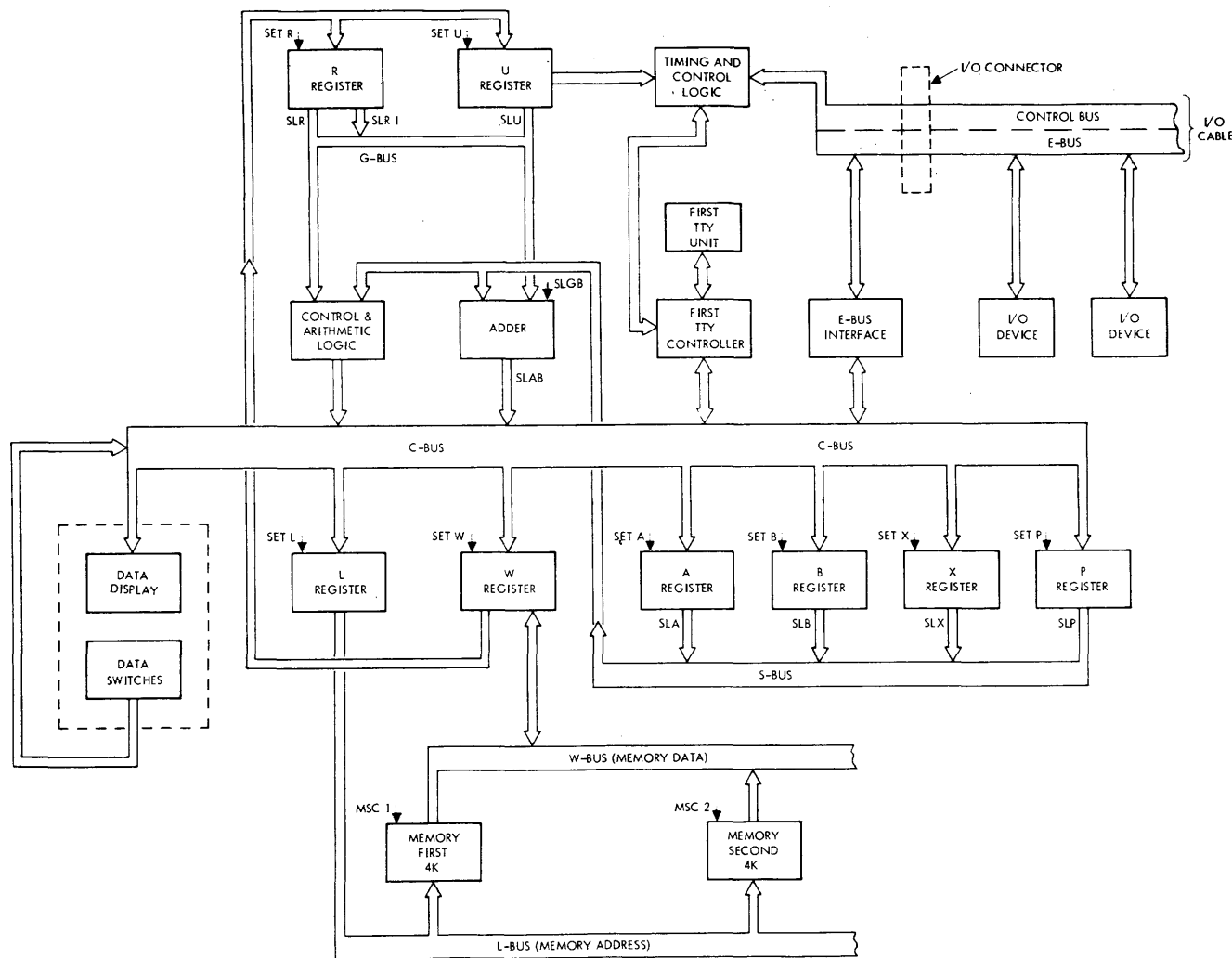


Figure IV-4. 620/i Organization

VT11-1285



CHAPTER IV 620 COMPUTER SYSTEMS

1.4.2 Control

The control section provides the timing and control signals required to perform all operations in the computer. The major elements are the instruction register and the timing and decoding logic.

- a. The instruction (U) register is 16 or 18 bits long. This register receives each instruction from memory and holds the instruction during its execution. The control fields of the instruction word are routed to the decoding and timing logic where the codes determine the required timing and control signals. The address field, used for various addressing operations, is also routed to the arithmetic/logic section.
- b. The decoding logic decodes the fields of the instruction word held in the instruction register to determine the control signal levels required to perform the operations specified by the instruction. These levels select the timing signals generated by the timing logic.
- c. The timing logic generates the basic 2.2-Mc system clock. From this clock, the timing logic develops the timing pulses which control the sequence of all operations in the computer.

1.4.3 Arithmetic/Logic

The arithmetic/logic section is the part of the computer that performs numeric and logical calculations. (Refer to figure IV-4 for the important components of this section.) The arithmetic unit is functionally composed of several subsections, a number (R) register, adder, and control and arithmetic logic. The R register receives the operands read from memory and holds these words during the execution of an arithmetic or logical instruction. The R register gates allow selection of the R register contents, the R register complement, or the instruction register for an operation. This selection depends on whether an operand stored in the R register or the A field of the instruction word stored in the instruction register is to be used. The adder generates the arithmetic sum and carry. The logic gates allow shifting of the bits, the forming of a logical product and logical masking; these gates are used to implement the shifts required for multiplication and division.



CHAPTER IV 620 COMPUTER SYSTEMS

1.4.4 Input/Output

The 620/i I/O section facilitates integration of the computer into an overall system. The I/O section of the computer communicates with the operation registers and the memory through the internal C bus (refer to figure IV-4). Data and control signals are transmitted to and from external peripheral devices through the I/O bus. Standard or special peripheral devices are in parallel on the I/O bus, and any number of logical devices up to a total of 64 can be added. Such devices could include teletypewriters, high-speed printers, analog/digital converters, disc memory, common carrier interface, magnetic tape transports, and plotters.

1.4.5 Bus Structure

There are four buses in the 620/i and 620/L computer systems:

- a. *The W bus* provides the parallel path and selection logic for routing data and instructions between memory, the I/O unit, the control unit, and the arithmetic unit; it also provides a direct path to memory for the IN MEMORY and OUT MEMORY I/O instruction; and, with the interlace option, allows I/O operations to occur simultaneously with extended arithmetic and shift commands.
- b. *The C bus* provides the parallel path and selection logic for routing data between the arithmetic unit, the I/O unit, the memory bus, and the operation registers. This bus permits data to be uniquely or commonly transferred to the operation registers, and performs the distribution function for microprogramming. The C bus also provides a bidirectional parallel word path to the party-line bus and the W bus. The C bus is the central communication avenue and connects with all internal elements of the 620/i or 620/L computers.
- c. *The S bus* provides the parallel path and selection logic for routing data between the operation registers and the arithmetic unit.
- d. *The party-line bus* provides a 16-bit parallel bidirectional I/O communication path. This bus includes the control lines for transfer ready, sense, control, interrupt address, and acknowledge and information drop-ins. The party-line bus is packaged as one cable. Each peripheral device has a party-line connector and a party-line extender connector; the device and the party-line form a link whereby additional subsystems can be added at the site on a plug-in basis.



CHAPTER IV 620 COMPUTER SYSTEMS

1.5 TIMING

The 620/i and 620/L systems operate on a basic 1.8-microsecond machine cycle; that is, a full memory cycle (read/restore or clear/write) is performed in each 1.8-microsecond time interval (except for some special cases in which this period is extended as discussed in subsequent paragraphs). All operations performed by the computer are accomplished within some multiple of this basic timing period.

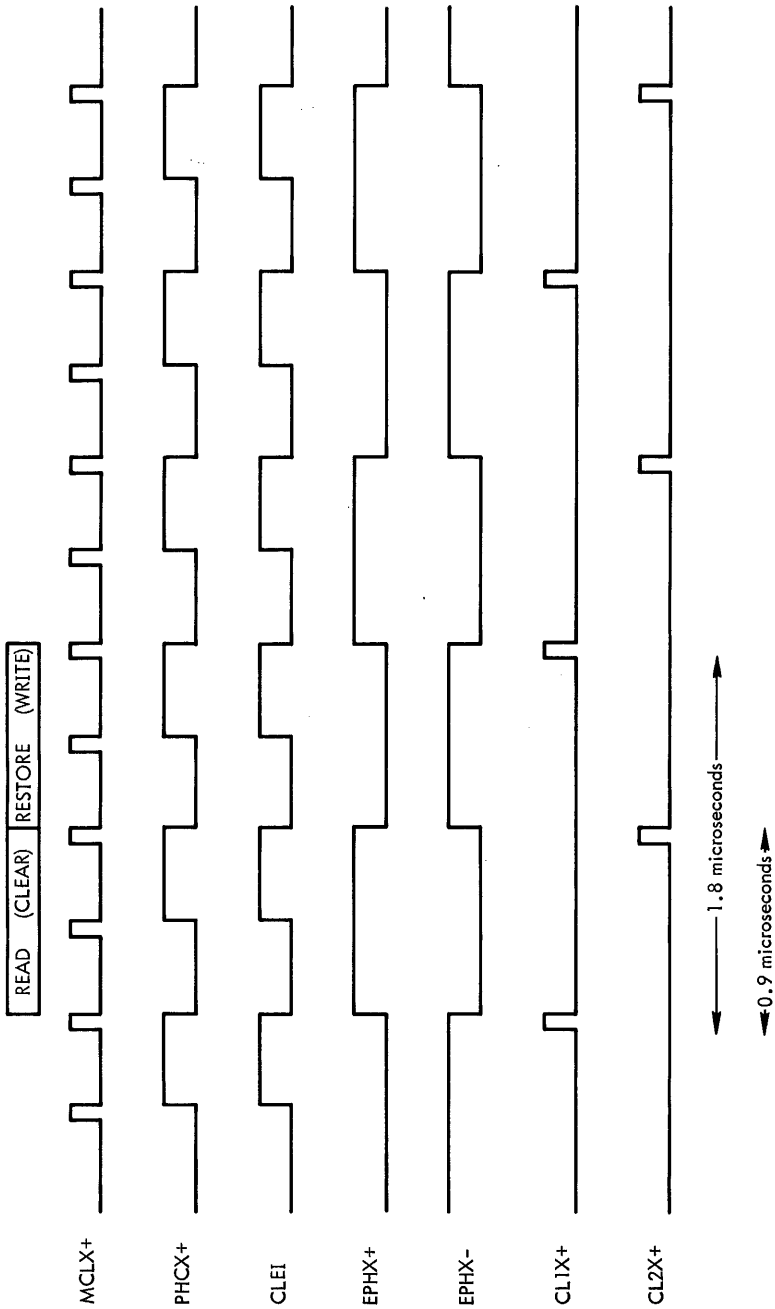
To execute the various operations, several suboperations are performed during the memory cycle time. Timing of these suboperations is controlled by the internal 2.2-MHz master clock. The period of this master clock is 0.45 microseconds, or one-fourth of the basic 1.8-microsecond machine cycle; this permits multiple suboperations to be executed during the memory cycle period. Note that the first half-cycle (0.9 microseconds) of the memory period is used to access a word (read) or to clear a cell (clear); the second half-cycle is used to restore a word (restore) or to write a new word (write) into the cell.

1.5.1 Clocks

The clocks which control the timing of all operations in the computer are generated by the timing and control logic. These clocks are illustrated in figure IV-5 and listed in table IV-3.



CHAPTER IV
620 COMPUTER SYSTEMS



VTII-0314

Figure IV-5. Basic Timing Clocks



CHAPTER IV

620 COMPUTER SYSTEMS

Table IV-3. Basic Timing Clocks

Clock	Description
Master Clock (MCLX +)	Crystal-controlled timing signal for entire system 2.2-MHz
Phase Clock (PHCX +)	1.1-MHz timing signal (counted down and synchronous with MCLX +); used to time the basic execute and ad- dress phases of the computer
Address Phase (EPHX-)	Basic timing phase, synchronous with restore or write half cycles of memory; all transfers of in- struction and operand addresses to memory are per- formed during this period
Execute Phase (EPHX +)	Basic timing phase, synchronous with read or clear half cycles of memory; all operations on words (transfers of data to and from memory and execution of instructions) are performed during this period
Clock 1	Basic timing pulse used to initiate memory cycle and all operations synchronous with start of memory cycle
Clock 2 (CL2X +)	Basic timing clock used to initiate all operations synchronized with start of memory write or restore half-cycle

1.5.2 Clock Modifiers

All functions performed by the 620/i and 620/L occur in two basic phases:

- The transfer of addresses to the memory L register (address phase)
- Operation upon words read from memory, or the storing of words into memory (execute phase).

These basic address and execution phases can be modified by certain program instructions or by signals received from devices external to the computer. The conditions under which the periods of the basic clocks are modified are:



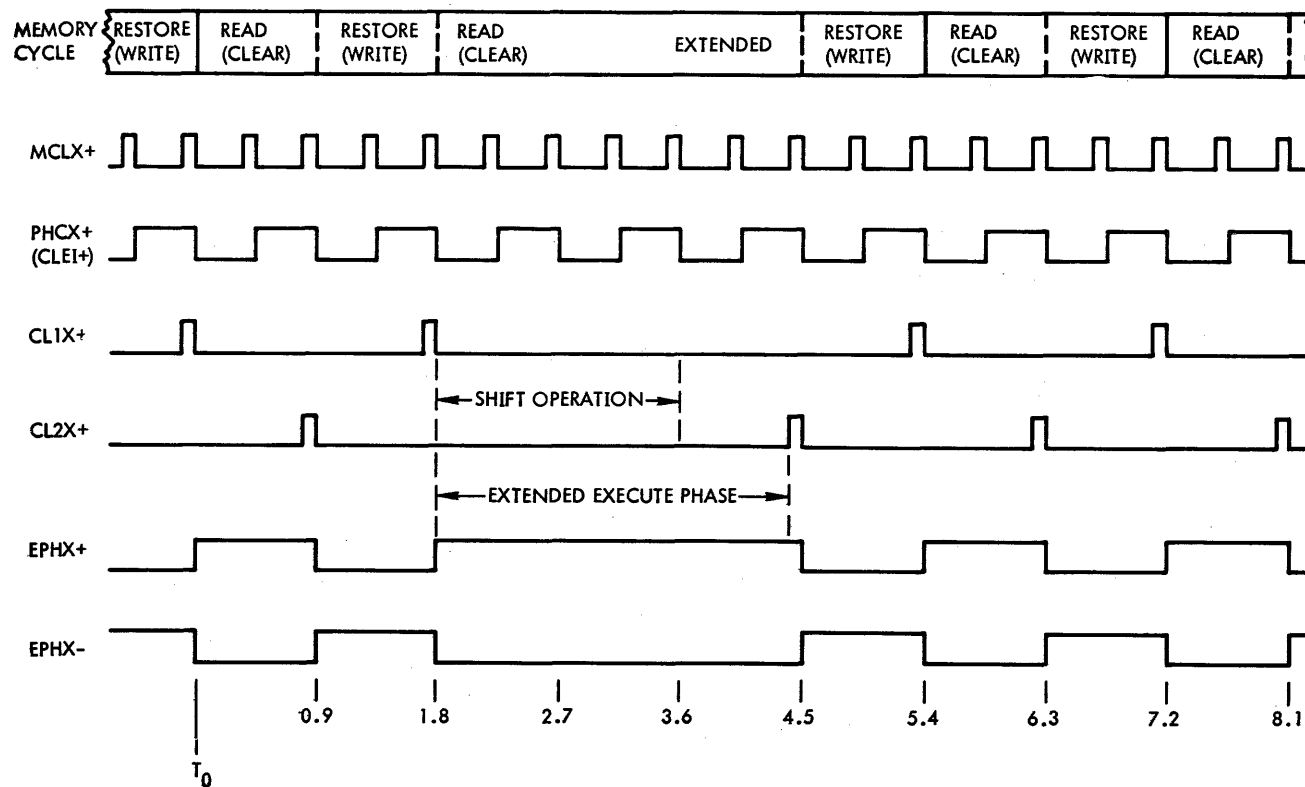
CHAPTER IV 620 COMPUTER SYSTEMS

Shift	During shifting operations with words contained in the A and B Registers, the execution phase (EPHX+) is extended by the number of master clock periods (0.45 microseconds) equal to the specified number of shifts
Interrupt	When an external interrupt is received, the address phase (EPHX-) is extended 0.9 microseconds to accommodate delays in receiving the interrupt address from the external device
Trap	When a buffer interlace controller requests a transfer to or from memory, EPHX- is extended 3.15 microseconds to permit the execution of the full trap sequence (routing of address and data from the external device)
Halt	On a halt instruction, clocks CL1X+ and CL2X+ are inhibited; this prevents any further operations until the STEP or RUN switch is pressed.

Modification of the execute phase of an instruction is illustrated in figure IV-6. This modified sequence is typical of a shift instruction. At time 0, the instruction has been fetched from memory. Starting at time 0.9, the instruction is executed; however, the normal 0.45-microsecond execute phase is extended 0.45 microseconds for each shift (six, in this illustration). Note that clocks 1 and 2 (CL1X+ and CL2X+) are inhibited during the extended execution period. In a similar manner, the address phase is extended when required by the conditions defined above.

1.5.3 Sequence Control

The basic clocks generated from the master clock are used to time three operating sequences: instruction cycle, operand cycle, and address cycle. All operations performed by the computer are timed by one or more of these timing sequences.

CHAPTER IV
620 COMPUTER SYSTEMS

VTII-1193

Figure IV-6. Example of a Modified Clock Sequence

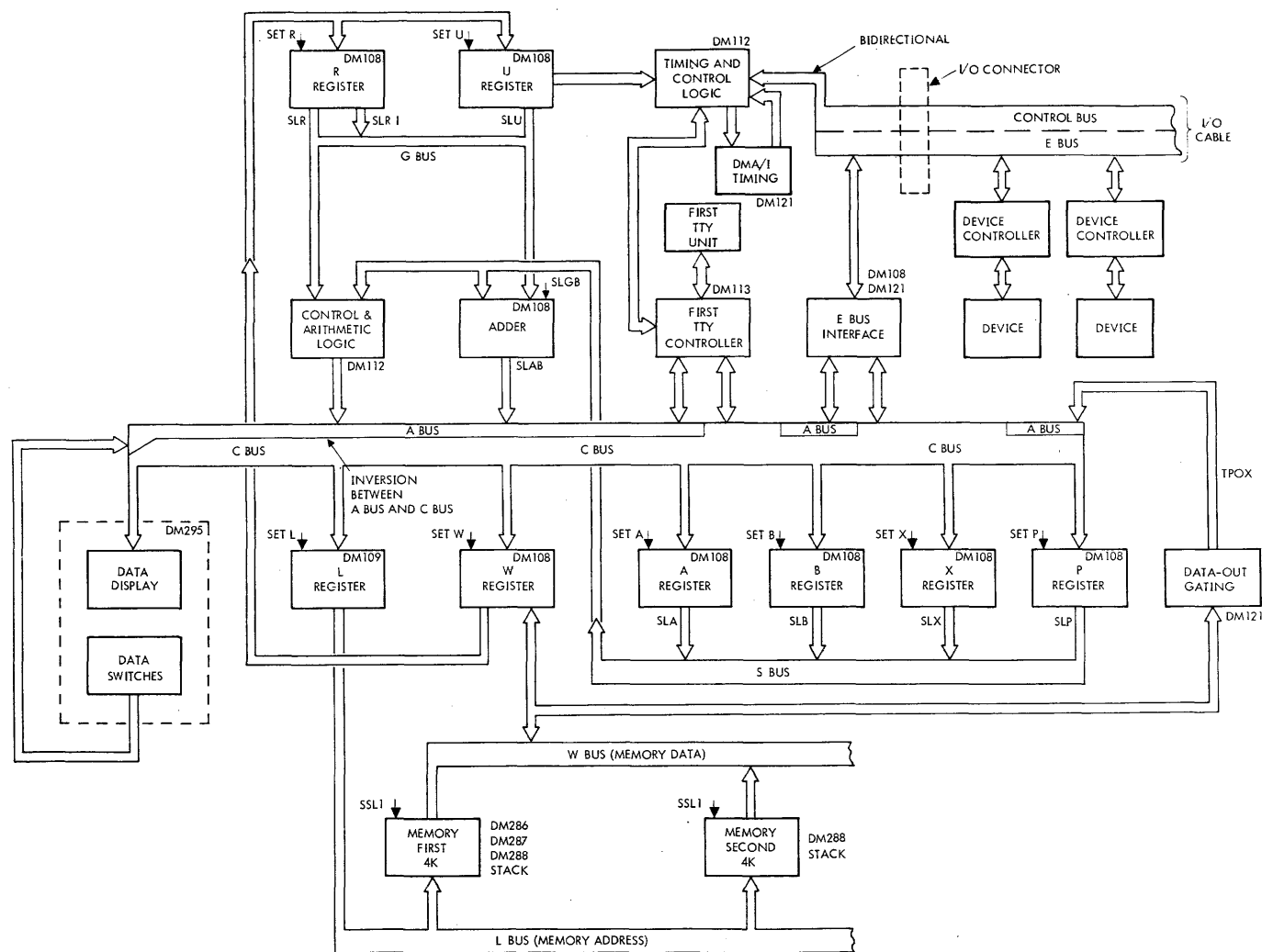


Figure IV-7. Data 620/L Organization

VT13-0268



CHAPTER IV 620 COMPUTER SYSTEMS

1.6 INFORMATION TRANSFER

All data communication between the basic functional elements of the machine is through the three data buses C, S, and W. The C and S buses are internal to the computer. The W bus is external and bidirectional; that is, a single set of lines is used to carry information both to and from the memory. The following paragraphs outline the major data transfer paths in the computer (refer to figure IV-7).

1.6.1 P Register to Memory

As an instruction cycle begins, the location of the next instruction is transferred from the P register to the L register. The contents of P are transferred through the S bus to the adder. The adder increments the location address with the arithmetic gates, and restores the incremented count to the P and L registers. The memory address register, L, now contains the address of the next instruction word to be fetched from memory, and the P register holds the updated address.

1.6.2 Memory to U Register

During the instruction cycle, the instruction word located by the address in the L register is read out on the W bus and read into the W register (memory data register). It's then transferred out to the U register.

1.6.3 U Register to Memory

For many instructions requiring an operand, the address of the operand is contained in the instruction word held in the U register. This operand address is transferred to the L register through gates in the arithmetic logic and the C bus. The address from U can be modified during the transfer to L as follows:

- a. *Direct Address.* No modification; bits 0 through 10 transferred from U to L directly address the operand in the first 2,098 memory locations.
- b. *Relative Address.* The effective operand address transferred to L is formed by adding bits 0 through 8 from U to the contents of P. Addition is performed by selecting the contents of P and U and bringing them into the adder. This permits addressing any word up to 512 locations ahead of the current program location.



CHAPTER IV 620 COMPUTER SYSTEMS

- c. *Index Address.* The effective operand address transferred to L is formed by adding bits 0 through 8 from U to either the contents of X or B.
- d. *Indirect Address.* Same transfer as direct address, but the word read from memory will be the address of an operand rather than the operand itself.

1.6.4 Memory to R Register

Operands read from memory into the W register are transferred to the R register. The operands are stored in R while an arithmetic or logical operation is being performed.

For indirect addressing, and for instructions whose operand address is stored in the memory location following the instruction word, the operand address will be read from memory into the W register and then transferred to the R register. The address is then routed to the L register through the C bus.

1.6.6 Operation Registers to Memory

The contents of any one of the operation registers are transferred to memory by selecting the register onto the S bus and routing the word through the adder C bus, and W register. The contents of the P register can be transferred to the L register to address an instruction as previously explained. The contents of the P register and other registers can be stored in memory by the same path, except that the word is entered into the W register. Note that an address cycle must precede this transfer to place the storage address in the L register.

1.6.7 Memory to Operation Registers

The contents of a memory location can be transferred to any of the operation registers through the W, G, and C buses. Note that an address transfer must precede the data transfer to place the memory address in the L register.

1.6.8 Input to Memory

Input data from the E bus can be routed directly to memory through the C and W buses. Data transfer must be preceded by an address transfer to load the memory location into the L register. When the transfer is under control of an instruction, the memory address will be generated as a normal operand address.



CHAPTER IV 620 COMPUTER SYSTEMS

1.6.9 Output from Memory

Output words can be transferred directly from memory to the I/O cable through the W, G, and C buses. A storage address must first be transferred to the L register by an instruction.

1.6.10 Input to Operation Registers

Input words can be transferred directly to the A or B register through the E and C buses. These transfers are always controlled by an instruction, with the instruction designating the operation register to receive the word.

1.6.11 Output from Operation Registers

Words can be transferred directly from the A or B registers to the I/O cable through the S, C and E buses. These transfers are controlled by an instruction which connects the selected register on the S bus.

1.6.12 Operation Register to Operation Register

The contents of an operation register can replace or modify the contents of the register itself or another register. The process of incrementing and restoring the contents of the P register has been previously described. The contents of the A, B, and X registers can be transferred, incremented, complemented, decremented, or shifted. All these operations involve selecting the register onto the S bus, processing in the adder, and transferring back through the C bus. Note that shifting is performed in this transfer path. The contents of the selected register are shifted left or right as they are gated from the arithmetic logic gates to the C bus. This transfer path is involved in all register change instructions.

1.7 DECODING

The operation code and M fields of the instruction words (refer to Chapter III, section 1.2.1) stored in the U register are decoded to provide static control levels used throughout the execution of the instruction. In the following discussion, reference will be made to the following logic diagrams contained in the 620/i maintenance manual Volume 2.

Op. Code Decoding

DM110, sheet 2

Address/Function Decoding

DM110, sheet 1

Note that the gating terms shown in these diagrams correspond to the U register bit positions. These bit positions correspond to the instruction fields discussed in chapter III and summarized in table IV-4.



CHAPTER IV 620 COMPUTER SYSTEMS

for double-word instructions where the second word is an address (e.g., jump) by placing a one in the I bit of the second word.

1.7.1 Operation Code Decoding

The instructions operation code contained in bits 12 through 15 of the instruction word is decoded in three functional categories: class, set, and group. These three categories, which encompass all types of instruction performed by the computer, have been chosen to minimize the gating required to implement the program operations by generating terms common to many instructions. The complete operation code decoding structure is shown in drawing DM110 in the maintenance manual. The three categories of decoding are summarized in tables IV-5, IV-6, and IV-7.

- a. *Class decoding* separates instructions into three classes: single-word addressing, single-word nonaddressing or double-word and I/O.
- b. *Set decoding* simplifies gating requirements for the execution of the single-word addressing instructions. Sets H1XX + through H4XX + define subcategories of the single-word addressing instructions. Timing functions are used to select the appropriate phase for executing the instruction.
- c. *Group decoding* is an arbitrary structure. One of the group terms is true for all single-word addressing instructions. These terms are used in various gating structures to implement the separate operations.

1.7.2 M Field Decoding

The M field of the instruction word (bits 9 through 11) is decoded to specify the following according to the instruction class defined in the operation code:

Class K1	Addressing Mode
Class K2	Instruction Type
Class K3	Instruction Type

Note that for class K1 instructions, the instruction type (load, store, arithmetic, or logic) is specified by the operation code; and for class K2, the instruction type is specified by the M field. Class K3 contains all I/O type instructions, and the M field specifies a subtype. M field decoding as a function of the class is summarized in table IV-8.

In class K1 instructions, the indirect addressing mode is specified by 07 in the M field (AC7X +); the indirect addressing level is extended by placing a one in the I bit of the indirect address words read from memory. Indirect addressing may also be accomplished

**CHAPTER IV**
620 COMPUTER SYSTEMS**Table IV-4. Instruction Storage in U Register**

U Register Output	Instruction	
	Bit No.	Field
U15X +	15	Operation Code
U14X +	14	
U13X +	13	
U12X +	12	
U11X +	11	M Field
U10X +	10	
U09X +	9	
U08X +	8	A Field
U07X +	7	
U06X +	6	
U05X +	5	
U04X +	4	
U03X +	3	
U02X +	2	
U01X +	1	
U00X +	0	



CHAPTER IV 620 COMPUTER SYSTEMS

Table IV-5. Operation Code Classes

Code	Class Desig.	Gating Terms	Description	Instruction Types
01-07, 11-17	K1	K1XX + K1XX +	All single- word address- ing instruc- tions	Load, Store, Arith- metic Logical
00	K2	K2XX +	Single-word nonaddress- ing and double-word.	Jump, Jump and Mark, Execute, immediate Register change, Logic, Shift, Arith- metic shift, control extended
10	K3	K3XX +	All I/O in- structions	Input/Output

Table IV-6. Operation Code Sets

Code (octal)	Set Desig.	Gating Terms	Description
00-03	H1	H1XX +	Instruction cycle execute phase of all load instructions.
04-07	H2	H2XX +	Operand cycle execute phase on all store instructions
11-15	H3	H3XX +	Instruction cycle execute phase of all arithmetic and logic instructions. (Except INR)
16-17	H4	H4XX-	

**CHAPTER IV
620 COMPUTER SYSTEMS****Table IV-7. Operation Code Groups**

Code (octal)	Group Desig.	Gating Terms
01, 05, 11, 15	G1	G1XX +
02, 06, 12, 16	G2	G2XX +
03, 07, 13, 17	G3	G3XX +
04, 14	G4	G4XX +

Table IV-8. M Field Decoding

Class Desig.	M Field (Octal)	Gating Terms	Addr. Mode, Type, or Subtype	Description
K1	0-3	ACOX + to AC3X +	Direct address	Refer to chapter III, section 1.2.1
	4	AC4X +	Relative ad- dress	
	5	AC5X +	Index, index/ indirect (X register)	
	6	AC6X +	Index, index/ indirect (B register)	
	7	AC7X +	Indirect address	



CHAPTER IV

620 COMPUTER SYSTEMS

Table IV-8. M Field Decoding (*continued*)

Class Desig.	M Field (Octal)	Gating Terms	Addr. Mode, Type, or Subtype	Description
K2	0	AC0X +	Control	HLT only
	1	AC1X +	Jump	All
	2	AC2X +	Jump and mark	All
	3	AC3X +	Execute	All
	4	AC4X +	Shift	Arithmetic and logic
	5	AC5X +	Register change	All
	6	AC6X +	Immediate	All
	7	AC7X +	Miscellaneous	Set/reset OF
K3	0	AC0X +	External control	EXC
	1	AC1X +	Sense	SEN
	2	AC2X +	Data input	Operation registers and memory
	3	AC3X +	Data output	Operation registers and memory
	4	AC4X +	Extended external control	



CHAPTER IV
620 COMPUTER SYSTEMS

SECTION 2
620/f, 620/f-100 SYSTEMS

2.1 INTRODUCTION

The Varian 620/f computer is a high-speed, general-purpose, digital computer for scientific and industrial applications, its features include:

Fast Operation	750-nanosecond memory cycle
Large Instruction Set	142 plus 8 optional instructions
Word Length	16 bits
Modular Core Memory	Expandable to 32,768 words in 4,096- or 8,192-word increments
Automatic Data Transfer	Direct memory access (DMA) facility provides automatic data transfers with rates to 275,000 words per second; priority memory access (PMA) for transfer rates to 1.3 million words per second
Multiple Addressing	Direct, indirect, relative, preindexed and postindexed, immediate, extended, and indirect indexed
Flexible I/O	64 devices can be placed on the partyline I/O bus. The I/O system can easily be expanded to include features such as automatic block transfer, multilevel priority interrupt, and cycle-stealing data transfers
Extensive Software	DAS 4A, DAS 8A, and DAS MR (macro) assemblers;



CHAPTER IV 620 COMPUTER SYSTEMS

binary load/dump (BLD II); debugging (AID II); computer diagnostics (MAINTAIN II); mathematical subroutines; real-time monitor (RTM); source program editor (EDIT); master operating system (MOS) for fixed- and moving-head discs, drum, and magnetic tape; ANSI FORTRAN IV; conversational BASIC; report generator (RPG IV, a business-oriented language); and an extensive library of programs in the VOICE users' group

Table IV-9 lists the 620/f specifications.

Table IV-9. 620/f Specifications

Parameter	Description
Type	General-purpose, parallel-operation digital computer
Memory (Read/Write)	A 3-wire/3D magnetic core memory with a 16-bit word length, 750-nanosecond full cycle time, 400-nanosecond access time, 4,096-word basic and expandable to 32,768 words in 4,096 increments, asynchronous with CPU operation
Word Length	16 bits
Machine Cycle Speed	750 nanoseconds
Operation Registers	A register: 16-bit accumulator and shift register

**CHAPTER IV**
620 COMPUTER SYSTEMS**Table IV-9. 620/f Specifications (continued)**

Parameter	Description	
Auxiliary Registers	B register:	16-bit accumulator and shift register (least significant half of double- length accumulator) or index register
	X register:	16-bit index register
	P register:	15-bit program counter and index register for relative addressing
	I register:	16-bit instruction register
	L register:	15-bit memory address register
	R register:	16-bit arithmetic buffer register
Arithmetic	D register:	16-bit input/output register
	Binary, two's complement notation	
	Add or Subtract	1.5 microseconds
	Multiply (optional)	6.4 microseconds
	Divide (optional)	6.4 microseconds
	Register Change	750 nanoseconds
Arithmetic Operation Times	Input/Output	From A or B reg- ister, 1.5 micro- seconds
		From memory, 2.25 microseconds
Logic Levels	Positive Logic:	
	(Internal)	
	True = +2.4V minimum, +5V maximum	
	False = -0.5V minimum, +0.5V maximum	
	Negative Logic:	
	(I/O Bus)	
	True = -0.5V minimum, +0.4V maximum	
	False = +2.8V minimum, +3.6V maximum	



CHAPTER IV
620 COMPUTER SYSTEMS

Table IV-9. 620/f Specifications (continued)

Parameter	Description
Addressing Modes:	Direct: to 2,048 words Relative to P register: to 512 words Pre- and postindexed with X register hardware: to 32,768 words (does not add to execution time) Pre- and postindexed with B register hardware: to 32,768 words (does not add to execution time) Multilevel indirect: to 32,768 words Immediate Indirect indexed: to 32,768 words Extended: to 32,768 words
Instructions	142 plus 8 optional instructions
Instruction Types	One-word addressing One-word nonaddressing Two-word addressing Two-word nonaddressing
Input/Output	Asynchronous
I/O Program Control Instructions	Data transfer in: One-word nonaddressing Two-word addressing Data transfer out One-word nonaddressing Two-word addressing External control One-word nonaddressing Program sense Two-word addressing
Computer Options	Memory protection (MP) Teletype Controller (TTY) Buffer interlace controller (BIC) Power failure/restart (PF/R) Real-time clock (RTC) Automatic bootstrap loader (ABL)

**CHAPTER IV**
620 COMPUTER SYSTEMS**Table IV-9. 620/f Specifications** *(continued)*

Parameter	Description
	Priority interrupt module (PIM)
	Priority Memory Access (PMA)
	Optional instruction set:
	Hardware multiply/divide (M/D)
	Bit test (BT)
	Skip if register equal (SRE)
Software	<p>SYMBOLIC ASSEMBLER: Modular two-pass symbolic assembler operating in the basic 4,096-word memory. Includes 17 basic pseudo-operations. The 8,192-word memory version includes over 30 pseudo-operations</p> <p>FORTRAN: Modular one-pass compiler; subset of ANSI FORTRAN for 8,192-word memory</p> <p>AID: Program analysis package that assists programmers in operating the machine and debugging other programs. Includes basic operational executive subroutines</p> <p>DIAGNOSTICS: Software package that provides fast off-line verification of CPU and peripheral operation and assists in isolating and correcting suspected faults</p> <p>SUBROUTINES: Complete library of basic mathematical, fixed- and floating-point, single- and double-precision, number conversion and peripheral communi-</p>



CHAPTER IV 620 COMPUTER SYSTEMS

Table IV-9. 620/f Specifications *(continued)*

Parameter	Description
	<p>cation subroutines plus provisions for adding application-oriented routines</p> <p>MOS: The master operating system (MOS) provides for automatic batch processing that includes a minimum 8K core memory</p> <p>BASIC: BASIC is an easy-to-use programming language for business and scientific applications, permitting an inexperienced operator to program the system with only a few hours training</p> <p>RPG IV (optional): The report program generator (RPG IV) system, a hardware/software package, produces reports, financial statements, sale records, and other commercial documents in tabular form</p>
Dimensions	<p>The mainframe and expansion chassis' I, II, and III are 10.5 inches (26.6 cm) high, 19 inches (48.1 cm) wide, and 21 inches (53.1 cm) deep (expansion chassis III is 15 inches (37.9 cm) deep). The mainframe power supply is approximately 5.25 inches (13.3 cm) high, 19 inches (48.1 cm) wide, and 21 inches (53.1 cm) deep. The expansion power supply is approximately 5.25 inches (13.3 cm) high, 19 inches (48.1 cm) wide, and 18 inches (45.7 cm) deep. The 620/f-100 mainframe power supply is located in the mainframe chassis with the CPU tray.</p>

**CHAPTER IV**
620 COMPUTER SYSTEMS**Table IV-9. 620/f Specifications (continued)**

Parameter	Description
Weight	The mainframe and expansion chassis each weigh approximately 65 pounds (29.3 kg). The mainframe power supply weighs approximately 80 pounds (36.2 kg). The expansion power supply weighs approximately 60 pounds (28.6 kg).
Input Voltage	105 to 125V ac or 210 to 250V ac at 50 or 60 Hz (For compatibility with the teletype, frequency must be either 50 or 60 (+1/2, -0) Hz.)
Input Current	The mainframe power supply requires approximately 15 amperes ac; each expansion frame power supply requires approximately 4 amperes ac
Temperature Operating Storage	0 to 50 degrees C -20 to 70 degrees C
Mainframe Power Supply Outputs	+3V at 5 amperes +5V at 50 amperes -5V at 2 amperes +12V at 4 amperes -20V programmed at 6 amperes +40V at 2 amperes (The -20V output is controlled by a sensistor in the memory stack to regulate the current in memory-inhibiting lines.)
Expansion Power Supply Outputs	+5V at 20 amperes -5V at 4 amperes +12V at 4 amperes -12V at 4 amperes
Humidity Operating Storage	To 90 percent without condensation To 95 percent without condensation



CHAPTER IV 620 COMPUTER SYSTEMS

Table IV-9. 620/f Specifications (continued)

Parameter	Description
Vibration	3 to 10 Hz at 1g force or 0.25 double amplitude, whichever is less. Exponentially raised frequency from 3 to 10 Hz and back to 3 Hz over a 10-minute period, three complete cycles. This specification applies for all three principal axes
Shock	4g for 5 to 11 milliseconds, essentially sine shock waveform (all three principal axes; both directions in each axis)

2.2 SWITCHES AND INDICATORS

Figure IV-8 shows the switches and indicators on the control panel of the 620/f computer. Their uses are discussed individually in the following subsections. Used with a teletypewriter and peripheral devices, the control panel contains all controls necessary to operate the 620/f computer.

The front panel of the power supply has an AC PWR ON indicator light.

2.2.1 Power Switch

The key-operated power switch controls the ac input to the 620/f power supply.

In the PWR OFF position, ac input to the power supply primary is disabled.

In the PWR ON position, there is ac power to the power supply primary and the system should be fully operational.

In the PWR ON DISABLE position, there is ac power to the power supply primary and the computer is operational. However, all control console switches are disabled except the power switch itself. Pressing any other switch while the power switch is in PWR ON DISABLE has no effect.

The control panel and power supply indicator lights are functional when the POWER switch is in PWR ON or PWR ON DISABLE.

The key can be removed from the power switch in any of the three positions. To turn off



CHAPTER IV
620 COMPUTER SYSTEMS

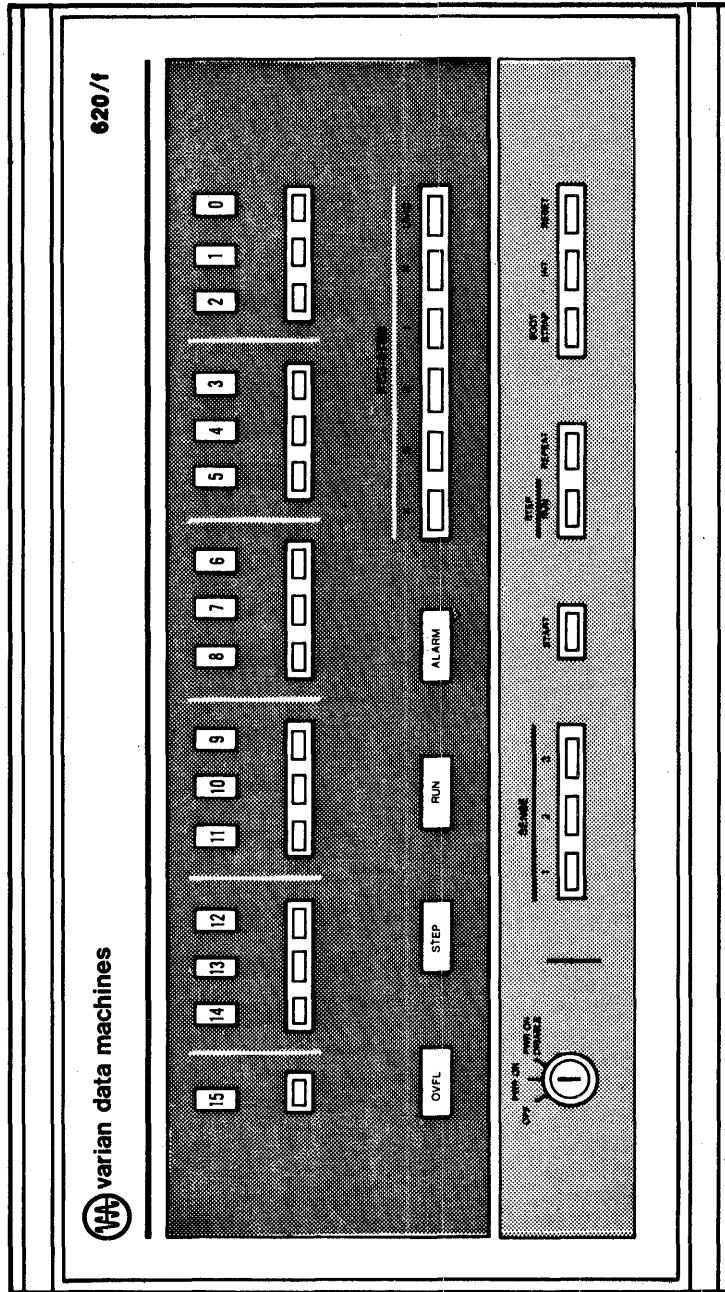


Figure IV-8. 620/f Computer Control Panel

V711-0703



CHAPTER IV 620 COMPUTER SYSTEMS

the computer, place the power switch in the PWR ON position, lift the STEP/RUN switch, then turn the power switch to PWR OFF.

2.2.2 STEP/RUN Switch and STEP and RUN Indicators

When the STEP/RUN switch is up, the 620/f is in step mode and the STEP indicator is lit. When the switch is down, the computer is in run mode. The RUN indicator lights after START is depressed.

If the computer is in step mode:

- a. Pressing the STEP/RUN switch to RUN position puts the computer in run mode.
- b. Pressing the START switch executes the instruction in the I register, and fetches the next instruction from the address specified by the contents of the P register and places it in the I register.

If the computer is in run mode:

- a. Lifting the STEP/RUN switch to STEP position halts the 620/f after completing the execution of the current instruction and fetches the next instruction and sets it in the I register. The RUN indicator goes out and the STEP indicator lights.
- b. Pressing the START switch starts the program at the address specified by the P register after executing the instruction in the I register.

2.2.3 BOOTSTRAP Switch

BOOTSTRAP is a momentary, spring-loaded switch that is functional in 620/f systems containing the optional automatic bootstrap. In other 620/f systems, this switch is present on the control panel, but it is not connected.

The bootstrap program enables the loading of the binary load/dump program into memory. Before the automatic bootstrap is loaded into memory, the binary load/dump tape should be inserted into the paper tape reader with the first binary frame at the read station.

To load the automatic bootstrap program:



CHAPTER IV 620 COMPUTER SYSTEMS

- a. Set the power switch to PWR ON.
- b. Set the STEP/RUN switch to RUN.
- c. Press and release BOOTSTRAP.

If the system does not contain an automatic bootstrap, load the provided bootstrap program manually.

2.2.4 START Switch

START is a momentary, spring-loaded switch. Pressing it when the 620/f is in the run mode starts the program. Pressing the START switch when the computer is in the step mode executes the instruction in the I register (except HLT), and fetches the next instruction from the address specified by the contents of the P register and places it in the I register.

2.2.5 REGISTER Switches

Pressing one of the five REGISTER switches selects the designated register (X, B, A, I, or P) for display or entry.

Only one register can be selected at a time. Simultaneously pressing two or more REGISTER switches disables the selection logic and ORs the front panel register display.

2.2.6 Register Entry Switches and Display Indicators

The 16 indicators across the top of the 620/f control panel display the contents of a selected register. Data are entered into registers on the corresponding register entry switches located under the indicators. The indicators and switches are read from left to right, bits 15 to 0. An illuminated indicator shows that that bit contains a one. For negative data, the sign bit (bit 15) is a one. The indicators and switches are divided into groups of three for ease in reading octal configurations.

2.2.6.1 REGISTER DISPLAY

To display the contents of a register, switch the STEP/RUN switch to STEP and press the REGISTER switch for the desired register.

The display indicators light when they correspond to register bits that contain ones. To remove the display, pull up on the REGISTER switch and the indicators go out.



CHAPTER IV 620 COMPUTER SYSTEMS

2.2.6.2 DATA OR INSTRUCTION ENTRY

To enter data or instructions in a register:

- a. Display the contents of the register.
- b. Enter ones by pressing down on the register entry switches corresponding to the bits to be set.
- c. Enter zeros in the other bits by pulling up on all other register entry switches. The indicator lights do not change when the register entry switches are manipulated. They still display the contents of the register.
- d. When the desired configuration is entered on the register entry switches, press LOAD. This loads the register with the configuration entered on the switches, and the indicators change to display this new configuration in the register.

To enter data into core memory:

- a. Load into the I register a storage instruction (STA, etc.).
- b. Select the register specified by the storage instruction in step a.
- c. Load the selected register using the data entry switches.
- d. Press START to execute the instruction in the I register. This stores the contents of the specified register at the effective memory address.

The TSA instruction can also transfer data entered on the control panel switches to the A register.

2.2.7 LOAD Switch

LOAD is a momentary, spring-loaded switch. When the 620/f is in step mode and a register has been selected, pressing this switch loads the register with the bit configuration entered on the register entry switches.



CHAPTER IV 620 COMPUTER SYSTEMS

2.2.8 REPEAT Switch

REPEAT is a toggle switch that is operative in both step and run modes. To repeat an instruction contained in the I register, press REPEAT, and then press START. The instruction is executed again and the program counter advances. However, the contents of the I register remain unchanged.

To run a program, REPEAT must be off.

2.2.9 SENSE Switches

The three SENSE switches are toggle switches permitting program modification by the operator. When the program contains instructions dependent on the setting of these switches, jumps and executions occur when the switch condition is met and do not occur when the switch condition is not met.

To set a SENSE switch, press down. To reset it, lift. Operations dependent on the position of this switch will be executed if the switch is in the position indicated by the instruction.

EXAMPLE

A program can be written so that the operator can obtain a partial total of a column of figures being added by use of the JSS1 (jump if SENSE switch 1 is set) instruction. The program writes individual entries as long as SENSE switch 1 is not set. When the operator wants a partial total, he sets the switch. The program then jumps to an instruction sequence that prints the desired information.

2.2.10 INT (Interrupt) Switch

INT is a momentary, spring-loaded switch used to interrupt the 620/f computer. It is functional only when the 620/f is in the run mode.

In systems that do not contain the optional priority interrupt module (PIM), pressing INT interrupts to memory address 0.

In systems containing a PIM, pressing INT interrupts to an even-numbered memory address specified by the PIM.



CHAPTER IV 620 COMPUTER SYSTEMS

2.2.11 RESET Switch

RESET is a momentary, spring-loaded switch used for initialization control and for stopping I/O operations. Pressing this switch halts the 620/f and initializes the computer and peripherals. This switch is electrically interlocked with the STEP/RUN switch and is disabled when the latter is in RUN.

Note that this switch is not a display reset.

2.2.12 OVFL (Overflow) Indicator

OVFL lights whenever an overflow condition exists.

2.2.13 ALARM Indicator

ALARM lights to signal an overheated system. If the POWER switch key is accessible, turn the power switch to PWR OFF and call the Varian customer service engineer.

If the power switch key is not accessible, turn off the power switch located on the back of the power supply, or pull the main plug, and call the Varian customer service engineer.

2.3 MANUAL OPERATIONS

With the 620/f in step mode, data or instructions can be manually transferred to or from memory or stored programs can be manually executed.

Note that the I register contains the instruction being executed, while the P register points to the address of the following instruction.

To load data or instructions into memory, to display the contents of memory, or to alter the contents of memory, follow the procedures in *Register-Entry Switches and Display Indicators*.

2.3.1 Loading Into Sequential Memory Addresses

To load a sequential group of memory addresses:

- a. Set STEP/RUN to STEP and press REPEAT.
- b. Load the P register with the base address.



CHAPTER IV 620 COMPUTER SYSTEMS

- c. Load into the I register a storage instruction (STA, etc.) with 100 in the M field (relative addressing), and zero in the A field.
- d. Select the register specified by the storage instruction in step c.
- e. Load the selected register using the data entry switches.
- f. Press START to execute the instruction in the I register.
- g. Repeat steps e and f until all instructions are loaded. The next cell to be loaded can be observed by displaying the P register.

2.3.2 Displaying From Sequential Memory Addresses

To display the contents of a group of sequential memory addresses:

- a. Place STEP/RUN in STEP, and press REPEAT.
- b. Load the P register with the base address.
- c. Load into the I register a loading instruction (LDA, etc.) with 100 in the M field (relative addressing), and zero in the A field.
- d. Select the register specified by the loading instruction in step c.
- e. Press START once for each memory location to be displayed.

2.3.3 Manual Execution of a Stored Program

To execute a stored program manually:

- a. Select step mode and turn off REPEAT.
- b. Set the P register to the first address of the program.
- c. Clear the I register.
- d. Press START.



CHAPTER IV 620 COMPUTER SYSTEMS

- e. Press START again to execute the instruction and to load the next instruction into the I register.
- f. Repeat step e once for each instruction.

2.3.4 Manual Repetition of Instructions

To repeat an instruction manually:

- a. Press the REPEAT switch down.
- b. Press START. This advances the P register but inhibits loading the I register. Thus, pressing START again executes the same instruction.

2.4 ORGANIZATION

The functional sections of the computer are illustrated in figure IV-9.

2.4.1 Control Section

The computer control section generates the basic 9.0-MHz system clock that provides the timing and control signals for all computer operations. It directs the transfer of data between the registers and controls CPU operations. It also interprets instructions read from memory and provides the necessary gating logic for executing them. Information from the instruction (I) register is used to generate the timing and control signals.

2.4.2 Decoding Section

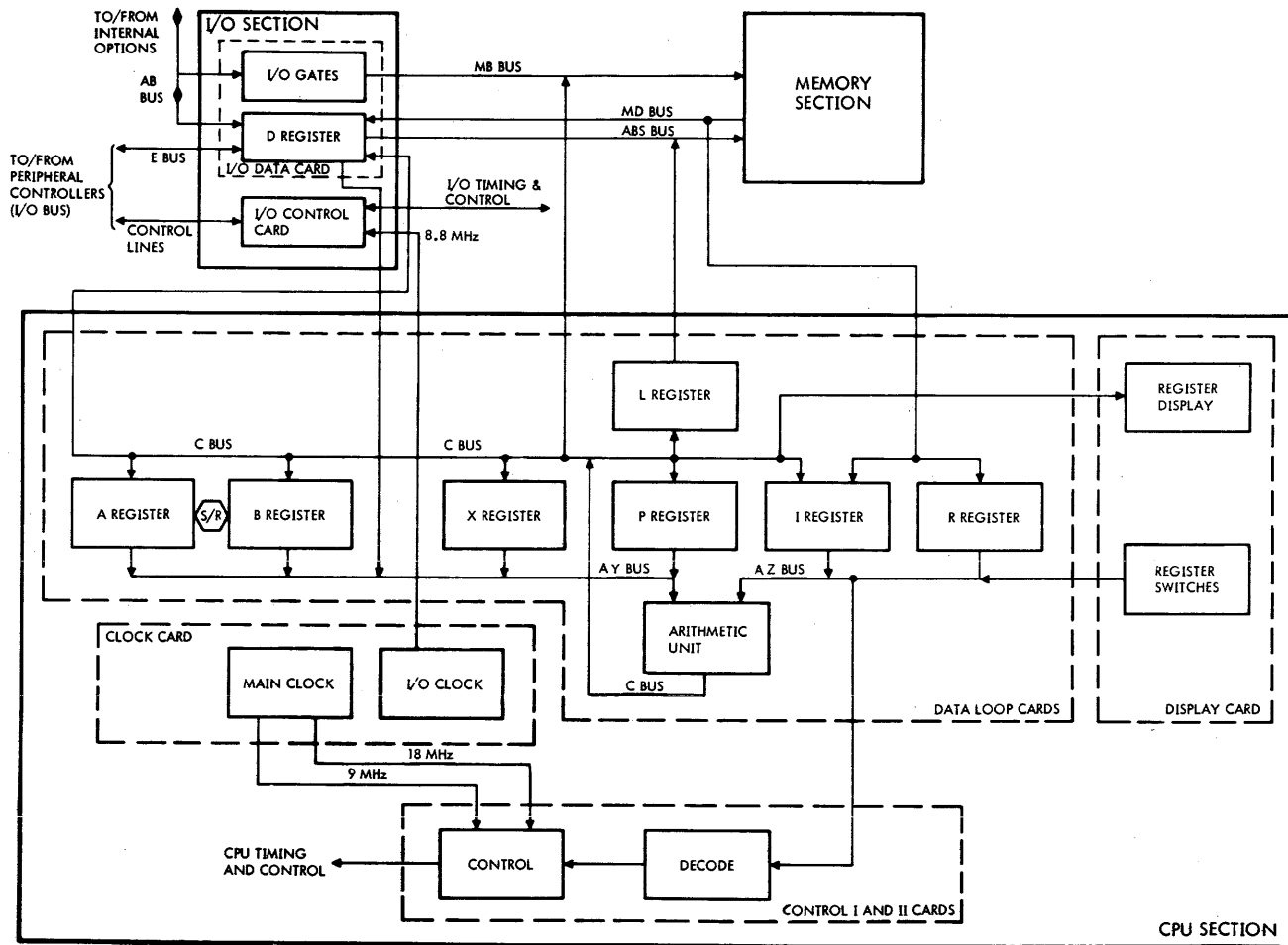
The decoding section decodes the fields of the instruction word held in the I register to determine the control signal levels. These levels select the timing signals generated by the timing unit in the control section.

2.4.3 Arithmetic Unit

The arithmetic unit contains the adder, gating, and control circuits required for all arithmetic and logic operations except shifting.



CHAPTER IV 620 COMPUTER SYSTEMS



VT12-0281B

Figure IV-9. 620/f Computer Functional Organization



CHAPTER IV 620 COMPUTER SYSTEMS

2.4.4 Operation Registers

The operation registers are designated A, B, X, and P. The A, B, and X registers are directly accessible to the programmer. The P register is accessible indirectly to the following:

- a. Jump instructions that modify the program sequence
- b. The relative addressing mode of the arithmetic/logic instructions that uses the contents of the P register to modify the operand address

A register. This 16-bit register is the upper half of the accumulator and accumulates the results of logical and addition/subtraction operations, the most significant half of the double-length product in multiplication, and the remainder in division. The A register can also be used for I/O transfer under program control.

B register. This 16-bit register is the lower half of the accumulator and accumulates the least significant half of the double-length product in multiplication and the quotient in division. It can also be used for I/O transfers under program control and as a second hardware index register.

X register. This 16-bit register permits indexing of operand addresses without adding time to the execution of indexed instructions.

P register. This 15-bit register holds the address of the current instruction and is incremented before each new instruction is fetched. A full complement of instructions is available for conditional and unconditional modification of this register. The P register is also used in relative addressing.

2.4.5 Auxiliary Registers

The auxiliary registers are designated I, L, R, and D.

I register. This 16-bit instruction register receives each instruction from memory through the W bus and holds the instruction during its execution. Instructions can be loaded in the I register from the C bus via the control panel register entry switches. The control fields of the instruction word are routed to the decoding section to determine the required timing and control signals. The five least significant bits of the I register are transferred into a shift counter to shift-count-control the shift instructions.



CHAPTER IV 620 COMPUTER SYSTEMS

L register. This 15-bit address register contains the address of the memory location currently being accessed during either a clear/write or read/restore cycle.

R register. This 16-bit buffer register holds the second and subsequent words of a double-word instruction. It also holds the multiplicand and divisor in arithmetic operations. The R register buffers the arithmetic unit from memory to permit interlace I/O operations on a memory cycle-stealing basis.

D register. This 16-bit register stores I/O information.

2.4.6 Data Switch Section

The data switch section provides gating logic for operand data being read from or written into memory via I/O. CPU information does not pass through this section.

2.4.7 Register Entry Switches/Display Indicators

The register-entry switches enter data and instructions, via the control panel, in the A, B, X, I, or P register. The display indicators display the contents of the A, B, X, I, or P register as selected on the control panel.

2.4.8 Shift-and-Rotate Circuit

The shift-and-rotate (S/R) circuit is a special data path to shift or rotate the contents of the A and B registers.

2.4.9 Internal Buses

The basic computer contains eight buses designated as C, AY, AZ, MB, MD, ABS, AB, and I/O.

C bus. The C bus provides the data path and selection logic for routing data from the arithmetic unit to the operation registers (A, B, X, and P), the auxiliary registers (D, I, and L), and register display.

AY bus. The AY bus routes selected data from either (or any combination of) the A, B, X, or P register, or I/O data (D) register to the AY input of the arithmetic unit.

AZ bus. The AZ bus routes selected data from the I and R registers and register-entry switches.



CHAPTER IV 620 COMPUTER SYSTEMS

MB and MD buses. The MB and MD buses provide data paths to and from memory, respectively.

ABS bus. The ABS bus routes address information into memory from the address (L) register and the D register.

AB bus. The AB bus provides data paths between the computer and the internal options (real-time clock, power failure/restart, and Teletype controller).

Input/output bus. The I/O bus is a party-line, bidirectional bus. It permits programmed data transfers between peripheral devices and the computer. The I/O bus also permits plug-in expansion of all peripheral controllers. Part of the I/O bus is an E bus used for bidirectional data transfer.

2.5 TIMING

The 620/f operates on a basic 750-nanosecond machine cycle. That is, a full memory cycle (read/restore or clear/write) is performed in each 750-nanosecond time interval (except in some special cases in which this period is extended as discussed in subsequent paragraphs). All operations performed by the computer are accomplished within some multiple of this basic timing period.

To execute the various operations, several suboperations are performed during the memory cycle time. Timing of these suboperations is controlled by the internal 18-MHz master clock. The period of this master clock is 55 nanoseconds; this permits multiple suboperations to be executed during the memory cycle period. Note that the first half-cycle (approximately 400 nanoseconds) of the memory period is used to access a word (read) or to clear a cell (clear); the second half-cycle is used to restore a word (restore) or to write a new word (write) into the cell.

2.5.1 Clocks

The clocks which control the timing of all operations in the computer are generated by the timing and control logic. These clocks are illustrated in figure IV-10 and listed in table IV-10.



CHAPTER IV 620 COMPUTER SYSTEMS

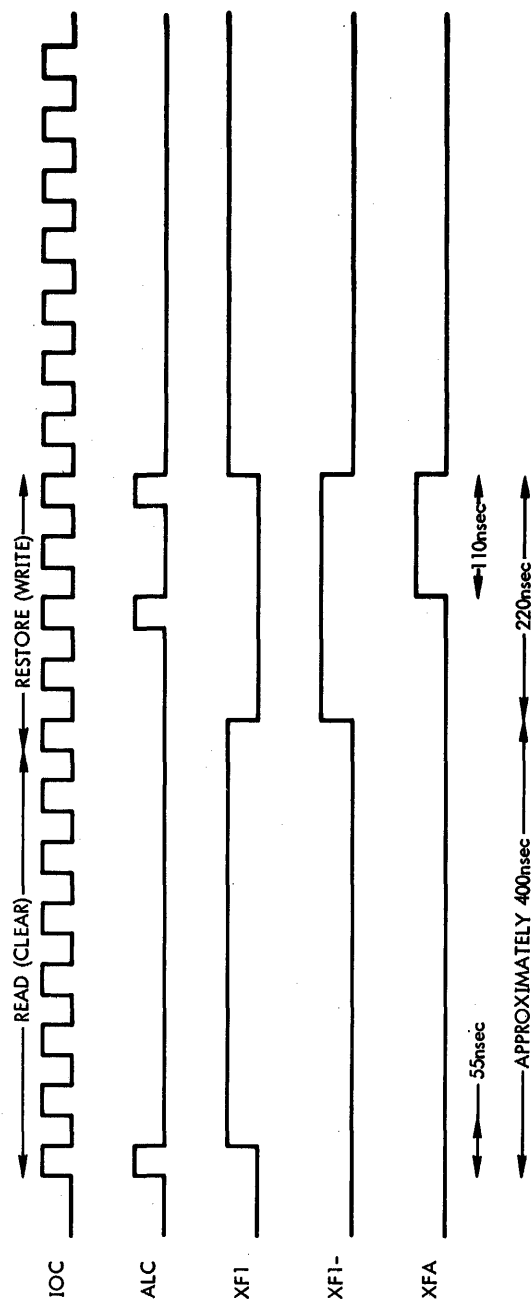


Figure IV-10. Basic Timing Clocks

VTII-1194



CHAPTER IV 620 COMPUTER SYSTEMS

Table IV-10. Basic Timing Clocks

Clock	Description
Master Clock (IOC)	Crystal-controlled timing signal (18 MHz) for the entire system
Alternate Clock (ALC)	9-MHz timing signal (counted down and synchronous with IOC); used to time the basic execute and address phases of the computer
Sequence State 1 (XF1 +)	Basic timing phase, synchronous with read or clear half-cycle of memory; all operations on words (transfers of data to and from memory) are performed during this period
Sequence State 2 (XF1-)	Basic timing phase, synchronous with restore or write half-cycle of memory; all transfer of instruction and operand addresses to memory are performed during this period
Sequence State 3 (XFA)	Basic timing phase used for execution or instructions and other operations

2.5.2 Clock Modifiers

All functions performed by the 620/f occur in two basic phases:

- a. Transfer of addresses to the memory L register (address phase)
- b. Operation upon words read from memory, or the storing of words into memory (execute phase)

These basic address and execution phases can be modified by certain program instructions or by signals received from devices external to the computer. The conditions under which the periods of the basic clocks are modified are:

Shift	During shifting operations with words contained in the A and B Registers, sequence states 2 and 3 are extended by the number of alternate clock periods (110 nanoseconds) equal to the specified number of shifts -1.
-------	---



CHAPTER IV

620 COMPUTER SYSTEMS

Interrupt	When an external interrupt is received, sequence states 2 and 3 are extended 0.9 microsecond to accommodate delays in receiving the interrupt address from the external device
Trap	When a buffer interlace controller requests a transfer to or from memory, sequence states 2 and 3 are extended 3.15 microseconds to permit the execution of the full trap sequence (routing of address and data from the external device)
Halt	On a halt instruction (ST1), the alternate clock is inhibited; this prevents any further operations until the STEP or RUN switch is pressed.

Modification of the execute phase of an instruction is illustrated in figure IV-11. This modified sequence is typical of a shift instruction. At time 0, the instruction has been fetched from memory. Starting at time 400, the instruction is executed; however, the normal 220-nanosecond sequence state 2 is extended 110 nanoseconds for each shift (six, in this illustration). In a similar manner, sequence state 3 is extended when required by the conditions defined above.

2.5.3 Sequence Control

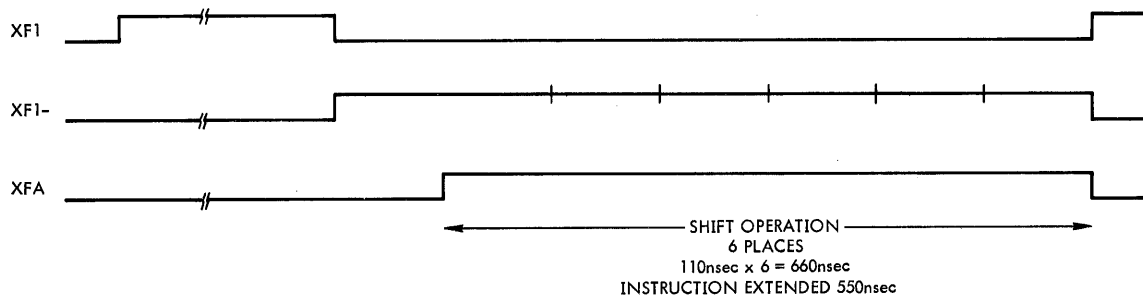
The basic clocks generated from the master clock are used to time three operating sequences: instruction cycle, operand cycle, and address cycle. All operations performed by the computer are timed by one or more of these timing sequences.

2.6 INFORMATION TRANSFER

All data communication between the basic functional elements of the machine is through the three data buses C, AY, and W. The C and AY buses are internal to the computer. The W bus is external and bidirectional; that is, a single set of lines is used to carry information both to and from the memory. The following paragraphs outline the major data transfer paths in the computer. Refer to figure IV-12.



CHAPTER IV 620 COMPUTER SYSTEMS



VTII-1195

Figure IV-11. Example of a Modified Clock Sequence

2.6.1 P Register to Memory

As an instruction cycle begins, the location of the next instruction is transferred from the P register to the L register. The contents of P are transferred through the AY bus to the arithmetic unit which increments the location address with the arithmetic gates, and restores the incremented count to the P and L registers. The memory address register, L, now contains the address of the next instruction word to be fetched from memory, and the P register holds the updated address.

2.6.2 Memory to I Register

During the instruction cycle, the instruction word located by the address in the L register is read out on the W bus and read in to the I register.

2.6.3 I Register to Memory

For many instructions requiring an operand, the address of the operand is contained in the instruction word held in the I register. This operand address is transferred to the L register through gates in the arithmetic unit and the C bus. The address from the I register can be modified during the transfer to L as follows:



CHAPTER IV 620 COMPUTER SYSTEMS

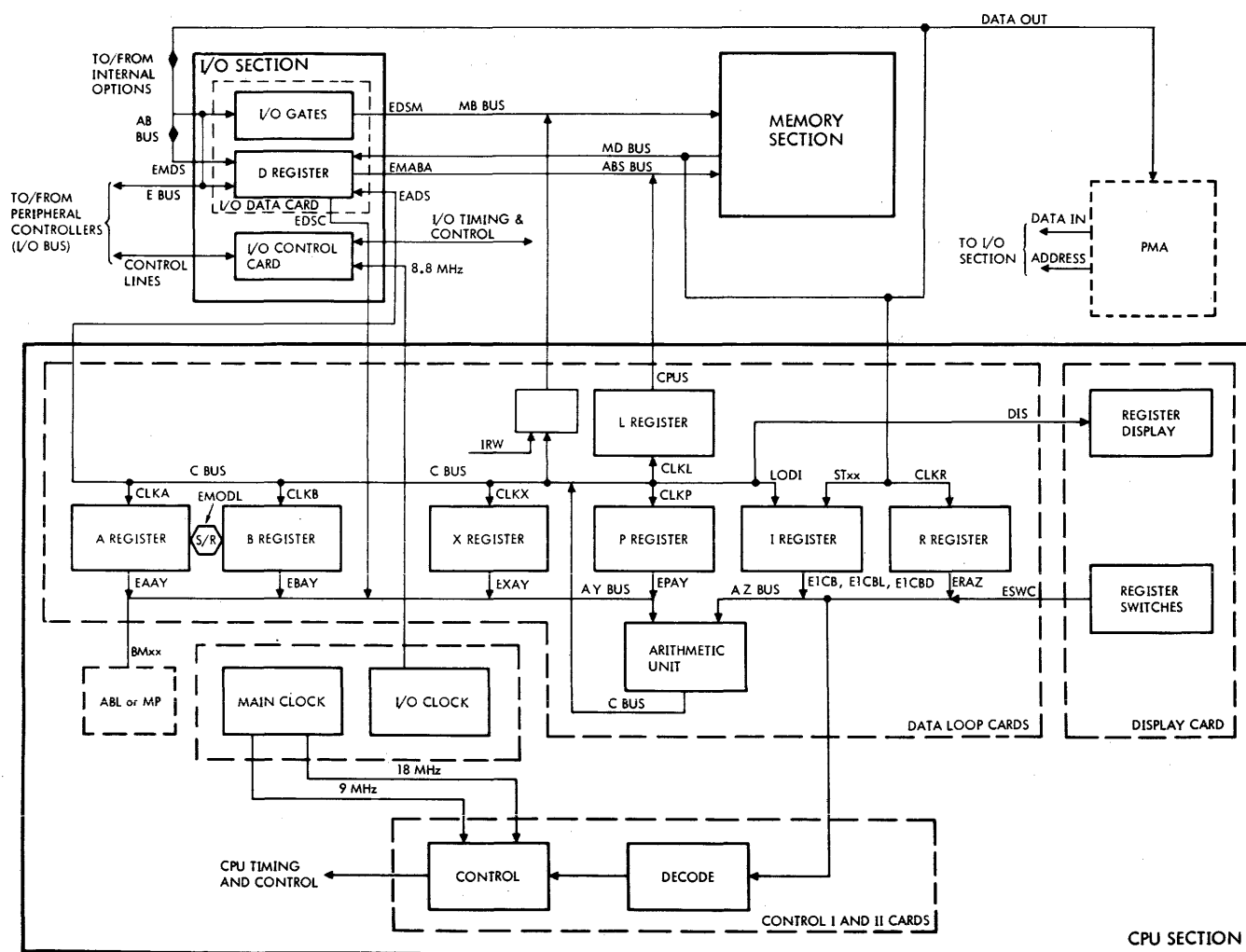


Figure IV-12. 620/f Organization

VT12-398



CHAPTER IV 620 COMPUTER SYSTEMS

- a. *Direct Address.* No modification; bits 0 through 10 transferred from I to L directly address operand in the first 2,098 memory addresses.
- b. *Relative Address.* The effective operand address transferred to L is formed by adding bits 0 through 8 from I to the contents of P. Addition is performed by selecting the contents of P and I and bringing them into the arithmetic unit. This permits addressing any word up to 512 locations ahead of the current program location.
- c. *Index Address.* The effective operand address transferred to L is formed by adding bits 0 through 8 from I to either the contents of X or B.
- d. *Indirect Address.* Same transfer as direct address but the word read from memory will be the address of an operand rather than the operand itself.

2.6.4 Memory to R Register

Operands read from memory are transferred to the R register. The operands are stored in R while an arithmetic or logical operation is being performed.

For indirect addressing, and for instructions whose operand address is stored in the memory location following the instruction word, the operand address will be read from memory and then transferred to the R register. The address is then routed to the L register through the C bus.

2.6.5 Arithmetic Unit to Operation Register

Outputs from the arithmetic unit, generated as a result of an arithmetic operation involving the R register and one of the operation registers, are stored in an operation register through the C bus.

2.6.6 Operation Register to Memory

The contents of any one of the operation registers are transferred to memory by selecting the register onto the AY bus and routing the word through the arithmetic unit, C bus, and W data switch register. The contents of the P register may be transferred to the L register to address an instruction as previously explained. The contents of P and other registers can be stored in memory by the same path, except that the word is entered into W the data switch register. Note that an address cycle must precede this transfer to place the storage address in the L register.



CHAPTER IV
620 COMPUTER SYSTEMS

2.6.7 Memory to Operation Registers

The contents of a memory location can be transferred to any of the operation registers through W, AZ, and C buses. Note that an address transfer must precede the data transfer to place the memory address in the L register.

2.6.8 Input to Memory

Input data from the E bus can be routed directly to memory through the data switch register and W bus. Data transfer must be preceded by an address transfer to load the memory location into the L register. When the transfer is under control of an instruction, the memory address will be generated as a normal operand address.

2.6.9 Output from Memory

Output words can be transferred directly from memory to the I/O cable through the W and C buses and the data switch register. A storage address must first be transferred to the L register by an instruction.

2.6.10 Input to Operation Register

Input words can be transferred directly to the A or B registers through the E and C buses and the data switch register. These transfers are always controlled by an instruction, with the instruction designating the operation register to receive the word.

2.6.11 Output from Operation Registers

Words can be transferred directly from the A or B registers to the I/O cable through the AY, C, and E buses. These transfers are controlled by an instruction which connects the selected register on the AY bus.

2.6.12 Operation Register to Operation Register

The contents of an operation register can replace or modify the contents of the register itself or another register. The process of incrementing and restoring the contents of P has been previously described. The contents of the A, B, and X registers can be transferred, incremented, complemented, or decremented. All these operations involve selecting the register onto the AY bus, processing in the arithmetic unit, and transferring back through the C bus.



SECTION 3

620/L-100 SYSTEM

3.1 INTRODUCTION

The **Varian 620/L-100 Computer** is a general-purpose digital computer, designed for a variety of system applications.

The computer processes 16-bit words in a full-cycle execution time of 950 nanoseconds, or over one million cycles per second.

The instruction set of the Varian 620/L-100 comprises 133 standard instructions, many of which can be microcoded to extend the effective repertoire to several hundred instructions.

Core memory can be expanded in 4,096 word (4K) increments, from a minimum of 4K to a maximum of 32K. Improved design allows the packaging of a fully expanded 32K system in two 10-1/2-inch high, standard rack enclosures.

The central processing unit (CPU) features four user-accessible operation registers, five buffer registers, an overflow indicator, and convenient operator's control panel.

Six addressing modes can be implemented: direct, multilevel indirect, immediate, indexed, relative, and extended forms that permit direct addressing of any area of the fully expanded 32K system.

One power supply can furnish all the power required to maintain the maximum 32K system plus a number of peripheral controllers.

The computer mainframe chassis accommodates the circuitry for the CPU, an 8K master memory, all the available mainframe (internal) option, and up to nine peripheral controllers.

Mainframe standard features include: hardware multiply/divide and extended addressing (M/D), memory protection (MP), real-time clock (RTC), and power failure/restart (PF/R).

The standard Varian 620/L-100 party-line input/output (I/O) bus can interface a maximum of ten peripheral controllers. Additional peripheral controllers can be accommodated by including an I/O buffer card.



System I/O options include: priority interrupt module (PIM) and buffer interlace controller (BIC). The PIM establishes eight levels of interrupt priority for selected peripheral controllers and places interrupt requests on the I/O bus in order of priority. The BIC implements the direct memory access (DMA) capabilities of the basic computer, permitting cycle-stealing I/O data transfers between memory and peripheral controllers at rates of up to 382,720 words per second.

Table IV-11. 620/L-100 Specifications

Description	System-oriented, general-purpose digital computer for on-line data processing	
Memory	Magnetic core, with a 16-bit word length, 950-nanosecond full-cycle time, 425-nanosecond access time, and expandable from the basic 4,096-word (4K) minimum to a maximum of 32,768 words (32K)	
Arithmetic	Parallel, binary, fixed-point, two's complement	
Word Length	16 bits	
Machine Cycle Speed (Fetch and Execute)	Addition/subtraction	1.9 microseconds
	Multiplication (optional):	9.5 microseconds
	Division (optional):	9.5-13.2 microseconds
	Register modification:	0.95 microseconds
	A/B register input/output:	1.9 microseconds
	Memory input/output:	2.85 microseconds
Instruction Set	115 standard, and 18 optional, instructions, many of which can be microcoded for extended operations	
Instruction Types	One- and two-word addressing, and one- and two-word nonaddressing instructions performing the following functions:	
	Load/store Shift/rotation Register modification Arithmetic Logic	Jump Jump and mark Execution Control Input/output



Addressing Modes	Direct, to 2,048 words Relative to P register, to 512 words Indexed with X or B register, to 32,768 words (does not add to execution time) Multilevel indirect Immediate Extended
Operation Registers	A register: 16-bit accumulator and shift register B register: 16-bit accumulator and shift register (low-order half of the double-length accumulator), or index register X register: 16-bit index register P register: 16-bit program counter
Auxiliary Registers	U register: 16-bit instruction register L register: 15-bit memory address register W register: 16-bit memory data register S register: 5-bit shift register R register: 16-bit operand register
Control Panel	Register entry switches and display indicators; overflow (OVFL), STEP, and RUN indicators; REGISTER select and bit RESET switches; three SENSE switches; instruction REPEAT, STEP, and RUN switches; SYSTEM RESET, and three-position power switch
Logic and Signals	Integrated circuits and 4.211 MHz clock Internal logic levels: 0V = false (zero), +5V = true (one) Memory data logic levels: 0V = true (one), +5V = false (zero) I/O bus logic levels: +3 = false (zero), 0V = true (one)
Input/Output	Programmed I/O operations: external control, pro- gram sense, data transfer in, and data transfer out Automatic data transfers: direct memory access (DMA) with transfer rates over 382,720 words per second



	Interrupt system: allows computer options and peripherals to interrupt CPU operations
Standard Features	Multiply/divide and extended addressing: simplifies the programming of arithmetic and addressing operations Real-time clock: user-selected variable time base for time and event accumulation Power failure/restart: protects a program in progress during power failures
Computer Option	Bootstrap protection: protects the memory address containing the bootstrap loader routine and the binary load/dump program
I/O Options	Priority interrupt module: establishes and implements interrupt priorities for peripherals Buffer interlace controller: permits direct access to memory for block data transfers
Input Voltage	105 to 125V ac, or 210 to 250V ac, at 50 or 60 Hz
Input Current	Power supply requires 5 amperes at 115V, and 3 amperes at 230V
Dimensions	Mainframe and expansion chassis: 10.5 inches (26.6 cm) high, 13 inches (32.9 cm) deep, and 19 inches (48.1 cm) wide Power supply: 10.5 inches (26.6 cm) high, 7.5 inches (18.9 cm) deep, and 17.75 inches (44.9 cm) wide
Weight	Mainframe and expansion chassis: approximately 35 pounds (15.9 kg) without circuit cards Power supply: approximately 36 pounds (16.3 kg)
Temperature	Operating: 0 to 50 degrees C Storage: -20 to 70 degrees C
Humidity	Operating: to 90 percent without condensation Storage: to 95 percent without condensation

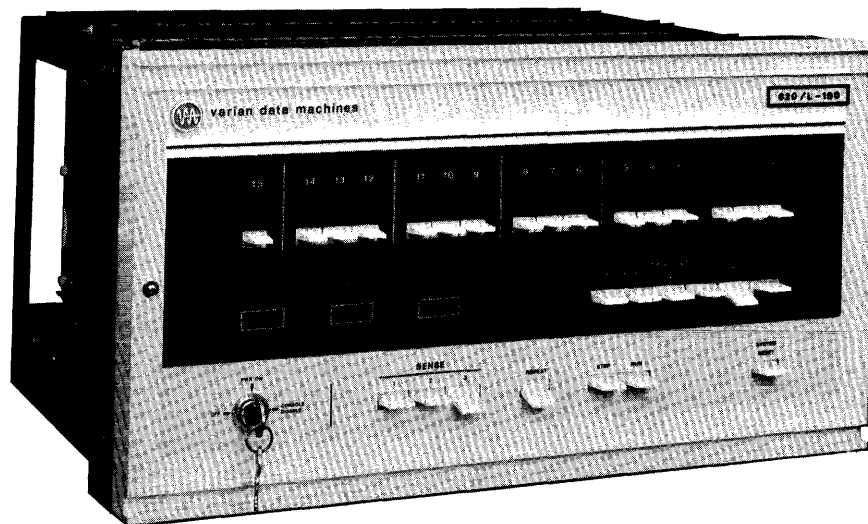
**Vibration**

3 to 10 Hz at 1g force or 0.25 double amplitude, whichever is less; exponentially raised frequency from 3 to 10 Hz and back to 3 Hz for 10 minutes, three complete cycles; applies to all three principal axes

Shock

4g for 11 milliseconds, essentially sine shock waveform (all three principal axes, both directions in each axis)

Figure IV-13 presents an outline of the 620/L-100 computer.



VHT-172

VHT-0172

Figure IV-13. Varian 620/L-100 Mainframe



3.2 SYSTEM OPERATION

Program Execution

The Varian 620/L-100 requires very little preparation before a program can be executed. Assuming that the system, including peripherals, is properly installed and connected to an ac power source, the following procedure is followed to make a cold start (i.e., when a new system is being initialized or the contents of memory are unknown).

- a. Turn on computer power by placing the power keyswitch on the control panel (figure IV-14) to PWR ON.
- b. Initialize the system by pressing SYSTEM RESET, then reset all registers using the REGISTER selection and BIT RESET switches.
- c. Load the appropriate bootstrap loader routine (table IV-12) from the control panel.
- d. Load the binary load/dump program (BLD II,) using the Teletype or high-speed paper tape reader (depending on the bootstrap loader selected). Verify after loading that the P register contains the proper starting address.
- e. Load the object program using the same paper tape reader as that used for BLD II.
- f. Press the RUN switch on the control panel.

This section describes control panel switches and indicators, and implementation of the above procedure and manual operations.

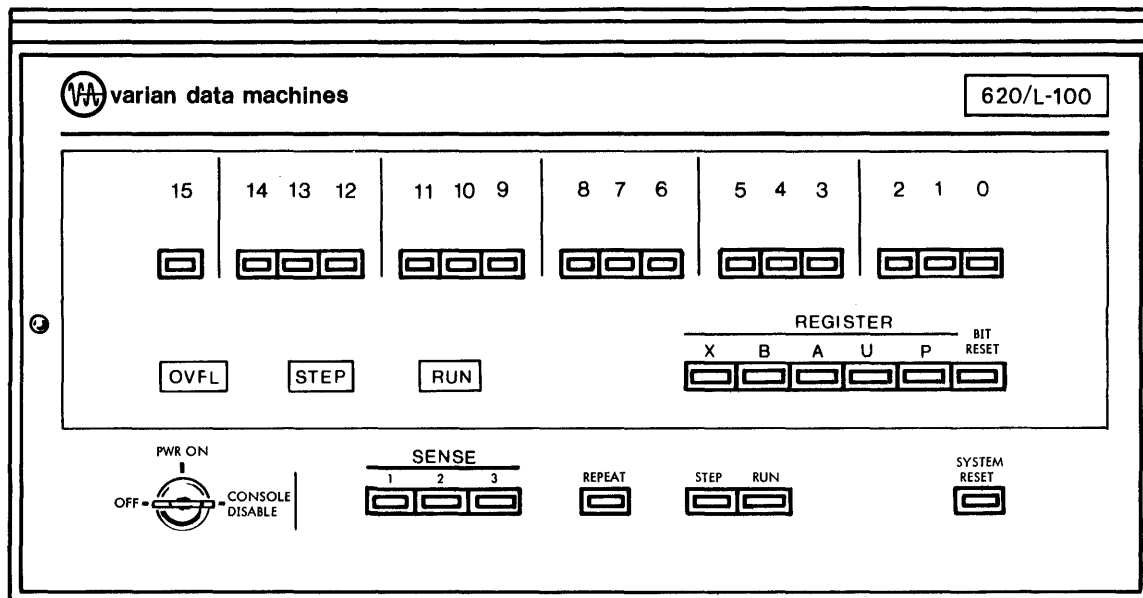
Switches and Indicators

The control panel of the Varian 620/L-100 is illustrated in figure IV-14.

Power Switch

The key-operated power switch controls the ac input to the power supply.

In the OFF position, ac input to the power supply primary is disabled. In the PWR ON position, there is ac power to the power supply primary and the system should be fully operational. In the CONSOLE DISABLE position, there is ac power to the power supply



VTII-1830

Figure IV-14. Varian 620/L-100 Control Panel

primary and the computer is operational. However, all control panel switches are disabled except the power switch itself. Pressing any other switch while the power switch is in CONSOLE DISABLE has no effect. The control panel indicators are functional when the power switch is in either the PWR ON or CONSOLE DISABLE position.

The power switch key can be removed in any of the three positions. To turn off the computer, place the switch in PWR ON, press the STEP switch, then turn the power switch to OFF.

STEP Switch and Indicator

Pressing the momentary, spring-loaded STEP switch when the computer is in run mode (RUN indicator on) halts the computer after execution of the current instruction. Pressing STEP when the computer is halted executes the instruction currently in the instruction (U) register (step mode).

When STEP is pressed, the STEP indicator lights; it goes out when the RUN switch is pressed.



RUN Switch and Indicator

Pressing the momentary, spring-loaded RUN switch executes the instruction currently in the U register and starts automatic processing of the stored program at the address specified by the P register.

When RUN is pressed, the RUN indicator lights; it goes out when the STEP switch is pressed.

REGISTER Selection Switches

Pressing one of the five toggle-action REGISTER selection switches selects the designated registers (X, B, A, U, and P) for display or entry.

Only one register can be selected at a time. Simultaneously pressing two or more REGISTER switches disables the selection logic and the register display indicators.

Register Entry Switches and Display Indicators

The 16 indicators across the top of the control panel display the contents of a selected register. Data are entered into registers on the corresponding register entry switches located under the indicators. The indicators and switches are read from left to right (bits 15 through 0). A lighted indicator shows that bit contains a one. For negative data, the sign bit (bit 15) is a one. The indicators and switches are divided into groups of three for ease in reading octal configurations.

To display the contents of a register, press STEP and select the desired register. The display indicators light when they correspond to register bits that contain ones. To remove the display, pull up on the REGISTER switch.

To enter data or instructions in a register:

- a. Display the contents of the selected register.
- b. Clear the register to all zeros by pressing the BIT RESET switch.
- c. Enter ones by pressing down on the register entry switches corresponding to the bits to be set. The associated display indicator lights for each switch pressed.



To enter data into computer memory:

- a. Load a storage instruction (e.g., STA) into the U register.
- b. Select the register specified by the storage instruction.
- c. Load the data word into the selected register using the data entry switches.
- d. Press STEP to execute the instruction in the U register. This stores the contents of the specified register at the effective memory address.

BIT RESET Switch

Pressing the momentary, spring-loaded BIT RESET switch when the computer is in step mode resets all bits of the selected register to zero. All register display indicators go out.

REPEAT Switch

The toggle-action REPEAT switch permits manual repetition of an instruction in the U register. When REPEAT is down, pressing STEP executes the instruction and advances the P register to the next program address. The U register contents remain unchanged. REPEAT is disabled when the computer is in run mode.

SENSE Switches

The three toggle-action SENSE switches permit program modification by the operator. When the program contains instructions dependent on the setting of these switches, jumps and executions occur when the switch condition is met and do not occur when the condition is not met.

To set a SENSE switch, press down. To reset it, lift. Operations dependent on the position of this switch are executed if the switch is in the position indicated by the instruction.

EXAMPLE

A program can be written so that the operator can obtain a partial total of a column of figures being added by use of the JSS1 (jump if SENSE switch 1 is set) instruction. The program writes individual entries as long as SENSE switch 1 is not set. When the operator wants a partial total, he sets the switch. The program then jumps to an instruction sequence that prints the desired information.



SYSTEM RESET Switch

The momentary, spring-loaded SYSTEM RESET switch is used for initialization control and for stopping I/O operations. Pressing this switch halts the computer and initializes it and all peripherals. *Note that SYSTEM RESET does not reset the registers.*

OVFL (Overflow) Indicator

OVFL lights when a program overflow condition exists.

3.3 MANUAL OPERATIONS

Loading the Bootstrap Loader

After computer power is turned on and the system initialized, load the bootstrap loader routine (table IV-12):

- a. In step mode, load a store A register relative to P instruction (054000) into the U register.
- b. Press the REPEAT switch.
- c. Load the starting memory address of the bootstrap loader (007756) into the P register.
- d. Load the first bootstrap loader instruction into the A register. If the high-speed paper tape reader is to be used for subsequent program input, select the column headed High-Speed Reader Code in table IV-12; if using the Teletype paper tape reader, select the column headed Teletype Reader Code.
- e. Press STEP to load the A register contents into the address specified by the P register, which is incremented by one after the instruction is loaded.
- f. Clear the A register by pressing BIT RESET.
- g. Repeat steps d, e, and f for each bootstrap loader instruction.



Table IV-12. Bootstrap Loader Routines

Address	High-Speed Reader Code	Teletype Reader Code	Symbolic Coding		
007756	102637	102601	READ	CIB	RDR
007757	004011	004011		ASLB	NBIT - 7
007760	004041	004041		LRLB	1
007761	004446	004446		LLRL	6
007762	001020	001020		JBZ	SEL
007763	007772	007772		(Memory address)	
007764	055000	055000		STA	0,1
007765	001010	001010		JAZ	LHLT + 1
007766	007000*	007000		(Memory address)	
007767	005144	005144		IXR	
007770	005101	005101	ENTR	INCR	1
007771	100537	102601	SEL	SEL	RDON
007772	101537	101201		SEN	IBFR,READ
007773	007756	007756		(Memory address)	
007774	001000	001000		JMP	* - 2
007775	007772	007772		(Memory address)	

NOTE

The bootstrap loader routine is always loaded into the highest addresses of the first 4K memory increment, regardless of available memory.

* Replace this code with 007600 if the test executive of MAINTAIN II (refer to document number 98 A 9952 060) is to be loaded and executed.

To determine that the bootstrap loader is correctly loaded:

- Initialize the system by pressing SYSTEM RESET.
- Clear all registers by momentarily pressing each REGISTER selection switch, pressing BIT RESET each time.
- Load LDA instruction 014000 (load A register relative to P) into the U register.



- d. Load the bootstrap loader's starting memory address (007756) in the P register, keeping the REPEAT switch in the down position.
- e. Select the A register and press STEP. The contents of each memory address are displayed sequentially each time STEP is pressed.
- f. If an error is found, load the correct instruction code into memory.

NOTE

The P register error address is always the error address plus one.

Loading, Displaying, and Altering Memory

To load data or instructions into memory, to display the contents of memory, or to alter the contents of memory, follow the procedures given in the description of the register entry switches and display indicators above.

Loading Sequential Memory Addresses

To load a sequential group of memory addresses, follow the procedures for loading the bootstrap loader routine using the A, B, or X register and loading the base address of the instruction group into the P register.

Displaying Sequential Memory Addresses

To display the contents of a group of sequential memory addresses:

- a. Press STEP and REPEAT.
- b. Load the base address of the instruction group into the P register.
- c. Load into the U register a relative-addressing load instruction (LDA, LDB, or LDX).
- d. Select the register specified by the instruction in step c.
- e. Press STEP once for each memory address to be displayed.



Executing a Stored Program

To execute a stored program manually:

- a. In step mode, load the first address of the program into the P register.
- b. Clear the U register.
- c. Press STEP.
- d. Press STEP again to execute the instruction and to load the next instruction into the U register.
- e. Repeat step d once for each instruction.

Repeating an Instruction

To repeat an instruction manually:

- a. In step mode, press REPEAT.
- b. Press STEP.

This procedure advances the P register each time STEP is pressed, but inhibits the loading of the U register with the next instruction in sequence.

3.4 CENTRAL PROCESSING UNIT

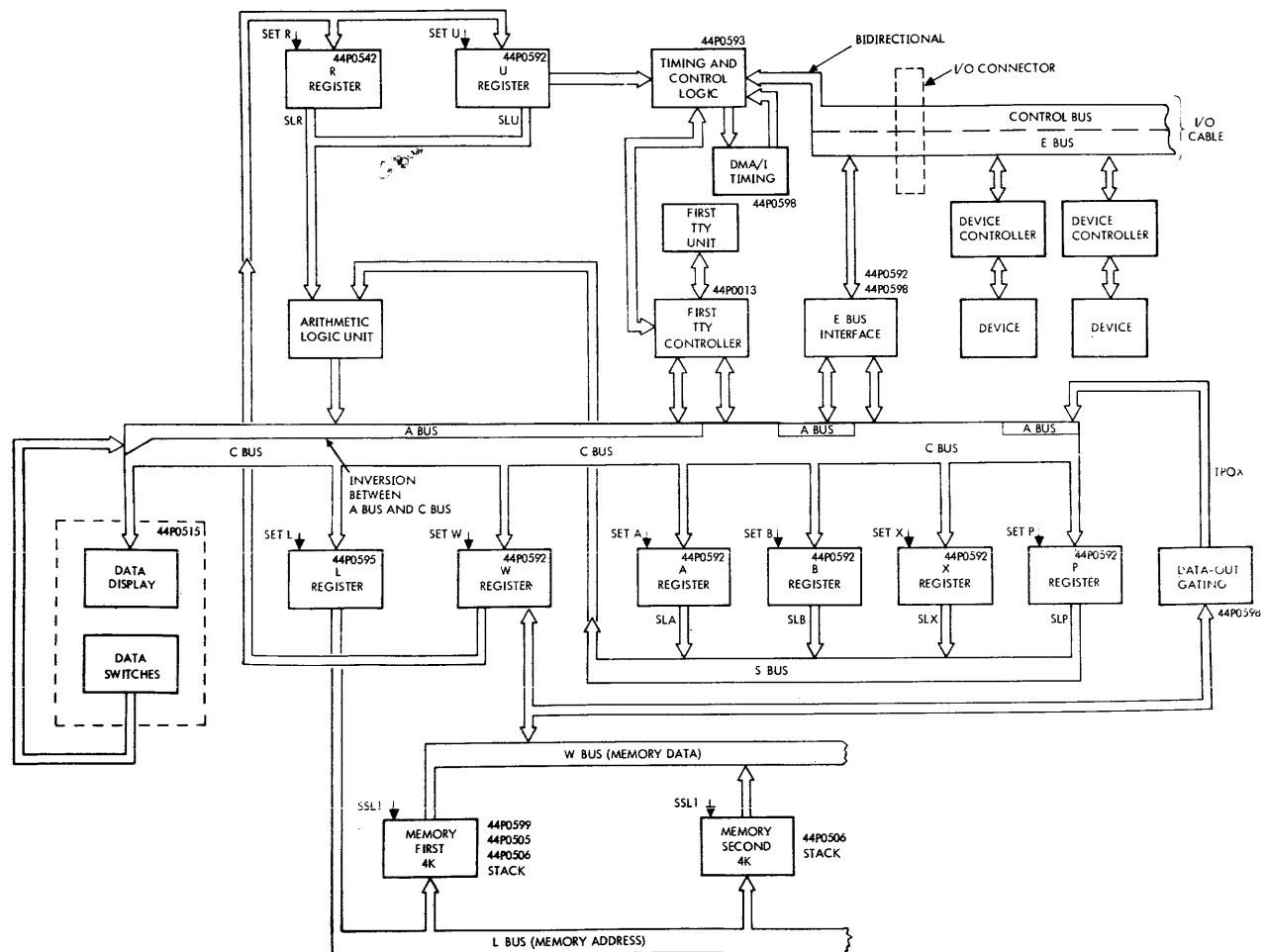
The Varian 620/L-100 computer is organized in three major functional sections:

- The Central Processing Unit (CPU)
- The Memory
- The Input/Output (I/O) System

Figure IV-15 illustrates the functional sections of the CPU and their interaction with memory and the I/O system.



IV-70



VT13-0288

Figure IV-15. Varian 620/L-100 Computer Organization



The CPU can be grouped, for descriptive purposes, into five functional sections: the control section, the arithmetic/logic section, operation registers, auxiliary register, and internal buses.

Control Section

The control section generates the timing and control signals for all computer operations. The major elements in this section are the instruction (U) register, the timing and decoding logic, and the shift control logic.

The U register receives each 16-bit instruction from memory through the W bus and holds the instruction during its execution.

The control fields of the instruction word are routed from the U register to the timing and decoding logic, where they are decoded to determine the signal levels required to perform the operations specified by the instruction.

The address field of the instruction word held in the U register is used for addressing operations. The information contained in this field is then routed to the arithmetic/logic section.

Timing logic generates the 4.211 MHz master clock from which the signals that control the sequence of computer operations are derived.

The shift control section contains the shift counter and logic to control shifting, multiplication, and division operations.

Arithmetic/Logic Section

The arithmetic/logic section comprises the operand (R) register and the arithmetic unit.

The R register receives operands from memory and holds them during instruction execution. The operand can be either data or address words. This register also permits transfers between memory and the I/O bus during the execution of the optional extended-addressing instructions.

The arithmetic unit contains gating required for arithmetic, logical, and shifting operations. Indexed- and relative-addressing modifications take place in this section without adding to the instruction execution time.



The arithmetic unit also controls the gating of words from the operation registers and the I/O bus to the C bus, where they are distributed to the operation registers or to memory buffers. This facility implements various microcoded instructions.

Operation Registers

The CPU contains four operation registers, designated A, B, X, and P.

The A, B, and X registers are directly accessible to the operator. The P register is indirectly accessible through the use of the jump instructions, which modify the program sequence.

A Register

This 16-bit register is the upper-half of the accumulator. It holds the results of arithmetic and logic operations referring to operands stored in memory. During multiplication, it holds the most significant half of the double-length product. The A and B registers can also be used for I/O transfers under program control.

B Register

This 16-bit register serves as an extension of the accumulator and as a second index register. Instructions that shift the contents of the A and B registers simultaneously are available.

X Register

This 16-bit register permits indexing of operand addresses without adding time to the execution of indexed instructions.

P Register

This 16-bit register holds the address of the current instruction. It is incremented before each new instruction is fetched. A full complement of instructions is available for conditional and unconditional modification of this register. The P register is also used in relative addressing.



Auxiliary Registers

The auxiliary registers are designated U, S, L, W, and R. None are directly accessible to the operator.

U Register

This 16-bit register holds the instruction being executed. The U register acts as a buffer between the control unit and memory to permit I/O operations on a memory-cycle-by-memory-cycle basis.

S Register

This five-bit register, in combination with the U register, works as a shift counter. The S register also acts as a buffer between memory and the control unit.

L Register

This 16-bit memory address register holds the address of the location in memory being accessed during memory cycles.

W Register

The W register is the 16-bit memory buffer register.

R Register

This 16-bit buffer holds the multiplicand and divisor in arithmetic operations. The R register acts as a buffer between the arithmetic unit and memory to permit I/O operations.

Internal Buses

The CPU contains five buses, designated C, S, W, L, and E.



C Bus

This bus provides the parallel path and selection logic for routing data between the arithmetic unit, the I/O bus, the operation registers, and the memory (W) register. The register display indicators on the computer control panel are also driven from the C bus. Collection and distribution of data simultaneously from and to operation registers is facilitated by the C bus.

S Bus

This bus provides the parallel path and selection logic for routing data from the operation registers to the arithmetic unit.

W Bus

The W register is directly connected to memory through the W bus to provide paths for data in and out of memory.

L Bus

The L register is directly connected to memory through the unidirectional L bus.

E Bus

This bus is a bidirectional input/output bus. It permits data transfers between peripheral devices and the computer. The E bus is an integral part of the I/O system.

Information Transfer

All communication between the functional section of the CPU is through the C, S, and W buses. The C and S buses are internal to the CPU. The W bus is external and bidirectional; that is, one set of lines carries information both to and from memory. The W bus provides a direct path to memory for data transfers and, in combination with the



buffer interlace controller (BIC), allows I/O operations to occur simultaneously with extended arithmetic and shifting operations.

P Register to Memory

As an instruction cycle begins, the address of the next instruction is transferred from the P to the L register. The contents of the P register are transferred through the S bus to the adder. The adder increments the address by one and transfers the incremented count to the P register. The L register then contains the address of the instruction to be fetched from memory, and the P register holds the updated address.

Memory to U Register

During the instruction cycle, the instruction address in the L register is read out on the W bus to the W register, from which it is transferred out to the U register.

U Register to Memory

For many instructions requiring an operand, the address of the operand is in the instruction word held in the U register. This operand address is transferred to the L register through gates in the arithmetic logic and the C bus. The address from the U register can be modified during the transfer to the L register as follows:

- a. **Direct Address.** No modification; bits 0 through 10 are transferred from the U register to the L register to directly address an operand in the first 2,098 memory locations.
- b. **Relative Address.** The effective operand address transferred to the L register is formed by adding bits 0 through 8 from the U register to the contents of the P register. This permits addressing a word up to 512 locations above the current program location.
- c. **Indexed Address.** The effective operand address transferred to the L register is formed by adding bits 0 through 8 from the U register to the contents of either the X register or the B register.
- d. **Indirect Address.** The word read from memory is the address of an operand rather than the operand itself.

Memory to R Register

Operands read from memory into the W register are transferred to the R register. They are stored in the R register during an arithmetic or logical operation.



For direct addressing (and for two-word addressing instructions in which the operand address is the second word), the operand address is read from memory into the W register and then transferred to the R register; it is then routed to the L register via the C bus.

Adder to Operation Register

Outputs from the adder, generated as a result of an arithmetic operation involving the R register and one of the operation registers, are transferred to an operation register via the C bus.

Operation Register to Memory

The contents of an operation register can be transferred to memory by gating those contents of the S bus and routing the word through the C bus and W register. Note that an address cycle must precede this transfer to load the storage address in the L register.

Input to Memory

Data from the E bus are routed directly to memory through the C and W buses. A data transfer is preceded by an address transfer to load the memory address into the L register. When the transfer is controlled by an instruction, the memory address is generated as a normal operand address.

Output From Memory

Data are transferred directly from memory to the I/O cable through the W and C buses. A storage address is first transferred to the L register by an instruction.

Input to Operation Registers

Data are transferred directly to the A or B register through the E and C buses. These transfers are always controlled by an instruction designating the register to receive the information.

Output From Operation Registers

Data are transferred directly from the A or B register to the I/O cable through the S, C, and E buses. These transfers are controlled by an instruction that connects the selected register to the S bus.



Register to Register

The contents of an operation register can be used to replace or modify the contents of any register. The process of incrementing and restoring the contents of the P register is described above. The contents of the A, B, and X registers can be transferred, incremented, complemented, or decremented. The overflow indicator can be set and reset. These operations are implemented by gating the register contents to the S bus, processing them in the adder, and returning the result via the C bus. Note that shifting occurs in this data path. The contents of the selected register are shifted to the left or right as they are gated from the arithmetic/logic gates to the C bus. Note that all register modification instructions use this data path.

Instruction Field Decoding

The operation code and mode (M) fields of the instruction word stored in the U register are decoded to provide static control levels used throughout the execution of the instruction.

Operation Code

The instruction's operation code has three functional categories: class, set, and group.

- a. Class designates one of three types of instructions: one-word addressing, one-word nonaddressing or two-word, and I/O.
- b. Set decodes simplify gating requirements for the execution of one-word addressing instructions. Timing specifications select the appropriate phase for executing the instruction.
- c. Group decoding is any arbitrary designation to describe the gating of computer operations according to the desired function. One of the group terms is true for all one-word addressing instructions.

M Field

The M field of an instruction word specifies the addressing mode or the instruction type, according to the instruction class defined in the operation code.



Timing

The Varian 620/L-100 operates on a basic 950-nanosecond machine cycle. That is, a full memory cycle (read/restore or clear/write) occurs in each 950-nanosecond interval. All computer operations take place within some multiple of this basic timing period.

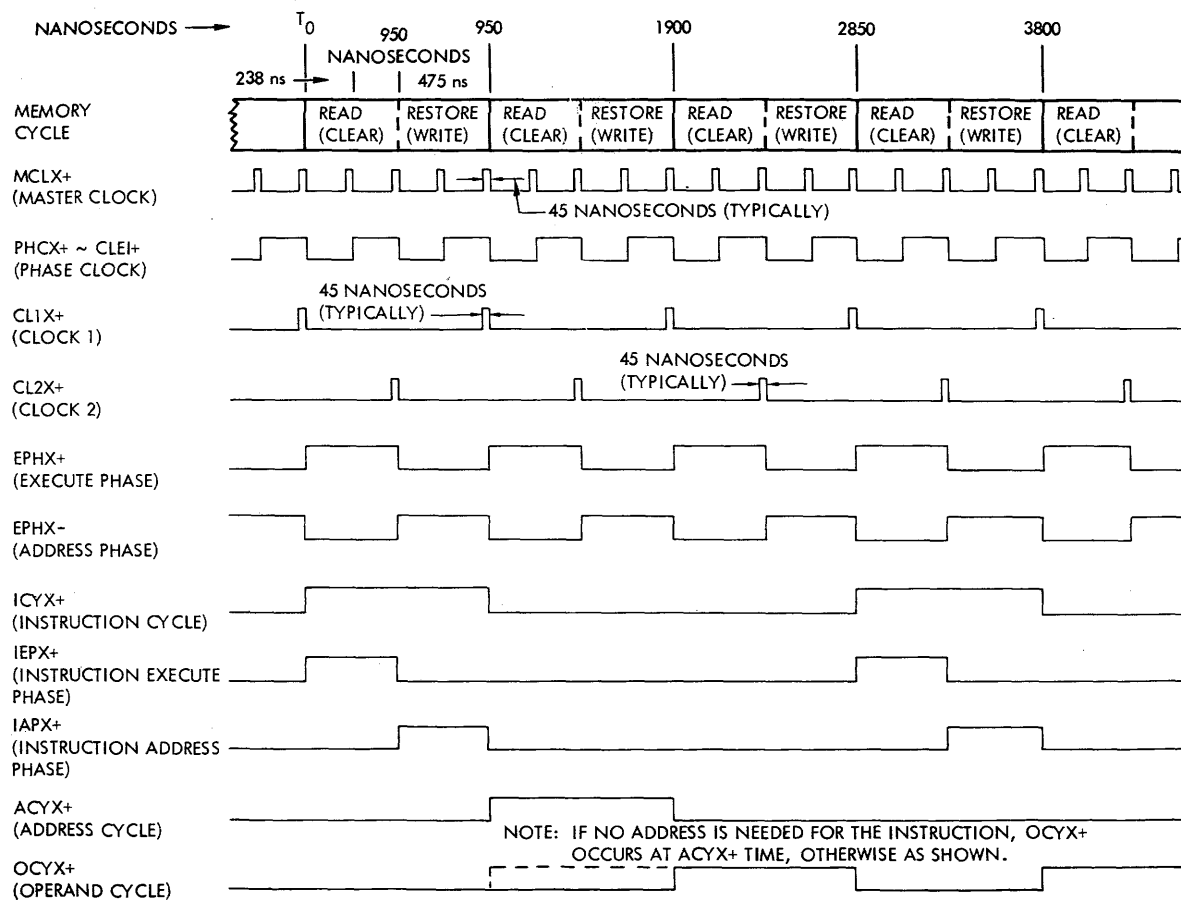
During a full-cycle memory operation, suboperation timing is controlled by an internal 4.211 MHz master clock. The pulsewidth of this master clock is 237 nanoseconds, or one-fourth of the basic 950-nanosecond machine cycle; this permits the execution of various suboperations during the memory cycle. Note that the first half-cycle (475 nanoseconds) of the period is used to access a word (read) or to load zeros into an address in memory (clear). The second half-cycle is used to reload a word (restore) or to write a new word (write) into the address.

System Clocks

The clock signals that control the timing of computer operations are listed in table IV-13 and their waveforms are illustrated in figure IV-16.

Table IV-13. Varian 620/L-100 System Clocks

Signal	Mnemonic	Description
Master Clock	MCLX +	Crystal-controlled 4.211 MHz timing signal for the entire system
Phase Clock	PHCX +	The 2.105 MHz timing signal (counted down and synchronized with MCLX +) used to time the basic address and execution phases of computer operations
Address Phase	EPHX-	Basic timing phase that corresponds to the memory restore or write half-cycle; instruction and operand addresses are transferred to memory during this period
Execution Phase	EPHX +	Basic timing phase that corresponds to the memory read or clear half-cycle; data are transferred to and from memory and instructions are executed during this period
Clock 1	CL1X +	Signal that initiates a memory cycle and operations that are synchronized with the start of the memory cycle
Clock 2	CL2X +	Signal that initiates operations that are synchronized with the start of a memory write or restore half-cycle



VTI2-0380

Figure IV-16. Basic Clock Waveforms





A memory cycle in the Varian 620/L-100 comprises two phases:

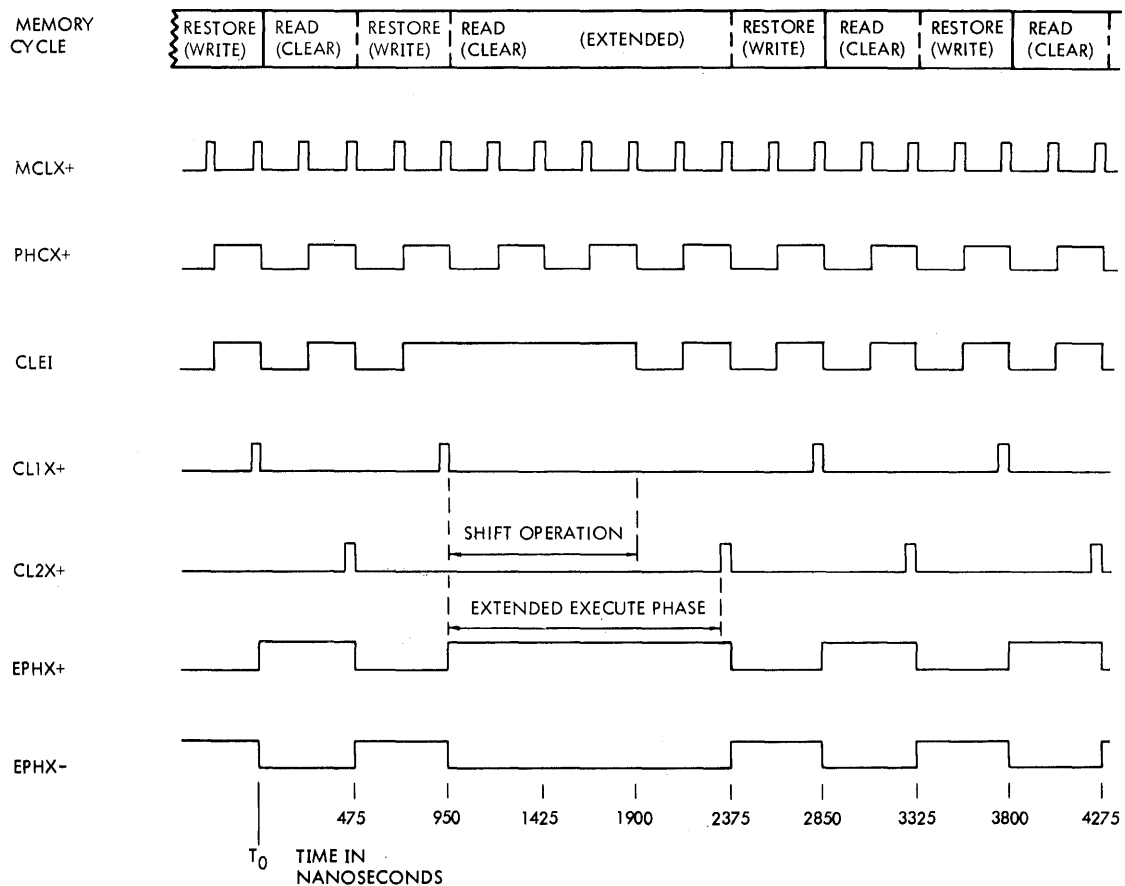
- a. Fetching an instruction from memory
- b. Executing the fetched instruction

Clock Modifiers

The memory-cycle phases are modified by certain instructions or by signals received from devices external to the computer. The conditions under which the clock periods are modified are:

Shift	During the shifting of words contained in the A and B registers, the execution phase is extended by the number of master clock periods (237.5 nanoseconds) equal to the number of shifts specified.
Interrupt	When an external interrupt is requested, the address phase is extended 475 nanoseconds to accommodate delays in receiving the interrupt address from the external device.
Trap	When a BIC requests a transfer to or from memory, the address phase is extended 1.66 microseconds to permit the execution of the trapping sequence (i.e., the routing of the address and data from the external device).
Halt	On a halt instruction, clock CL1X+ and CL2X+ prevent any further operations until the STEP or RUN switches are pressed to resume program execution.

Modification of the execution phase of an instruction is illustrated in figure IV-17. The illustration is typical of a shift instruction. At time 0, the instruction has been fetched from memory. Starting at time 475, the instruction is executed. However, the normal 237.5-nanosecond execution phase is extended 237.5 nanoseconds for each shift (six are illustrated). Note that CL1X+ and CL2X+ are inhibited during the extended execution phase. In a similar manner, the address phase is extended when required by the conditions defined above.



VTI2-0384

Figure VI-17. Example of a Modified Clock Sequence

Operation Sequences

The basic clock signals generated from the 4.211 MHz master clock time three operation sequences: instruction cycle (ICYX +), operand cycle (OCYX +), and address cycle (ACYX +). All computer operations are timed by one or more of these signals.

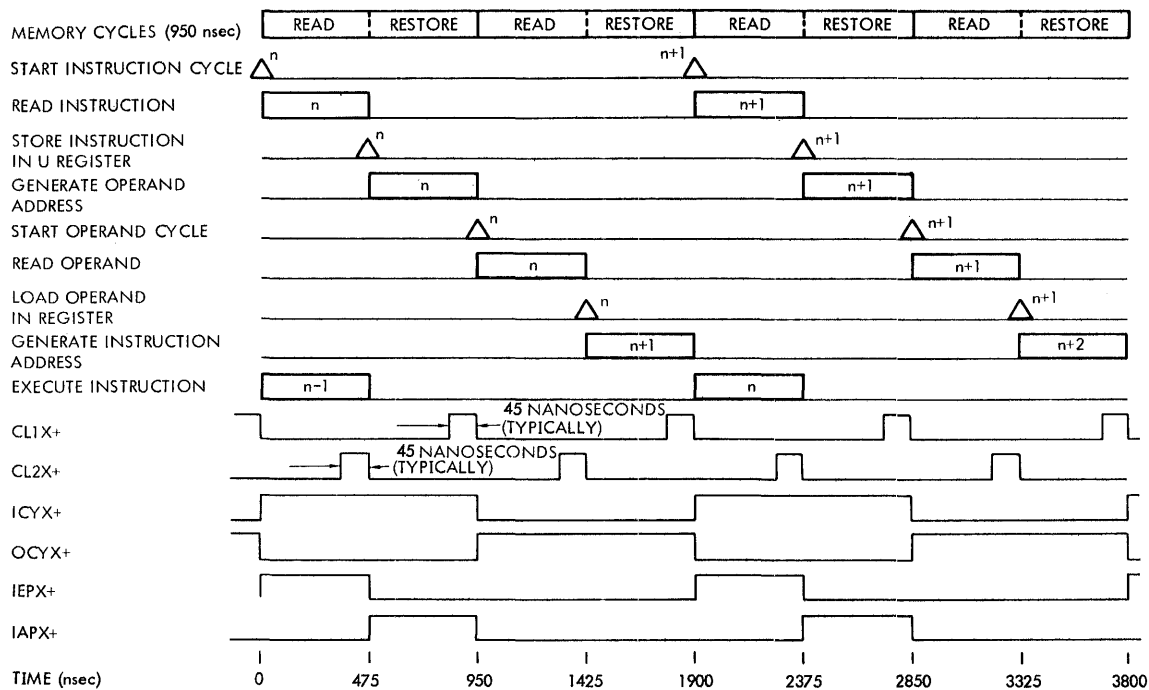
The following paragraphs describe typical operation sequences. Variations of these sequences depend on the instruction being executed. However, a study of these fundamental operations will aid the user in understanding the timing of a specific instruction sequence.



Accessing an Operand in Memory

The simplest and most basic operation sequence is one in which a one-word, directly addressed operand is read from memory. This is typical of the load/store, arithmetic (excluding multiplication and division), and logic instructions. The timing of the suboperations of this sequence is illustrated in figure IV-18. At time 0, the instruction cycle (ICYX+) for the n th (current) instruction is initiated. Note that instruction $n - 1$ is being executed (IEPX+) while the n th instruction is being read from memory. At time 475, the instruction is transferred to the U register. During the instruction address phase (IAPX+), when the instruction just read is being restored to memory, the operand address is generated.

Since the operand is not indirectly addressed in the illustrated case, the operand cycle (OCYX+) is initiated at time 950-nanoseconds. After the operand has been read from memory and stored in the R register, the address of the next instruction ($n + 1$) is



VT12-0381A

Figure IV-18. Accessing an Operand in Memory

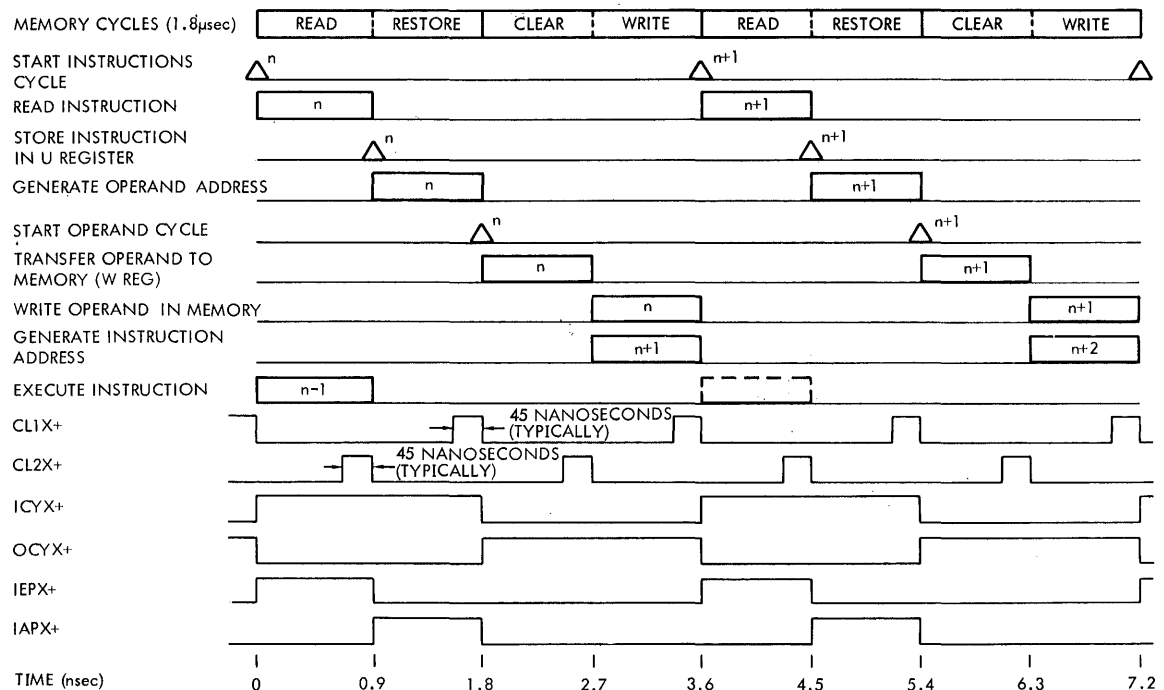


generated (normally by incrementing the P register) and transferred to the L register. This suboperation is executed while the operand is being restored to memory. The instruction cycle (ICYX +) for $n + 1$ is then initiated at time 1900.

Note that the operation to be performed on the operand contained in the R register is executed during the IEPX + phase of the instruction cycle for $n + 1$. This operation could be, for example, adding the operand value to the contents of the A register and storing the result in that register (ADD instruction), or simply transferring the operand to one of the operation registers (LDA, LDB, and LDX instructions).

Storing an Operand in Memory

The sequence for storing an operand in memory (STA, STB, and STX instructions), is essentially identical to that for accessing an operand, except that the specified memory address is cleared and the operand written into it. The sequence of suboperations is shown in figure IV-19.



VT12-0141A

Figure IV-19. Storing an Operand in Memory



The n th instruction is accessed and the operand address generated during the instruction cycle as described above; execution of the $n - 1$ instruction occurs during IEPX- of the n th cycle as indicated. However, during OCYX+, the operand is transferred to memory while the referenced address is being cleared.

During the last half of the cycle, the operand is stored in the address just cleared. During this time, the address for the next instruction is generated. Note that there is no execution, as such, for this type of instruction (indicated by dashed lines in the illustration) because the execution has already been accomplished, in effect, by the transfer and storage of the operand in memory.

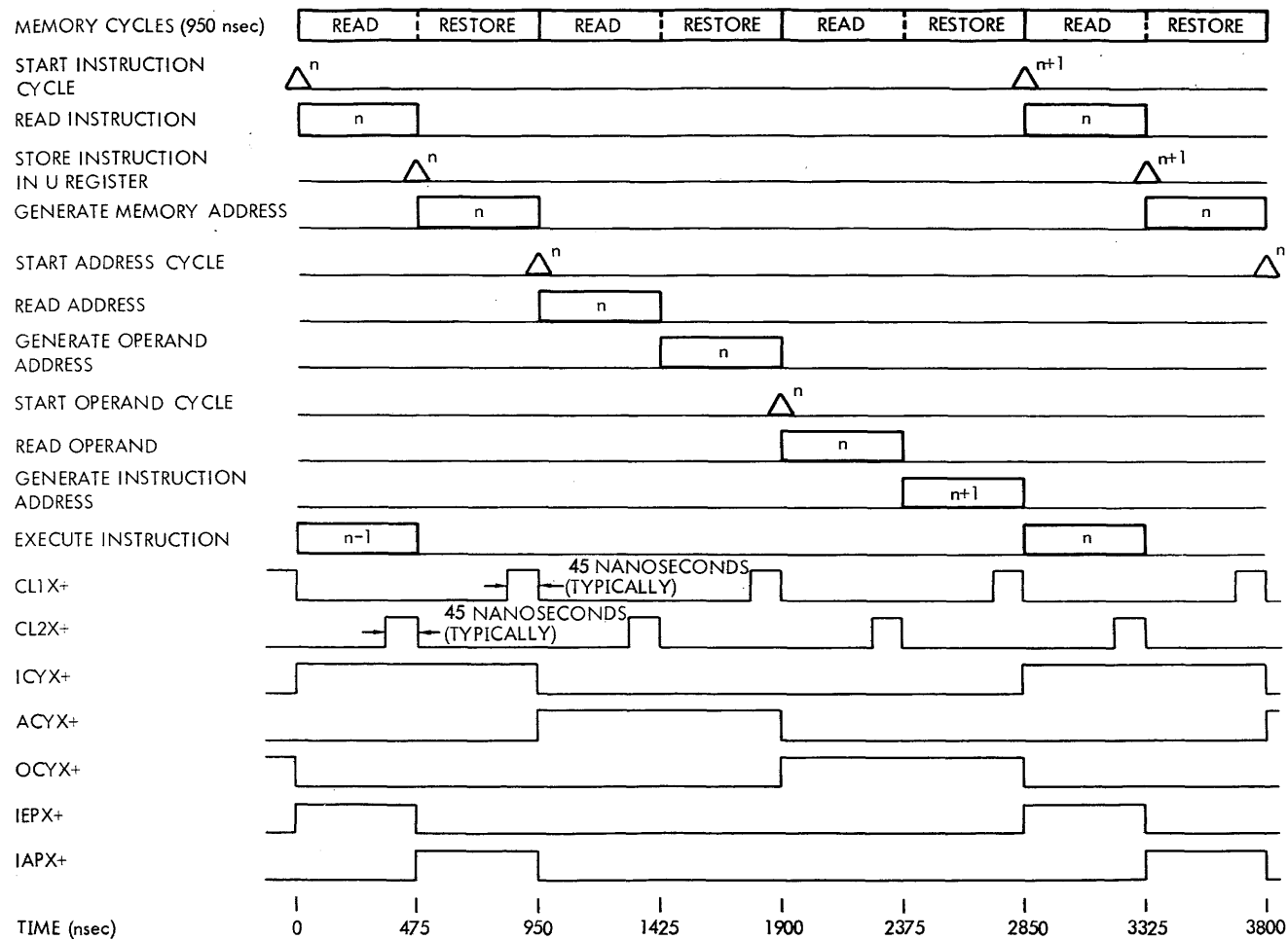
Accessing an Operand Indirectly

In this case, an address cycle (ACYX +) is required to read the indirect address word from memory before performing the operand cycle (OCYX-).

The sequence of suboperations for accessing an operand indirectly is illustrated in figure IV-20.

During the instruction cycle, the n th instruction is read from memory and stored in the U register. The previous instruction, $n - 1$, is executed during IEPX+. During the instruction address phase (IAPX+), the location of the (indirect) address word is generated. This address word is read from memory and stored in the R register.

For the case illustrated in figure IV-20, the address word contains the address of the operand. If this were not the case, another address cycle would be initiated to access a second address word, etc. The operand address is transferred to the L register during the last half of ACYX+ to locate the operand read out during the succeeding OCYX+. The address for instruction $n + 1$ is generated and instruction n is executed, completing the sequence.



VT12-0383A

Figure IV-20. Accessing an Operand Indirectly



CHAPTER V

LOGIC DESCRIPTIONS



varian data machines



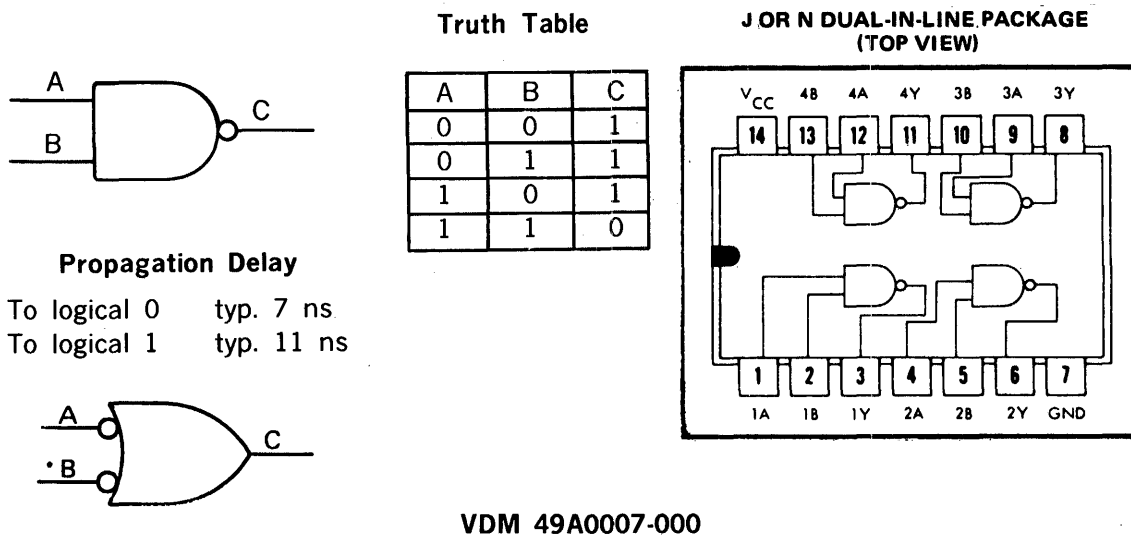
CHAPTER V LOGIC DESCRIPTIONS

LOGIC DESCRIPTIONS

DTL and TTL integrated circuits (ICs) are used throughout the 620 series computer systems. These circuits are general-purpose digital logic units packaged to simplify maintenance. The IC board layout uses a "bit-slice" technique in which all register and gating circuits associated with six bits are packaged on one board. figure V-1 through V-49 describe the basic logic packages used in the 620 series computers.

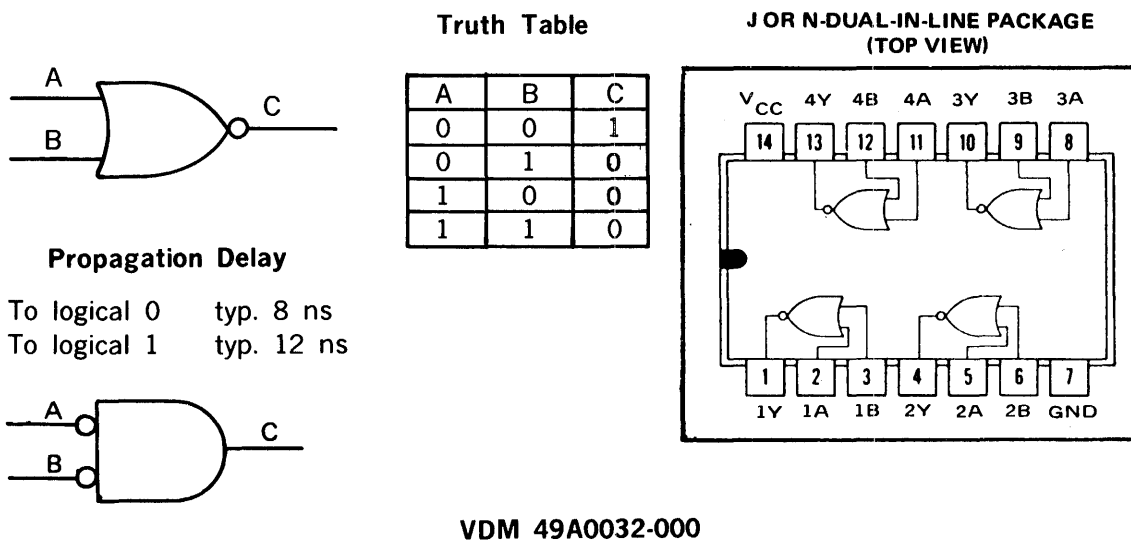
The following abbreviations are used in the following figures:

AY	General Instrument
HD	Harris Semiconductor
TI	Texas Instruments part
TR	Western Digital
SN74	In TI part number, indicates TTL logic (same number used by National Semiconductor)
SN15	In TI part number, indicates DTL logic
MC	Motorola part
N	Signetics
U	Fairchild part one-shot Number followed by PC or DC Fairchild
Logical 0 (normal)	Ground
Logical 1 (normal)	+ 5V
Logical 0 (I/O bus)	+ 3V
Logical 1 (I/O bus)	Ground

CHAPTER V
LOGIC DESCRIPTIONS

VTII-1723

Figure V-1. Quadruple 2-Input NAND Gate (SN7400N, 7400PC, N7400A)

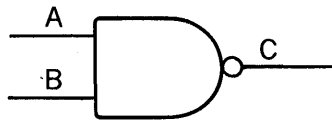


VTII-1724

Figure V-2. Quadruple 2-Input Positive NOR Gate (SN7402N, MC7402P, 7402PC)

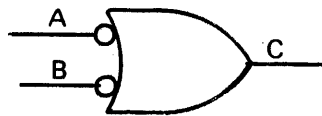


CHAPTER V LOGIC DESCRIPTIONS



Propagation Delay

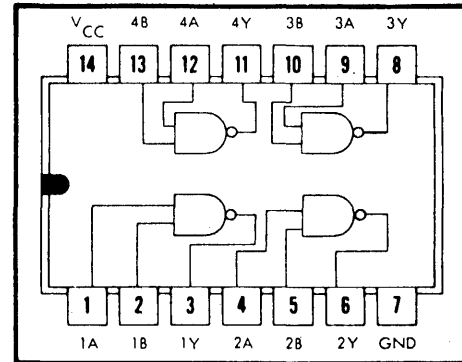
To logical 0 typ. 8 ns
To logical 1 typ. 35 ns



Truth Table

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

J OR N DUAL-IN-LINE PACKAGE (TOP VIEW)

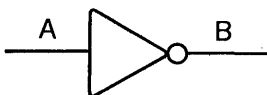


positive logic: $Y = \overline{AB}$

VDM 49A0081-001

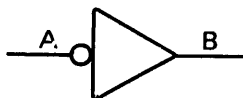
VTII-1725

Figure V-3. Quadruple 2-Input Positive NAND Gate (Open Collector) (SN7403N)



Propagation Delay

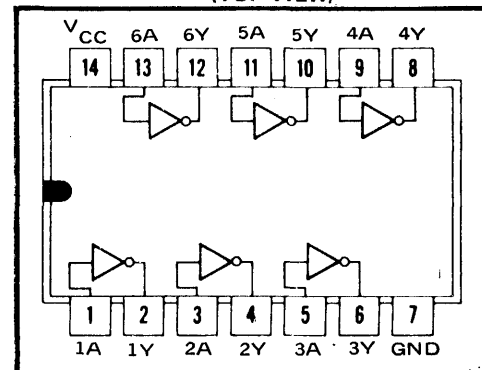
To logical 0 typ. 8 ns
To logical 1 typ. 12 ns



Truth Table

A	B
0	1
1	0

J OR N DUAL-IN-LINE PACKAGE (TOP VIEW)



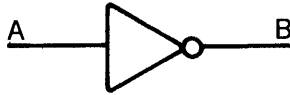
VDM 49A0040-000

VTII-1726

Figure V-4. Hex Inverters (SN7404N, MC7404P, N7404A, 7404PC)



CHAPTER V LOGIC DESCRIPTIONS



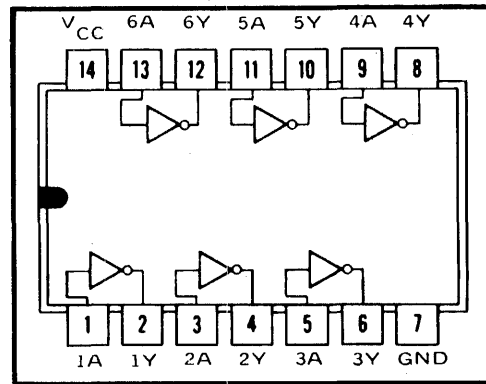
Truth Table

A	B
0	1
1	0

Propagation Delay

To logical 0 typ. 8 ns
To logical 1 typ. 40 ns

JORN DUAL-IN-LINE PACKAGE
(TOP VIEW)

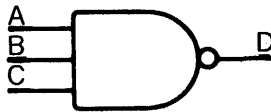


positive logic: $Y = \bar{A}$

VTII-1727

VDM 49A0575-000

Figure V-5. Hex Inverter with Open-Collector Circuit (SN7405J, MC7405L)



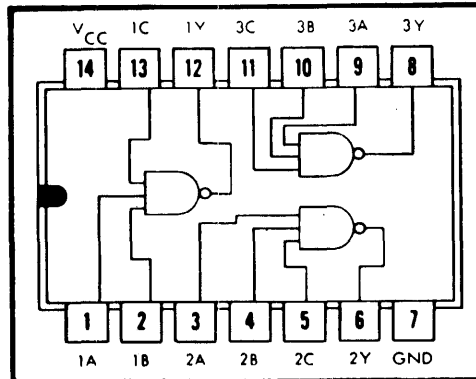
Truth Table

A	B	C	D
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Propagation Delay

To logical 0 typ. 7 ns
To logical 1 typ. 11 ns

JORN DUAL-IN-LINE PACKAGE
(TOP VIEW)



positive logic: $Y = \overline{ABC}$

VTII-1728

VDM 49A0005-000

Figure V-6. Triple 3-Input Positive NAND Gate (SN7410N, 7410PC, N7410A)



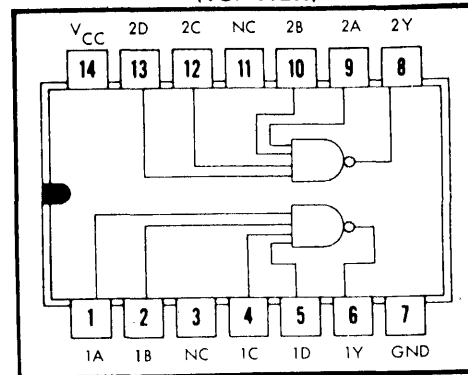
CHAPTER V LOGIC DESCRIPTIONS

Truth Table

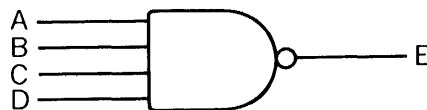
A	B	C	D	E
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
etc.	etc.	etc.	etc.	etc.
1	1	1	1	0

J OR N DUAL-IN-LINE PACKAGE

(TOP VIEW)

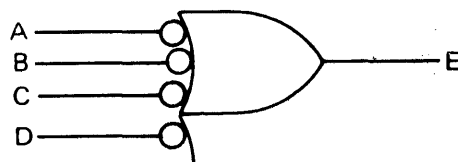


positive logic: $Y = \overline{ABCD}$



Propagation Delay

To logical 0 typ. 8 ns
To logical 1 typ. 12 ns



VDM 49A0006-000

VTII-1696

Figure V-7. Dual 4-Input Positive NAND Gate (SN7420N, MC7420, 7420PC)

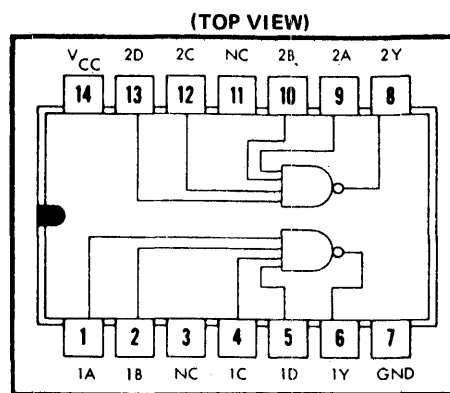


CHAPTER V LOGIC DESCRIPTIONS

Truth Table

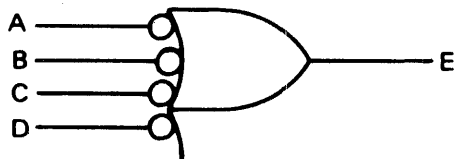
A	B	C	D	E
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
etc.	etc.	etc.	etc.	etc.
1	1	1	1	0

J OR N DUAL-IN-LINE PACKAGE



Propagation Delay

To logical 0 typ. 8 ns
To logical 1 typ. 13 ns



VDM 49A0504-000

VT11-1697

Figure V-8. Dual 4-Input Positive NAND Buffer (SN7440N, MC7440L, 7440DC)



CHAPTER V LOGIC DESCRIPTIONS



Propagation Delay

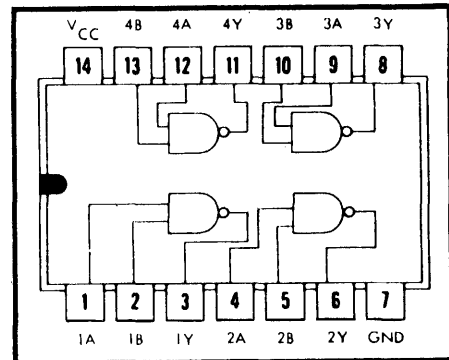
To logical 0 typ. 6.2 ns
To logical 1 typ. 5.9 ns



Truth Table

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

J OR N DUAL-IN-LINE PACKAGE (TOP VIEW)



VDM 49A0039-000

VTII-1715

Figure V-9. Quadruple 2-Input Positive NAND Gate (SN74H00N, MC3000P)



Propagation Delay

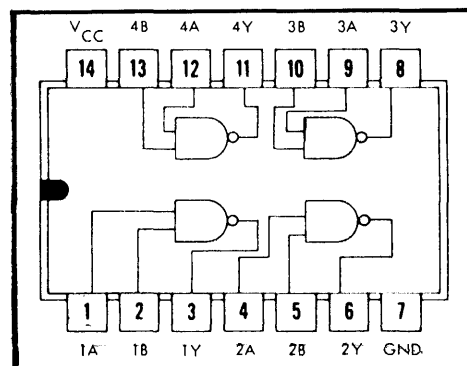
To logical 0 typ. 7.5 ns
To logical 1 typ. 10 ns



Truth Table

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

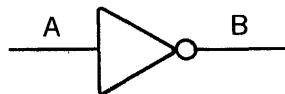
J OR N DUAL-IN-LINE PACKAGE (TOP VIEW)



VDM 49A0042-000

VTII-1698

Figure V-10. Quadruple 2-Input Positive NAND Gate with Open Collector (SN74H01N, MC3004P)

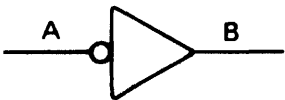
CHAPTER V
LOGIC DESCRIPTIONS

Truth Table

A	B
0	1
1	0

Propagation Delay

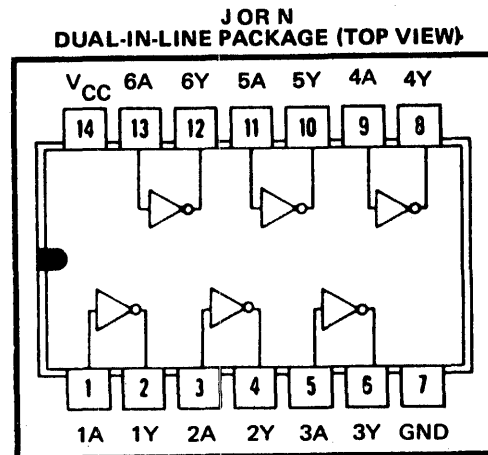
To logical 0 typ. 6.5 ns
To logical 1 typ. 9 ns



VDM 49A0023-000

VTII-1716

Figure V-11. Hex Inverter (SN74H04N, MC30018)

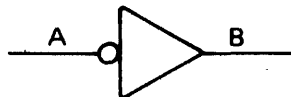


Truth Table

A	B
0	1
1	0

Propagation Delay

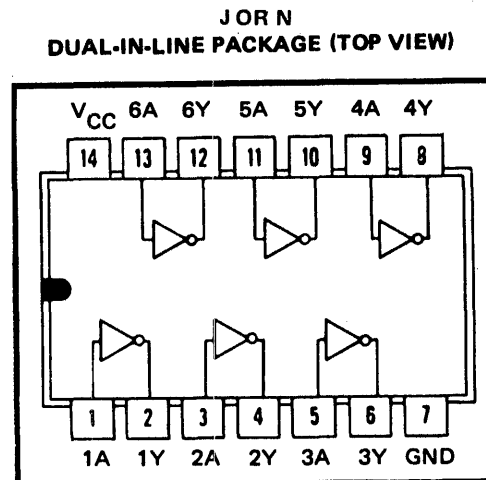
To logical 0 typ. 10 ns
To logical 1 typ. 13 ns



VDM 49A0061-000

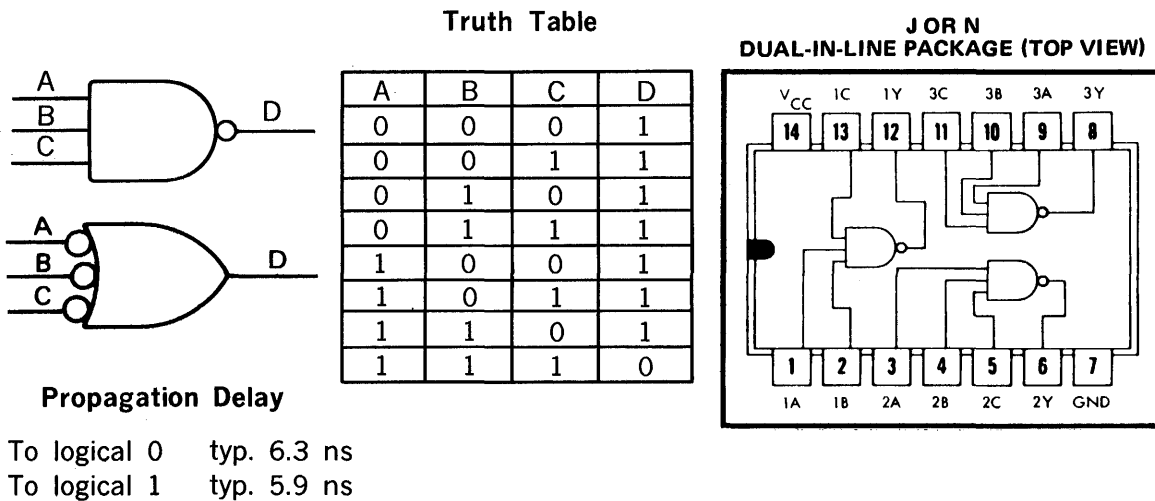
VTII-1699

Figure V-12. Hex Inverter with Open-Collector Output (SN74H05N)





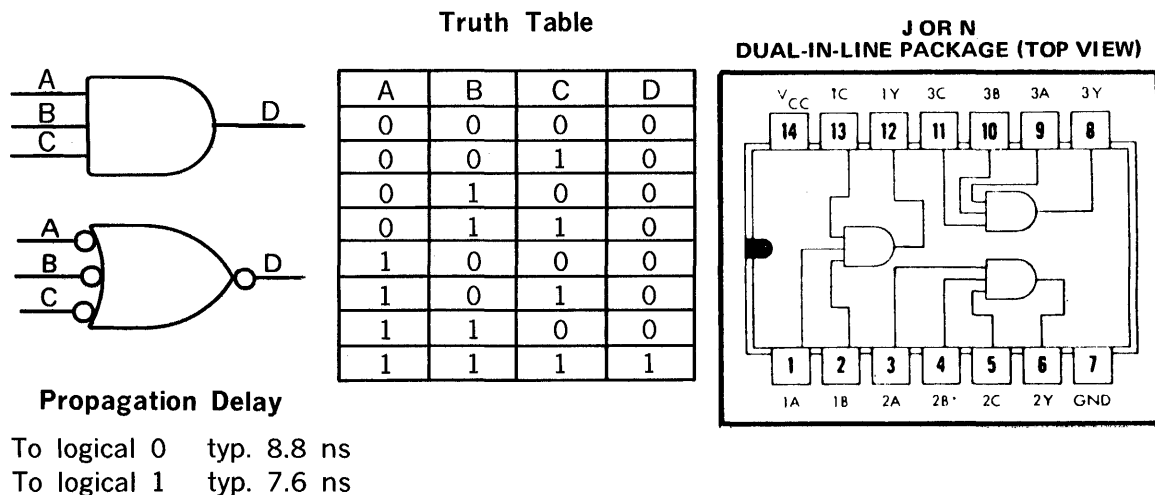
CHAPTER V
LOGIC DESCRIPTIONS



VDM 49A0054-000

VTII-1717

Figure V-13. Triple 3-Input Positive NAND Gate (SN74H10N)



VDM 49A0022-000

VTII-1700

Figure V-14. Triple 3-Input Positive AND Gate (SN74H11N)

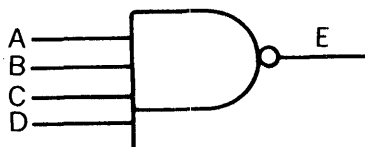
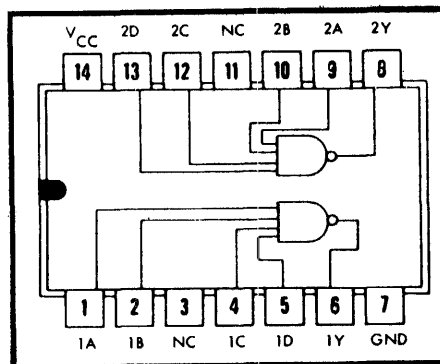


CHAPTER V LOGIC DESCRIPTIONS

Truth Table

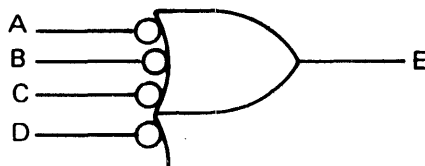
A	B	C	D	E
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
etc.	etc.	etc.	etc.	etc.
1	1	1	1	0

J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)



Propagation Delay

To logical 0 typ. 7 ns
To logical 1 typ. 6 ns



VDM 49A0056-000

VTII-1701

Figure V-15. Dual 4-Input Positive NAND Gate (SN74H20N, MC3010)

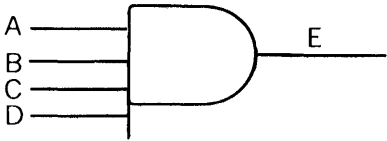
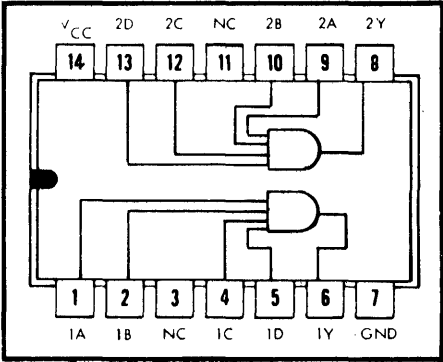


CHAPTER V
LOGIC DESCRIPTIONS

Truth Table

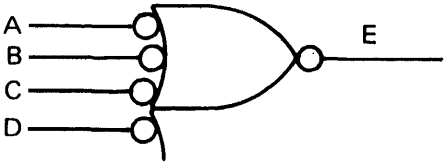
A	B	C	D	E
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
etc.	etc.	etc.	etc.	etc.
1	1	1	1	1

J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)



Propagation Delay

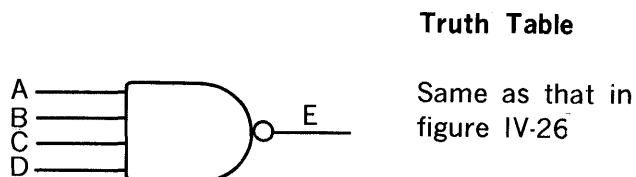
To logical 0 typ. 8.8 ns
To logical 1 typ. 7.6 ns



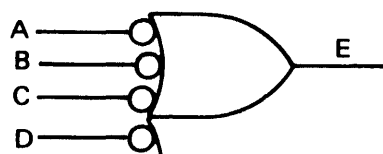
VDM 49A0094-001

VT11-1702

Figure V-16. Dual 4-Input Positive AND Gate (SN74H21N, MC3011)

CHAPTER V
LOGIC DESCRIPTIONS**Propagation Delay**

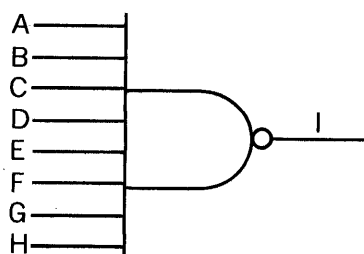
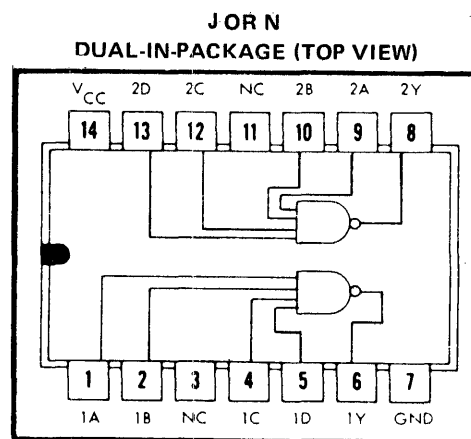
To logical 0 typ. 7.5 ns
To logical 1 typ. 10 ns



VDM 49A0038-000

VTII-1718

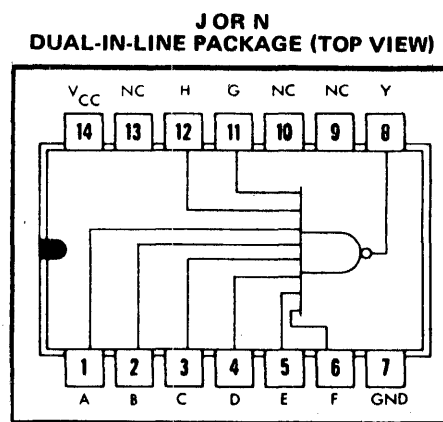
Figure V-17. Dual 4-Input Positive NAND Gate (SN74H22N)

**Propagation Delay**

To logical 0 typ. 8.9 ns
To logical 1 typ. 6.8 ns

Truth Table

Same as that in figure IV-26 except for extra inputs



positive logic: $Y = \overline{ABCDEFGH}$

VDM 49A0060-000

VTII-1703

Figure V-18. 8-Input Positive NAND Gate (SN74H30N)

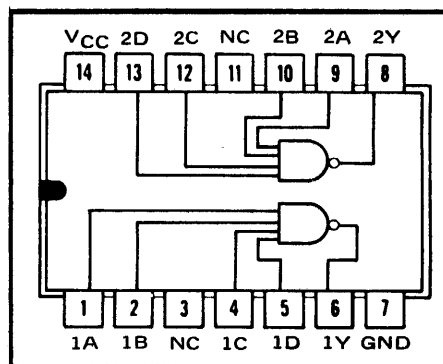


CHAPTER V LOGIC DESCRIPTIONS

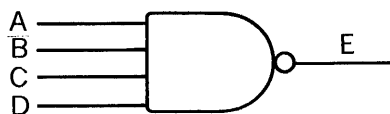
Truth Table

A	B	C	D	E
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
etc.	etc.	etc.	etc.	etc.
1	1	1	1	0

JOR N
DUAL-IN-LINE PACKAGE (TOP VIEW)

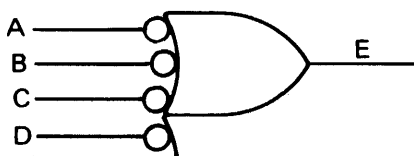


positive logic: $Y = \overline{ABCD}$



Propagation Delay

To logical 0 typ. 6.5 ns
To logical 1 typ. 8.5 ns



VDM 49A0019-000

VTII-1704

Figure V-19. Dual 4-Input Positive NAND Buffer (SN74H40N, MC3024P)



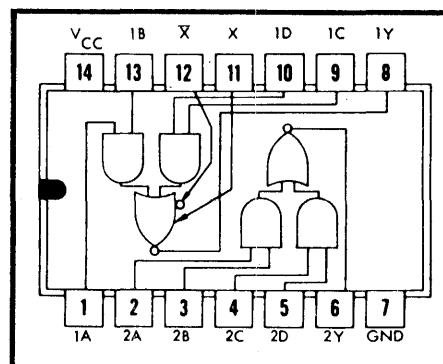
CHAPTER V LOGIC DESCRIPTIONS

Truth Table

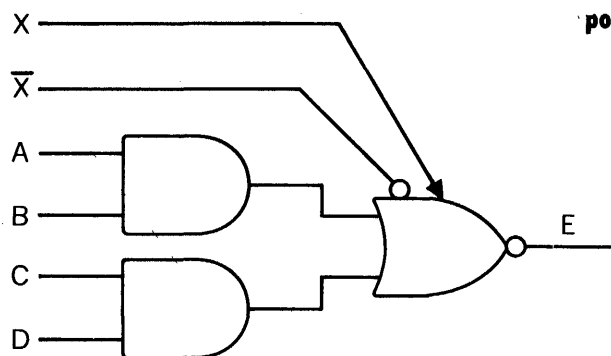
A	B	C	D	E
0	0	1	1	0
1	1	0	0	0
1	1	1	1	0

All other cases E is equal to a logical 1

J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic: $Y = \overline{(AB)} + (CD) + (X)$
(X = Output of SN74H60 or SN74H62)



$X = 1$ and $\overline{X} = 0$
causes $E = 0$

Propagation Delay

To logical 0	typ. 6.2 ns
To logical 1	typ. 6.8 ns

Note

- Both expander inputs are used simultaneously for expanding.
- If expander is not used, leave X and X pins open.
- Expander inputs X and X are functional on SN75H50N circuits only. Make no external connections to X and X pins of SN74H51.
- A total of four SN74H60 expander gates or one SN74H62 expander gate can be connected to the expander inputs.

VDM 49A0093-001 and 49A0041-000

VTII-1705

Figure V-20. Dual 2-Wide 2-Input AND-OR-Invert Gates (SN74H50 and 51N MC3020 and 3023)



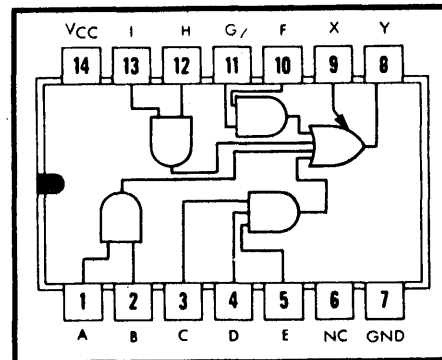
CHAPTER V LOGIC DESCRIPTIONS

Truth Table

Any AND gate with all logical 1 inputs causes K to be logical 1

Logical 1 on J input causes K to be logical 1

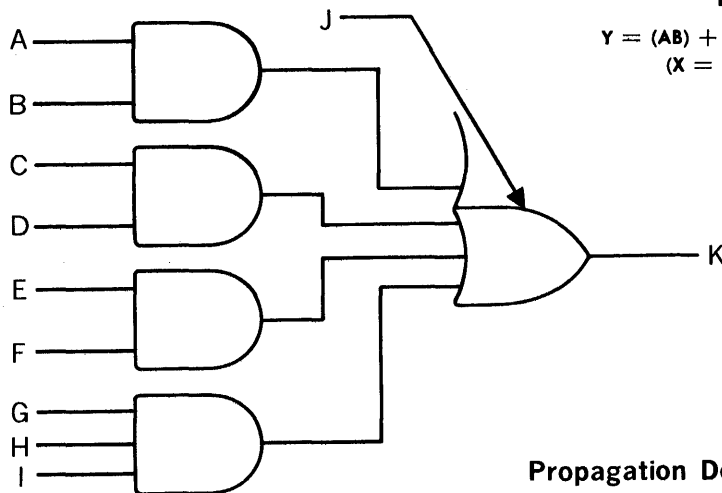
J OR N DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic:

$$Y = (AB) + (CDE) + (FG) + (HI) + (X)$$

(X = Output of SN74H61)



Propagation Delay Using Expander Pin

To logical 0	typ. 9.8 ns
To logical 1	typ. 14.8 ns

Note

1. A total of six expander gates can be connected to input J.
2. No internal connection.

VDM 49A0095-000

VT11-1729

Figure V-21. Expandable 2-2-2-3-Input AND-OR Gate (SN74H52N, MC3031P)

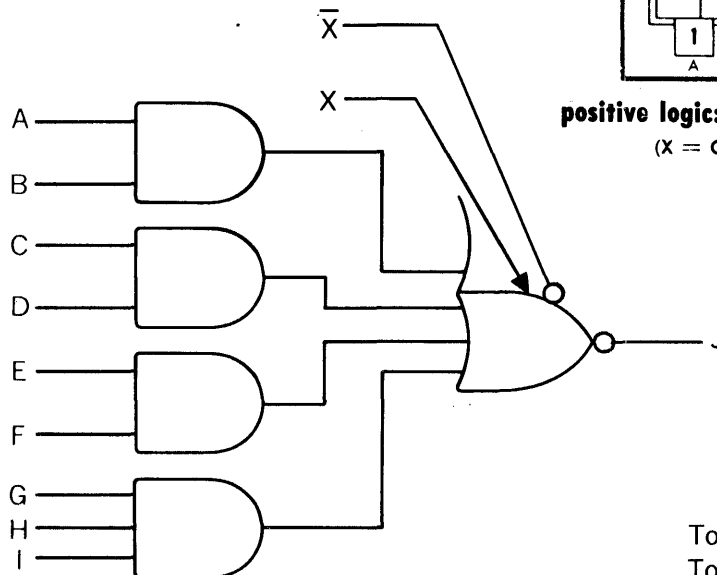


CHAPTER V LOGIC DESCRIPTIONS

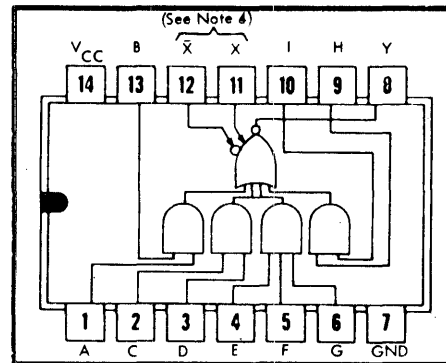
Truth Table

Any AND gate with all logical 1 inputs causes J to be logical 0

Logical 1 on X and logical 0 on \bar{X} causes J to be logical 0



J OR'N DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic: $Y = (\overline{AB}) + (CD) + (EFG) + (HI) + (X)$
(X = Output of SN74H60 or SN74H62)

Propagation Delay

To logical 0	typ. 7.4 ns
To logical 1	typ. 11.4 ns

Note

- Both expander inputs are used simultaneously for expanding.
- If expander is not used, leave X and \bar{X} pins open.
- Expander inputs X and \bar{X} are functional on SN74H53 circuits only. Make no external connection to X and \bar{X} pins of SN74H54.
- A total of four SN74H60 expander gates or one SN74H62 expander gate can be connected to the expander inputs.

VDM 49A0106-000

VTII-1730

Figure V-22. Expandable 2-2-2-3-Input AND-OR Invert Gate (SN74H53N, MC3032)

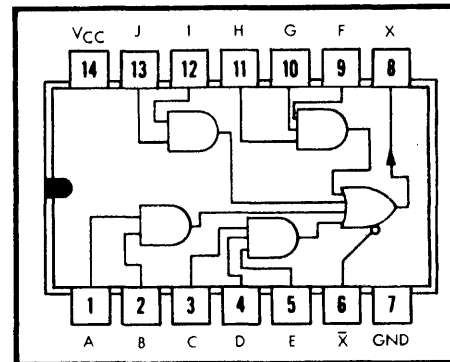


CHAPTER V LOGIC DESCRIPTIONS

Truth Table

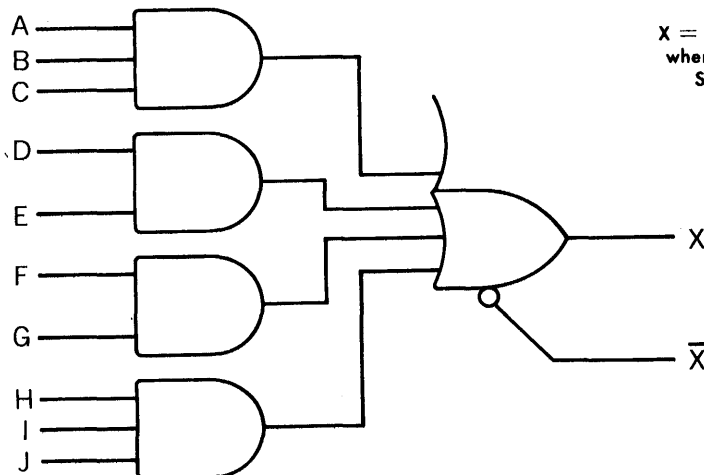
Any AND gate with all logical 1 inputs causes X to be logical 1 and \bar{X} to be logical 0 when X and \bar{X} are connected to SN74H50 or SN74H53

JOR N DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic:

$X = (AB) + (CDE) + (FGH) + (IJ)$
when connected to X and \bar{X} pins of
SN74H50 or SN74H53 circuit.



Propagation Delay

As listed in figures
IV-31 or IV-33

Note

1. Connect to X input of SN74H50 or SN74H53.
2. Connect to \bar{X} input of SN74H50 or SN74H53.

VDM 49A0098-000

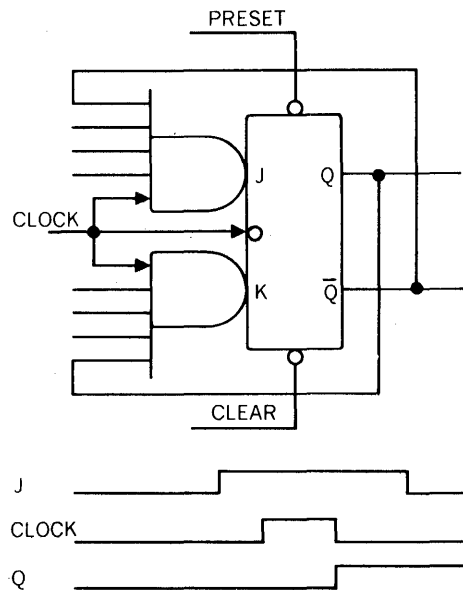
For Use With SN74H50 and SN74H53 Circuits

VTII-1731

Figure V-23. 3-2-2-3-Input AND-OR Expander (SN74H62N, MC3018P)



CHAPTER V LOGIC DESCRIPTIONS



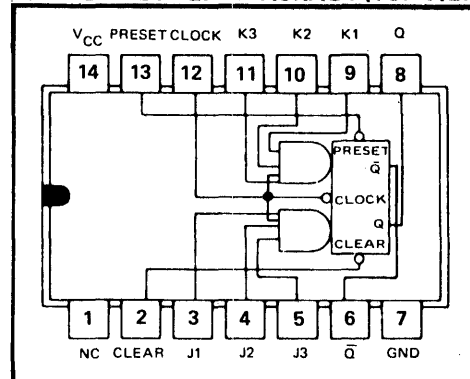
TRUTH TABLE		
t_n		t_{n+1}
J	K	Q
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

NOTES:

1. $J = J_1 \cdot J_2 \cdot J_3$
2. $K = K_1 \cdot K_2 \cdot K_3$
3. t_n = Bit time before clock pulse.
4. t_{n+1} = Bit time after clock pulse.
5. NC = No Internal Connection.

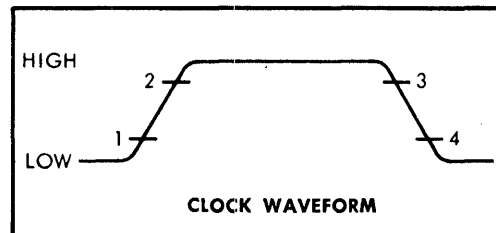
J can be removed after the leading edge of clock

J OR N DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic:

- Low input to preset sets Q to logical 1
- Low input to clear sets Q to logical 0
- Preset and clear are independent of clock



These J-K flip-flops are based on the master-slave principle. The AND gate inputs for entry the master section are controlled by the clock pulse. The clock pulse also regulates the circuitry which connects the master and slave sections. The sequence of operation is as follows:

1. Isolate slave from master
2. Enter information from AND gate inputs to master
3. Disable AND gate inputs
4. Transfer information from master to slave

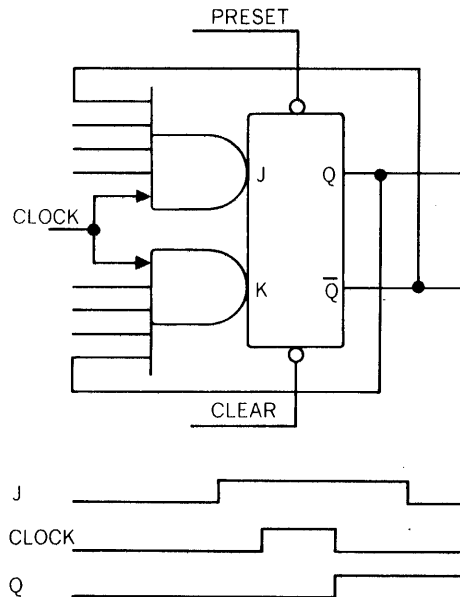
VDM 49A0003-000

VTII-1732

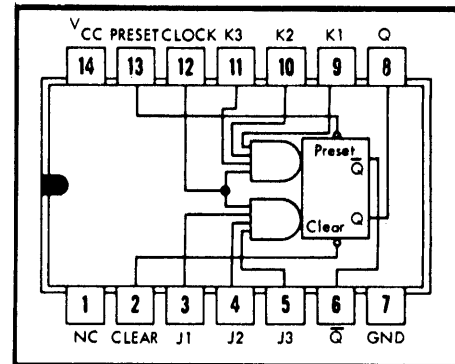
Figure V-24. J-K Master-Slave Flip-Flop (SN7472N)



CHAPTER V LOGIC DESCRIPTIONS



**J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)**



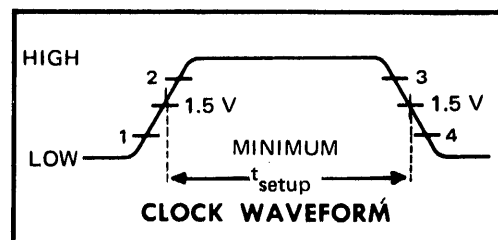
positive logic:

Low input to preset sets Q to logical 1
Low input to clear sets Q to logical 0
Preset and clear are independent of clock

TRUTH TABLE		
t_n		t_{n+1}
J	K	Q
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

Minimum clock = 12 ns

NOTES: 1. $J = J1 \cdot J2 \cdot J3$
2. $K = K1 \cdot K2 \cdot K3$
3. t_n = Bit time before clock pulse.
4. t_{n+1} = Bit time after clock pulse.



These J-K flip-flops are based on the master-slave principle. The AND gate inputs for entry into the master section are controlled by the clock pulse. The clock pulse also regulates the circuitry which connects the master and slave sections. The sequence of operation is as follows:

1. Isolate slave from master
2. Enter information from AND gate inputs to master
3. Disable AND gate inputs
4. Transfer information from master to slave

Logical state of J and K inputs must not be allowed to change when the clock pulse is in a high state.

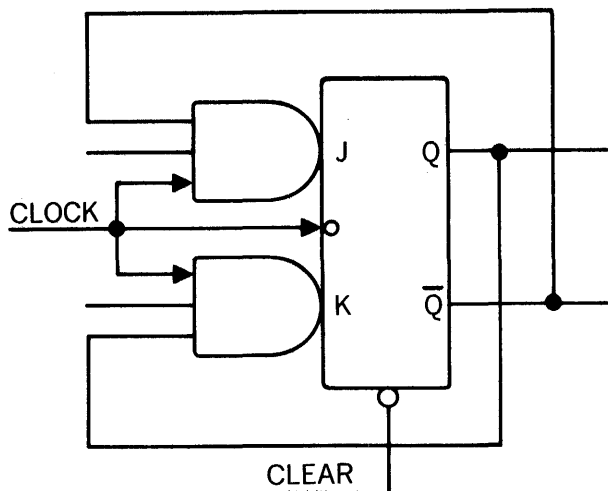
VDM 49A0520-000

VTII-1733

Figure V-25. J-K Master-Slave Flip-Flop (SN74H72N)



CHAPTER V LOGIC DESCRIPTIONS



TRUTH TABLE (Each Flip-Flop)		
t_n		t_{n+1}
J	K	Q
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

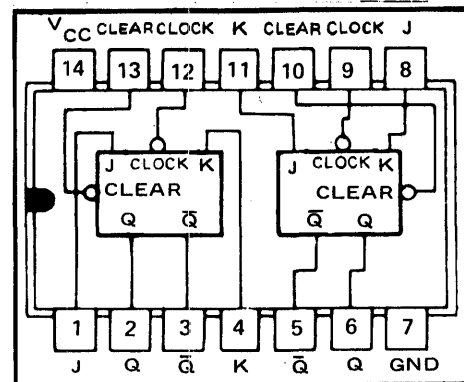
NOTES:

1. t_n = Bit time before clock pulse.
2. t_{n+1} = Bit time after clock pulse.

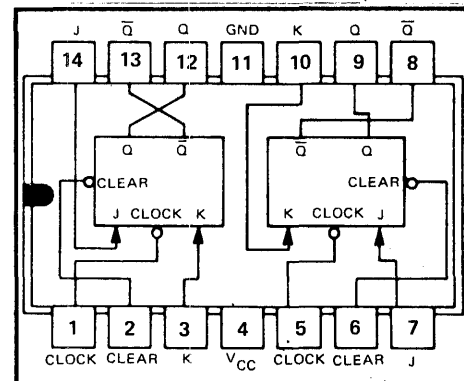
These J-K flip-flops are based on the master-slave principle. The AND gate inputs for entry into the master section are controlled by the clock pulse. The clock pulse also regulates the circuitry which connects the master and slave sections. The sequence of operation is as follows:

1. Isolate slave from master
2. Enter information from AND gate inputs to master
3. Disable AND gate inputs
4. Transfer information from master to slave

SN54107, SN74107
J OR N DUAL-IN-LINE PACKAGE (TOP VIEW)



SN5473, SN7473
J OR N DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic:

Low input to clear sets Q to logical 0.
Clear is independent of clock.

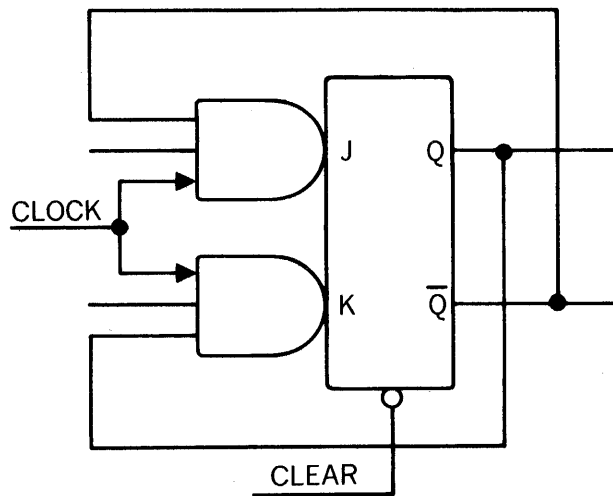
VDM 49A0002-000 and 49A0100-000

VTH-1734

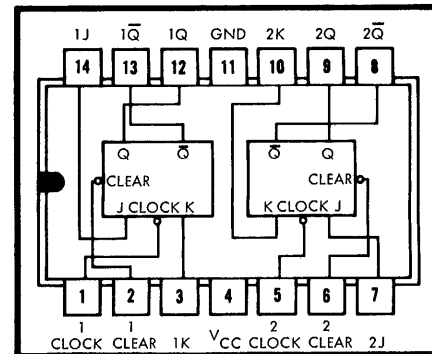
Figure V-26. Dual J-K Master-Slave Flip-Flops (SN7473N, 74107N)



CHAPTER V LOGIC DESCRIPTIONS



J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)



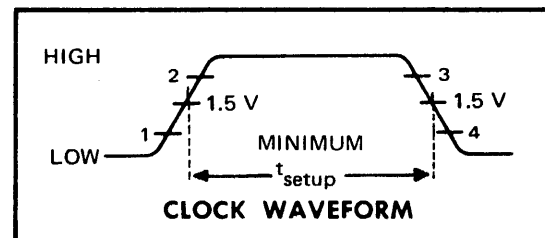
positive logic:

Low input to clear sets Q to logical 0
Clear is independent of clock

TRUTH TABLE		
t_n		t_{n+1}
J	K	Q
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

NOTES:

1. t_n = Bit time before clock pulse.
2. t_{n+1} = Bit time after clock pulse.



These J-K flip-flops are based on the master-slave principle. The AND gate inputs for entry into the master section are controlled by the clock pulse. The clock pulse also regulates the circuitry which connects the master and slave sections. The sequence of operation is as follows:

1. Isolate slave from master
2. Enter information from AND gate inputs to master
3. Disable AND gate inputs
4. Transfer information from master to slave

Logical state of J and K inputs must not be allowed to change when the clock pulse is in a high state.

Minimum clock time = 12 ns

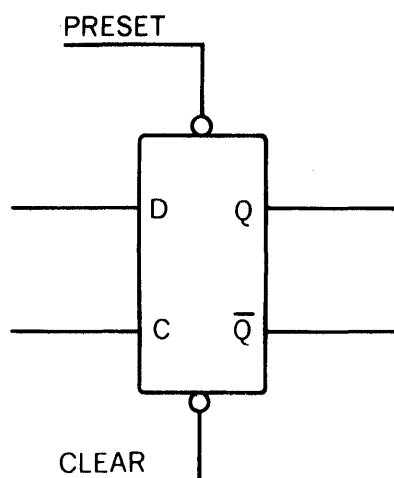
VDM 49A0036-000

VTII-1735

Figure V-27. Dual J-K Master-Slave Flip-Flops (SN74H73N)

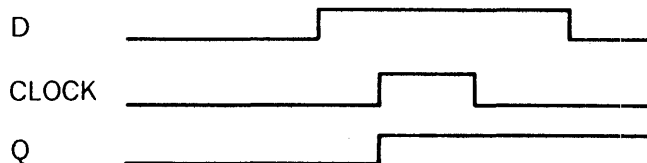


CHAPTER V LOGIC DESCRIPTIONS

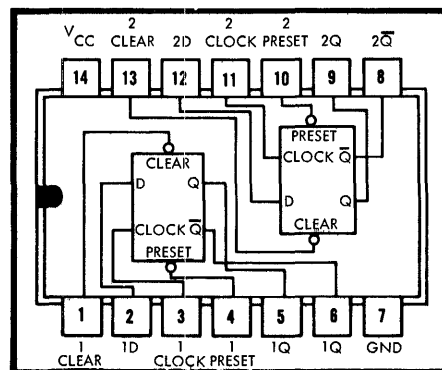


Propagation Delay Using Clock Input

To logical 0 typ. 20 ns
To logical 1 typ. 14 ns



J OR N DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic:

Low input to preset sets Q to logical 1
Low input to clear sets Q to logical 0
Preset and clear are independent of clock

TRUTH TABLE (Each Flip-Flop)

t_n	t_{n+1}	
INPUT	OUTPUT	OUTPUT
D	Q	\bar{Q}
0	0	1
1	1	0

NOTES: 1. t_n = bit time before clock pulse.
2. t_{n+1} = bit time after clock pulse.

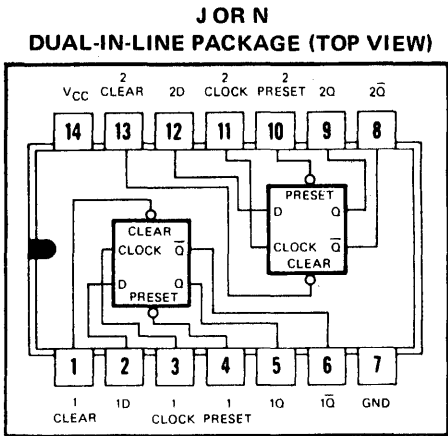
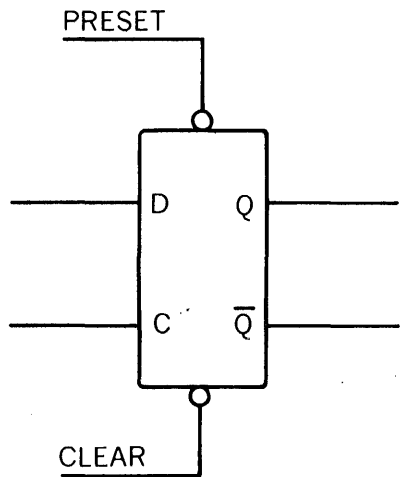
Clock triggering occurs at a voltage level of the clock pulse and is not directly related to the transition time of the positive-going pulse. After the clock input threshold voltage has been passed, the data input (D) is locked out.

VDM 49A0012-000

VTII-1736

Figure V-28. Dual D-Type Edge-Triggered Flip-Flop (SN7474N)

CHAPTER V
LOGIC DESCRIPTIONS



Low input to preset sets Q to high level
Low input to clear sets Q to low level
Preset and clear are independent of clock

Propagation Delay Using Clock Inputs

To logical 0 typ. 8.5 ns
To logical 1 typ. 13 ns

TRUTH TABLE (Each Flip-Flop)

t_n	t_{n+1}	
INPUT	OUTPUT	OUTPUT
D	Q	\bar{Q}
0	0	1
1	1	0

Width of clock pulse = minimum of 15 ns

NOTES: A. t_n = bit time before clock pulse.
B. t_{n+1} = bit time after clock pulse.

Information at input D is transferred to the Q output on the positive-going edge of the clock pulse. Clock triggering occurs at a voltage level of the clock pulse and is not directly related to the transition time of the positive-going pulse. When the clock input is at either the high or low level, the D input signal has no effect.

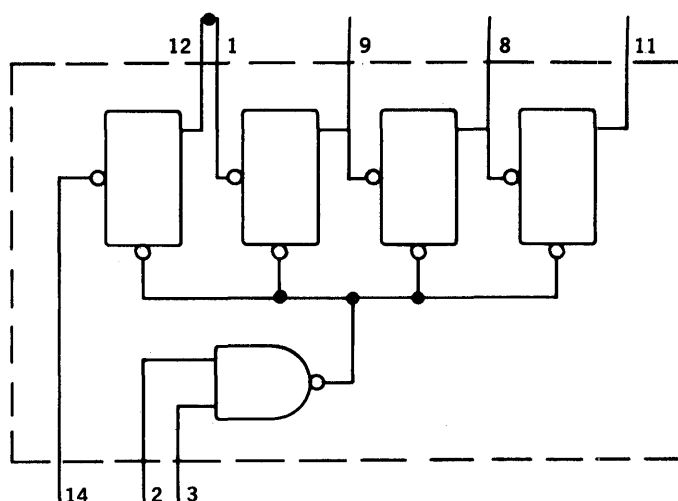
VDM 49A0082-001

VTII-1737

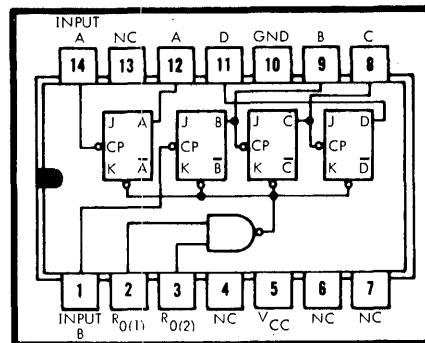
Figure V-29. Dual D-Type Edge-Triggered Flip-Flop (SN74H74N)



CHAPTER V LOGIC DESCRIPTIONS



J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)



TRUTH TABLE (See Notes 1, 2, and 3)

COUNT	OUTPUT			
	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Propagation Delay From Input Clock A to Output D

To logical 0 typ. 75 ns
To logical 1 typ. 75 ns

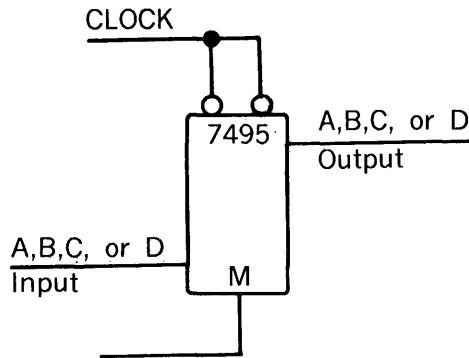
- NOTES: 1. Output A connected to input B
2. To reset all outputs to logical 0 both $R_0(1)$ and $R_0(2)$ inputs must be at logical 1.
3. Either (or both) reset inputs $R_0(1)$ and $R_0(2)$ must be at a logical 0 to count.

VTII-1738

Figure V-30. 4-Bit Binary Counter (SN7493N)

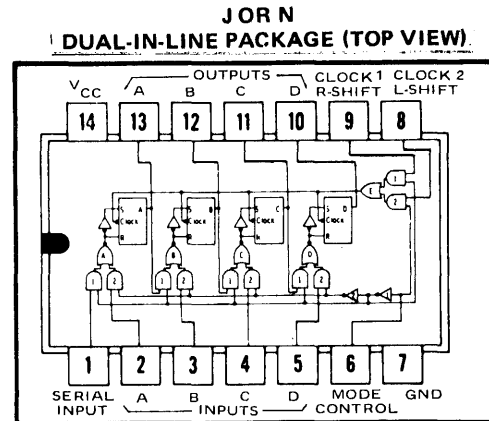


CHAPTER V LOGIC DESCRIPTIONS



Propagation Delay From Clock 1 or 2

To logical 0	typ. 24 ns
To logical 1	typ. 26 ns



positive logic:

Mode control = 0 for right shift
Mode control = 1 for left shift or parallel load

Information transferred when clock input goes low

When a logical 0 level is applied to the mode control input, the number 1 AND gates are enabled and the number 2 AND gates are inhibited. In this mode, the output of each flip-flop is coupled to the R-S inputs of the succeeding flip-flop and right-shift operation is performed by clocking at the clock 1 input. In this mode, serial data is entered at the serial input. Clock 2 and parallel inputs A through D are inhibited by the number 2 AND gates.

When a logical 1 level is applied to the mode control input, the number 1 AND gates are inhibited (decoupling the outputs from the succeeding R-S inputs to prevent right-shift) and the number 2 AND gates are enabled to allow entry of data through parallel inputs A through D and clock 2. This mode permits parallel loading of the register; or, with external interconnection, shift-left operation. In this mode, shift-left can be accomplished by connecting the output of each flip-flop to the parallel input of the previous flip-flop (D output to input C, etc.) and serial data are entered at input D.

VDM 49A0090-001

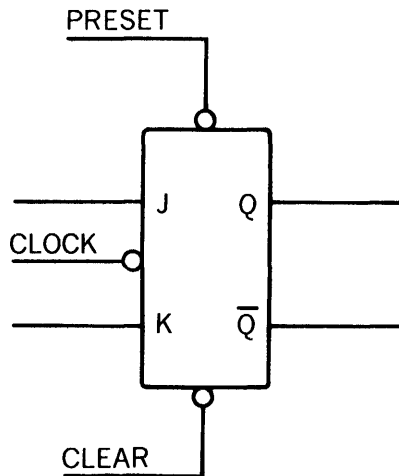
Parallel-In Parallel-Out Register

VTII-1739

Figure V-31. 4-Bit Right-Shift Left-Shift Register (SN7495N)



CHAPTER V LOGIC DESCRIPTIONS

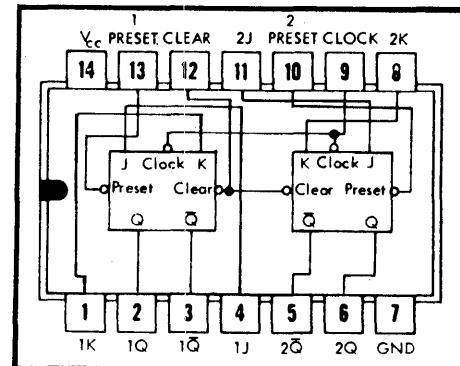


TRUTH TABLE		
t_n		t_{n+1}
J	K	Q
0	0	Q_n
0	1	0
1	0	1
1	1	\bar{Q}_n

NOTES:

1. t_n = Bit time before clock pulse
2. t_{n+1} = Bit time after clock pulse

J OR K DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic:

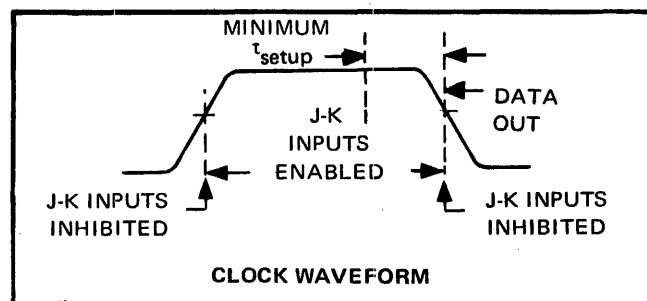
- Low input to preset sets Q to logical 1
- Low input to clear sets Q to logical 0
- Preset and clear are independent of clock

Propagation Delay Using Clock

To logical 0	typ. 16 ns
To logical 1	typ. 10 ns

Minimum T Setup

Logical 1	10 ns
Logical 0	13 ns



These dual monolithic J-K flip-flops are negative edge-triggered. They feature individual J, K, and asynchronous preset inputs to each flip-flop as well as common clock and asynchronous clear inputs. When the clock goes high, the inputs are enabled and data will be accepted. The logical state of the J and K inputs may be allowed to change when the clock pulse is in a high state and bistable will perform according to the truth table as long as minimum set-up times are observed. Input data are transferred to the outputs on the negative edge of the clock pulse.

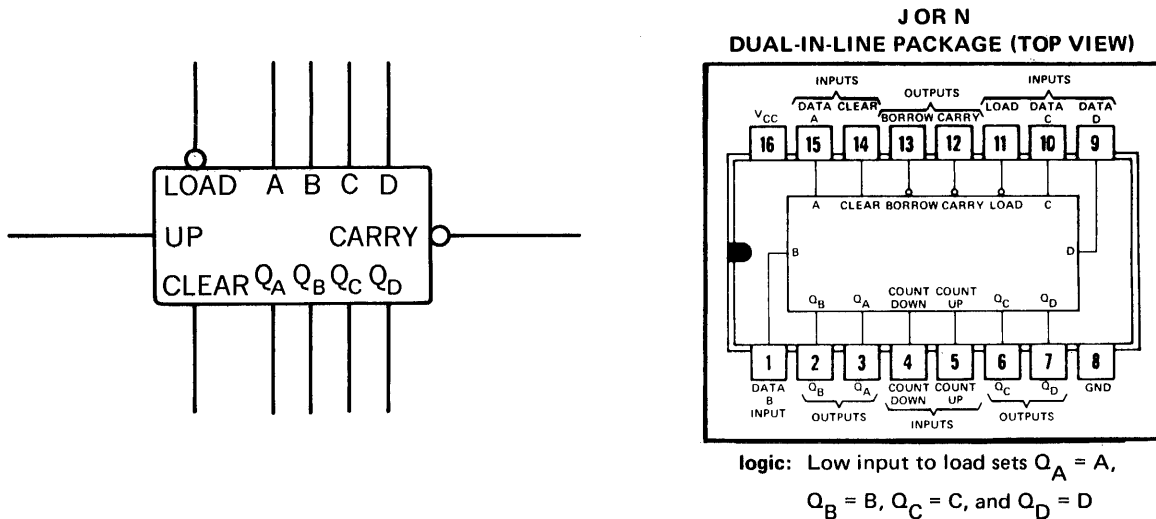
VDM 49A0099-000

VT11-1740

Figure V-32. Dual J-K Edge-Triggered Flip-Flop (SN74H108N)



CHAPTER V LOGIC DESCRIPTIONS



The outputs of the four master-slave flip-flops are triggered by a low-to-high transition of either count (clock) input. The direction of counting is determined by which count input is pulsed while the other count input is high.

All four counters are fully programmable; that is, the outputs may be preset to any state by entering the desired data at the data inputs while the load input is low. The output will change to agree with the data inputs independently of the count pulse. This feature allows the counters to be used as modulo-N dividers by simply modifying the count length.

A clear input has been provided which forces all outputs to the low level when a high level is applied. The clear function is independent of the count and load inputs. An input buffer has been placed on the clear, count, and load inputs to lower the drive requirements to one normalized Series 54/74 load. This is important when the output of the driving circuitry is somewhat limited.

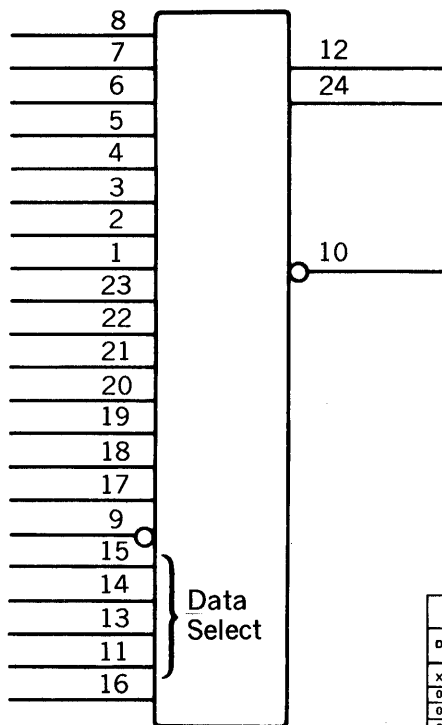
**VDM 49A0091-000
Dual Clock With Clear**

VTII-1741

Figure V-33. Synchronous 4-Bit Up/Down Counter (SN74193J)



CHAPTER V LOGIC DESCRIPTIONS



Allows user to sample bit specified by data select for a 1 or 0

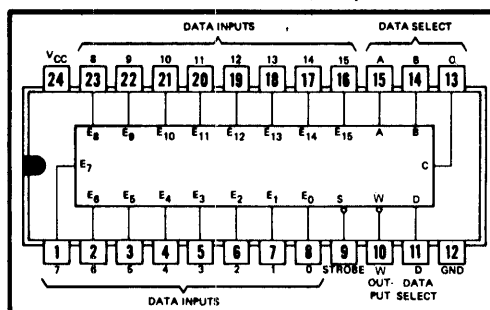
Example:

If data select was equal to 7, then pin 1 would be tested for a 1 or 0 and the results of the test put on pin 10.

Propagation Delay

Through 4 selects	28 ns
Through 3 selects	20 ns
Data input to output	10 ns

N DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic:

$$W = S(ABCDE_0 + ABCDE_1 + ABCDE_2 + ABCDE_3 + ABCDE_4 + ABCDE_5 + ABCDE_6 + ABCDE_7 + ABCDE_8 + ABCDE_9 + ABCDE_{10} + ABCDE_{11} + ABCDE_{12} + ABCDE_{13} + ABCDE_{14} + ABCDE_{15})$$

TRUTH TABLE

INPUTS																OUTPUT						
D	C	B	A	STROBE	E ₀	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	E ₇	E ₈	E ₉	E ₁₀	E ₁₁	E ₁₂	E ₁₃	E ₁₄	E ₁₅	W	
X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
0	0	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
0	0	0	1	0	0	X	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
0	0	0	1	0	1	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
0	0	1	0	0	0	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	X	1
0	0	1	0	0	1	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	0
0	0	1	1	0	0	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	1
0	0	1	1	0	1	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	X	0
0	1	0	0	0	0	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	X	1
0	1	0	0	0	1	X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	X	0
0	1	0	1	0	0	X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	X	1
0	1	0	1	0	1	X	X	X	X	X	1	X	X	X	X	X	X	X	X	X	X	0
0	1	1	0	0	0	X	X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	1
0	1	1	0	0	1	X	X	X	X	X	X	1	X	X	X	X	X	X	X	X	X	0
0	1	1	1	0	0	X	X	X	X	X	X	X	0	X	X	X	X	X	X	X	X	1
0	1	1	1	0	1	X	X	X	X	X	X	X	1	X	X	X	X	X	X	X	X	0
1	0	0	0	0	0	X	X	X	X	X	X	X	0	X	X	X	X	X	X	X	X	1
1	0	0	0	0	1	X	X	X	X	X	X	X	1	X	X	X	X	X	X	X	X	0
1	0	0	1	0	0	X	X	X	X	X	X	X	0	X	X	X	X	X	X	X	X	1
1	0	0	1	0	1	X	X	X	X	X	X	X	1	X	X	X	X	X	X	X	X	0
1	0	1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	0	1	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
1	0	1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	0	1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
1	1	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
1	1	0	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	0	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
1	1	1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	1	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
1	1	1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
1	1	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
1	1	1	1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0
1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	0
1	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	1
1	1	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	0
1	1	1	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	1
1	1	1	1	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	X	X	X	X	X	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	X	X	X	X	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	X	X	X	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	X	X	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	X	0

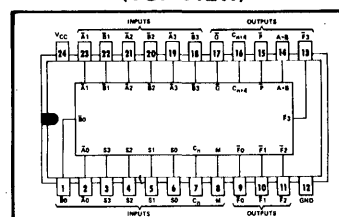
When used to indicate an input condition, X = LOGICAL 1 OR LOGICAL 0

VDM 49A0097-000

VTII-1742

Figure V-34. Data Selector/Multiplexor (SN74150N)

**N, DUAL-IN-LINE PACKAGE
(TOP VIEW)**



logic: see function tables

PIN DESIGNATIONS

DESIGNATION	PIN NOS.	FUNCTION
A3, A2, A1, A0	19, 21, 23, 2	WORD A INPUTS
B3, B2, B1, B0	18, 20, 22, 1	WORD B INPUTS
S3, S2, S1, S0	3, 4, 5, 6	FUNCTION-SELECT INPUTS
C _n	7	CARRY INPUT
M	8	MODE CONTROL INPUT
F3, F2, F1, F0	13, 11, 10, 9	FUNCTION OUTPUTS
A = B	14	COMPARATOR OUTPUT
F	15	CARRY PROPAGATE OUTPUT
C _{n+4}	16	CARRY OUTPUT
G	17	CARRY GENERATE OUTPUT
V _{CC}	24	SUPPLY VOLTAGE
gnd	23	GROUND

TABLE OF LOGIC FUNCTIONS

FUNCTION SELECT				OUTPUT FUNCTION		FUNCTION SELECT				OUTPUT FUNCTION	
S3	S2	S1	S0	LOW LEVELS ACTIVE	HIGH LEVELS ACTIVE	S3	S2	S1	S0	NEGATIVE LOGIC	POSITIVE LOGIC
L	L	L	L	F = A minus 1	F = A	L	L	L	L	F = \bar{A}	F = \bar{A}
L	L	L	H	F = AB minus 1	F = A+B	L	L	L	H	F = $\bar{A}\bar{B}$	F = $\bar{A}\bar{B}$
L	L	H	L	F = $\bar{A}\bar{B}$ minus 1	F = A+B	L	L	H	L	F = $\bar{A}+B$	F = $\bar{A}B$
L	L	H	H	F = minus 1 (2's complement)	F = minus 1 (2's complement)	L	L	H	H	F = Logical 1	F = Logical 0
L	H	L	L	F = A plus (A+B)	F = A plus AB	L	H	L	L	F = $\bar{A}\bar{B}$	F = $\bar{A}B$
L	H	L	H	F = AB plus (A+B)	F = (A+B) plus AB	L	H	L	H	F = \bar{B}	F = \bar{B}
L	H	H	L	F = A minus B minus 1	F = A minus B minus 1	L	H	H	L	F = $\bar{A} \odot \bar{B}$	F = $\bar{A} \odot B$
L	H	H	H	F = A+B	F = AB minus 1	L	H	H	H	F = A+B	F = AB
H	L	L	L	F = A plus (A+B)	F = A plus AB	H	L	L	L	F = $\bar{A}\bar{B}$	F = $\bar{A}+B$
H	L	L	H	F = A plus B	F = A plus B	H	L	L	H	F = $\bar{A} \odot B$	F = $\bar{A} \odot \bar{B}$
H	L	H	L	F = $\bar{A}\bar{B}$ plus (A+B)	F = (A+B) plus AB	H	L	H	L	F = \bar{B}	F = \bar{B}
H	L	H	H	F = A+B	F = AB minus 1	H	L	H	H	F = A+B	F = AB
H	H	L	L	F = A plus A [†]	F = A plus A [†]	H	H	L	L	F = Logical 0	F = Logical 1
H	H	L	H	F = AB plus A	F = (A+B) plus A	H	H	L	H	F = $\bar{A}\bar{B}$	F = A+B
H	H	H	L	F = $\bar{A}\bar{B}$ plus A	F = (A+B) plus A	H	H	H	L	F = AB	F = A+B
H	H	H	H	F = A	F = A minus 1	H	H	H	H	F = A	F = A

With mode control (M) and C_n low

[†] Each bit is shifted to the next more significant position.

With mode control (M) high: C_n irrelevant

For positive logic: logical 1 = high voltage
logical 0 = low voltage

The SN74181 are high-speed arithmetic logic unit (ALU)/function generators which have a complexity of 75 equivalent gates on a monolithic chip. This circuit performs 16 binary arithmetic operations on two 4-bit words as shown in the function table. These operations are selected by the four function-select lines (S0, S1, S2, and S3) and include addition, subtraction, decrement, and straight transfer. When performing arithmetic manipulations, the internal carries must be enabled by applying a low-level voltage to the mode control input (M). A full carry look-ahead scheme is made available in the SN74181 for fast, simultaneous carry generation with a group carry propagation (P) and carry generate (G) for the four bits in the package. When used in conjunction with the SN74182 full carry look-ahead circuits, high-speed arithmetic operations can be performed. For example, the typical addition time for the SN74181 is 24 nanoseconds for four bits. When expanding to 16-bit addition with the SN74182, only 13 nanoseconds further delay is added so that the total addition time is 35 nanoseconds, or 2.2 nanoseconds per bit. One SN74181 is needed for every 16 bits (four SN74181 circuits).

VDM 49A0096-000



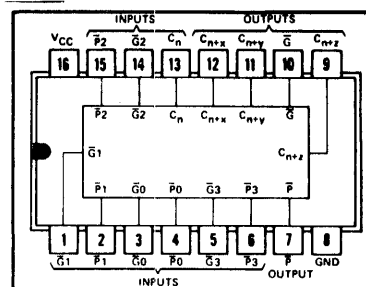
CHAPTER V

LOGIC DESCRIPTIONS

PIN DESIGNATIONS

DESIGNATION	PIN NOS.	FUNCTION
$\bar{G}_0, \bar{G}_1, \bar{G}_2, \bar{G}_3$	3, 1, 14, 5	ACTIVE-LOW CARRY GENERATE INPUTS
$\bar{P}_0, \bar{P}_1, \bar{P}_2, \bar{P}_3$	4, 2, 15, 6	ACTIVE-LOW CARRY PROPAGATE INPUTS
C_n	13	CARRY INPUT
$C_{n+x}, C_{n+y}, C_{n+z}$	12, 11, 9	CARRY OUTPUTS
\bar{G}	10	ACTIVE-LOW CARRY GENERATE OUTPUT
\bar{P}	7	ACTIVE-LOW CARRY PROPAGATE OUTPUT
V_{CC}	16	SUPPLY VOLTAGE
GND	8	GROUND

N DUAL-IN-LINE PACKAGE (TOP VIEW)



logic: see description

The SN74182 is a high-speed, look-ahead carry generator capable of anticipating a carry across four binary adders or group of adders. It is cascadable to perform full look-ahead across n-bit adders, with only 13 nanoseconds delay for each level of look-ahead. Carry, generate-carry, and propagate-carry functions are provided as enumerated in the pin designation table above.

The SN74182, when used in conjunction with the SN74181 arithmetic logic unit (ALU), provides full high-speed carry look-ahead capability for up to n-bit words. Each SN74182 generates the look-ahead (anticipated carry) across a group of four ALUs and, in addition, other carry look-ahead circuits may be employed to anticipate carry across sections of four look-ahead packages up to n-bits. Applications data for the SN74181 illustrates cascading of SN74182 circuits to perform multi-level look-ahead.

Carry inputs and outputs of the SN74181 are in their true form and the carry propagates (P) and carry generates (G) are in negated form; therefore, the carry (input, outputs, generate, and propagate) functions of the look-ahead circuit are implemented in the compatible forms. Reinterpretations of carry functions at the SN74181 are also applicable and compatible with the look-ahead package. Logic equations are:

$$\begin{aligned}
 C_{n+x} &= G_0 + P_0 C_n \\
 C_{n+y} &= G_1 + P_1 G_0 + P_1 P_0 C_n \\
 C_{n+z} &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_n \\
 \bar{G} &= \bar{G}_3 + P_3 \bar{G}_2 + P_3 P_2 \bar{G}_1 + P_3 P_2 P_1 \bar{G}_0 \\
 \bar{P} &= P_3 P_2 P_1 P_0
 \end{aligned}$$

VDM 49A0102-000

VTII-1744

Figure V-36. Look-Ahead Carry Generator (SN74182N)

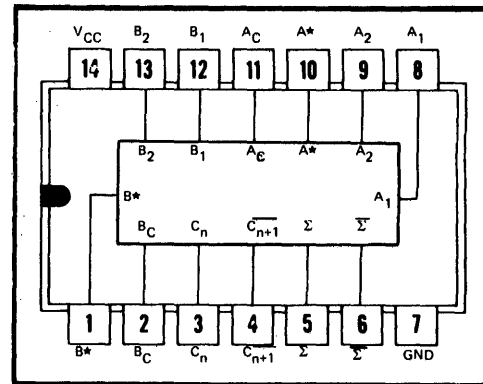


CHAPTER V LOGIC DESCRIPTIONS

TRUTH TABLE
(See Notes 1, 2, and 3)

C_n	B	A	$\overline{C_{n+1}}$	$\overline{\Sigma}$	Σ
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	0	1

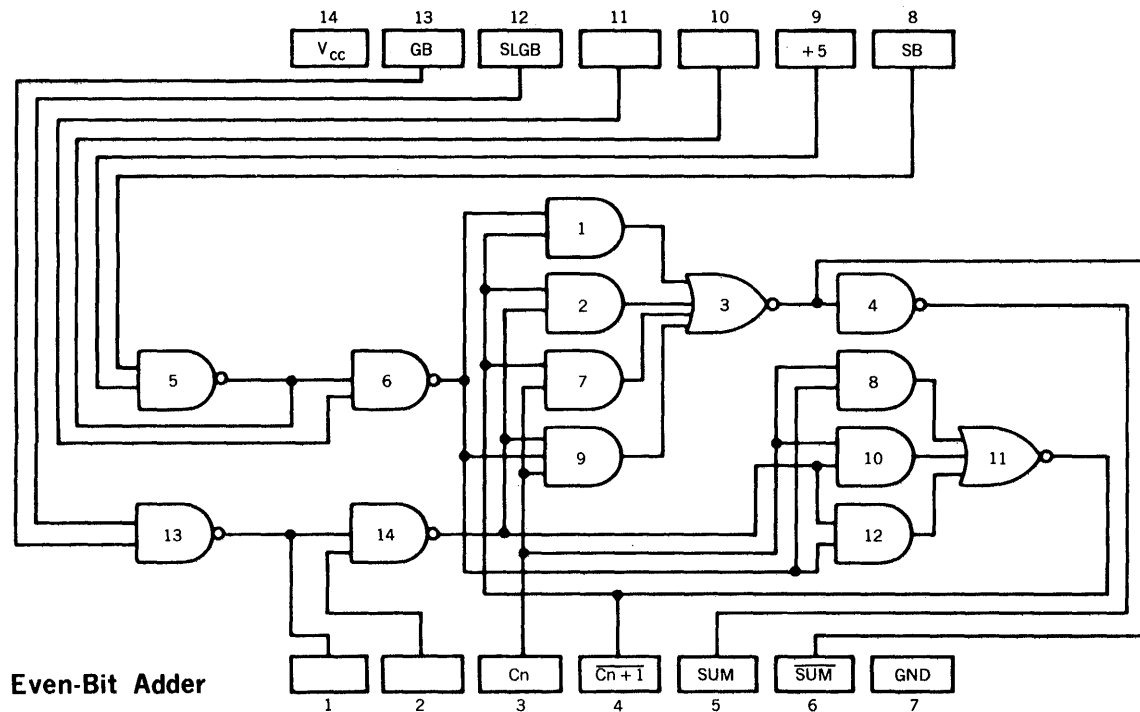
JORN
DUAL-IN-LINE PACKAGE (TOP VIEW)



NOTES:

1. $A = A^* \cdot A_C$, $B = B^* \cdot B_C$ where $A^* = A_1 \cdot A_2$, $B^* = B_1 \cdot B_2$
2. When A^* or B^* are used as Inputs, A_1 and A_2 or B_1 and B_2 respectively must be connected to GND.
3. When A_1 and A_2 or B_1 and B_2 are used as Inputs, A^* or B^* respectively must be open or used to perform Dot-OR logic.

positive logic: See truth table



VDM 49A0001-000

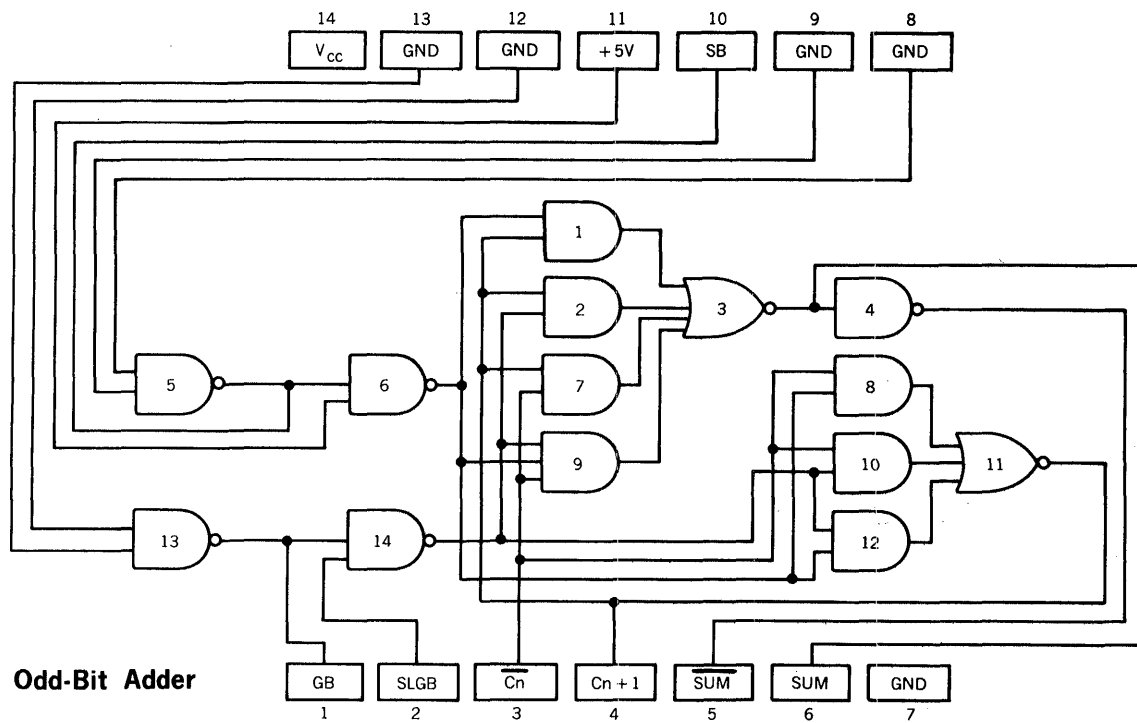
VTH-1745

Figure V-37. Gated Full Adder (SN7480N)

(continued next page)



CHAPTER V LOGIC DESCRIPTIONS



Operation: This gated full adder is used in the 620/i and 620/L systems for both even and odd bits, differing only in the input pins used.



CHAPTER V LOGIC DESCRIPTIONS

Truth Table

Access time = 40 ns

The truth table shown is the high-speed paper tape bootstrap for the 620/f (16 words of 16 bits).

32 words of 8 bits
16 words of 16 bits

H = logical 1
L = logical 0

Word	Byte	INPUTS						MSB	OUTPUTS								LSB
		BINARY SELECT							ENABLE								
		E	D	C	B	A	G										
0	0	L	L	L	L	L	L	1	0	0	0	0	1	0	1		
	1	L	L	L	L	H	L	1	0	0	1	1	1	1	1		
1	2	L	L	L	H	L	L	0	0	0	0	1	0	0	0		
	3	L	L	L	H	H	L	0	0	0	0	1	0	0	1		
2	4	L	L	H	L	L	L	0	0	0	0	1	0	0	0		
	5	L	L	H	L	H	L	0	0	1	0	0	0	0	1		
3	6	L	L	H	H	L	L	0	0	0	0	1	0	0	1		
	7	L	L	H	H	H	L	0	0	1	0	0	1	1	0		
4	8	L	H	L	L	L	L	0	0	0	0	0	0	1	0		
	9	L	H	L	L	H	L	0	0	0	1	0	0	0	0		
5	10	L	H	L	H	L	L	0	0	0	0	0	0	0	0		
	11	L	H	L	H	H	L	1	0	0	0	1	1	0	0		
6	12	L	H	H	L	L	L	0	1	0	1	1	0	1	0		
	13	L	H	H	L	H	L	0	0	0	0	0	0	0	0		
7	14	L	H	H	H	L	L	0	0	0	0	0	0	1	0		
	15	L	H	H	H	H	L	0	0	0	0	1	0	0	0		
8	16	H	L	L	L	L	L	0	0	0	0	1	1	1	0		
	17	H	L	L	L	H	L	0	0	0	0	0	0	0	0		
9	18	H	L	L	H	L	L	0	0	0	0	1	0	1	0		
	19	H	L	L	H	H	L	0	1	1	0	0	1	0	0		
10	20	H	L	H	L	L	L	0	0	0	0	1	0	1	0		
	21	H	L	H	L	H	L	0	1	0	0	0	0	0	1		
11	22	H	L	H	H	L	L	1	0	0	0	0	0	0	1		
	23	H	L	H	H	H	L	0	1	0	1	1	1	1	1		
12	24	H	H	L	L	L	L	1	0	0	0	0	0	1	1		
	25	H	H	L	L	H	L	0	1	0	1	1	1	1	1		
13	26	H	H	L	H	L	L	0	0	0	0	0	0	0	0		
	27	H	H	L	H	H	L	1	0	0	0	0	0	0	0		
14	28	H	H	H	L	L	L	0	0	0	0	0	0	1	0		
	29	H	H	H	L	H	L	0	0	0	0	0	0	0	0		
15	30	H	H	H	H	L	L	0	0	0	0	0	0	0	0		
	31	H	H	H	H	H	L	1	0	0	0	1	1	0	0		
	ALL	X	X	X	X	X	H	H	H	H	H	H	H	H	H		

H = high level, L = low level, X = irrelevant

VDM 49A0113-000 (as selected)

VTII-1747

Figure V-38. 256-Bit Read-Only Memory (SN7488N)

(continued next page)

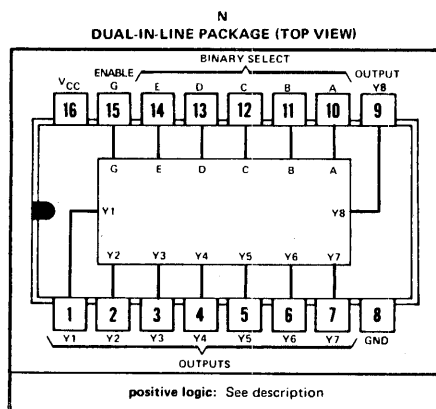


CHAPTER V LOGIC DESCRIPTIONS

The SN7488 circuit is a customer-programmed, 256-bit, read-only memory organized as 32 words of eight bits each. This monolithic, high-speed, transistor-transistor logic (TTL), 32-word memory array is addressed in straight 5-bit binary with full on-chip decoding. An overriding memory-enable input is provided which, when taken high, will inhibit the 32 address gates and cause all eight outputs to remain high. Data, as specified by the customer on the illustrated truth table/order blank, are permanently programmed into the monolithic structure for the 256 bit locations. This organization is expandable to n-words of N-bit length.

The addressing of an eight-bit word is accomplished through the buffered, binary select inputs which are decoded by the 32 five-input address gates. When the memory-enable input is high, all 32 gate outputs are low, turning off the eight output buffers.

Data are programmed into the memory at the emitters of 32 eight-emitter transistors. The programming process involves connecting or not connecting each of the 256 emitters. If an emitter is connected, a low-level voltage is read out of that bit location when its decoding gate is addressed. If the emitter is not connected, a high-level voltage is read when addressed. Those decoding-gate output emitters which are used are connected to their respective bit lines to drive the eight output buffers. Since only one decoding gate is addressed at a time, only one of the 32 transistors can supply current to the output buffers at a time.

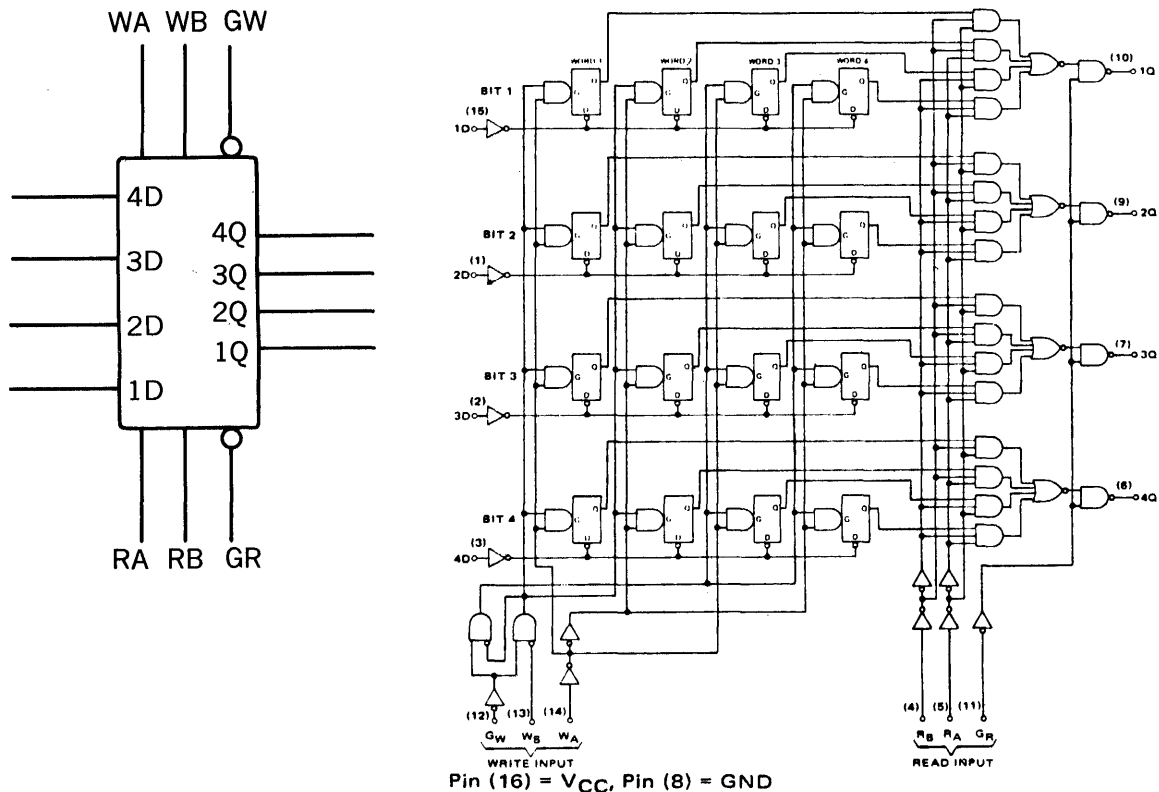


VTII-1748

Figure V-38. 256-Bit Read-Only Memory (SN7488N) (continued)



CHAPTER V LOGIC DESCRIPTIONS



WRITE FUNCTION TABLE
(SEE NOTES A, B, AND C)

WRITE INPUTS			WORD			
W _B	W _A	W _G	0	1	2	3
L	L	L	Q = D	Q _n	Q _n	Q _n
L	H	L	Q _n	Q = D	Q _n	Q _n
H	L	L	Q _n	Q _n	Q = D	Q _n
H	H	L	Q _n	Q _n	Q _n	Q = D
X	X	H	Q _n	Q _n	Q _n	Q _n

READ FUNCTION TABLE
(SEE NOTES A AND D)

READ INPUTS			OUTPUTS			
R _B	R _A	R _G	1Q	2Q	3Q	4Q
L	L	L	W0B1	W0B2	W0B3	W0B4
L	H	L	W1B1	W1B2	W1B3	W1B4
H	L	L	W2B1	W2B2	W2B3	W2B4
H	H	L	W3B1	W3B2	W3B3	W3B4
X	X	H	H	H	H	H

NOTES: A. H = high level, L = low level, X = irrelevant

B. (Q = D) = The four selected internal flip-flop outputs will assume the states applied to the four external data inputs.

C. Q_n = No change.

D. W0B1 = The first bit of word 0, etc.

VDM 49A0108-000
Organized 4 Words of 4 Bits

VTII-1749

Figure V-39. High-Speed Buffer Memory/Register File (SN74170N)

(continued next page)



CHAPTER V LOGIC DESCRIPTIONS

The SN74170 MSI 16-bit TTL register files are organized as 4 words of 4 bits each and separate on-chip decoding is provided for addressing the four word locations to either write-in or retrieve data. This permits simultaneous writing into one location and reading from another word location.

Four data inputs are available which are used to supply the 4-bit word to be stored. Location of the word is determined by the write address inputs A and B in conjunction with a write-enable signal. Data applied at the inputs should be in its true form. That is, if a high-level signal is desired from the output, a high-level is applied at the data input for that particular bit location. The latch inputs are arranged so the new data will be accepted only if both internal address gate inputs are high. When this condition exists, data at the D input are transferred to the latch output. When the write enable input, GW, is high, the data inputs are inhibited and their states can cause no change in the information stored in the internal latches. When the read enable input, GR, is high, the data outputs are inhibited and remain high.

The individual address lines permit direct acquisition of data stored in any four of the latches. Four individual decoding gates are used to complete the address for reading a word. When the read address is made in conjunction with the read-enable signal, the word appears at the four outputs.

This arrangement (data-entry addressing separate from data-read addressing and individual sense lines) eliminates recovery times, permits simultaneous reading and writing, and is limited in speed only by the write time (45 nanoseconds) and the read time (35 nanoseconds). The register file has a nondestructive readout in that data are not lost when addressed.



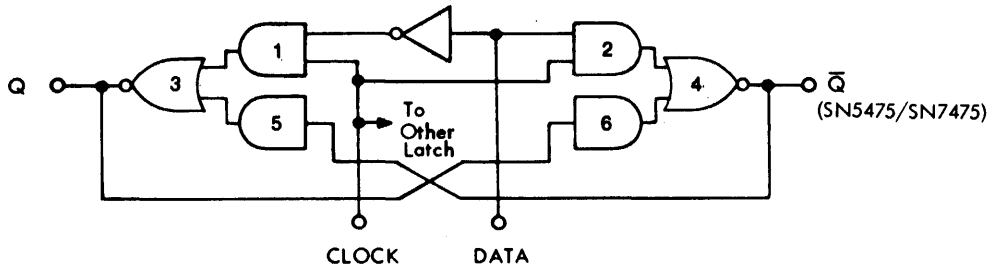
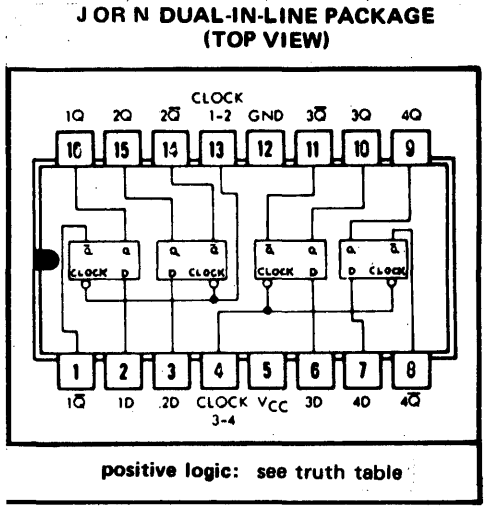
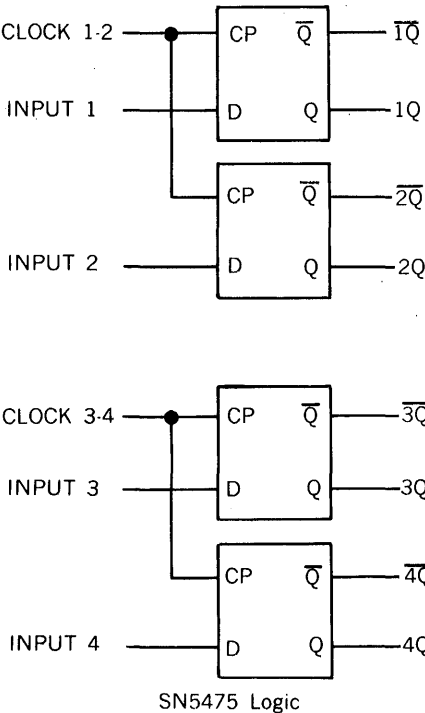
CHAPTER V
LOGIC DESCRIPTIONS

This complex-function IC is a monolithic, quadruple, bistable latch with complementary Q and Q outputs.

Information at the Q output follows that present at the data input (D) as long as the clock remains high. When the clock goes low, the information that was present at the time of transition is retained until the clock returns to high.

TRUTH TABLE (Each Latch)	
t_n	t_{n+1}
D	Q
1	1
0	0

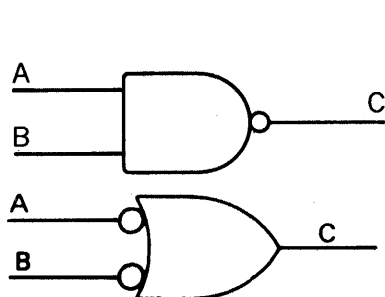
- NOTES: 1. t_n = bit time before clock pulse transition.
2. t_{n+1} = bit time after clock pulse transition.
NC—No Internal Connection



VDM 49A0000-000
VTII-1706 Figure V-40. Quadruple Bistable Latch (SN7475N)



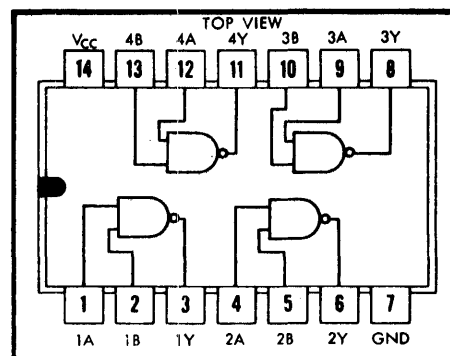
CHAPTER V LOGIC DESCRIPTIONS

**Truth Table**

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Propagation Delay

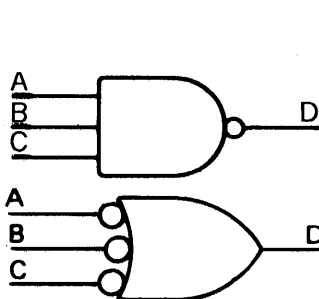
To logical 0 typ. 20 ns
To logical 1 typ. 40 ns



positive logic: $Y = \overline{AB}$

VDM 49A0008-000

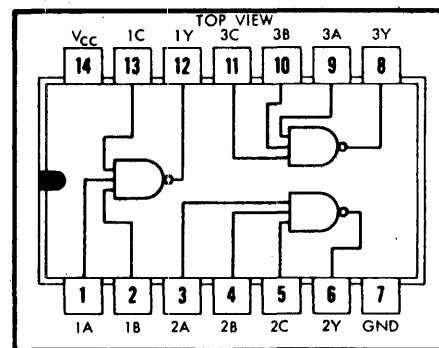
VTII-1719

Figure V-41. Quadruple 2-Input NAND Gate (SN15846N)**Truth Table**

A	B	C	D
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Propagation Delay

To logical 0 typ. 20 ns
To logical 1 typ. 40 ns



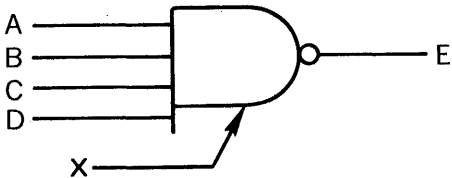
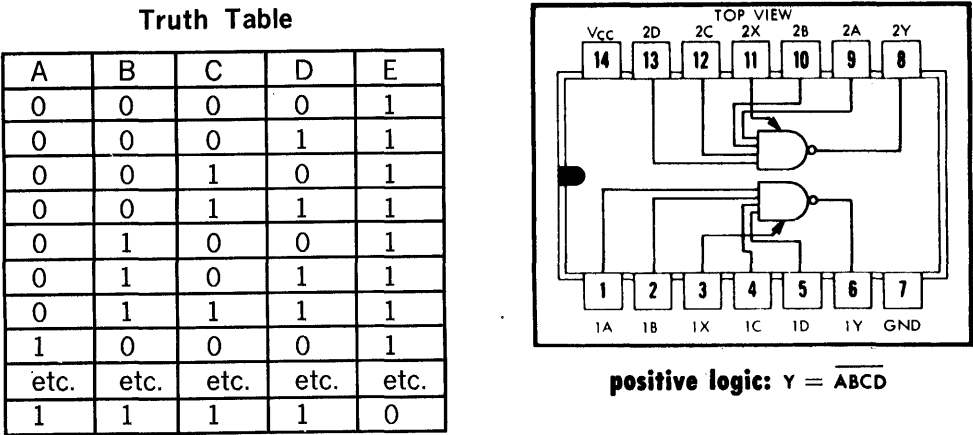
positive logic: $Y = \overline{ABC}$

VDM 49A0009-000

VTII-1707

Figure V-42. Triple 3-Input NAND Gate (SN15862N)

CHAPTER V
LOGIC DESCRIPTIONS



Propagation Delay

To logical 0	typ. 22 ns
To logical 1	typ. 32 ns

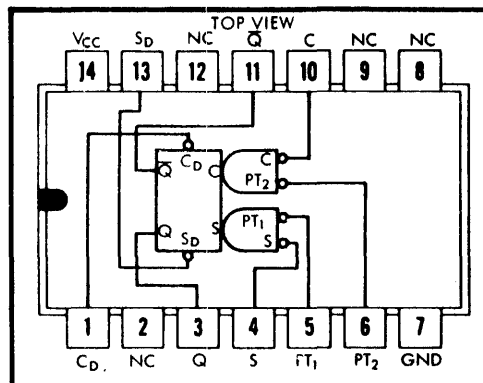
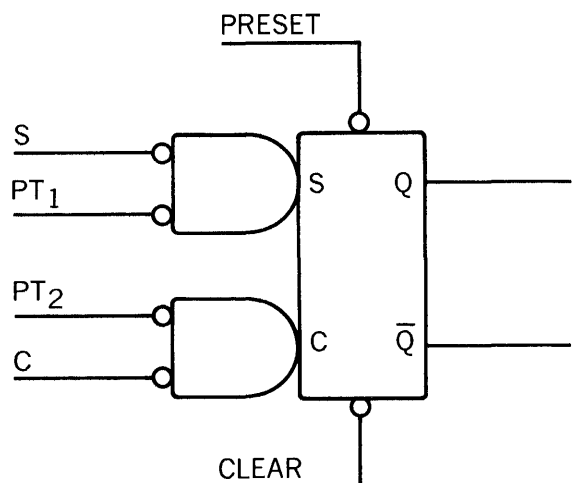
VDM 49A0010-000
Open Collector

VT11-1708

Figure V-43. Dual 4-Input NAND Power Gate (SN6006N)

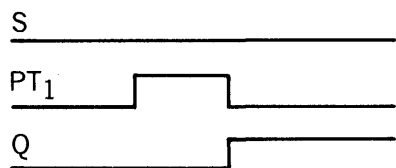


CHAPTER V LOGIC DESCRIPTIONS



positive logic: See asynchronous truth table

TRUTH TABLES



ASYNCHRONOUS			
DIRECT INPUT		OUTPUT	
S_D	C_D	Q	\bar{Q}
1	1	Q_n	\bar{Q}_n
0	1	0	1
1	0	1	0
0	0	1	1

SYNCHRONOUS					
t_n PULSE INPUT				t_{n+1} OUTPUT	
S	C	PT_1	PT_2	Q	\bar{Q}
1	X	X	1	Q_n	\bar{Q}_n
X	1	1	X	Q_n	\bar{Q}_n
0	1	0	X	1	0
0	X	0	1	1	0
1	0	X	0	0	1
X	0	1	0	0	1
0	0	0	0	Indeterminate	

Note

1. X indicates that either a logical 1 or a logical 0 may be present.
2. Logical 1 is more positive than logical 0.
3. Logical states shown for pulse inputs PT_1 and PT_2 indicate that a transition to that state has just occurred.
4. Truth tables reflect individual conditions at the inputs. Either direct input may be used to inhibit its corresponding pulse input.

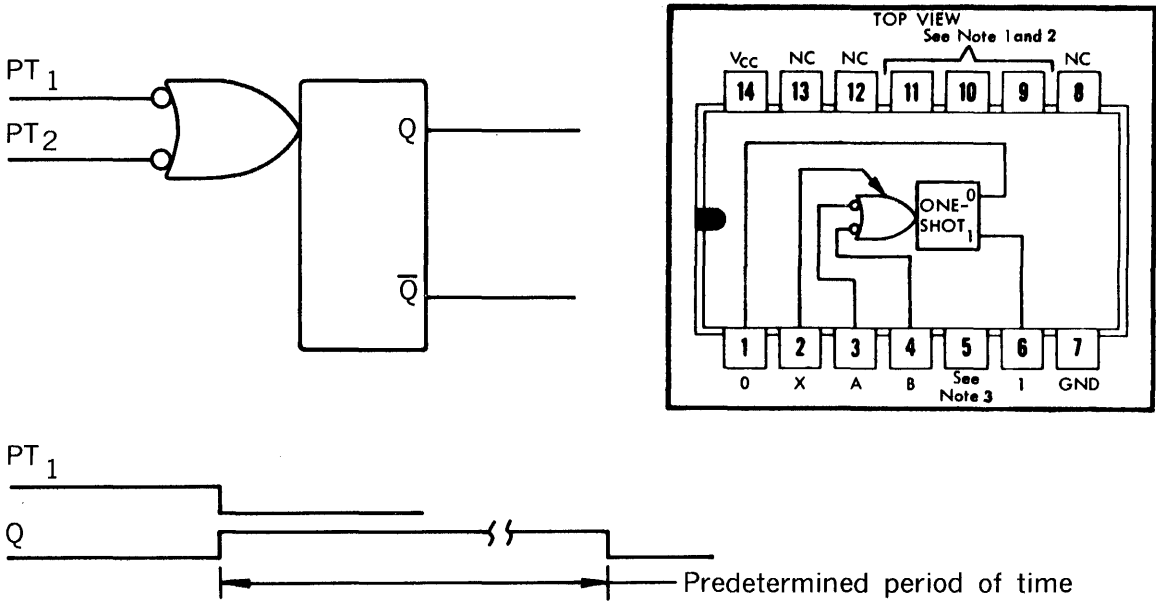
VDM 49A0014-000

VTII-1709

Figure V-44. Pulse-Triggered Binary (SN15850N)



CHAPTER V
LOGIC DESCRIPTIONS



TRUTH TABLE

t_n INPUT		t_{n+1} INPUT		OUTPUT
A	B	A	B	
1	1	1	1	INHIBIT
1	1	1	0	ONE-SHOT
1	1	0	1	ONE-SHOT
1	1	0	0	ONE-SHOT
0	1	X	X	INHIBIT
1	0	X	X	INHIBIT
0	0	X	X	INHIBIT

NOTES: a. t_n = time before input transition.
b. t_{n+1} = time after input transition.
c. X indicates that either a logical 1 or a logical 0 may be present.

VDM 49A0018-000

VT11-1710

Figure V-45. Monostable Multivibrator (SN15851N)



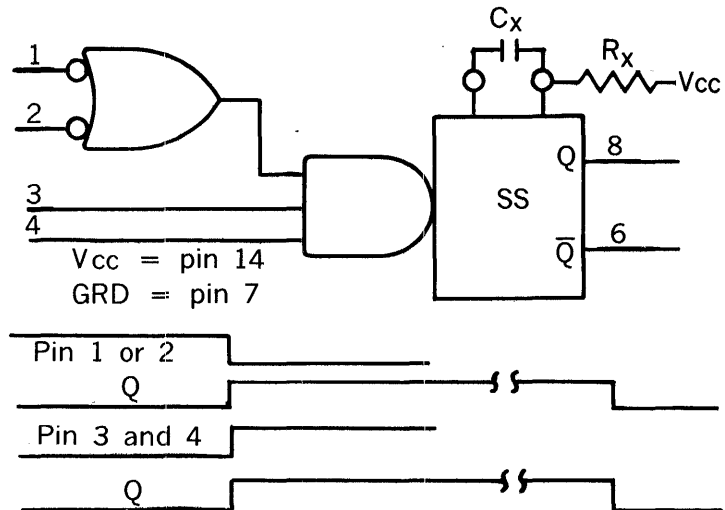
CHAPTER V LOGIC DESCRIPTIONS

Truth Table

Pin 1 or pin 2 going from a logical 1 to a logical 0 will cause a single-shot

Pin 3 and pin 4 going to a logical 1 will cause a single-shot

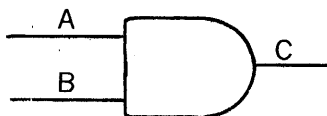
The single-shot can be re-triggered before time-out causes Q to stay high



VTII-1722

VDM 49A0524-000

Figure V-46. Retriggerable Monostable Multivibrator (Fairchild U6A960159X)

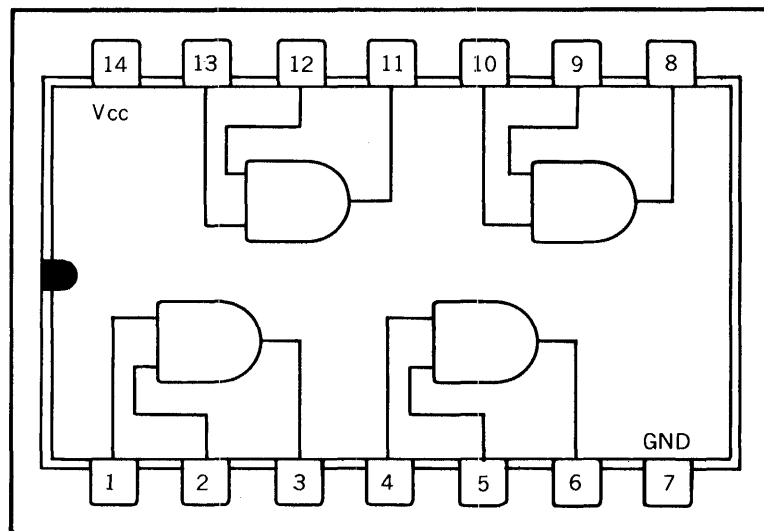


Propagation Delay

To logical "0" typ. 12 ns
To logical "1" typ. 15 ns

Truth Table

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1



Positive logic: $3 = 1 \cdot 2$

VDM 49A0104-000

VTII-1711

Figure V-47. Quadrate 2-Input AND Gate (MC3001P)

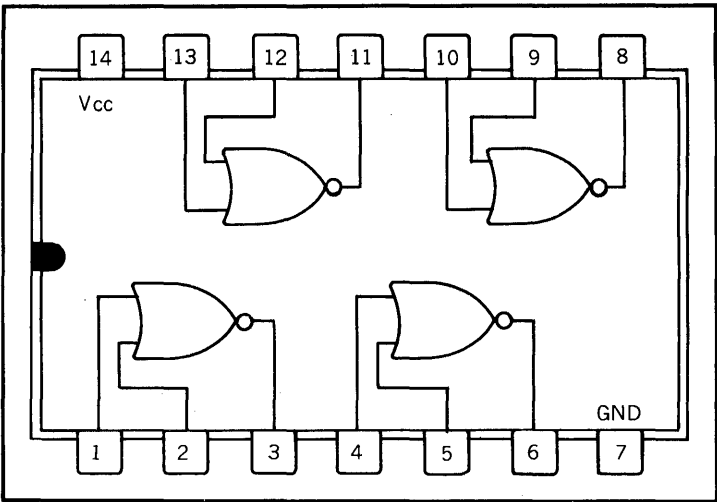
CHAPTER V
LOGIC DESCRIPTIONS



Propagation Delay
To logical "0" typ. 12 ns
To logical "1" typ. 15 ns

Truth Table

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

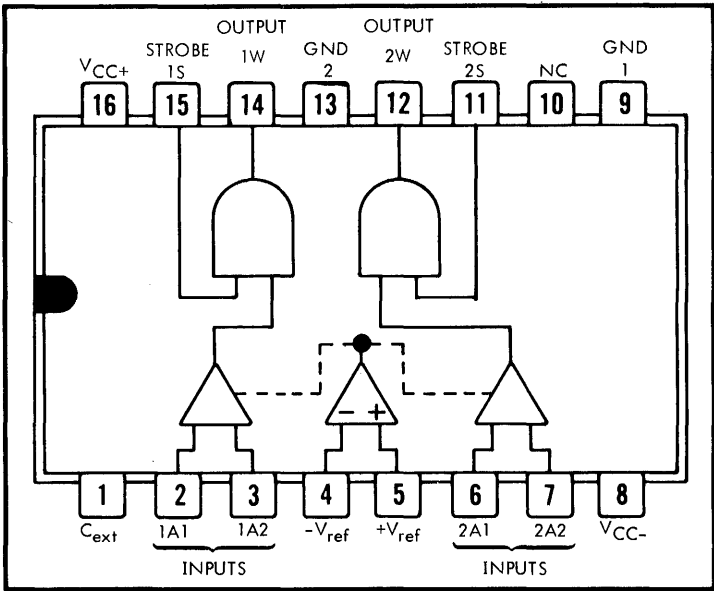


Positive logic: $3 = \overline{1 + 2}$

VDM 49A0105-000

Figure V-48. Quadruple 2-Input NOR Gate (MC3002P)

J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)



positive logic: $W = AS$

NC- No internal connection

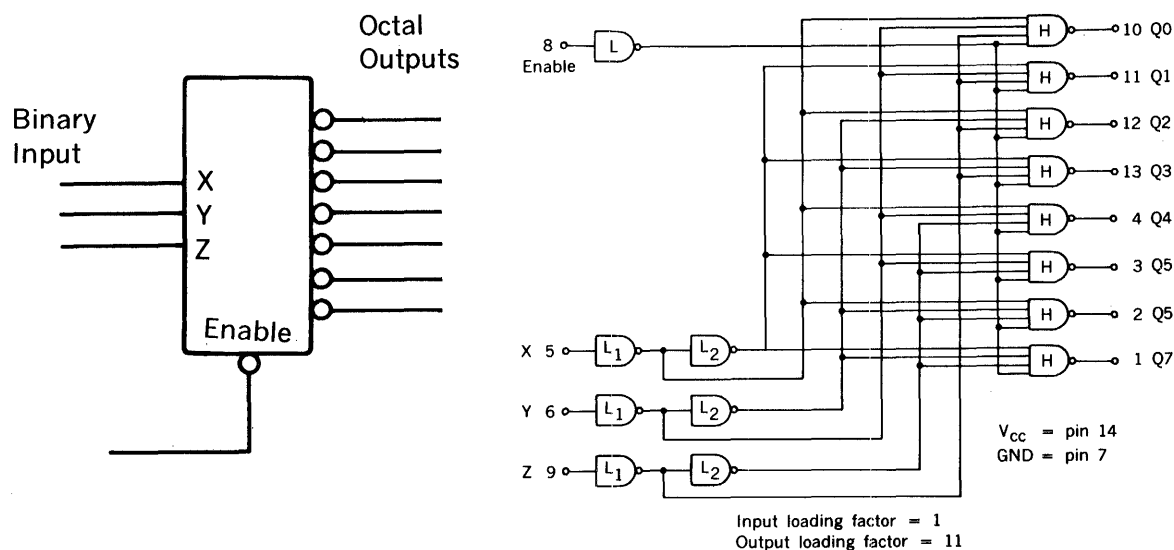
Propagation Delay
From input A1-A2 to
output W typ 20 ns
same for strobe

TRUTH TABLE

INPUTS		OUTPUT
A	S	W
H	H	H
L	X	L
X	L	L

VDM 49A0043 (SN7525) VDM 49A0043 (SN7524)

Figure V-49 Dual Sense AMPS

CHAPTER V
LOGIC DESCRIPTIONS

Truth Table

X	Y	Z	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

1 = high state

0 = low state

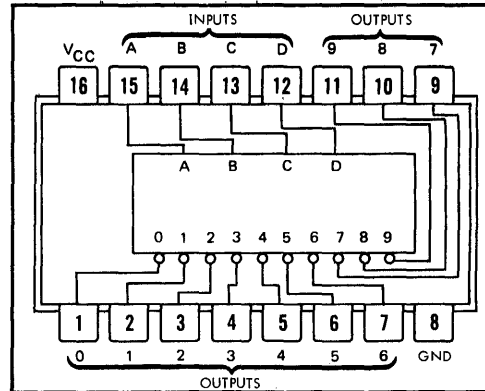


CHAPTER V
LOGIC DESCRIPTIONS

Propagation Delay

To logical 0 typ. 23 ns
To logical 1 typ. 26 ns

**J OR N DUAL-IN-LINE
OR W FLAT PACKAGE (TOP VIEW)**



positive logic: see truth tables

**SN5442/SN7442
BCD
INPUT**

D	C	B	A
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

**ALL TYPES
DECIMAL
OUTPUT**

0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	1	1	1
1	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

VTII-1713

VDM 49A0544 and 49A0044
Figure V-51. 3-Line to 8-Line Decoders



varian data machines