

MASTER

AECU-3635(2nd Ed.)

UNIVERSITY OF ILLINOIS  
GRADUATE COLLEGE  
DIGITAL COMPUTER LABORATORY

REPORT NO. 80

**ON THE DESIGN  
OF A VERY HIGH-SPEED COMPUTER**

OCTOBER 1957

SECOND EDITION

(Revised in Part April 1958)

DO NOT  
PHOTOSTAT

This work was supported in part by the  
Atomic Energy Commission and the Office of Naval Research  
under AEC Contract AT(11-1)-415

## DISCLAIMER

**This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.**

## **DISCLAIMER**

**Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.**

UNIVERSITY OF ILLINOIS

AEU-3635

GRADUATE COLLEGE

DIGITAL COMPUTER LABORATORY

URBANA, ILLINOIS

REPORT NO. 80

ON THE DESIGN OF A VERY HIGH-SPEED COMPUTER

October 1957

Second Edition

(Revised in Part - April 1958)

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

This work was supported in part by the  
Atomic Energy Commission and the Office of Naval Research  
under AEC Contract AT(11-1)-415

574 001

## PREFACE TO FIRST EDITION

This report summarizes the work of the Digital Computer Laboratory of the University of Illinois on a study of the feasibility of constructing a computer about one hundred times faster than present computers such as Illiac using presently available components and techniques.

Some promising designs and design features considered by the Digital Computer Laboratory are either not discussed in this report or are relegated to somewhat subordinate places in the report because they involve equipment which in the opinion of this Laboratory has a low probability of being built so as to have acceptable reliability with presently available components and techniques.

The first part of this report, Chapters 1, 2 and 3, deals with general features of the proposed computer and postulates the existence of various components and parts. The second part of the report, Chapters 4, 5, 6, 7, and 8, discusses these parts in some detail, beginning with a discussion of the circuits from which the different units of the computer would be built. A summary of the proposed machine's specifications is found on page ix.

The work herein reported contains results of efforts of all the personnel of the Digital Computer Laboratory and some members of the Computation Centre of the University of Toronto. The report was prepared by a committee consisting of D. B. Gillies, R. E. Meagher, D. E. Muller, R. W. McKay<sup>(1)</sup>, J. P. Nash<sup>(2)</sup>, J. E. Robertson and A. H. Taub (Chairman).

The study was supported jointly by the Atomic Energy Commission and the Office of Naval Research under AEC Contract AT(11-1)-415, by the University of Illinois and by the University of Toronto.

October, 1957

## PREFACE TO SECOND EDITION

The distribution and subsequent mailings of the original printing of this report exhausted the supply (300 copies). This printing is called a second edition because Chapter 4 has been considerably revised. Many new details as well as corrections are contained in this chapter on basic circuits which has been revised by W. J. Poppelbaum. Because circuit times and other circuit constants are affected, numerical data are changed elsewhere in the text.

A number of other corrections of only a minor nature are included in this edition. The page numbers following Chapter 4 are changed with respect to the first edition.

April, 1958

1. On the staff of the University of Toronto in residence at the Digital Computer Laboratory, University of Illinois, for the year 1956-57.
2. Resigned in April 1957.

CONTENTS

PREFACE .....	1
SUMMARY AND CONCLUSIONS .....	viii
SUMMARY OF PROPOSED MACHINE'S SPECIFICATIONS.....	ix
Chapter 1.      PROBLEMS REQUIRING FAST COMPUTERS	
1.1      Introduction.....	1
1.2      Hyperbolic and Parabolic Partial Differential Equations .	3
1.3      Elliptic Partial Differential Equations and Linear Equations .....	6
1.4      Sorting and Meshing Problems.....	7
Chapter 2.      REQUIREMENTS ON A VERY FAST COMPUTER	
2.1      Memory Capacity Requirements. ....	10
2.2      The Class of Problems That Can Use Backup Memories.....	14
2.3      Speed Requirements on the Memory.....	18
2.4      Numerical Requirements on Word Length . ....	19
2.5      Floating Point Arithmetic . ....	21
Chapter 3.      PROPOSED ORGANIZATION OF THE COMPUTER	
3.1      Introduction.....	26
3.2      A Design Criterion.....	26
3.3      The Speed of a Computer .....	28
3.4      Red Tape.....	30
3.5      Storage of Addresses.....	31
3.6      Storage of Intermediate Results.....	31
3.7      Access for Instructions and Operands.....	32
3.8      Summary of Useful Additional Storage.....	33

Chapter 3. (Continued)

3.9	Size of the Core Memory . . . . .	34
3.10	A Computer with Small Buffer Storage . . . . .	36
3.11	Words and Instructions . . . . .	37
3.12	Controls . . . . .	39
3.13	Example . . . . .	40
3.14	The Design of the Two Controls . . . . .	43
3.15	Order Code . . . . .	49
3.16	Programming Examples . . . . .	57
3.16.1	The Scalar Product of Two Vectors . . . . .	60
3.16.2	Evaluation of a Polynomial . . . . .	61
3.16.3	Continued Fraction . . . . .	62
3.16.4	Reverse the Digits of a Word . . . . .	63
3.16.5	Sideways Addition . . . . .	64
3.16.6	Square Root of Y . . . . .	64
3.16.7	Matrix Multiplication . . . . .	66

Appendix to Chapter 3.

3.17	Memory . . . . .	68
3.18	Execution of Instructions . . . . .	69
3.19	Jump Instructions . . . . .	70
3.20	B-Instructions and Block Transfers of Data . . . . .	71
3.21	Basic Form of Instructions . . . . .	71
3.22	B-Line Instructions . . . . .	72
3.23	C-Jump Instructions . . . . .	72

Appendix to Chapter 3. (Continued)

3.24	Arithmetic Instructions.....	73
3.25	Input-Output Instructions.....	73
3.26	Register Arrangements.....	73
3.27	Instruction Code .....	74
3.28	Programming Examples .....	75

Chapter 4. BASIC CIRCUITS

4.1	Introduction .....	81
4.2	Asynchronous vs. Synchronous Operation .....	82
4.3	Direct-Coupling vs. AC-Coupling.....	84
4.4	Two-Level DC vs. Pulse Representation.....	84
4.5	Transistors vs. Tubes.....	86
4.6	Reliability and Feasibility of a Transistorized Asynchronous DC-Coupled Machine.....	89
4.7	Tolerances, Critical Levels and Discrimination Levels .....	90
4.8	The Last Moving Point Philosophy .....	94
4.9	Characteristic Times: Switching Time and Operation Time .....	97
4.10	Flow-Gating.....	100
4.11	Principles of Circuit Analysis and Synthesis .....	104
4.12	Component Specifications .....	106
4.13	General Remarks about the Next Section .....	109
4.14	NOT Circuit (Class 1).....	110
4.15	Level Restorer (Class 1) .....	112
4.16	OR Circuit (Class 1).....	113



Chapter 4 (Continued)

4.17	AND Circuit (Class 1) . . . . .	115
4.18	Flow-Gating Flipflop (Class 2) . . . . .	116
4.19	Schmitt Trigger (Class 3) . . . . .	117
4.20	Eccles Jordan (Class 3) . . . . .	118
4.21	EXCLUSIVE OR (Class 4) . . . . .	119
4.22	Driver (Class 4) . . . . .	120
4.23	C-Element (Class 4) . . . . .	122
4.24	Shifting Registers . . . . .	123
4.25	Summary and Closing Remarks . . . . .	127

Appendix to Chapter 4		129
-----------------------	--	-----

Chapter 5. ASYNCHRONOUS CIRCUITS

5.1	Introduction . . . . .	133
5.2	Speed and Complexity . . . . .	133
5.3	Design of Asynchronous Circuits . . . . .	134
5.4	Location of Malfunctions and their Repair . . . . .	136
5.5	Reliability . . . . .	136
5.6	Asynchronous Principles . . . . .	137
5.7	Method of Combining Blocks . . . . .	140
5.8	Method of Simulating Synchronous Circuits . . . . .	143
5.9	Method of Design from Change Charts . . . . .	147

Chapter 6. MEMORY

6.1	Introduction . . . . .	150
6.2	Comparison of Some Types of Memory . . . . .	151

Chapter 6 (Continued)

6.3	Word-Arrangement Memory . . . . .	154
6.4	Experimental Results . . . . .	159

Appendix to Chapter 6.

	Possibility of a Non-Destructive Readout . . . . .	164
--	--	-----

Chapter 7. INPUT, OUTPUT AND AUXILIARY STORAGE

7.1	Introduction . . . . .	165
7.2	Data Rate Characteristics of Scientific and Data Processing Computers . . . . .	167
7.3	Magnetic Tape Characteristics . . . . .	169
7.4	Magnetic Drum Storage . . . . .	170
7.5	Input-Output Requirements for Data Generated or Consumed by Human Users . . . . .	171
7.6	Other Modes of Data Generation or Consumption . . . . .	172
7.7	Other Aspects of Machine Balance . . . . .	172
7.8	Summary and Conclusions . . . . .	173

Chapter 8. ARITHMETIC UNIT

8.1	Introduction . . . . .	174
8.2	Separate Carry Storage in a Binary Arithmetic Unit . . . . .	180
8.3	Separate Carry Storage for an Arbitrary Radix . . . . .	182
8.4	Binary Subtraction . . . . .	184
8.5	Carry Assimilation . . . . .	185
8.6	Analysis of Overflow: Introduction . . . . .	187
8.7	Conventional Overflow Analysis . . . . .	188
8.8	Number Representation with Separate Carry Storage . . . . .	189
8.9	Method of Analysis for Overflow . . . . .	190

Chapter 8. (Continued)

8.10	Overflow Detection: Left Shift of One Digital Position. . . . .	192
8.11	Overflow Detection: Addition or Subtraction. . . . .	193
8.12	Overflow Analysis: Addition or Subtraction Followed by a Right Shift. . . . .	195
8.13	Overflow Analysis: Left Shift Followed by an Addition or Subtraction Such That the Result is in Range . . .	196
8.14	Multiplication: Introduction . . . . .	198
8.15	The Recoding of the Multiplier. . . . .	200
8.16	The Multiplication Right Shift: Introduction . . . . .	202
8.17	The Multiplication Right Shift: The Nature of Corrections if A and Q are of Opposite Sign . . . . .	202
8.18	The Multiplication Right Shift: Assimilation of Carries for Digits Transferred from A and C to Q. . .	204
8.19	Multiplication Overflow . . . . .	205
8.20	Division: Introduction . . . . .	205
8.21	Non-Restoring Division. . . . .	207
8.22	The Standardized Division . . . . .	209
8.23	Division Overflow . . . . .	211
8.24	Quaternary Operation. . . . .	211
8.25	Estimates of Operation Times and Hardware Requirements. . . . .	212
8.26	Interpretation of Speed and Hardware Estimates. . . . .	218
8.27	Floating Point Arithmetic . . . . .	219

NOTE: Figures for each chapter have number designations which give first the chapter number, then the point and then the figure number within the chapter.

Footnotes are numbered sequentially beginning with 1 in each chapter separately.

## SUMMARY AND CONCLUSIONS

On the basis of the work reported herein the Digital Computer Laboratory of the University of Illinois concludes that it is feasible to construct a very fast digital computer in which the transistor circuits developed in that Laboratory would be used.

The results of two design studies are discussed. One involves a minimum of buffer storage in the form of transistor registers and is outlined in the body of Chapter 3, while the other involves a moderate amount of buffer storage in the form of a small-capacity, high-speed, random-access buffer memory and is discussed in the appendix to Chapter 3. The former design is emphasized because it is felt that its equipment requirements can be presently met.

In it, two controls are used, arithmetic control and advanced control, as well as buffer storage for instructions and operands, and by such means various units of the computer are kept in simultaneous operation. For example, B-modifications, memory accesses and complicated arithmetic operations, such as a double-length, add-product instruction, may be performed concurrently. Short but powerful inner loops may be stored outside the memory and acted upon by the control. Many of the gains in speed in the control and arithmetic unit are dependent upon asynchronous operation of these units.

The relative speed of the proposed computer compared to that of existing machines depends upon the problem being solved. For problems dominated by arithmetic operations it is estimated that the proposed computer will be 100 to 200 times faster than computers such as Illiac. For problems dominated by logical and combinatorial operations, this factor of gain in speed will be at least 50.

The proposed computer has a random-access word-arrangement memory of 8192 words of 52 bits each with an access time of 1.5  $\mu$ s.

The arithmetic unit is designed so that the digits of a multiplier are sensed and acted upon in such a way that the use of the adder is reduced. Furthermore, "carry registers" are used in this unit, and carries are assimilated only when necessary. It is expected that the proposed computer will have an average multiplication time between 3.5 and 4  $\mu$ s, addition times (with assimilation) of .3  $\mu$ s, and division times of 7 to 20  $\mu$ s.

The computer, aside from input-output facilities, will contain approximately 15,400 transistors, 34,000 diodes, and 42,000 resistors. The transistors are expected to be Western Electric transistors, type GF-45011.

The basic circuits built from these transistors have operation times of from 5 to 40  $\cdot 10^{-9}$  second depending upon the circuit.

## SUMMARY OF PROPOSED MACHINE'S SPECIFICATIONS

This report covers many aspects of high-speed computing machines. As the study shows, more than one decision concerning machine organization or design is acceptable, and the best decision concerning each question must remain unanswered until more detailed plans are drawn up. In this section, a set of specifications is written down which serves to show in a concise way a definite machine based upon the best estimates now given in the text.

The reliability of these estimates varies with different parts of the machine. Some parts may be greatly influenced by future developments and may be realized by techniques different from those now contemplated and, hence, will involve a number of active elements quite different from the amounts listed below. The operation-time estimates are based upon small-scale experimental studies. Although factors affecting these times are taken into account insofar as possible in the extrapolation to the large-scale prototype, it is necessary to recognize that unforeseen characteristics of the prototype may require revision of the time estimates.

A tabular summary is presented where possible.

### 1. Organization

In accordance with the historical division of computing machines the machine may be considered to have: an arithmetic unit, a memory, a control and an input-output unit or units. However, the rather long memory access time ( $1.5 \mu\text{s}$ ) compared to the short addition time ( $0.32 \mu\text{s}$ ) requires temporary storage facilities intimately connected with the memory and the arithmetic unit. These will be called fast-access registers or fast-access memory. The interconnections are shown in block diagram form.

Reading from the memory is carried out at the same time the arithmetic unit is in operation. Thus, the memory may load order register  $O_1$  and operand register X during the time the arithmetic unit is performing a multiplication. Except for the brief period when the arithmetic unit is

receiving or sending data to the fast-access registers, the arithmetic unit operates separately from the memory reading and writing operations. A part of the control called the advanced control arranges to keep registers  $O_1$ ,  $O_2$ , X, Y, and Z filled with data which the arithmetic unit is expected to need. Combinations of orders such as multiply and add help reduce accesses to the main memory.

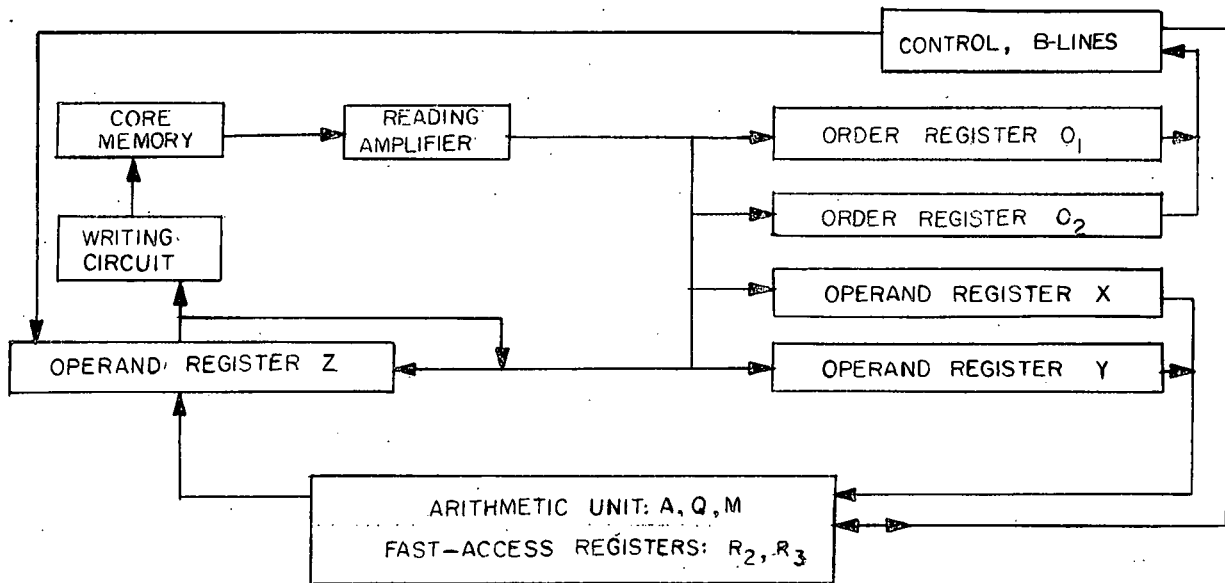


Figure 1. Block Diagram

## 2. Fast-Access Memory

Three operand registers (X, Y and Z)	Non-shifting, 52 bits each
Two instruction registers ( $O_1$ and $O_2$ )	Non-shifting, 52 bits each
Twelve B-registers or counters	Non-shifting, 13 bits each
Word length	52 bits

Each word is 1) 52-bit fixed point number  
or 2) 42-bit floating point number, 10-bit exponent  
or 3) four 13-bit control groups.

A short instruction has 7 bits for function, 4 bits for local fast register addressing, and 2 bits which are used for function or address bits.

A long instruction is a short instruction plus a 13-bit address.

Access-time, fast memory to arithmetic unit	0.05 $\mu$ s
Number of orders outlined in text	100

#### Equipment for Fast-Access Memory

Number of transistors	2000 $\pm$ 25%
Number of diodes	8000 $\pm$ 25%
Number of resistors	6300 $\pm$ 30%

### 3. Arithmetic Unit

The arithmetic unit is characterized by an adder with carry completion sensing, separate carry register so that carries are not assimilated except when necessary and a multiplication scheme which senses two digits and shifts two places for each partial step. Recoding of the multiplier digits reduces the number of uses of the adder during multiplication to  $52/3 = 17$  on the average.

#### Numbers

Number system	Binary, parallel
Word length	52 bits
Addition	Separate carry register, carry completion sensing
Number representation	2's complement, fixed and floating point
Electrical bit representation	DC voltage
Multiplication	Reversed ternary multiplier recoding
Multiplication shifts	Two at a time
Division	Conventional non-restoring with remainder (type 1) or non-restoring with standardization of partial remainders (type 2)
Overflow detection	Available on all orders

## Registers

Accumulator and quotient	Shifting registers, 52 bits each
Number register (M)	Shifting register, 52 bits
Operand registers $R_2$ and $R_3$	Non-shifting, 52 bits each

## Fixed Point Operation Times without Access

Addition without carry assimilation	0.19 $\mu$ s $\pm$ 10%
Addition with average carry assimilation	0.32 $\mu$ s $\pm$ 10%
Multiplication without assimilation	3.8 $\mu$ s average, $\pm$ 10% 4.8 $\mu$ s maximum, $\pm$ 10%
Division, type 1 without assimilation	10-20 $\mu$ s $\pm$ 10%
Division, type 2 without assimilation	7-14 $\mu$ s $\pm$ 10%
Average carry assimilation	0.13 $\mu$ s $\pm$ 10%
Maximum carry assimilation	0.67 $\mu$ s $\pm$ 10%

## Equipment Cost for Arithmetic Unit

Number of transistors	6,800 $\pm$ 25%
Number of diodes	6,300 $\pm$ 25%
Number of resistors	11,500 $\pm$ 30%

## 4. Control

The machine has an arithmetic control which forms instructions from  $O_1$ ,  $O_2$ , and the B-registers and properly sequences the arithmetic operations. The advanced control looks at instructions just ahead but not yet executed by the arithmetic unit and places additional data from the memory in  $O_1$ ,  $O_2$ , X or Y as necessary. Of course on conditional control transfer orders or other orders dependent upon calculations in process, the advanced control must wait for its information and no advantage is gained by the fast registers. Advanced control places operands in X, Y from addresses contained in  $O_1$  or  $O_2$  and then provides appropriate addresses to the arithmetic control to indicate that the operands are in X and Y.



A memory control to properly sequence the reading and writing from the memory and an input-output control are necessary.

#### Equipment for Controls

	<u>No. Transistors</u>	<u>No. Diodes</u>	<u>No. Resistors</u>
Arithmetic control	1200 $\pm$ 30%	4000 $\pm$ 30%	5000 $\pm$ 50%
Advanced Control	2400 $\pm$ 30%	8000 $\pm$ 30%	10,000 $\pm$ 50%
Memory control	1000 $\pm$ 40%	3500 $\pm$ 40%	4000 $\pm$ 60%
Input-output control	1000 $\pm$ 40%	3500 $\pm$ 40%	4000 $\pm$ 60%
Totals	5600 $\pm$ 34%	19,000 $\pm$ 34%	23,000 $\pm$ 54%

#### 5. Memory

A core memory using the word-arrangement (or word addressing scheme) is favored because this produces the shortest access times with the conventional cores known to be available. In this addressing scheme, there is a smaller drive current restriction than in the coincident current system and hence the core turn-over time may be shortened.

#### Characteristics

Main internal random-access unit	8192 words, 52 bits each
Random-access time	1.5 $\mu$ s
Type of memory core	General Ceramics S4, size F-394
Type of switch core	General Ceramics S5, size F-394
Type of addressing	By words

#### Equipment for Memory

Number of large vacuum tubes	300 $\pm$ 20%
Number of small vacuum tubes	1000 $\pm$ 20%
Number of transistors	1000 $\pm$ 20%
Number of cores (two per bit stored)	851,968
Number of cores for switches	212,992

Auxiliary memories are listed under input-output.

## 6. Basic Circuits

The basic circuits use diffused-base transistors with provision to prevent the transistors from going into saturation at any time. This produces the shortest transistor switch time. The circuits are more complicated than the simplest circuits and require a larger number of parts. The number of tolerance problems is necessarily greater, and these are being checked with the Illiac. Diodes, of a particularly fast type, are also used as switching elements.

Circuits	Direct-coupled asynchronous
Switching elements	Western Electric GF-45011 transistors, Qutronics Q5-250 or Q10-600 diodes or equivalent.

	<u>No. of Transistors</u>	<u>No. of Diodes</u>	<u>No. of Resistors</u>	<u>Operation time μs</u>
Flipflop	4	11	15	30
NOT	2	4	5	15
AND	2	0	3	5
OR	2	2	3	5
Double gate	4.3	1.2	8	--
Half-Adder	8	6	14	25
Complement Circuit	6	2	9	10

Although these circuits, and the logical developments giving rise to the operation times listed under the arithmetic unit, are direct-coupled, they are not under all conditions asynchronous. Chapter 5 is devoted to a discussion of this subject and a rigorous definition for asynchronism is presented. Although the more rigorous definition requires more care in application and more parts in most circuits, there are good prospects that it can be done and is worthwhile. Succeeding developments may adopt this new definition whenever possible.

## 7. Input-Output

For balance only the fastest possible types of input-output would be appropriate. This includes only magnetic drums and the fastest magnetic tape units. These units are greatly affected by commercial developments and no specific items are listed.

### Equipment for Off-Line Operation

- Punched card equipment
- Punched paper tape equipment
- Card and paper tape to magnetic tape converters
- Fast printer

### Equipment for On-Line Operation

- Magnetic tape units with 30 or more channels per tape
- Magnetic drum, 50,000 words
- Cathode ray tube display equipment
- Switching elements need not have an operation time shorter than 1  $\mu$ s.

## CHAPTER 1

### PROBLEMS REQUIRING FASTER COMPUTERS

#### 1.1 Introduction

Since the publication in 1946 of the fundamental report by A. W. Burks, H. H. Goldstine and J. von Neumann<sup>(1)</sup> a class of machines based directly on the ideas of this report have been built and put into operation. Other machines influenced in various degrees by this report have also been put into use. The experience obtained with these machines enables one to begin to discern the limitations of present-day computers in carrying out presently-formulated computational problems in various areas of applied mathematics, the physical sciences, and engineering as well as those computational problems arising in these and other fields whose solutions do not yet have a precise formulation in terms of arithmetic algorithms. Problems involving various types of partial or ordinary differential equations may be considered to be of the first class whereas problems involving a large degree of combinatorics and not necessarily arithmetic manipulation of various quantities may be said to belong to the second class.

It is the purpose of this chapter to discuss some problems of each type referred to above and to attempt to characterize them in terms of the requirements they impose on the memory, arithmetic unit and control of a computer. We shall be mainly concerned with the size and speed the first two of these organs must have.

In the analysis given below it will be assumed that for some problems, for example those which involve the solution of ordinary or partial differential equations, the time spent by present-day computers in obtaining the solution may be estimated by the formula:

$$T = F * \text{Multiplication time} * \text{Number of Multiplications}$$

where  $F$  is a number between one and ten, the multiplication time is the time

---

1. "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," A. W. Burks, H. H. Goldstine, John von Neumann, Institute for Advanced Study (Princeton, N. J.) June 1946.

necessary for the arithmetic unit to multiply two numbers (exclusive of the time required to obtain these numbers from the memory), and the number of multiplications is the total number occurring in the problem.

Actually T, the time spent by a computer in solving a problem, is composed of four parts:

- (1) the input-output time: the time spent in transferring the problem into and out of the machine,
- (2) the instruction access time: the time spent in transferring instructions from the memory to the control,
- (3) the operand access time: the time spent in transferring operands and partial results from the memory to the arithmetic unit and the time spent in the reverse process,
- (4) the arithmetic time: the time spent by the arithmetic unit in doing useful and necessary arithmetic.

For present-day computers these times are additive, although logically they need not be. Indeed, one possibility that many designers are exploring is the possibility of overlapping these times by building machines capable of performing a number of these tasks simultaneously, that is, in parallel.

As was pointed out by H. H. Goldstine,<sup>(2)</sup> the formula for T given above follows from the assumption that on the average, each multiplication can be imbedded in a sequence of (A+1) instructions consisting of one multiplication and A non-multiplicative instructions, each of which takes a time L to execute. The time to execute these instructions is then

$$t = M + AL + (A+1) (a_i + a_o) = FM$$

where M is the multiplication time,  $a_i$  and  $a_o$  are the instruction and operand access-times respectively. Let

$$F = 1 + \frac{AL}{M} + \frac{(A+1) (a_i + a_o)}{M}$$

2. H. H. Goldstine: "Systematics of Automatic Electronic Computers," Proceedings of the Darmstadt Colloquium, October 1955.

We then have

$$T = Nt = FMN$$

where  $N$  is the number of multiplications. The estimates on the size of  $F$  are obtained by estimating the various quantities entering on the right-hand side of the equation for  $F$ .

Thus for a machine such as Illiac,

$$a_i \approx \frac{1}{2} a_0 \approx .025M$$

$$\frac{FL}{M} = .08.$$

Hence if  $A = 11$  we have

$$F \approx 2.8 .$$

## 1.2 Hyperbolic and Parabolic Partial Differential Equations

We begin the listing of problems which are beyond the capabilities of present-day computers with some time-dependent problems in hydrodynamics. These problems involve the solution of a system of hyperbolic partial differential equations in which the independent variables are one-, two- or three-space variables and time. If the integration is done in terms of Lagrange coordinates, the number of dependent variables is  $2m + 1$  where  $m$  is the number of spatial dimensions, since the Eulerian coordinates must be computed, the velocity field must be determined, and one thermodynamic variable in addition to the density must be calculated. Each of these  $2m + 1$  independent quantities must be evaluated as functions of the time.

The calculation can be arranged so that the value of any dependent variable at time  $t + \Delta t$  at certain values of the spatial coordinates is determined in terms of the values of some or all the dependent variables at time  $t$ . However, only the values of the dependent variables at time  $t$  at

spatial points in a region near the point of interest are needed in this calculation; that is, not all the data at time  $t$  is needed to calculate the values of a dependent variable at a single spatial point at the time  $t + \Delta t$ . This observation is important for the consideration of the memory requirements of a computer.

The size of the time-step (i.e. the size of  $\Delta t$ ) must be related to the size of the spatial mesh. Thus in a one-dimensional problem in which we evaluate the dependent variable at mesh points

$$X = X_0 + j\Delta X \quad (j \text{ an integer})$$

we must have

$$C \Delta t \leq \Delta X$$

where  $C$  is the (variable) sound velocity.

If the problem is such that one would expect spatial variations in the dependent variables similar to those in the function

$$\sin 2\pi kX,$$

then one should have

$$2\pi k \Delta X \leq \frac{\pi}{2} .$$

It is not unreasonable to require that the extent of each spatial variable be divided into between 10 and 100 mesh points; that is,

$$\frac{1}{10} \geq \Delta X \geq \frac{1}{100} .$$

The amount of data which has to be stored at a given time is then

$$30 \leq D \leq 7 \cdot 10^6 ,$$

where the lower limit holds for a one-dimensional problem with a 10-point spatial mesh and the upper limit holds for a three-dimensional problem with a cubical mesh having 100 points to a side.

The number of multiplications involved in the calculation of the values of all the dependent variables at a mesh point depends on the number of dimensions. It may be assumed to be 30 for a one-dimensional problem, 50 for a two-dimensional one and 70 for a three-dimensional one. Hence,  $N$ , the total number of multiplications per time-step, will be

$$9 \cdot 10^2 \leq N \leq 4.9 \cdot 10^8 .$$

The computation time in seconds to perform the required calculations for one time-step with an existing computer with a multiplication time of  $5 \cdot 10^{-4}$  seconds and  $F = 2.5$  is then

$$1.1 \leq T \leq 6 \cdot 10^5 .$$

If the computation is to be done for a number of time-steps  $M$  sufficient for signals to traverse the region under consideration  $K$  times, that is if

$$CM \Delta t \approx Km \Delta x$$

where  $m$  is the number of mesh points in each dimension, since  $C \Delta t \approx \Delta x$ , we would have

$$M \approx Km .$$

Hence for a single problem the computation time in seconds would be

$$11K \leq t_c \leq 6K \cdot 10^7 .$$

Quite often numerical surveys are made in which calculations are done again and again, each time with various parameters assigned different values.

It is clear that for moderate values of  $K$  the time required for a single calculation is excessive in the three-dimensional case. There are, of course, many one-dimensional problems in which the number of mesh points is considerably larger than 10 and  $K$  is of the order of 30. Thus even in a one-dimensional problem it is quite possible to have

$$t_c \approx 25 \text{ hours}$$

for a single problem with an existing machine by taking  $m \approx 170$  and  $K = 30$ .



In two- and three-dimensional hydrodynamical problems done by use of Lagrangian coordinates there arises a difficulty in the calculation due to the fact that "particles" of the fluid which were originally neighbors do not remain neighbors. This implies that after some time-steps the calculation summarized above has to be interrupted and a new Lagrangian mesh has to be formed. If this process is to be done by the computer, the time estimates given above have to be increased. Moreover existing computers are not very efficient in performing combinatorial manipulations required in setting up the new Lagrangian net.

For parabolic differential equations the situation is much the same as that for hyperbolic ones with one very important difference. Instead of having the time-step linearly related to the spatial mesh size, we have

$$\Delta t \approx k(\Delta X)^2 .$$

This introduces another power of the number of mesh points in the expression for the number of multiplications required to integrate the equations for a given interval of time.

In summary it may be said that there are many problems involving hyperbolic and parabolic partial differential equations in which the total number of multiplications involved is between  $10^7$  and  $10^{11}$  and that in these problems 50,000 to 100,000 words of data are used. However the calculation may be so organized that relatively small amounts of this data are required at once. Therefore, it is not clear that a random access memory sufficient to hold 100,000 words of data is required.

### 1.3 Elliptic Partial Differential Equations and Linear Equations

The solution of elliptic partial differential equations may be reduced to solving a set of linear equations:

$$Ax = b$$

for the  $n$  components of  $x$  where  $A$  is a given  $n \times n$  matrix and the  $n$  components

of  $b$  are given. When such a system of linear equations arises from a partial differential equation, the dimensionality of the problem is related to the number of mesh points used in approximating the differential operators by difference operators. Values of  $n = 400$  or greater are not uncommon.

The matrices  $A$  which arise in these problems usually have the property that the non-zero elements in any row of the matrix are relatively few in number. Moreover the location of these non-zero elements relative to the diagonal element is the same for each row. However the values of these elements will not be the same unless the coefficients of the differential equation are constants. For non-linear differential equations, these coefficients are not the same on successive iterations.

Because of these properties of the matrix  $A$ , the usual methods of solving the linear equations involve iteration processes. If we assume that 20 iterations of a matrix are required and that there are 10 non-zero elements in a given row of the  $400 \times 400$  matrix we find that there are 80,000 multiplications required to find the solution of the problem once the coefficients of the matrix are known. In some problems these quantities may depend on their position in the matrix and on the approximate values of the  $x$ 's. Therefore on each iteration the coefficients of the matrix  $A$  have to be evaluated. If the evaluation of each non-zero term involves 10 multiplications, we must perform  $400 \cdot 10 \cdot 10 = 40,000$  multiplications to evaluate the matrix on each iteration and hence  $4.4 \cdot 10^4$  multiplications per iteration or  $8.8 \cdot 10^5$  multiplications per 20 iterations.

Hence for a non-linear elliptic problem the number of multiplications is comparable to the number involved in hyperbolic and parabolic differential equations. However the amount of data that need be stored and manipulated may be much smaller than in the latter cases.

#### 1.4 Sorting and Meshing Problems

In scientific computations and in other problems one is quite often confronted with the need of comparing sets of quantities represented by numbers which are either stored in the machine or on some medium which may be read by the machine. One way of handling such problems is to order the data in some

fashion. Because this problem is very common it is worthwhile discussing it to see how time-consuming it can become and what factors affect the time involved in completing it.

Von Neumann and Goldstine have described a method for reducing a set of numbers (or a set of complexes of numbers) to a set ordered by size. The method consists of an application of a meshing process: two sets of ordered numbers are meshed as follows: let the first set of numbers be  $X_1, \dots, X_n$  and the other set be  $Y_1, \dots, Y_m$ . Then we form a single set

$$Z_1, Z_2, \dots, Z_{n+m}$$

where

$$Z_k = X_i \text{ or } Y_j$$

where  $X_i$  is the first of the remaining  $X$ 's

$Y_j$  is the first of the remaining  $Y$ 's, and  $k = i + j - 1$ .

The choice as to whether an  $X$  or  $Y$  is chosen depends on whether

$$X_i \geq Y_j$$

or

$$X_i < Y_j .$$

Sorting is accomplished by successive meshing. Thus the numbers  $X_1, \dots, X_n$  are first meshed, as if they were groups consisting of single elements, into  $n/2$  groups of two elements:  $X_1, X_2; X_3, X_4; \dots; X_{n-1}, X_n$ . These are then meshed into  $n/4$  groups of four elements. This process continues until there are  $n/2^k$  groups of  $2^k$  elements each. If  $n$  is a power of 2,  $n = 2^m$  the process stops when  $k = m$ .

The arithmetic being done in this problem consists of approximately

$$n \log_2 n$$

comparisons, for on every meshing operation there is a comparison for the

determination of each element of the new set. However, the time for completing the sorting cannot be estimated accurately by multiplying this quantity by the time for doing the arithmetical comparison. One must take account of the time required for "red tape" operations (the time required to locate and acquire the proper numbers for the comparison), and the time to determine where the winner of the comparison test is to be placed, and to store this number in the memory.

Thus, the time to complete the sorting will be of the form

$$Kn \log_2 n$$

where  $K$  is a time usually much greater than the time to perform a simple addition in a machine.

It is expected that in quite modest problems  $n \approx 1000 \approx 2^{10}$  and, hence, the time to sort is  $Kn \log_2 n \approx K \cdot 10^4$ . As a rough estimate we may take  $K$  to be 10 times the addition time of the computer and, hence,  $K = 4 \cdot 10^{-4}$  seconds for a machine such as Illiac. Thus, the sorting of 1000 pieces of data would take 4 seconds. Although this time itself may appear small, it may have to be repeated a very large number of times and may influence greatly the feasibility of doing a particular problem on a computer. If  $n \approx 10^6$ , the time estimate would be multiplied by  $2 \cdot 10^2$  and would require 8,000 seconds or 2 hours to carry out the task.

## CHAPTER 2

### REQUIREMENTS ON A VERY FAST COMPUTER

#### 2.1 Memory Capacity Requirements

In the discussion of problems described in sections 1.2 and 1.3 we have seen that the total number of multiplications may vary between  $10^7$  and  $10^{11}$  and that in some of these problems 50,000 or 100,000 words of data are used. However, not all of this data is needed at any one stage of the computation and therefore need not necessarily be stored entirely in the high-speed random-access memory. Indeed it is the purpose of this section to show that the use of a lower speed non-random-access memory (a "backup" memory) in conjunction with a faster but smaller memory does not involve a great percentage increase of computing time for those problems satisfying the following property:

The problem is such that if  $N$  words of data are in the high-speed memory we may then calculate  $N-k$  new words which replace the same number of words previously held. Thus in one sweep through the memory  $N-k$  words are calculated and these may be stored in positions previously occupied by other words. If  $N-k$  is as large as required by the problem, this process is repeated many times and the total time for the calculation is

$$(N-k) \cdot n \cdot F \cdot M \cdot T = \tau_1 T$$

where

$n$  = the number of multiplications for each of the  $(N-k)$  words

$T$  = the number of times the memory is swept through,

If, however,  $N-k$  is not as large as required we must add to this the time necessary to load and unload the memory. Let us say that the time necessary to unload the high-speed memory is made up of two parts: (a) a time to get access to another memory,  $\alpha$ , and (b) a time to read  $N-k$  words into this memory,  $(N-k)\beta$ .

Thus unloading time is

$$\alpha + (N-k)\beta.$$

Assuming that reading and writing in the latter memory take equal times and that we have to read N words from it in order to properly load the high-speed memory, we have as our loading time

$$\alpha + N\beta.$$

Hence the total loading and unloading time is

$$2(\alpha + (N-k)\beta) + k\beta.$$

The ratio of this time to the computation time per load in the memory is

$$R = \frac{2(\alpha + (N-k)\beta) + k\beta}{(N-k)nFM}$$

$$R = \frac{2}{nFM} \left( \frac{\alpha + \beta \frac{k}{2}}{N - k} + \beta \right).$$

The total computation time is now

$$(1 + R)\tau_1 T.$$

Hence the ratio R is a measure of the cost of using a pair of memories instead of a single memory. We may write

$$\alpha/M = \alpha_1$$

$$\beta/M = \beta_1.$$

Then

$$R = \frac{2}{nF} \left( \frac{\alpha_1 + \frac{\beta_1 k}{2}}{N - k} + \beta_1 \right) = \frac{2}{nF} \left( \beta_1 + \frac{\frac{\alpha_1}{k} + \frac{\beta_1}{2}}{\frac{N}{k} - 1} \right).$$

In this formula  $n$ ,  $N$ , and  $k$  depend on the problem being solved and  $\alpha_1$  and  $\beta_1$  depend on the equipment being used. For a given problem we may estimate the size of high-speed memory  $N$  so that  $R$  is the order of 10%. The important thing to note is that the rate of change of  $R$  with  $N$  is a slowly-varying function beyond the value of 10%.

Goldstine<sup>(1)</sup> has shown that for a problem in hydrodynamics with time and two spatial variables for which 50,000 words of storage are needed, a core memory with room for 1750 words of data requires less than 10% of computing time for consulting the secondary memory when

$$\alpha_1 = 1700 \text{ or less}$$

$$\beta_1 = .8 \text{ or less}$$

$$k = 500$$

$$n = 20$$

$$F = 2.5$$

However, Goldstine did not consider the complications ensuing when slip-streams occur in the computations.

Suppose now we examine the question as to whether a random-access memory of 30,000-word capacity is needed.

In two- and three-dimensional hydrodynamics problems in which the quantities are time-dependent, the amount of data which has to be processed in one time-step will exceed this capacity by a factor which is at least between one and two and may be five. Therefore a backup memory will be needed in any case. We are thus dealing with the factor  $R$  and the question is what is the value of  $R$  for  $N \approx 30,000$  and say  $N \approx 2,000$ .

---

1. H. H. Goldstine: "Systematics of Automatic Electronic Computers,"  
Proceedings of the Darmstadt Colloquium, October, 1955.

Suppose we have a drum in which

$$\alpha = 17000 \mu s = \text{access-time to a word,}$$

$$\beta = 8 \mu s = \text{read-time for a word.}$$

If then the multiplication time is assumed to be  $5 \mu s$ ,

$$\alpha_1 = 3400.$$

$$\beta_1 = 1.6$$

$$R = \frac{2}{nF} \left( 1.6 + \frac{\frac{3400}{k} + .8}{\frac{N}{k} = 1} \right)$$

For  $k = 500$

$$R = \frac{2}{nF} \left( 1.6 + \frac{7.6}{\frac{N}{500} = 1} \right)$$

$$R_{30000} = \frac{2}{nF} (1.7) = .068$$

$$R_{2000} = \frac{2}{nF} (3.6) = .144$$

where  $\frac{2}{nF} = .04$ , ( $F = 2.5$ ,  $n = 20$ ).

Hence the ratio of the times to do the same problem with a memory of 32,768 words and one with 4768 words (2768 words of code being used) is

$$\frac{1.14}{1.07} = 1.07.$$

A 7% increase in time is gained by an almost eight-fold increase in size of memory.



For  $k = 1000$  we have

$$R = \frac{2}{nF} \left( 1.6 + \frac{4.2}{\frac{N}{1000} - 1} \right)$$

$$R_{30000} = \frac{2}{nF} \left( 1.6 + \frac{4.2}{3} \right) = \frac{2}{nF} (1.74) = .07$$

$$R_{2000} = \frac{2}{nF} \left( 1.6 + \frac{4.2}{1} \right) = \frac{2}{nF} (5.8) = .23$$

Here then we have a similar phenomenon. The ratio in times is

$$\frac{1.23}{1.07} = 1.15$$

a 15 speed-up being paid for by an almost eight-fold increase in size of memory. The doubling of the factor  $k$ , the amount of dead-weight data being carried in the transfers, is responsible for the change in time.

## 2.2 The Class of Problems that can Use Backup Memories

It is clear from the discussion of parabolic and hyperbolic differential equations that these can be formulated so that they satisfy the assumption made in the previous section. That is, if  $N$  words of data are in the high-speed memory we may then calculate  $N-k$  new words which replace the same number of words previously held. Hence a backup memory can be used to good advantage on such problems.

It may also be used on algebraic problems arising in the solution of elliptic partial differential equations or integral equations. Such problems involve the solution of linear algebraic equations of the form

$$Ax - b = 0 \tag{2.1}$$

$$Ax - \lambda x = 0. \tag{2.2}$$

These problems may either be solved by iterative methods or by direct methods. The former are useful for large matrices in which many elements vanish. In such cases the  $(n \times n)$  matrix  $A$  which may have  $n^2$  elements in reality is represented by many fewer elements and may be stored in a relatively modest high-speed memory. Thus for such cases where the matrix has relatively few non-zero elements and where an iterative method is used, that is the matrix  $A$  is not destroyed in the process of solution, there is no apparent need for a very high-capacity high-speed random-access memory.

Even when direct methods are used in solving linear equations such as equation (2.1) the price paid for using a modest-capacity low-access-time memory and a backup store in contrast to a high-capacity low-access time memory need not be excessive. This statement is justified as follows:

The direct method for solving equations (2.1) is the elimination method. Let us suppose that the matrix we are dealing with has  $n$  rows and  $n$  columns. It is to be reduced to a triangular matrix with the elements below the main diagonal zero by multiplying the rows by suitable numbers and subtracting them from subsequent ones.

We denote by  $p$  the capacity of the high-speed memory (for numerical storage) and write

$$n = pm$$

where we assume that  $m$  is an integer.

We propose to deal with  $m$  rows of the matrix  $A$  simultaneously. The first  $m$  rows of the  $(n+1) \times n$  augmented matrix are read into the machine and as much elimination as can be done is done. The reduced data is then written on the drum. The next  $m$  rows of the original matrix are written into the machine. The first  $m$  rows are called back from the drum, used to reduce the second  $m$  rows, these partially reduced rows are further reduced into their final form and then written onto the drum. The third set of  $m$  rows is then called in and similar processes are applied. This continues until the whole matrix has been triangularized.

In this process the triangularized augmented matrix is written on the drum, that is

$$W = \frac{n(n+3)}{2}$$

numbers are written onto the drum and

$$R = (p-1) \frac{m}{2} (2n - m + 3) + (p-2) \frac{m}{2} (2n + 3 - 3m) + \dots \frac{m}{2} [2n + 3 - (2p-3)m]$$

numbers are read from the drum where

$$p = \frac{n}{m}.$$

The time taken to perform W and R is the extra cost in time paid for by not having enough capacity in the high-speed memory.

Now

$$R = \frac{m}{2} (2n+3) \sum_{r=1}^{p-1} (p-r) - \frac{m^2}{2} \sum_{r=1}^{p-1} (p-r)(2r-1) = \frac{m}{2} (2n+3) \frac{p(p-1)}{2} - \frac{m^2}{2} \frac{p(p-1)(2p-1)}{6}$$

$$R = \frac{n(n-m)}{4m} \left[ 2n+3 - \frac{(2n-m)}{3} \right]$$

$$R = \frac{n(n-m)}{4m} \left( \frac{4}{3}n + \frac{9+m}{3} \right) \approx \frac{n^3}{3m}.$$

We may neglect W in comparison to R.

The number of multiplications involved in the triangularization of a matrix is to the same approximation

$$\frac{n^3}{3}.$$

Hence the ratio of the reading from the drum time to the computation time is

$$\frac{1}{mF} \beta_1$$

where we have assumed that we read in such large batches from the drum that the factor involving  $\alpha_1$  may be neglected.

If  $m$  is large enough this quantity may be made small so that the cost of not having a sufficiently high-capacity memory is not excessive even in this problem.

Even for sorting problems the use of a relatively small high-speed memory and a slower backup memory is not too costly in time. Let us suppose that we have a machine and a fast-access memory which is capable of sorting  $N_0$  numbers. Let us see how this may be used to sort  $N$  numbers which are originally stored on a drum. Let us write

$$N = rN_0$$

and assume that  $r$  is an integer.

We may sort the  $N$  numbers into  $r$  sets of  $N_0$  numbers and involved in this is a reading of  $N$  numbers from the drum and a writing of these. This reading and writing can be done in blocks of  $N_0$  numbers. The question arises as to what to do next. Now mesh the following numbers: A set of size  $K$  from the first  $N_0$ , a set of the same size from second  $N_0$ .... where

$$rK = N_0 .$$

In the process of sorting these numbers into  $N_0$  ordered numbers, one set of  $K$  numbers will be exhausted. Print out the first  $K$  sorted numbers and fill the space previously occupied by the exhausted set of  $K$  numbers with the next  $K$  numbers from the exhausted set. Thus the extra time involved in having a

backup store is the time required for two readings and writings of the data.  
 The percentage increase in time is

$$\frac{2RN + 2WN}{KN \log_2 N} = \frac{2(R+W)}{K \log_2 N}$$

### 2.3 Speed Requirements on the Memory

In section 1.1 it was stated that the quantity

$$F = 1 + A \frac{L}{M} + (A+1) \frac{a_i}{M} + (A+1) \frac{a_0}{M}$$

was one of the quantities determining the time for completing a computation and it was shown that for a machine such as Illiac this quantity was about 2.8 when  $A = 11$ .

With the present progress in circuitry it seems possible to design a computer with

$$M = 4 \mu s$$

$$L = .3 \mu s$$

$$a_0 = 1.5 \mu s$$

$$a_i = .75 \mu s \quad (\text{when orders are stored in pairs}).$$

Hence

$$\frac{L}{M} = .075$$

$$\frac{a_0}{M} = .375$$

$$\frac{a_i}{M} = .188$$

The corresponding figures for the Illiac are .08, .05 and .025 respectively. Hence we see that a machine with the same logical design as Illiac but using the latest circuit techniques would have a factor of

$$F = 1.56 + A(.64) = 8.6$$

for  $A = 11$ . The corresponding figure for Illiac is

$$F = 1.08 + A(.16) = 2.8,$$

that is, for the new machine the factor  $F$  would be three times larger. Since the time to do a computation is measured by  $F$  times the multiplication time the increase in the factor  $F$  vitiates in part the speed-up in multiplication time that can be achieved. If the multiplication time is decreased by a factor of 200, the total time to do a problem is decreased by a factor of only 67. This is because the terms  $\frac{a_0}{M}$  and  $\frac{a_i}{M}$  are so large for the present machine.

If  $F$  is to be the same for a machine with a multiply time of  $4\mu s$  as for Illiac and if the logic of the two machines were to be the same we would have to have a word access-time from the memory of about  $.15\mu s$ , that is, ten times faster than seems possible with present techniques involving core memories.

It is the purpose of the next chapter to discuss methods of reorganizing the computer and adding extra equipment so that a reasonable fraction of this factor of three in loss of speed is recaptured. The computers there proposed manage to recoup this factor on some problems. They are no slower than sequential machines on other problems and therefore at the worst would be at least sixty times as fast as Illiac.

#### 2.4 Numerical Requirements on Word Length

The word length for a given machine is influenced by arithmetic and organizational considerations. Mainly because of the former it is proposed to use a word length of 52 bits. There are two major arguments for this based on fixed point and floating point arithmetic respectively.

If a floating point number is represented as an exponent and fractional part packed into one word, then the word length should be sufficient to maintain accuracy in floating point arithmetic. A 42-bit fractional part seems to be

just adequate for this purpose. It is therefore proposed to use 10 bits as an exponent for floating point numbers and 42 bits as the fractional part. We shall discuss floating point arithmetic further in subsequent sections of this chapter.

Even with fixed point arithmetic there is an argument for increasing the word length for the representation of numbers over that used in computers such as Illiac. In essence this is the following: With faster computers larger problems are done in which more computations are carried out and in which round-off plays a larger role. Hence more guard digits are needed. An estimate of how many additional digits are needed may be obtained as follows:

Von Neumann and Goldstine<sup>(2)</sup> have shown that in inverting a matrix the limiting value of  $n$ , the size of the matrix, for which the results are accurate, is related to the number base  $\beta$  and the number of places carried,  $s$ , by the inequality

$$n < .15\beta^{s/4}.$$

Further the amount of time needed to invert a matrix is proportional to the number of multiplications, that is,  $n^3$ .

Hence for a machine about 128 times faster than Illiac, we should be able to invert a matrix of size  $2^{7/3}n$  in the same time as that with which Illiac deals with an  $n^{\text{th}}$  order matrix. If the results for this larger matrix are to be as accurate as those obtained from the Illiac on the smaller matrix we must have the number of bits carried on the fast machine,  $s'$ , given by

$$s' = s + 4 \left(\frac{7}{3}\right) \approx 50.$$

The problem seems to be the most stringent algebraic one for determining the number of bits needed.

2. H. H. Goldstine and J. von Neumann, "Numerical Inverting of Matrices of High Order," Bulletin American Mathematical Society, vol. 53, no. 11, pp. 1021-1099, November 1947.

## 2.5 Floating Point Arithmetic

Because many problems of the complexity and magnitude requiring a computer of the type discussed in this report are difficult to scale without some preliminary calculations, it seems worthwhile to have automatic facilities for floating point operation incorporated into the computer. However, as has been pointed out by Metropolis<sup>(3)</sup> it is not clear that the usual methods for performing floating point operations are the best ones that can be used. Numerical experiments are being planned at the Digital Computer Laboratory to help determine a reasonable set of rules for the automatic floating point operations to be incorporated into the computer.

The difficulty in using floating point arithmetic arises because the usual automatic floating point operations give no indication of relative error of the fractional part of the machine representation of numbers and indeed do not preserve relative errors in a computation.

Real numbers may be written as

$$x = 2^{x_e} x_f$$

where  $x_e$  is the exponent of the number and  $x_f$  is the fractional part. A computer represents the number by

$$\bar{x} = 2^{x_e} \bar{x}_f$$

where

$$\bar{x}_f = x_f + \epsilon_x$$

$\epsilon_x/x_f \approx \epsilon_x/\bar{x}_f$  is then the relative error in the fractional part of the number and

$$\bar{x} = 2^{x_e} (x_f + \epsilon_x)$$

3. N. Metropolis: "Floating Point Arithmetic," to appear in IRE Transactions on Electronic Computers.



The quantities  $x_e$  and  $x_f$  are not uniquely determined by  $x$ . For fixed point operation the computer user is required to restrict  $x$  so that

$$-1 \leq x < 1,$$

and then

$$x_e = 0.$$

This implies that a problem is scaled before the computer is used. In some cases this is done incorrectly in which case overflow occurs in the computation. Overflow may occur in accumulating a sum or in an improper division (one in which the denominator is less than the numerator).

To avoid scaling difficulties floating point operations are used. In these the restriction is made that

$$\frac{1}{2} \leq x_f < 1,$$

which fixes  $x_e$ .

The problem in doing arithmetic is to keep track of the size of  $\epsilon_x$  as compared to the size of  $x_f$ . The  $\epsilon_x$  accrue during arithmetic operations and their determination is complicated. The initial error made by inserting the fractional part of a number in a machine is bounded by  $\delta = 2^{-s}$  where  $s$  is the number of binary places to which numbers are stored (or one more). Then

$$|\epsilon_x| \leq \delta$$

and

$$\left| \frac{\epsilon_x}{x_f} \right| = \frac{\delta}{x_f} \approx \frac{\delta}{x_f}$$

when we may neglect  $\frac{\delta \epsilon_x}{x_f}$ .

In any computation difficulties ensue when  $x_f$  and  $\epsilon_x$  are of the same order of magnitude. In fixed point work this can be sensed from the magnitude of  $\bar{x}_f$ , that is, a computation in which too small values of  $\bar{x}_f \approx k\delta$ , where  $k$  is a "small" integer, occur, the  $\bar{x}_f$  have to be treated with great care. In floating point the ability to sense troublesome  $\bar{x}_f$  is lost, since all  $\bar{x}_f$  are made to lie between 1/2 and 1.

That  $x_f$  and  $\epsilon_x$  may become of the same order of magnitude is clear from the subtraction process. Suppose we have two standardized numbers  $\bar{x}$  and  $\bar{y}$  with  $x_e \geq y_e$  then one could form the difference between these as

$$\bar{d} = \bar{x} - \bar{y} = 2^{x_e - p} \left[ 2^p(x_f - 2^{y_e - x_e} y_f) + \epsilon_d \right]$$

where

$$\epsilon_d = 2^p(\epsilon_x - 2^{y_e - x_e} \epsilon_y) + \epsilon',$$

and  $\epsilon'$  is a rounding error. Now it is quite possible that although  $x_f$  and  $y_f$  are large compared to  $\epsilon_x$  and  $\epsilon_y$  respectively the quantity  $(x_f - 2^{y_e - x_e} y_f)$  is of the order of  $\epsilon_d$ . The exponent  $p$  occurring in the above measures the number of "significant" zeros in the difference. Metropolis<sup>(3)</sup> suggests storing  $p$  as zeros in front of the first non-zero digits of  $\bar{d}_f$ , that is, he proposes not to automatically standardize numbers of the arithmetic operations.

Then rules for performing arithmetic have to be provided. A provisional incomplete set of rules<sup>(4)</sup> is:

Addition and Subtraction: Position number with smaller exponent by shifting it to the right. Carry and addition or subtraction. If overflow occurs shift one place right and adjust exponent. Never shift to the left and standardize the result.

3. N. Metropolis: "Floating Point Arithmetic", to appear in I.R.E. Trans. on Electronic Computers.
4. These rules are based on those given by Metropolis in (3). They differ from his in the multiplication process as is discussed in the text. The Digital Computer Laboratory is very grateful to Dr. N. Metropolis for supplying a preprint of his paper to that Laboratory.

Multiplication: Given  $\bar{x} = 2^x e \bar{x}_f$

$$\bar{y} = 2^y e \bar{y}_f$$

and also  $\bar{x}_f \geq \bar{y}_f$ . Form

$$\begin{aligned} \overline{\bar{x}\bar{y}} &= 2^{x+y} e^{-p+y} e (2^p \bar{x}_f \bar{y}_f)_R \\ &= 2^{x+y} e^{-p} \left[ \left( 2^p (x_f + \epsilon_x)(y_f + \epsilon_y) \right) + \epsilon' \right] \\ &\approx 2^{x+y} e^{-p} (2^p x_f y_f + 2^p \epsilon_x y_f + 2^p \epsilon_y x_f + \epsilon') \end{aligned}$$

where  $p$  is determined so that

$$\frac{1}{2} \leq 2^p \bar{x}_f < 1,$$

the relative error is

$$\frac{2^p \epsilon_x y_f + 2^p \epsilon_y x_f + \epsilon'}{2^p x_f y_f} \approx \frac{2^p \epsilon_x \bar{y}_f + 2^p \epsilon_y \bar{x}_f + \epsilon'}{2^p \bar{x}_f \bar{y}_f}.$$

In order that the round-off process will not contribute appreciably to the error in the product on a single multiplication, we require

$$\frac{2^p \epsilon_y \bar{x}_f}{2^p \bar{x}_f \bar{y}_f} > \frac{\epsilon'}{2^p \bar{x}_f \bar{y}_f}.$$

This requirement states that the dominant error in the product be larger than the round-off error, that is, the size of the relative error in the product will be that in the factors. Now  $2^p \bar{x}_f$  is made to lie between 1/2 and 1 by the definition of p. Hence

$$\frac{1}{2} \epsilon_y \leq 2^p \bar{x}_f \epsilon_y \leq \epsilon_y.$$

Thus if  $\epsilon_y$  is of the order of  $\epsilon^n$  this criterion will be violated. Note that  $\epsilon_y$  will become of the order of  $\epsilon^n$  after a sequence of multiplications of this sort since it gets decreased in each multiplication.

Muller has suggested that a product be formed as

$$\overline{xy} = 2^{x+y} e^{-p-1} \left[ 2(2^p \bar{x}_f \bar{y}_f) \right]_R.$$

This has the effect of increasing p by 1 and now achieves the desired result since it replaces the above inequality by

$$\epsilon_y \leq 2^{p+1} \bar{x}_f \epsilon_y \leq 2\epsilon_y,$$

but the relative error is unaltered.

Division: The rules for this operation given by Metropolis for forming  $\bar{x}/\bar{y}$  are:

- (1) Shift  $\bar{x}$  to the right if necessary until  $|\bar{x}| < |\bar{y}|$ .
- (2) Standardize  $\bar{y}$ .
- (3) Form the rounded quotient of the resulting numbers.

These rules do not have the property that

$$\epsilon(\bar{x}/\bar{y})_f \approx \frac{\epsilon_x}{x_f} - \frac{\epsilon_y}{y_f}$$

and involve both right and left shifts. For these reasons numerical experiments are being planned to see if a better set of rules for division can be evolved.

## CHAPTER 3

### PROPOSED ORGANIZATION OF THE COMPUTER

#### 3.1 Introduction

The present state of computer components and computer circuits is such that it seems possible to build an arithmetic unit with a multiplication time<sup>(1)</sup>  $M = 3.5 \mu s$  and an addition time  $L = .35 \mu s$ . The description of an arithmetic unit with these characteristic times will be found in Chapter 8. These times are between 100 and 200 times faster than the corresponding times of current computers such as Illiac.

However, corresponding progress in speeding up the memory access-time has not been made. A large-capacity random-access memory will probably use magnetic cores and if these are put together in the form of a "word-arrangement" memory, (cf. Chapter 6) the access-time per word would be  $a = 1.5 \mu s$ . This seems to be the shortest access-time possible for a core memory using presently available cores. This access-time is about 18 times as fast as the access-time of Illiac's memory. Thus progress in memories lags by a factor of between five and ten behind progress in arithmetic and control units.

For this reason the present-day machine designer must reconsider the organization of the computer and see if it is possible to mitigate the effects of the unbalance between the speeds of the arithmetic unit and the memory. It is the purpose of this chapter to discuss various methods by which this can be done and to propose machine organizations which incorporate these methods.

#### 3.2 A Design Criterion<sup>(2)</sup>

The designer may have to choose one from many alternative designs each with a different number of switching elements and a different speed of

1. The best present estimates of the average multiplication time places it between  $3.5 \mu s$  and  $4 \mu s$ . This chapter is based on the lower figure. Its conclusion would not be materially altered if the higher figure were used.
2. It is proposed to apply this criterion in later work.

computation. The latter quantity may depend on the problem posed to the machine. If for a given class of problems the speed of computation can be estimated sufficiently accurately then the criterion discussed below may be used to choose between various designs. It should be noted that increased complexity may produce a slowing down of each individual element because the speed of an element is usually proportional to the number of elements to which it is connected.

For a computer with  $n$  equally reliable switching elements with an average life of  $\alpha$  hours each, a breakdown will occur once every  $\alpha/n$  hours, on the average. The computing time lost per breakdown, called  $F(n)$  hours, consists of the time to find and correct the faulty component, and the time to repeat that portion of the calculation which was wrong. The fraction of time during which the computer does useful calculation is

$$\frac{\frac{\alpha}{n}}{\frac{\alpha}{n} + F(n)} = \frac{1}{1 + \frac{nF(n)}{\alpha}}$$

A problem which requires  $T$  hours of faultless computer time would require, on the average,  $T\left(1 + \frac{nF(n)}{\alpha}\right)$  hours to be solved.

Suppose that the same problem required  $T'$  faultless hours on a computer with  $n'$  switching elements. Which computer is more efficient? For the same equipment, one could construct  $n'/n$  computers of the first sort, which would solve the problem in  $\frac{n}{n'}$   $T$  hours of faultless calculation. Therefore the computer with  $n'$  elements is more efficient provided that

$$T'\left(1 + \frac{1}{\alpha} n'F(n')\right) < \frac{n}{n'} T\left(1 + \frac{1}{\alpha} nF(n)\right)$$

or provided  $n'T'\left(1 + \frac{1}{\alpha} n'F(n')\right) < nT\left(1 + \frac{1}{\alpha} nF(n)\right)$ . Therefore the most efficient computer is one for which the function  $nT\left(1 + \frac{1}{\alpha} nF(n)\right)$  is a minimum.

That is, we must have

$$\frac{dT}{T} = - \frac{1 + \frac{n}{\alpha} \left(2F(n) + n \frac{dF}{dn}\right)}{1 + \frac{n}{\alpha} F(n)} \frac{dn}{n}$$

The experience of the Digital Computer Laboratory with Illiac and with the circuits contemplated for the new computer suggests that the quantity  $nF(n)/\alpha$  will be of the order of .05. If we assume that  $n \frac{dF}{dn}$  can be neglected with respect to  $F$  we have

$$\frac{dT}{T} = - \frac{1 + 2 \frac{n}{\alpha} F(n)}{1 + \frac{n}{\alpha} F(n)} \frac{dn}{n} \approx - \left( 1 + \frac{n}{\alpha} F(n) \right) \frac{dn}{n} .$$

Thus a 1% increase in  $n$  should yield a 1.05% increase in speed to justify itself. If

$$n \frac{dF}{dn} \approx F ,$$

that is,  $F \approx \rho n$ , where  $\rho$  is a constant, we have

$$\frac{dT}{T} = - \frac{1 + 3 \frac{n}{\alpha} F(n)}{1 + \frac{n}{\alpha} F(n)} \frac{dn}{n} = - \left( 1 + \frac{2n}{\alpha} F(n) \right) \frac{dn}{n} ,$$

and setting  $nF(n)/\alpha = .05$  gives

$$- \frac{dT}{T} \approx 1.1 \frac{dn}{n} .$$

### 3.3 The Speed of a Computer

The speed of a computer, the reciprocal of the quantity  $T$  discussed above, is a very difficult quantity to estimate for it depends on the problem being solved, the mathematical methods used, and the sequence of instructions given the machine as well as on the various properties of the machine. Moreover, the mathematical methods employed and the sequence of instructions given to the computer depend on the properties of the machine. For example, an integration of a partial differential equation may be done on one machine using a coarse mesh and high-order integration formulas and on another machine with a fine mesh and a low-order integration formula because the different methods are best suited to different machines.

It was pointed out in Section 1.1 that for present-day computers,  $T$ , the time spent by a computer in solving a problem for which the instructions and operands are in the high-speed random-access memory of the computer, may be written as a sum of two terms:

$$T = a + S$$

where  $a$  is the arithmetic time, the time spent by the arithmetic unit in doing useful and necessary arithmetic, and  $S$  is the sum of the time spent in transferring instructions from the memory to the control and the time spent in transferring operands and partial results to and from the arithmetic unit and the random-access internal memory.

If a computer can perform arithmetic and can consult the memory simultaneously,  $T$  satisfies the inequality

$$\text{Max } (a, S) \leq T \leq a + S .$$

The upper limit in this inequality holds for a sequential computer and the lower limit holds for a computer ( and a problem) such that every memory access is perfectly overlapped with arithmetic, or conversely.

For those problems for which we may write (cf. Section 1.1)

$$T = NFM,$$

where  $N$  is the number of multiplications,  $M$  is the multiplication time and

$$F = 1 + A \frac{L}{M} + \frac{(A + 1)(a_i + a_o)}{M} .$$

We assume that the instructions are stored by pairs and use the times given in Section 3.1 as those pertaining to the computer under consideration.

It then follows that

$$\frac{a}{M} = N(1 + A \frac{L}{M}) = N(1 + \frac{A}{10})$$

$$\frac{S}{M} = \frac{N(A + 1)(a_i + a_o)}{M} = \frac{\frac{3}{2} N(A + 1) a_o}{M} = .65N(A + 1) .$$



Then we have

$$1.18 \leq \frac{S}{a} \leq 6.5$$

where the lower limit obtains for  $A = 1$  and the upper holds for  $A$  infinite.

For

$A = 5$	$S/a = 2.6$
$A = 8$	$S/a = 3.25$
$A = 10$	$S/a = 3.58$

For a computer such as Illiac  $a/M$  is small (approximately .04). The ratio of  $S/a$  is correspondingly small and the access-time is not as important a consideration in the design of a computer as it is when present-day techniques are used.

The main problem confronting the computer designer is to decrease the effect of  $S$ . This can be accomplished in general by reducing the number of accesses to the main memory and by attempting to overlap the time that is used in referring to the memory with the time that is used in doing useful arithmetic.

For problems in which

$$T = NFM$$

this means that the number  $A$  is to be decreased and that the machine is to be organized so that the equation for  $F$  is to be replaced by an expression such as

$$F = \text{Max} \left( 1 + A/10, .65(A + 1) \right)$$

### 3.4 Red Tape

A large portion of the instructions in any program deals with counting of iterations, constructing and modifying other instructions which are to be used subsequently and in general determines what is to be done next rather

than perform arithmetic. This type of work is commonly referred to as red tape work.

The control unit of a computer can be provided with an adder and registers called B-lines (or index registers or modifying registers) so that the majority of the red tape work could be done outside the arithmetic unit and concurrent with its operation. These facilities would produce a substantial decrease in the quantities  $S$  and  $a$ . For those problems for which we may write  $S$  and  $a$  in terms of the number of instructions  $A$  as above, the effect of doing red tape work outside the arithmetic unit is to decrease the quantity  $A$  by a factor of about 2. Thus for a problem in which  $S/a = 3.58$  for a machine such as Illiac, the addition of such a control will reduce  $S/a$  to 2.6.

### 3.5 Storage of Addresses

Addresses to the memory are needed by instructions when the instructions are executed. In many computers these are stored with the instructions in the memory. It was pointed out in the previous section that these addresses can be supplied to the instructions in modified form at the time they are needed, that is, when the instruction is being executed by the control.

This suggests that space for storing these constructed addresses not be provided in the main memory. This space can then be used for more efficient storage of instructions.

The computer design described below makes use of this feature and as a result is able to keep the arithmetic unit usefully occupied for a longer time per memory access for instructions than it otherwise would.

The number of memory words of instructions needed for a given problem is thereby reduced, and as a result the quantity  $S$  is reduced.

### 3.6 Storage of Intermediate Results

The numerical operands are of two sorts: initial or final numbers, and intermediate results. The distinction between these two classes may be

illustrated by the following example. Compute

$$\sum_{i=1}^n a_i b_i .$$

Writing the sum recursively as

$$S_0 = 0$$

$$S_i = a_i b_i + S_{i-1}$$

then the sequence of numbers  $S_i$  ( $i = 1, \dots, n-1$ ) are intermediate results.  $S_0$  and the  $2n$  numbers  $a_i$  and  $b_i$  are initial essential numbers and the number  $S_n$  is the final result. In this example the minimum number of memory accesses for numbers would then be  $2n + 1$  if the result was to be stored in the memory.

If there were some extra storage space outside the main memory (for this example carried out in single precision a single register would presumably do) the number of memory accesses involving intermediate results could be reduced substantially. The design described below uses a number of such registers.

### 3.7 Access for Instructions and Operands

The history of words in the memory representing instructions in a computer provided with B-registers is quite distinct from that of words representing operands. Instructions normally come from consecutive memory locations and go into the control. They are seldom written into the memory. Most instructions obeyed by the control are situated in one or more inner loops which are obeyed repetitively a large number of times. However, the number of instructions in an inner loop is highly variable. For example in problems involving linear algebra such as matrix multiplication inner loops are very short and could be replaced by one special-purpose instruction each; in partial differential equations inner loops can consist of 50, 100, 200 or more instructions.

The designer may take advantage of these facts to decrease the harmful effects of the access-time for instructions and for operands. If very fast-access storage is provided outside the memory, this may be used to hold the instruction being executed and other instructions. This memory may be filled by using multiple readouts of instructions in one read-time from the 1.5  $\mu$ s memory. In that way the access-time per instruction may be decreased. If this storage can hold an entire inner loop which is to be executed  $n$  times, multiple readouts of instructions do not influence the time considerations very much since each instruction is used  $n$  times per readout.

Since inner loops are so variable in size and since the acquisition of a moderately large-capacity very high-speed memory seems difficult to achieve at present, because of technical reasons, it seems advisable to examine other methods for taking advantage of the behavior of instructions described above.

If a few registers were provided outside the 1.5  $\mu$ s memory these could be used to store the current instruction and some following ones. Moreover, one could attempt to fill them from the memory while the arithmetic unit was engaged in multiplication or division, that is, while it could not be calling for new operands.

Further, the information contained in these registers and B-registers would be available to the control which could be consulting the memory for operands needed in the execution of subsequent instructions while the arithmetic unit operated. Storage for these operands would have to be provided.

### 3.8 Summary of Useful Additional Storage

In order to decrease the effects of the term  $S$  in the expression for the time of a computation, we have seen uses for the following storage facilities in addition to those provided on a machine such as Illiac.

- I - B-registers for address modification
- II - Very fast-access memory for
  - a. storage of intermediate results,
  - b. storage of instructions which may be obtained while the arithmetic unit is occupied,
  - c. storage of operands, needed later, which may be obtained while the arithmetic unit is occupied.

The capacities of the memories which could be used effectively vary from problem to problem. Hence, if one single memory could be provided for all three purposes and parts of it used interchangeably, this would have many advantages over fixing the amount of storage once and for all for each of the three purposes listed. Another approach would be to provide a minimum amount of equipment for each of these purposes.

Both approaches have been studied here. The latter is described in the subsequent sections of this chapter and a version of the former in an appendix to this chapter.

### 3.9 Size of the Core Memory

For many problems, the speed of a computer depends, in a rather complicated way, on the speed of its main random-access memory. A large core memory lends itself to more complicated numerical methods and table look-up procedures, whose use can result in a faster program. Furthermore, for problems whose temporary storage requirements exceed the capacity of the core memory, data must be held on the drum or on magnetic tapes, and be sent to or from the core memory in blocks. Unless the core memory is large, an inordinate amount of time may be consumed in transferring data to and from these auxiliary memories. The data transfer problem will be described for the drum. For magnetic tapes similar considerations would apply.

From the discussion given in Section 2.1 it is evident that the percentage decrease in total time to do a problem on a computer with a

drum backup memory becomes smaller as the capacity of the main random-access memory increases beyond a critical point. The reason for this fact is connected with the distribution of time spent in reading and writing on a drum. Suppose that the average random-access time (the time for the drum to rotate to the point where the first of a sequence of consecutive words is under the reading heads) is 8.5 ms or 8500  $\mu$ s, and that thereafter successive words are read at the rate of one every 6  $\mu$ s. The time required to read a block of N consecutive words to or from the drum is then

$$8500 + 6N \mu\text{s},$$

and the average time per word for blocks of various sizes is

$$(8500/N + 6)\mu\text{s}, \text{ that is,}$$

$$8506 \mu\text{s} \quad \text{if} \quad N = 1$$

$$91 \mu\text{s} \quad \text{if} \quad N = 100$$

$$23 \mu\text{s} \quad \text{if} \quad N = 500$$

$$10.25 \mu\text{s} \quad \text{if} \quad N = 2000$$

$$6 \mu\text{s} \quad \text{if} \quad N = \infty$$

The percentage decrease in this time obtained by increasing N beyond 2000 is small. This implies that there should be space in the core memory for at least one 2000-word block of temporary storage. For some problems for which not all data are consecutive, or for which further space must be allowed for the results of calculation which are later to be transferred to the drum, space for several blocks, each of the order of 2000 words, should be provided.

When the time for useful and necessary calculation is of the same order of magnitude as the time required for drum operations, a considerable increase in over-all speed is obtained if drum reads and writes are performed concurrently with arithmetic. The core memory, normally being used by control and the arithmetic unit, would then be only occasionally interrupted to either supply a word to the drum or accept a word from the drum. In such a mode of operation, space for one further block of data in addition to that already mentioned, is required to hold the data for this autonomous transfer.

Finally, there is a class of problems, which includes sorting and filing, for which the data rate of no auxiliary memory now under development seems sufficient. A mode of operation in which the computer is time-shared between two different problems would increase the duty cycle of the arithmetic unit. For such a mode of operation to be practical, there should be enough space in the core memory for two problems.

It is believed that an 8192-word core memory is sufficiently large to satisfy these requirements.

### 3.10 A Computer with Small Buffer Storage

We begin the discussion of this type of computer by listing the equipment required.

#### Main Memory

An 8192-word core memory with an access-time of 1.5  $\mu$ s for both reading and writing is used. The first 1.0  $\mu$ s of a read operation accounts for the readout, and a further enforced 0.5  $\mu$ s is required to regenerate the word. A memory register, called Z, is used for writing and regeneration. When a number to be written has been placed in Z, the arithmetic unit may proceed with further calculation.

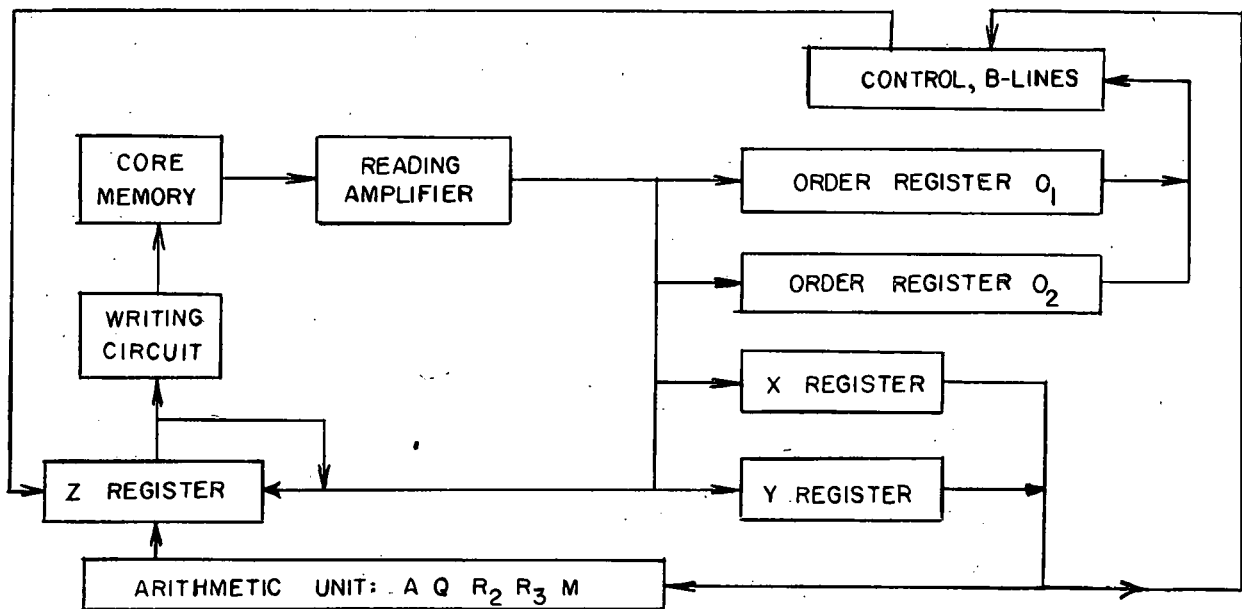
#### Fast-Access Registers

Addressable: The two registers A, Q of the double-length accumulator, and two further registers  $R_2$ ,  $R_3$ .

Non-Addressable (from the point of view of a programmer): Two operand registers X, Y, the memory register Z, and two instruction registers  $O_1$ ,  $O_2$ .

B-Lines: Up to twelve 13-bit address registers which may be used interchangeably as counters, address-modifiers and addresses.

The connections with the core memory are shown in the following diagram.



A word read from the core memory normally goes to one of  $O_1$ ,  $O_2$ , X, Y and also goes to Z for regeneration. For the writing operation, which necessarily is preceded by a read operation, a gate prevents the word read from reaching Z, and Z is set from the arithmetic unit. Since the first part of a write operation consists of reading out the word in the memory location referred to, it is possible to read one word into say X or Y and write another word back, thereby exchanging two words in 1.5  $\mu$ s. A gate from Z back to the reading bus allows a word previously read from the core memory and placed in Z to be transferred later to one of  $O_1$ ,  $O_2$ , X, Y.

### 3.11 Words and Instructions

A 52-bit word may represent one of the following: a 52-bit fixed point number, a packed floating point number consisting of a 42-bit numerical part and a 10-bit exponent, four 13-bit control groups.

An instruction consists of either one or two control groups. (In case of two control groups, the possibility is permitted that it consists of the last 13 bits of one word and the first 13 bits of the next.) Some instructions always require one control group, some always require two, and some may be short or long depending on whether one of the first 13 bits is 0 or 1.



A short instruction (one control group) consists of three parts:

F	7 function bits
B	4 bits
C	2 bits called $C_1$ and $C_2$

The significance of B and C depends on the value of F.

A long instruction (two control groups) consists of F, B, C, and also

N	a 13-bit address.
---	-------------------

The significance of N also depends on the value of F.

### Arithmetic Instructions

B represents an address. If  $B = 0, 1, 2, 3$ , that address is  $0, 1, 2, 3$ , but if  $B = 4, 5, \dots, 15$ , that address consists of the 13-bit integer in B-line  $B_4, B_5, \dots, B_{15}$ . If  $C_1 = 0$  the instruction is short. A short instruction with  $B = 0, 1, 2, 3$  refers to  $A, Q, R_2, R_3$  respectively. If  $C_1 = 1$  the instruction is long, and the address defined by B is added to N. If  $C_2 = 1$ , the number in the B-line, after it is used to form the address, is increased by one. Therefore any arithmetic instruction can also control counting in a B-line.

### Jump Instructions (control transfer instructions)

Most jump instructions are long. N refers to the word containing the next instruction if the condition is satisfied, and C gives the position inside that word (which of the 4 control groups is to be the first control group of that instruction). The address N of an unconditional jump instruction is modified by the address represented by B, as explained above. Conditional jump instructions have unmodified addresses. There are two types: B-line tests and arithmetic tests. For these, B refers to the B-line to be tested, or one of up to 16 arithmetic conditions to be satisfied. This latter feature increases the number of possible orders by using B, in some cases, for 4 further function bits.

The only short jump instructions are the so-called "short loop" instructions, which increase or decrease the integer in the designated B-line by 1 (modulo  $2^{13}$ ). If the result is not zero, control is transferred to the

first control group of the current word or the previous word, depending on whether  $C_2 = 0$  or 1. These generalized repeat instructions allow short inner loops to be held in  $O_1$  and  $O_2$  during their execution. The instructions need be read only once from the main memory.

### B-line Arithmetic

The short B-line instructions transfer the least significant 13 bits of the accumulator (A) to the designated B-line, or replace the contents of the accumulator with the non-negative integer held in the B-line. There are three classes of long B-line instructions depending on whether N represents

- an integer operand
- a core memory address
- the name of another B-line.

### 3.12 Controls

The contents of the order registers  $O_1$  and  $O_2$  are used by two controls, called the advanced control and the arithmetic control. The function of advanced control is to scan instructions not yet executed by arithmetic control, determine what data are required from the memory, and acquire them. So far as possible, both controls operate simultaneously. For example, during the time of an average multiplication (3.5  $\mu$ s) there is more than enough time to read two words from the core memory. Ideally, the memory and the arithmetic unit would both operate continuously, and the overall speed of the computer operating in this parallel mode would be twice that of a sequential machine with the same number of extra registers. In practice, one control will sometimes have to wait for the other, so the ratio of speeds is not quite as favorable as 2:1. One control will have to wait for the other if one of the following situations occurs:

- (a) its operation requires the use of equipment currently being used by the other control,
- (b) its correct operation depends on a result not yet obtained by the other control,
- (c) there is no space to hold further data.

Interlocks must be provided to detect these cases.

For a computer with a B-modification system, most operands from core memory locations have addresses depending on the contents of a B-line, and a very high percentage of conditional jump instructions are B-line tests.

Advanced control executes all B-line instructions.

A problem arises in the execution of an instruction requiring a write to the core memory. Operands may be obtained in advance of a computation, but results must be stored after the computation is finished. In the proposed design the advanced control may defer the execution of a write instruction and may read further operands during the time required to compute the result to be stored.

### 3.13 Example

Calculate  $c_i = a_i b_i$  for  $i = 0, 1, 2, \dots, n-1$ . Assume that  $\{a_i\}$ ,  $\{b_i\}$ ,  $\{c_i\}$  are stored consecutively and that  $B_4$  holds the address of  $a_0$   
 $B_5$  holds the address of  $b_0$   
 $B_6$  holds the address of  $c_0$   
 $B_7$  holds  $8192-n$ , that is,  $-n \bmod 2^{13}$ .

The program requires one word:

F	B	C
→ Replace A by	4	01
Multiply A by	5	01
Store the result in location	6	01
Short loop	7	00

#### Remarks

1. For the first three (arithmetic) instructions  $C_1 = 0$  so these are short instructions.
2. The capacity of  $O_1 O_2$  is two words which is more than enough to hold this loop. Only one instruction read is required for this loop, an instruction read preliminary to executing it for  $i = 0$ . For  $i \neq 0$ , advanced control recognizes that the instructions are still in the fast memory.

3. Since  $C_2 = 1$  for the arithmetic instructions, each execution increases the address in a B-line by one, which is equivalent to changing the subscript  $i$  to  $i + 1$  for that variable.

4. The address of the last instruction is given by  $C = 00$  which means "the first control group of this word" as shown by the arrow.

5. For definiteness, suppose that multiplication requires  $3.5 \mu\text{s}$  exclusive of core memory access, and that the three other instructions each require  $.25 \mu\text{s}$  for operations exclusive of main memory accesses. The time for arithmetic, per execution of the loop,  $a = 3.5 + 3(.25) = 4.25 \mu\text{s}$ . The time for core memory accesses consists of one initial read for the instructions and then two reads and one write per execution of the loop. The average memory time is therefore

$$S = \frac{1.5 + 3n(1.5)}{n} = \left( \frac{1.5}{n} + 4.5 \right) \mu\text{s}.$$

For  $n$  reasonably large, this average value is very close to  $4.5 \mu\text{s}$ . Since  $a = 4.25$  and  $S = 4.5$ , a sequential computer requires  $8.75 \mu\text{s}$  for each execution of this loop, and a computer capable of simultaneous arithmetic and memory operations should require  $\max(4.25, 4.5) = 4.5 \mu\text{s}$ .

6. A sequential machine executing this program would use the memory as follows:

```
read the word of instructions
read  $a_0$ 
read  $b_0$ 
write  $c_0$ 
read  $a_1$ 
read  $b_1$ 
write  $c_1$ 
etc.
```

If the memory were used in this way, a delay of at least  $3.5 \mu\text{s}$  (one multiply time) must elapse between "read  $b_0$ " and "write  $c_0$ " since multiplication cannot begin until  $b_0$  has been obtained, and "write  $c_0$ " cannot begin until the multiplication is completed.

Advanced control will be designed to use the core memory in the following order:

```
read instruction word
read a0
read b0
read a1
read b1
write c0
read a2
read b2
write c1
etc.
```

The first multiplication takes place while  $a_1$  and  $b_1$  are being read, and the second multiplication takes place during part of the time required to write  $c_0$  and read  $a_2$  and  $b_2$ .

It should also be noted that the successful overlap of memory use and arithmetic requires, in this example, that advanced control read the operand required for the next use of the multiply instruction during the current multiplication. This is possible because the advanced control performs B-operations and executes jump instructions. Therefore, after arithmetic control has obtained its operand, advanced control is allowed to process the instruction again.

#### Conditional arithmetic jumps

When advanced control encounters a conditional arithmetic jump, it cannot predict with certainty which instruction will be executed next until the previous arithmetic has been completed. In this situation, if instruction 96 (see order code) is used by the programmer, advanced control waits for arithmetic control whereas if instruction 95 is used by the programmer, advanced control reads the word to be jumped to and waits to make sure that it is the next word of instructions to be executed.

### 3.14 The Design of the Two Controls

The controls are designed in such a way that there are no rules or conventions which the coder must follow. If the coder knows something about the operation of the two controls, he may be able to write a faster program, but it is not necessary even to know of the existence of  $O_1$ ,  $O_2$ , X, Y, Z or advanced control in order to write a correct program. One of the most important interlocks between the two controls is this: if advanced control discovers the need for a word from some memory location, and it has previously encountered a store instruction to that memory location which has not been performed, then advanced control waits. In other words, advanced control does not take an old value if a new one is in the course of being computed.

Considerable equipment is devoted to reducing the number of core memory accesses and to performing them as early as possible. Function decoding is more complex for a highly compressed order code. Addresses are held in fast registers (B-lines), powers of 2 are generated inside of the arithmetic unit, extra registers are provided for the storage of intermediate results, and advanced control is provided with equipment to recognize when a core memory word is actually held in X or Y already. Because  $O_1$  and  $O_2$  have a capacity of 8 control groups, advanced control can process control groups up to 7 ahead of arithmetic control. Since at least two operands could be obtained during the time of one multiplication, two data buffer registers X, Y are provided. If it does not conflict with other considerations it is planned to use Z in the following way: if there is nowhere else to put a word to be read from the core memory, it is temporarily left in Z, and later transferred to its correct destination.

Core memory write instructions are executed when arithmetic control reaches them, but advanced control need not wait for arithmetic control. Logically, it would be possible to defer a single write operation still further, provided there was a register to hold the word to be written. Then the core memory would be written into only when it was not needed for anything else. Since this requires an extra register, this feature has not been included.

To simplify the design of the controls, operands are read from the core memory in the order in which they are required by arithmetic control, and the next word of instructions is read only after advanced control has processed the current instruction word. Because some loops of instructions are held in  $O_1, O_2$  during several executions of each instruction, neither control can alter control groups in  $O_1, O_2$  during the execution of an instruction.

The following additional equipment is required to sequence the two controls:

$D_1$  is a 3-bit counter which holds the "address" in  $O_1, O_2$  of the control group currently being executed by arithmetic control. The four control groups in  $O_1$  are numbered 0 to 3, and those in  $O_2$  are numbered 4 to 7. The most significant bit of  $D_1$  indicates the register ( $O_1$  or  $O_2$ ), and the remaining two bits indicate the position inside the register.  $D_1$  is used to operate a selector to read a control group to the order register of arithmetic control. Long instructions require two uses of this selector.

$D_2$  is a 3-bit counter, similar to  $D_1$ , used by advanced control.

$D_3$  is a 13-bit control counter used by advanced control.  $D_3$  gives the location of the next word of instructions.

$A_x$  is a 13-bit register, which holds the address of the word in the register X.

$A_y$  is a 13-bit address register for the word in Y.

$A_z$  is a 14-bit register. One bit is set to 1 when advanced control encounters a write instruction, and the address from that instruction is copied into the remaining 13 bits of  $A_z$ . The extra bit is set to zero when the write instruction is obeyed by arithmetic control. Therefore  $A_z$  holds the address of any unexecuted write instruction.

$K_x$  is a 2-bit counter which is increased by one whenever advanced control finds a use for X, and is decreased by one each time arithmetic control uses X.

$K_y$  is a 2-bit counter for Y whose action is analogous to  $K_x$ .

Registers  $O_1$  and  $O_2$  are each extended by 4 bits, one for each control group. The extra bit in each control group may be set by advanced control to indicate, in questionable cases, which of X, Y holds the operand. The extra bit is read by arithmetic control.

### Sequencing of Instructions

Suppose that the instruction currently being obeyed by advanced control is not a jump instruction. If it is a multiply or divide instruction, and if also  $A_z$  indicates that there is an unexecuted store operation to be done, after the operand for this multiply or divide instruction has been obtained, advanced control waits until the store operation has been initiated before proceeding to the next instruction. Otherwise on completion of the instruction,  $D_2$  is increased by one, mod 8, provided the result is not equal to  $D_1$ . If  $D_2 = D_1$ , the next instruction is being executed by arithmetic control, so advanced control waits until  $D_2 \neq D_1$ . If, by counting, the most significant digit of  $D_2$  was changed, then either

- (a) advanced control has processed all instructions in  $O_1$  and  $O_2$  and must read another word of instructions,
- or (b)  $O_1, O_2$  contain an inner loop of between 1-1/4 and 2 words, and advanced control has passed from the last control group of the first of these words to the first control group of the second word.

A flipflop is provided for the purpose of detecting (b) above. (b) can only occur if one of the short loop instructions, ( $b' = b + 1$  or  $b' = b - 1$ , and jump to the left-hand side of the last word if  $b'$  is non-zero), has been obeyed as a jump. When such a jump is executed, this flipflop is set to one. If, as a result of counting, the most significant bit of  $D_2$  is changed, then instead of reading a word of instructions, this short loop flipflop is set to zero, and advanced control proceeds to execute the next control group.

In case (a) above, advanced control next compares the most significant digits of  $D_1$  and  $D_2$ . If they are not equal, advanced control reads the word from the address given by  $D_3$  into the register ( $O_1$  or  $O_2$ ) given by the most significant bit of  $D_2$ , increases  $D_3$  by one, and proceeds to execute the control



group given by  $D_2$ . If the most significant bits of  $D_1$  and  $D_2$  are equal, arithmetic control is still executing a control group in the register ( $O_1$  or  $O_2$ ) to be read into next, so advanced control first reads the word (whose address is given by  $D_3$ ) into Z, then waits until the most significant digits of  $D_1$  and  $D_2$  disagree, and then proceeds as before.

Two operations, "read from the core memory", and "read into Z from the core memory", require further explanation. When a word is to be read from the core memory, its address is compared with  $A_z$ . If the extra bit in  $A_z$  is 1 and the addresses coincide, the number wanted has not yet been computed, and the read is inhibited and advanced control waits. Then, when the extra bit becomes zero, indicating that arithmetic control has placed the word in Z, it is copied from Z to its destination. If the extra bit of  $A_z$  is zero, or the addresses differ, the word is read from the core memory in the usual way.

After arithmetic control has obtained the operand for a control group,  $D_1$  is increased by one. Before executing another control group,  $D_1$  and  $D_2$  are compared. If  $D_1 = D_2$ , arithmetic control waits until  $D_1 \neq D_2$ . Note that  $D_2$  is increased provided it does not become equal to  $D_1$ , whereas  $D_1$  is increased regardless of  $D_2$ .

### Jump Instructions

When advanced control encounters an unconditional jump instruction, it waits until the first bit of  $D_1$  and of  $D_2$  agree, then it complements the first bit of  $D_2$ , replaces the other 2 bits of  $D_2$  by the digits of C, replaces  $D_3$  by the address from the current instruction, and performs the operations previously described for reading a word to one of  $O_1$  or  $O_2$ . When arithmetic control encounters an unconditional jump, it complements the first bit of  $D_1$  and replaces the other two bits of  $D_1$  by C, compares  $D_1$  with  $D_2$  and executes the next control group when  $D_1 \neq D_2$ .

Conditional jump instructions are executed by advanced control. If the condition depends on the contents of a B-line, advanced control may execute the jump immediately, whereas if the condition depends on a result in the arithmetic unit, advanced control waits until  $D_1 = D_2$  before making the test.

Two flipflops,  $F_1$  and  $F_2$ , are required for conditional jump instructions.  $F_1$  is zero unless advanced control has encountered a conditional jump instruction and arithmetic control has not.  $F_1$  is set to zero when arithmetic control obeys a conditional jump instruction. If advanced control encounters a conditional jump instruction and  $F_1$  is already one, it waits until  $F_1$  becomes zero before setting it to one and proceeding.  $F_2$  is used to indicate to arithmetic control whether the jump was executed or not.

When advanced control encounters a conditional jump instruction, it acts as follows:

- (1) If the jump is an arithmetic test likely to be obeyed, (order 95) it reads the word, whose address is given by the current instruction, into Z.
- (2) It waits until  $F_1 = 0$ .
- (3) If the test involves an arithmetic result, it waits until  $D_1 = D_2$ .
- (4) It makes the test. If the jump is not obeyed it sets  $F_2 = 0$  and proceeds to the next control group. If the jump is obeyed it sets  $F_2 = 1$  and proceeds to (5).
- (5) If the instruction is not a "short loop" instruction, it follows the procedure described above for unconditional jumps. If it is a short loop instruction, it complements the most significant digit of  $D_2$  if  $C = 01$ , or leaves it the same if  $C = 00$ , sets the two other bits of  $D_2$  to zero, waits until  $D_1 \neq D_2$ , and then proceeds to execute the next control group.

When arithmetic control encounters a conditional jump instruction, it sets  $F_1 = 0$  and either counts  $D_1$  (if  $F_2 = 0$ ) or changes  $D_1$  in a similar fashion to  $D_2$  (if  $F_2 = 1$ ).

### Store Instructions

When advanced control encounters a store instruction, it waits, (if necessary), until the extra bit of  $A_z$  is zero indicating that a previously called-for store operation has now been executed, and then copies the address into  $A_z$ , sets the extra bit of  $A_z$  to one and proceeds to the next control group.

When arithmetic control encounters a store instruction, it waits until the end of any memory operation then in progress, copies the word into Z and compares  $A_z$  with  $A_x$  and  $A_y$ . If  $A_z = A_x$ , for example, then Z is copied into X. After this possible "updating" operation, it signals the core memory to write Z into the address given by  $A_z$ . When the write has been completed, the core memory re-sets the extra flipflop of  $A_z$  to zero. Arithmetic control is free to execute further control groups as soon as the core memory action has been initiated, and advanced control is prevented from using the core memory by a "memory busy" signal.

### The Use of X and Y

Advanced control does a partial decoding of other instructions to indicate whether

- (a) the instruction transfers information between the arithmetic unit and a B-line,
- (b) an operand is apparently required from the core memory.

If (a) holds, advanced control waits until  $D_1 = D_2$ , and then executes the instruction.

If (b) holds, advanced control compares the address with  $A_x$  and  $A_y$ .

The purpose of the following rule is to allow the programmer to save accesses to and from the memory in problems where an operand is used a number of times in a set of orders. If the address of the storage location referred to is not equal to either  $A_x$  or  $A_y$ , that is X and Y are available to advanced control for reading or writing, one on these is chosen as follows: If the latest use of one of these registers was by the instruction: copy previous operand (that is order 58), choose that register; otherwise choose the register not used last. A flipflop, set by each use of X or Y, indicates which register satisfies this rule. Suppose, for definiteness, that X is chosen. If  $K_x = 0$ , advanced control reads the word from the memory into X, sets  $A_x$  equal to the address, sets  $K_x = 1$ , and sets to zero the 14th bit of the current control group in  $O_1, O_2$  indicating to arithmetic control that the operand is to be found in X. If  $K_x \neq 0$ , advanced control reads the word into Z, and waits until  $K_x = 0$ , before copying Z into X and performing the other operations just mentioned.

If the address is equal to one of  $A_x$  or  $A_y$ , say  $A_y$ , advanced control waits (if necessary), until  $K_y < 3$ , and then increases  $K_y$  by one, and sets the 14th bit of the current control group in  $O_1, O_2$  to 1, indicating that the operand is to be found in Y.

The use of these features of advanced control is illustrated in the second part of section 3.16.2.

### 3.15 Order Code

Capital letters refer to registers and addresses, and lower case letters refer to the contents of registers and memory locations. Unprimed lower case letters refer to the initial contents, and primed lower case letters refer to the final contents. For example,  $a' = m$  means copy into A the word in memory location M, and  $a' = a + m$  means add into A the word in memory location M. M is the address after modification. For arithmetic instructions,

if $C_1 = 0$ (short instruction)	$M = B$	for $B = 0, 1, 2, 3$
	$M = b$	for $B = 4, 5, \dots, 15$
if $C_1 = 1$ (long instruction)	$M = B + N$	for $B = 0, 1, 2, 3$
	$M = b + N$	for $B = 4, 5, \dots, 15$ .

If  $C_1 = 0$  and  $B = 0, 1, 2, 3$ , then m is the contents of  $A, Q, R_2$  or  $R_3$ ; otherwise m refers to the core memory. An unmodified address ( $M=N$ ) is obtained by setting  $B = 0, C_1 = 1$ .

When N refers to a B-line, the name of the B-line is  $B_N$ . Thus  $b' = b + b_N$  means add the address from  $B_N$  to the address in B-line B.

The subscript R indicates that a number is rounded to single length accuracy. One rounded multiplication instruction is described by  $a' = (a \cdot m)_R, q' = a$ . The initial contents of A is transferred to Q and multiplied by m. During multiplication the digits of Q are circulated, and digits are not shifted from A into Q.

The symbol (aq) designates the value of the double length number in the AQ registers.  $(aq) = a + 2^{-51} \bar{q}$ , where  $\bar{q}$  consists of q with a zero sign digit.

R is an extra register in the arithmetic unit, possibly one-half of the shifting number register. When the contents of the accumulator are replaced by another number (orders 1, 2, 3, 4), the previous contents are then copied into R. This facilitates the double precision accumulation of products.

In the accumulator A, intermediate results are correctly represented provided that they do not lie outside of the range  $-2 \leq a < 2$ . A number is said to have overflowed when it lies outside of the range  $-1 \leq a < 1$ . An overflow indicator will be set if the result of an arithmetic operation is in a state of unassimilated overflow, during improper division<sup>(1)</sup>, if an unassimilated overflow is detected during arithmetic left shift, or if assimilation causes overflow. The overflow indicator will be cleared by any logical instruction ("and", "exclusive or", "not" or logical shift left or right), by an arithmetic shift right of at least one place, or a "test and clear overflow" instruction. An attempt to store a number into the core memory when the overflow indicator is set causes the computer to stop.

There are two categories of shift orders, arithmetic shifts and logical shifts. An arithmetic shift multiplies or divides a or (aq) by  $2^M$ , that is, the sign digit of A is duplicated during right shift, and digits are shifted around the sign digit of Q. A logical shift considers (aq) to consist of 104 consecutive digits, which are translated to the left or right. Zeros are inserted into the left hand end of A on right shift, and on left shift overflow is not set. Double length logical shifts shift into the sign digit of Q.

### Stops

It is preferable if, before stopping, the computer prints information telling why it has stopped. This could be accomplished by reserving a small portion of the core memory for a multiple-entry printing routine. The various stop conditions cause the contents of the control counter to be copied into  $B_{15}$ , and cause automatic jumps to distinct entries to this printing routine. After printing  $b_{15}$ , and other information pertinent to the stop, the computer either encounters one absolute stop or the program might continue.

---

(1) See chapter 8 for a discussion of assimilated and unassimilated overflow.

If the function digits of F consist of seven zeros or seven ones, the computer will stop. The purpose of this is to stop as soon as possible if control begins to execute data as instructions--many fixed point data are small positive or negative numbers. Similarly any unassigned value of F causes a stop. If there are parity checks on the drum or core memories, the wrong parity should cause a stop.

Except as otherwise noted, the following orders are short or long depending on whether  $C_1 = 0$  or 1.

Arithmetic orders

1.  $a' = m$   $r' = a$
2.  $a' = -m$   $r' = a$
3.  $a' = |m|$   $r' = a$
4.  $a' = -|m|$  ,  $r' = a$
5.  $a' = a + m$
6.  $a' = a - m$
7.  $a' = a + |m|$
8.  $a' = a - |m|$
9.  $a' = 2m$
10.  $a' = -2m$
11.  $a' = a + 2m$
12.  $a' = a - 2m$
13.  $a' = 2^{-M}$
14.  $a' = -2^{-M}$
15.  $a' = a + 2^{-M}$
16.  $a' = a - 2^{-M}$
17.  $a' = (a \cdot m)_R$  ,  $q' = a$
18.  $a' = (q \cdot m)_R$  ,  $q' = q$
19.  $(aq)' = a \cdot m$
20.  $(aq)' = q \cdot m$

21.  $(aq)' = (a \cdot m) + (rq)$  add product
22.  $(aq)' = (q \cdot m) + 2^{-51}a$  semi-accumulative multiplication
23.  $a' = \left( \frac{(aq)}{m} \right)_R$
24.  $a' = \left( \frac{a}{m} \right)_R$
25.  $a' = \text{remainder}, q' = \frac{(aq)}{m}$ . The sign of the remainder agrees with the sign of the divisor. If the divisor is positive, the remainder is less than the divisor. If the divisor is negative, the remainder is greater than or equal to the divisor.
26.  $a' = 2^M a, q' = q$  M can be positive, negative or zero.
27.  $a' = 2^{-M} a, q' = q$  For 26, 27  $|M| \leq 64$ , and
28.  $(aq)' = 2^M (aq)$  for 28-31  $|M| \leq 128$ .
29.  $(aq)' = 2^{-M} (aq)$
30. set  $a = 0$ , then 28 above
31. set  $a = 0$ , then 29 above
32.  $q' = m$
33.  $a' = m, m' = a$  (exchange)
34.  $a' = m' = a + m$
35.  $a' = m' = a - m$
36.  $a' = a, m' = a + m$
37.  $a' = a, m' = a - m$
- 38.\*  $a' = 2^k a, b' = b - k$ , where  $\frac{1}{2} \leq |a'| \leq 1$ . This is a single length standardize instruction. If  $a = 0$ ,  $k = 64$  and  $a' = 0$ .
- 39.\*  $(aq)' = 2^k (aq), b' = b - k$ , where  $\frac{1}{2} \leq |(aq)'| \leq 1$ .
- If  $(aq) = 0$ ,  $k = 128$  and  $(aq)' = 0$ .
- 40.\*  $(aq)' = 0$

\*Denotes short orders

Floating Point Orders ( $a_F$  is the contents of the accumulator considered as a floating point number, and  $m_F$  is the floating point number held in M.)

- 41.  $a_F' = m_F$
- 42.  $a_F' = -m_F$
- 43.  $a_F' = a_F + m_F$
- 44.  $a_F' = a_F - m_F$
- 45.  $a_F' = a_F - |m_F|$
- 46.  $a_F' = a_F \cdot m_F$
- 47.  $a_F' = a_F + m_F$
- 48.  $a_F' = 2^M a_F$  (add M to the exponent)
- 49.  $a_F' = 2^{-M} a_F$  (subtract M from the exponent)
- 50.  $m_F' = a_F$
- 51.\*  $a_F' = (a, b)$  (Copy the contents of the designated B-line into the accumulator exponent register, that is, convert from fixed point to unstandardized floating point.)
- 52.\* Standardize  $a_F$ , counting in the exponent register.
- 53.\*  $b' = \text{exponent}$ ,  $a_F' = a_F$ . This can be used to convert a floating point result to fixed point.

\*Denotes short orders

Store Orders (see also 32-37, 50, 76, 78)

- 54.  $m' = a$
- 55.  $m' = q$
- 56.  $m' = (aq)_R = a'$       $q' = 0$
- 57.  $m' = a$ ,  $a' = 0$



58.  $m'$  = immediately preceding operand. Commonly  $M = 1, 2,$  or  $3$ . This allows the contents of the X or Y register to be transferred to Q,  $R_2$  or  $R_3$  without affecting the accumulator.
59. "store and carry". Assimilate a, and store its non-sign digits in location M with a zero sign digit.  
Then  $a' = 0, 2^{-51}, -2^{-51},$  or  $-2^{-50}$  depending on whether
- $a \geq 0$  and no overflow
  - $a \geq 0$  and overflow
  - $a < 0$  and no overflow
  - $a < 0$  and overflow.

### Logical Orders

60.  $a' = a \oplus m$  ("exclusive or")  
 $q' = a \& m$  ("and" or digitwise product)
61.  $a' = \sim m = -m - 2^{-51}$  ("not" or digitwise complement)
62.  $a' = 2^M a$  overflow is not set
63.  $a' = 2^{-M} a$  the sign digit is not duplicated
64.  $(aq)' = 2^M(aq)$
65.  $(aq)' = 2^{-M}(aq)$
- (aq) is considered to be 104 consecutive digits without regard for sign.

### B-line Orders

66.  $b' = N$  The unmodified address consisting of the
- 67.<sup>+</sup>  $b' = -N$  second control group is used as an
- 68.<sup>+</sup>  $b' = b + N$  integer operand.
- 69.<sup>+</sup>  $b' = b - N$
- 70.<sup>+</sup>  $b' = b_N$
- 71.<sup>+</sup>  $b' = -b_N$  The second control group is interpreted
- 72.<sup>+</sup>  $b' = b + b_N$  as the name of a second B-line.
- 73.<sup>+</sup>  $b' = b - b_N$

- 74.\*  $b' = b - 1$
- 75.\*  $b'$  is the 13-bit group from word N (unmodified), position C.
- 76.\* store b in word N position C.
- 77.\* set B, B+1, B+2, B+3 to be the four 13-bit groups of the word at N.  
B is restricted to be one of 4, 8 or 12.
- 78.\* store the contents of B, B+1, B+2, B+3 in word N. (B = 4, 8, or 12).
- 79.\*  $a' = b \cdot 2^{-51}$  (b is considered a positive integer)
- 80.\*  $a' = a + b \cdot 2^{-51}$
- 81.\*  $a' = a - b \cdot 2^{-51}$
- 82.\*  $b'$  = least significant 13 bits of a
- 83.\*  $b'$  = least significant 13 bits of q

+Denotes long orders  
\*Denotes short orders

#### B-Conditional Jump Orders

- 84.\*  $b' = b + 1$ , if  $b' \neq 0$  jump to the leftmost control group of:  
this word if  $C_2 = 0$ ,  
the preceding word if  $C_2 = 1$ .
- This is called the "short loop" order.
- 85.\*  $b' = b - 1$ , otherwise similar to 84.
- 86.\*  $b' = b + 1$ , jump to word N, position C if  $b' \neq 0$
- 87.\*  $b' = b - 1$ , jump to word N, position C if  $b' \neq 0$
- 88.\* Jump if  $b \neq 0$
- 89.\* Jump if  $b = 0$
- 90.\* Jump if the most significant bit of b is zero
- 91.\* Jump if the most significant bit of b is one

+Denotes long orders  
\*Denotes short orders

### Unconditional Jump Orders

- 92.<sup>+</sup> Jump to word M position C. (M is the modified address)
- 93.<sup>+</sup> Set  $B_{15}$  to be the address of the next instruction, and jump to word M position C. This is an "enter subroutine" order.
94. Jump to the instruction given by  $B_{15}$ . The effect of executing a 94 order is to transfer control to the first instruction following the preceding 93 order. 94 is a "leave subroutine" order.

+Denotes long orders

### A-Conditional Jump Orders

- 95.<sup>+</sup> The test is determined by the value of the integer B. Advanced control is to read out the word into Z on the assumption that the jump will probably be executed.

If B = 0, jump if  $a \geq 0$

B = 1, jump if  $a < 0$

B = 2, jump if  $a = 0$

B = 3, jump if  $a \neq 0$

B = 4, jump if overflow, and reset indicator

B = 5, jump if no overflow, otherwise reset indicator

B = 6, jump if overflow

B = 7, jump if no overflow

- 96.<sup>+</sup> Similar to 95, except that advanced control assumes that the jump will not be executed, and does not read the word in advance.

+Denotes long orders

### Miscellaneous

97.  $a' = a$ , set overflow if  $a - m$  is negative. This is a comparison of the sizes of two numbers which does not alter the contents of any register.

- 98.\*      Modify the next instruction by the contents of B-line B, in addition to any other modification called for.
- 99.\*      Read, into A, a word whose bits define the states of the various switches on the machine.
- 100.      Input-output and drum instructions. These have not yet been assigned.

\*Denotes short orders

### 3.16 Programming Examples

Suppose that two vectors  $\underline{a}$ ,  $\underline{b}$  are stored in core memory locations  $A_0, A_0+1, \dots, A_0+n-1$ ;  $B_0, B_0+1, \dots, B_0+n-1$ , and that  $\lambda$  is a number whose magnitude is less than 1. It is required to compute the vector  $\underline{c} = \underline{a} + \lambda \underline{b}$ , and store it in locations  $C_0, C_0+1, \dots, C_0+n-1$ .

Assume       $q = \lambda$

$$b_4 = A_0$$

$$b_5 = B_0$$

$$b_6 = C_0$$

$$b_7 = -n \text{ mod } 2^{13}.$$

The program is

F	B	C	Remarks
$\rightarrow a' = (q \times m)_R, q' = q$	5	01	leftmost control group of a word
$a' = a + m$	4	01	
$m' = a$	6	01	
$\leftarrow b' = b + 1, \text{ jump unless } b = 0$	7	00	short loop order.

The inner loop requires one word consisting of four short instructions. The input routine can assign storage locations to the program in such a way that the first instruction of this loop is a left-hand control group. If an overflow test were included in this loop it would still require only 1-1/2 words.

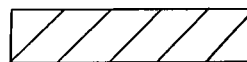
The table on the following page shows the time sequence of events for the core memory, advanced control and arithmetic control for the first few instructions encountered in this loop. Each typewritten line, reading down, represents  $\frac{1}{4}\mu\text{s}$  of time, and events on the same horizontal line take place simultaneously. The following assumptions are made:

1. The last  $0.5\mu\text{s}$  of a core memory read operation, after the word has been read out and while the core memory is regenerating the location, is available for advanced control to proceed with further non-memory operations.
2. Operation times are:  $1.5\mu\text{s}$  for core memory read or write,  $3.5\mu\text{s}$  for multiplication,  $0.25\mu\text{s}$  for simple operations such as addition, transfer, jump.

These times differ from the times that are expected to be taken by the proposed computer to do the operations referred to. However, the diagram is simpler to draw with these values of the times and for any values for these quantities corresponding diagrams can be made.

3. At the beginning (the top of the page), the core memory, advanced control and arithmetic control are all waiting. Normally, advanced control has performed operations ahead of arithmetic control, so this assumption is conservative.

Time during which one unit must wait is marked



The first and second executions of the loop and part of the third execution, by arithmetic control, are shown. The first and last executions differ from the intermediate ones, which are all similar to the second. After the first execution of the loop, the core memory operates continuously, the arithmetic unit must wait only  $0.25\mu\text{s}$  out of every  $4.5\mu\text{s}$ , and advanced control, although supervising memory operations and performing B-line arithmetic, is idle a considerable fraction of the time. This is not inefficient, because the purpose of advanced control is to maximize the duty cycles of the memory and the arithmetic unit, which it certainly does for this problem.

time / $\mu$ s	Core Memory	Advanced Control	Arithmetic Control
0	read word of instructions		
1.0	regenerate	word of instructions to $O_1$	
1.5	read location $B_0$	Add 1 to B-line 5	
2.5	regenerate	$b_0$ to X	Multiply q by x
3.0	read location $A_0$	Add 1 to B-line 4	
4.0	regenerate	$a_0$ to Y	
4.5	read location $B_0+1$	set $A_z = \text{address } C_0$ Add 1 to B-line 6 count in B-line 7 and jump	
5.5	regenerate	$b_1$ to X	
6.0			Add y to accumulator
6.25	store z in location $C_0$		store in z
			obey jump
			Multiply q by x
7.75	read location $A_0+1$	Add 1 to B-line 4	
8.75	regenerate	$a_1$ to Y	
9.25	read location $B_0+2$	set $A_z = \text{address } C_0+1$ Add 1 to B-line 6 count in B-line 7 and jump	
10.25	regenerate	$b_2$ to X	Add y to accumulator
10.75	store z in location $C_0+1$		store in Z
			Multiply q by x
12.25	etc.	etc.	etc.

### 3.16.1 The Scalar Product of Two Vectors

If  $\underline{a}$ ,  $\underline{b}$  are  $n$ -vectors stored in locations  $\{A_0+ki\}$ ,  $\{B_0+i\}$ ,  
 ( $i = 0, 1, \dots, n-1$ ), form

$$a \cdot b = \sum_{i=0}^{n-1} N(A_0 + ki) N(B_0 + \ell i),$$

where  $k$  and  $\ell$  are constants and the notation  $N(x)$  means "the fixed point number held in memory location  $x$ ". (This problem arises in the multiplication of the transpose of one rectangular matrix by another matrix, if both matrices are stored by rows).

Assume  $(aq) = 0$  at the start of the loop,

and  $b_4 = A_0$

$$b_5 = B_0$$

$$b_6 = k$$

$$b_7 = \ell$$

$$b_8 = -n \text{ mod } 2^{13}.$$

The program is

F	B	C	N <sup>(3)</sup>
$a' = m \quad r' = a$	4	00	
$(aq)' = m \cdot a + (rq)$	5	00	
$b' = b + b_N$	4	10	6
$b' = b + b_N$	5	10	7
short loop	8	01	

The inner loop is 1-3/4 words. The result is the unrounded double length sum of products. The first instruction following this loop would be either "test overflow", or "round and store". If  $k, \ell$  are constant throughout all uses of the loop, the third and fourth instructions could be of the  $b' = b + N$  type, and only 3 B-lines would be required.

3. In this example the digits under N label a B-line,  $B_N$ .

### 3.16.2 Evaluation of a Polynomial

Form  $y_n = \sum_{i=0}^n a_i t^{n-i}$ , by means of the sequence

$$y_0 = a_0$$

$y_{i+1} = ty_i + a_{i+1}$ , where  $\{a_i\}$  are stored in consecutive core memory locations beginning at  $A_0$ , and

$$b_4 = A_0$$

$$b_5 = -n \text{ mod } 2^{13}$$

$$q = t$$

	F	B	C
$a' = m$		4	01
$a' = (q \cdot m)_R, q' = q$		0	00
$a' = a + m$		4	01
short loop		5	00

The inner loop itself requires 3/4 word. The instruction  $a' = (q \cdot m)_R, q' = q$  0 00 forms the product  $(a \cdot q)_R$ , and is required to be a left-hand control group.

An alternative method which illustrates the selection of X or Y by advanced control assumes:

$$b_4 = A_0$$

$$b_5 = -n \text{ mod } 2^{13}$$

$$b_6 = \text{address of } t.$$



F	B	C
a' = m	4	01
→ a' = (m · a) <sub>R</sub>	6	00
a' = a + m	4	01
└ short loop	5	00

The inner loop requires 3/4 word. The multiply instruction is required to occupy a left-hand control group of a word. Although the program apparently calls for two memory references per execution of the loop, one for  $a_i$  and one for the number  $t$ , in fact,  $t$  is read out just once and held permanently in one of the registers X or Y. This is true because the two last-used operands are still available, and  $t$  is always one of them. The program calls for operands to be used in this order:

$a_0, t, a_1, t, a_2, t, a_3, \dots$

and when advanced control decides which register (X or Y) to hold  $a_{i+1}$ , it chooses the register not used last, that is, not containing  $t$ , so  $t$  is held permanently throughout the execution of the loop.

### 3.16.3 Continued Fraction

Form  $F_n(x)$ , the  $n$ -th convergent of  $a_0 + \frac{b_0 x}{a_1 + \frac{b_1 x}{a_2 + \dots}}$ .

Define

$$y_0 = a_n$$

$$y_1 = b_{n-1} x + a_{n-1} y_0$$

and, for  $i \geq 2$ ,

$$y_i = y_{i-2} x b_{n-i} + y_{i-1} a_{n-i}.$$

Thus

$$F_n(x) = y_n / y_{n-1}.$$

Suppose that  $a_n, b_{n-1}, a_{n-1}, b_{n-2}, \dots, a_1, b_0, a_0$  are stored in consecutive memory locations beginning at the address in  $B_4$ , and suppose that  $B_5$  has  $-(n-1)$  initially, and that  $x$  is in  $A$ .

F	B	C	
$m' = a$	3	00	Store $x$ in $R_3$
$a' = m$	4	01	[ Store $y_0 (=a_n)$ in $R_2$
$m' = a$	2	00	
$a' = m$	4	01	[ $b_{n-1} x$
$(aq)' = m \cdot a$	3	00	
$a' = m, r' = a$	4	01	$y_1$ unrounded
$(aq)' = m \cdot a + (rq)$	2	00	
$\rightarrow m' = (aq)_R$	0	00	Round $y_{i-1}$
$q' = m$	2	00	$q = y_{i-2}$
$m' = a$	2	00	$r_2 = y_{i-1}$
$a' = (q \cdot m)_R$	4	01	
$(aq)' = a \cdot m$	3	00	$y_{i-2} x b_{n-i}$
$a' = m, r' = a$	2	00	[ Add $y_{i-1} a_{n-i}$ to obtain
$(aq)' = a \cdot m + (rq)$	4	01	
short loop	5	01	
$a' = \left( \frac{(aq)}{m} \right)_R$	2	00	$F_n(x) = y_n / y_{n-1}$

This program requires 4 words, and its inner loop is exactly two words.

### 3.16.4 Reverse the Digits of a Word

Suppose that  $x$ , the number in the accumulator, has digits  $x_0, x_1, \dots, x_{51}$ . It is required to form instead  $y$ , the number whose digits are  $x_{51}, x_{50}, \dots, x_0$ .

	F	B	C	N	
	$b' = -N$	4	10	52	
→	$(aq)' = 2^{-M}(aq)$ logical	1	00		shift 1 right
	$a' = m, m' = a$	2	00		exchange
	$(aq)' = 2^M(aq)$ logical	1	00		shift 1 left
	$a' = m, m' = a$	2	00		exchange
└	short loop	4	01		

The program requires 1-3/4 words: one long instruction and 5 short ones. The logical shifts are used in order not to set overflow during left shifts. If the exchange instruction were not available, the inner loop would require two more instructions, and it would be necessary to use  $R_3$  as well as  $R_2$ .

### 3.16.5 Sideways Addition

Count the number of ones in the word in Q and place the result in  $B_4$ .

	F	B	C	N	
	$b' = N$	4	10	0	clear $B_4$
	$b' = -N$	5	10	52	set the count
→	clear A, $(aq)' = 2^M(aq)$ logical	1	00		
	$a' = a + b$	4	00		
	$b' = a$	4	00		
└	short loop	5	00		

The program requires 2 words, and the inner loop requires one word.

### 3.16.6 Square Root of Y

If  $a_0 = \frac{1}{2} + \frac{1}{2}x$  and  $a_{n+1} = \frac{1}{2}(a_n + \frac{x}{a_n})$ , then  $\{a_n\} \rightarrow \sqrt{x}$ . Furthermore, it may be shown that if  $x \geq 1/4$ ,  $a_5$  is a very accurate approximation to  $\sqrt{x}$ .

This suggests the following method for obtaining  $\sqrt{y}$  as fast as possible: define  $x = 4^n y$ ,  $\frac{1}{4} \leq x < 1$ , then  $\sqrt{y} = 2^{-n} a_5$  where  $\{a_i\}$  are defined as before.

The only difficult operation is " $4^n$ ", since the computer standardizes by 2 and not 4, and most of the complexity of the program which follows is due to this type of standardization. Suppose that  $y$  is in A.

F	B	C	N	
$b' = -N$	4	00	5	
$b' = N$	5	00	0	
$a' = 2^k a, b' = b - k$	5	00		
$m' = a$	2	00		
$b' = -b_N$	5	00	5	
$a' = b$	5	00		
$(aq)' = 2^{-M}(aq)$	1	00		Halve
$b' = a$	5	00		
Clear A, $(aq)' = 2^M(aq)$	1	00		Double
$b' = a$	6	00		
$a' = m$	2	00		$2^k y \rightarrow A.$
$a' = 2^{-M} a$	6	00		
$m' = a$	2	00		x
$a' = 2^{-M} a$	1	00		
$a' = a + 2^{-M}$	1	00		$a_0$
$m' = a$	3	00		stores $a_i$
$a' = m$	2	00		x
$a' = \begin{pmatrix} a \\ m \end{pmatrix}_R$	3	00		
$a' = a + m$	3	00		
$a' = 2^{-M} a$	1	00		
short loop	4	01		
$a' = 2^{-M} a$	5	00		

The program requires  $6\frac{1}{4}$  words, and the inner loop is  $1\frac{1}{2}$  words.

### 3.16.7 Matrix Multiplication

Let  $\|a_{ij}\|$ ,  $\|b_{ij}\|$  be  $m \times n$  and  $n \times p$  matrices stored by rows beginning at memory locations  $A_0$ ,  $B_0$  respectively. It is required to store the product  $\|c_{ij}\|$  by rows in locations beginning at  $C_0$ , given that

$$b_4 = A_0$$

$$b_5 = B_0$$

$$b_6 = C_0$$

$$b_7 = m$$

$$b_8 = n$$

$$b_9 = p$$

and  $b_{15} = \text{link}.$

The B-registers will be used for the following purposes:

$$b_4 = \text{address of } a_{ij}$$

$$b_5 = \text{address of } b_{jk}$$

$$b_6 = \text{address of } c_{ik}$$

$$b_7 = (m)$$

$$b_8 = n$$

$$b_9 = p$$

$$b_{10} = (n)$$

$$b_{11} = (p)$$

$$b_{12} = np-1$$

$$b_{15} = \text{link},$$

where a number in parenthesis indicates a counter whose initial value is that number, and whose final value is zero.  $b_{10}$  is the count for each scalar product,  $b_{11}$  is the count across any one row, and  $b_7$  is the over-all count.

	F	B	C	N	
$a' = b$		8	00		
$q' = b$		9	00		
$(aq)' = m \cdot q$		0	00		np-1 to $b_{12}$
$b' = q$		12	00		
$b' = b - 1$		12	00		
$b' = b_N$		11	10	9	
unconditional jump		0	$C^{(4)}$	N	enter loop
----		--	--	-	waste 1/2 word
$b' = b - b_N$		4	10	8	reset $A_0 + kn$
$b' = b_N$		10	10	8	
Clear AQ		0	00		
$a' = m, r' = a$		4	01		
$(aq)' = m \cdot a + (rq)$		5	00		inner loop 1-1/4 words
$b' = b + b_N$		5	10	9	
$b' = b - 1$ jump unless 0		10	01		
$m' = (aq)_R$		6	01		store $c_{ik}$
$b' = b - b_N$		5	10	12	
$b' = b - 1$ jump unless 0		11	C	N	row count (p)
$b' = b - b_N$		5	10	9	
$b' = b - 1$ jump unless 0		7	C	N	over-all count (m)
obey link		15	00		

The entire program requires 7-3/4 words including calculating the constants for this triple count. If it were desired to preserve the contents of the B-lines, about 2 more words would be required.

4. The notation C N is used here to mean an address indicated by the arrow which would probably be denoted symbolically by the programmer, and assigned by the input routine.

## APPENDIX TO CHAPTER 3

The design of a computer with a 1.5 $\mu$ s main memory depends critically on what other equipment is available, and the cost of building two memories, one for data and one for instructions, or a memory with multiple word simultaneous read-out. It seems quite possible that work now going on in memory development will lead to faster memories. If this occurs the design described above will be modified.

The computer described may be thought of as having a "tactical" data-anticipation system. Advanced control, with no help from the coder, scans a few instructions ahead, and tries to obtain operands before they are required by arithmetic control. A "strategic" data-anticipation system, on the other hand, transfers blocks of data and instructions from the main memory into a large, fast, buffer memory, long in advance of their use. For this, the coder must somehow indicate what data are wanted, and where they are to be put.

An earlier design based on a 64-word rapid-access buffer memory and the simultaneous transfer of 4 words between the main memory and this buffer memory will now be described. Its description is included because it indicates an alternative method of mitigating the memory access problem. The detailed description given refers to a computer with an internal memory of 4096 40-bit words. These numbers differ from those now thought desirable for a computer. However, since the material is being included for illustrative purposes it was not thought necessary to modify these numbers.

### 3.17 Memory

- C: Slow memory 4,096-word core
- D: Fast memory 64-word diode capacitor

The 64-word memory D will be divided into 8 blocks of 8 words each. Reading between C and D will be by 4-word half blocks.

The 8 blocks of D will be divided into 3 categories:

1. Block 0 for temporary storage,
2. Blocks 1 and 2 for instructions,
3. Blocks 3, 4, 5, 6 and 7 for data, constants, etc.

Transfers to blocks 0, 3, 4, 5, 6 and 7 will be made by half block transfers, 4 words at a time. Transfers to blocks 1 and 2 will be made by pairs of 4-word transfers so that a complete block is filled by a single transfer order.

The 512 blocks of C will be addressed in half-block groups of 4 words designated by the 10 most significant bits of a 12-bit register.

### 3.18 Execution of Instructions

All instructions are executed from the 16 locations (32 instructions) of  $D_1$  and  $D_2$ . Instructions are transferred into these blocks in pairs of 4-word transfers so that a single transfer instruction brings in 16 new instructions. The control counter is a 12-bit counter that indicates the location in C of the next instruction pair. The least significant octal digit of the control counter gives the position within the 8-word block of C and  $D_1$  while the 3 most significant octal digits identify the block (1 out of 512) in C.

A flipflop  $F_1$  is needed to specify whether the next instruction pair will be taken from  $D_1$  or from  $D_2$ . Whenever the least significant octal digit of the control counter is zero, indicating the beginning of a new block,  $F_1$  is changed and a new block of 8 words is read into the  $D_1$  designated by  $F_1$  ( $D_1$  or  $D_2$  according as  $F_1$  is 0 or 1, respectively).

A second flipflop,  $F_2$ , is required to inhibit the transfer of new instructions from C to D if a loop of instructions is being repetitively executed. The control of  $F_2$  is by means of D-jump instructions, i.e., jump instructions referring to instructions already in D. (For details see the discussion of jump instructions.) In ordinary operation flipflops  $F_1$  and  $F_2$  agree and are changed together as determined by the control counter. When a D-jump occurs  $F_1$  is set



to designate the block of the new instruction to be executed, but  $F_2$  is not changed. As long as  $F_1$  and  $F_2$  disagree, the carry from the least significant octal digit of the control counter is inhibited and no new transfer from C to D is allowed. When  $F_1$  and  $F_2$  agree once more, the normal sequencing mode is resumed.

### 3.19 Jump Instructions

The ideal jump instruction would always refer only to a core location but would sense when the wanted instruction is in D and in such a case jump without requiring an access to C. If, however, the wanted instruction is not in D, the 8-word block containing it is transferred to  $D_1$ , and  $F_1$  and  $F_2$  are set to zero.

The sensing of the presence of the wanted instruction in D can be done for certain cases by adding another 3-bit counter and making a comparison between the jump address, the control counter and this 3-bit counter which, in conjunction with  $F_1$ , is a 4-bit counter specifying a position in  $D_1$  and  $D_2$ . If the jump address is less than the control counter by no more than the 4-bit counter, the wanted instruction is in D. If the jump address is greater than the control counter, the wanted instruction may or may not be in D and we cannot tell without additional equipment.

Therefore two kinds of jump instructions will be provided:

1. C-jumps which have 12-bit addresses and cause the 8-word block of C containing the wanted instruction to be placed in  $D_1$ .
2. D-jumps which have 4-bit addresses and cause a jump within  $D_1$  and  $D_2$ .

The C-jumps may or may not have the automatic sensing described above, depending upon cost, time, and difficulty experienced by the programmer in keeping track of instructions in D.

### 3.20 B-Instructions and Block Transfers of Data

As mentioned in section 3.17, the memory D will be divided into 8 blocks of 8 words. There will be a connection between the B-lines and the transfer of words from C to  $D_3$ ,  $D_4$ ,  $D_5$ ,  $D_6$  and  $D_7$ . This connection is established through the instructions used to set the B-lines, of which there are eight designated  $B_0$ ,  $B_1$ , ...  $B_7$ . Such instructions have a 3-bit B-designation and a 12-bit address:

An instruction setting  $B_i$  ( $i = 3, 4, 5, 6, 7$ ) to  $n$  may cause the 4 words of the half-block of C designated by the 10 most significant bits of  $n$  to be placed in the first or second half of  $D_i$  depending on whether the least significant of the 10 bits is 0 or 1. The position in the half-block of  $D_i$  of the word designated by  $n$  will be given by the two least significant bits of  $n$ .

Another way to view the transfer is that the most significant 3 octal digits of  $n$  specify one of 512 8-word blocks of C while the least significant octal digit of  $n$  specifies a position both in the designated block of C and in  $D_i$ .

Whenever the modified address is such that the least significant octal digit is 3, indicating that the first half of  $D_i$  is used up, a new access may be started to fill the second half from C. Similarly, if the least significant octal digit of the modified address is 7, an access may be started to fill the first half of  $D_i$  from C. Thus the block is automatically kept filled if desired and 4 back values of current data are always available after the usual starting procedure. Similarly the current block can be transferred from D to C.

One bit of the B-instruction will be used to inhibit the automatic transfer from C to  $D_i$  when desired. This makes it possible to use the designated  $B_i$  as a counter and to reduce the number of transfers from C.

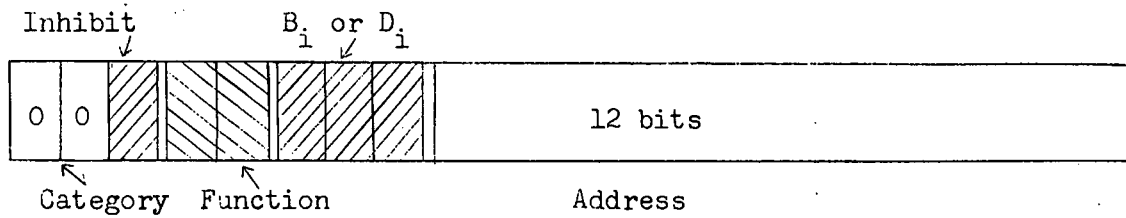
### 3.21 Basic Form of Instructions

The word length is assumed to be 40 bits with two 20-bit instructions per word. Instructions are divided into four basic categories which are distinguished by two bits:

- 00 B-line
- 01 Transfer and C-Jump
- 10 Arithmetic and D-Jump
- 11 Input-Output

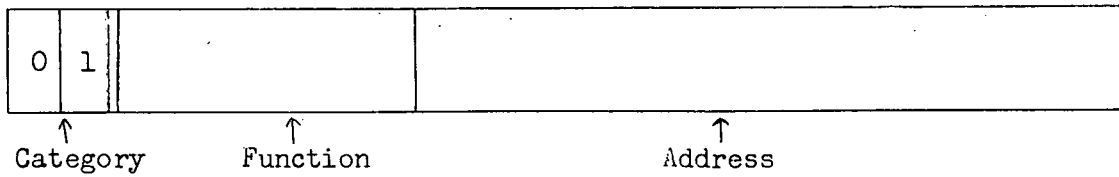
### 3.22 B-Line Instructions

These instructions will have a 12-bit C address, a 3-bit designation for  $B_i$  and  $D_i$ , a bit to inhibit the automatic transfer if desired, two bits for the function and two bits for the category.



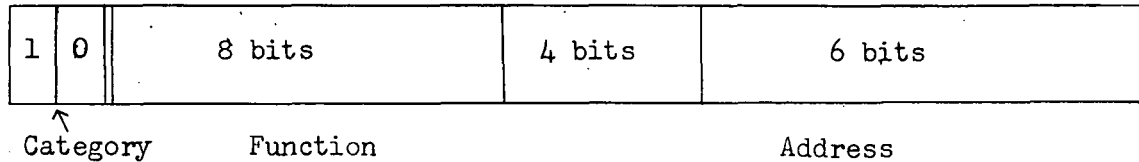
### 3.23 C-Jump Instructions

These instructions will have a 12-bit C address, a 2-bit category number and 6 bits for the function.



### 3.24 Arithmetic Instructions

These instructions will require a 6-bit address (3 bits to designate  $D_i$  and  $B_i$  and 3 bits to give position in  $D_i$ ), 2 bits for the category and 8 bits for the function. This leaves 4 bits unassigned. One of these will be used to augment the 6-bit address to permit specifying a shift of up to 127 places and the other 3 are left unassigned.



### 3.25 Input-Output Instructions

No consideration has been given to these.

### 3.26 Register Arrangements

The assumption is made that only A and Q are used, but the interconnections are different from those in the Illiac. The following things are assumed:

1. Certain instructions cause automatic transfers between A and Q, including interchanging them.
2. In most multiply instructions one operand is in A and is transferred to Q before the multiplication begins.
3. In division the dividend is either A or AQ and the quotient is put in A.
4. It is possible to shift A without shifting Q.
5. The "add product" instruction is an exception to the use of A and Q and requires one extra memory register.

### 3.27 Instruction Code

The examples use a notation that is self-explanatory except for the arithmetic instructions, which are interpreted as follows. The two digits are octal digits, the first referring to a block  $D_i$  and B-line  $B_i$  ( $i = 3, \dots, 7$ ) or block alone if  $i = 0, 1, 2$ . The second digit refers to position in the block. Thus, for example, the instruction

42  $\longrightarrow$  Acc

means that after the position (here 2 in the instruction) has been modified by  $B_4$ , place the contents of the appropriate position of block  $D_4$  in the accumulator.

The symbols Tc and Tu refer to conditional and unconditional C to D transfers, a conditional transfer being an automatic one dependent upon the modified address as described on page 71. A conditional transfer refers to the next block in C while an unconditional transfer refers to the block specified by the modified address. Similarly Sc and Su refer to conditional and unconditional writes from D to C.

The instructions used in the examples have been the following ones. It is expected, of course, that there will be many others.

1. B-line instructions
  - a. Set  $B_i$  positively or negatively
  - b. Add or subtract to or from  $B_i$
  - c. Count  $B_i$
  - d. Jump if  $B_i \neq 0$  or if  $B_i = 0$
2. Arithmetic instruction
  - a. Clear add and subtract
  - b. Hold add and subtract
  - c.  $C(n) \cdot A$
  - d. Add product

- e.  $C(n) \cdot Q$ , rounded or unrounded
  - f.  $C(AQ) + C(n)$
  - g. Shift
  - h. Store, rounded or unrounded
3. Jump instructions
- a. Jump if  $C(A) \geq 0$  or if  $C(A) < 0$

### 3.28 Programming Examples

#### Matrix Multiplication

$$C_{ij} = \sum_k a_{ik} b_{kj} \quad i, j, k \text{ in } 0 \leq t \leq n - 1$$

A stored by rows, then  $a_{ik}$  in  $(a + ni + k)$

B stored by columns, then  $b_{kj}$  in  $(b + nj + k)$

C stored by rows so  $C_{ij}$  goes to  $c + ni + j$

$$\text{Hence } \sum_k N(a + ni + k) N(b + nj + k) \longrightarrow c + ni + j$$

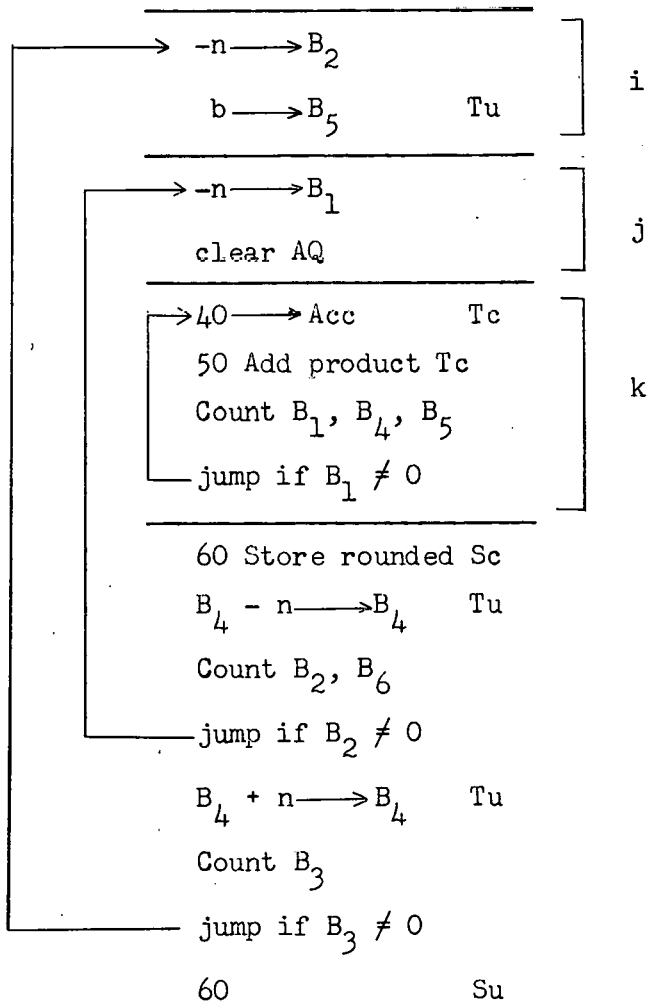
then  $j + 1 \longrightarrow j$  i remains constant, unless  $j = n$ , then  $0 \longrightarrow j$   $i + 1 \longrightarrow i$ .

Given indices  $a'$   $b'$   $c'$  where  $a'$  begins at  $a$ ,  $b'$  at  $b$ ,  $c'$  at  $c$  and counts

$k - n$	$B_1$	$B_4$	$a'$
$j - n$	$B_2$	$B_5$	$b'$
$i - n$	$B_3$	$B_6$	$c'$

$-n \rightarrow B_3$   
 $a \rightarrow B_4$  Tu  
 $c \rightarrow B_6$

Can be trivially altered  
 to do  $(m \times n) \cdot (n \times p)$ .



Evaluation of Polynomial

$$P(x) = \sum_{i=0}^n a_i x^{n-i}$$

define  $b_0 = a_0$

$$b_{i+1} = x b_i + a_{i+1}$$

$$b_n = P(x).$$

Suppose  $x$  in Acc and  $a_i$  in  $a + i$  when  $a, n$  are known

Store Acc in 00

$-n \rightarrow B_1$

$a \rightarrow B_3$  Tu

$30 \rightarrow \text{Acc}$  Tc

count  $B_3$

$\rightarrow 00$  multiply  
 $30$  Add Tc  
 count  $B_1, B_3$   
 jump if  $B_1 \neq 0$

Square Root

$y$  in AQ

define  $a_1 = 1/2 + y/2$

$a_{i+1} = (y/a_i - a_i)/2 + a_i$  stop at  $k$  when  $y/a_k - a_k \geq 0$ .

10 1/2                    || constant

11 Acc  $\rightarrow$  00            ← entry

    Q  $\rightarrow$  01

    shift right 1

    add 10

    jump

    02 add

$\rightarrow$  02 store

    00  $\rightarrow$  A

    01  $\rightarrow$  Q

    02 divide

    02 subtract

    shift right 1

    jump if A < 0 ←



Relaxation

a o  
b o o o  $1/4(N(a) + N(b) + N(b + 2) + N(c)) \longrightarrow b + 1$   
c o stop after doing n.

a  $\longrightarrow$  B<sub>3</sub> Tu  
b  $\longrightarrow$  B<sub>4</sub> Tu  
c  $\longrightarrow$  B<sub>5</sub> Tu  
(42  $\longrightarrow$  Acc Tu)  
-n  $\longrightarrow$  B<sub>1</sub>  
-  $\longrightarrow$  30  $\longrightarrow$  Acc Tc  
40 add  
42 add Tc  
50 add Tc  
shift right 2 (single length)  
41 store rounded S<sub>c</sub>  
Count B<sub>1</sub>, B<sub>3</sub>, B<sub>4</sub>, B<sub>5</sub>  
jump if B<sub>1</sub>  $\neq$  0  
(40  $\longrightarrow$  Acc Su)

Relaxation: Extrapolated Liebmann

a o  
b o o o  
c o

$$N(b + 1) + 1/4(N(a) + N(b) + N(b + 2) + N(c) - 4N(b + 1)) (1 + \epsilon) \longrightarrow b + 1$$

10	ε	]	constant
11	a → B <sub>3</sub>		Tu
	b → B <sub>4</sub>		Tu
	c → B <sub>5</sub>		Tu
	(42 → Acc		Tu)
	-n → B <sub>1</sub>		
	→ 30 → Acc		Tc
	40 add		
	42 add		Tc
	50 add		Tc
	shift right 2		
	41 subtract		
	00 store rounded		
	10 multiply		
	00 add		
	41 add		
	41 store rounded	S <sub>c</sub>	
	count B <sub>1</sub> , B <sub>3</sub> , B <sub>4</sub> , B <sub>5</sub>		
	jump if B <sub>1</sub> ≠ 0		
	(40 → Acc		Su)

To Merge 2 Sequences of Numbers

Given k numbers beginning at a

ℓ numbers beginning at b

store the merged list beginning at c.

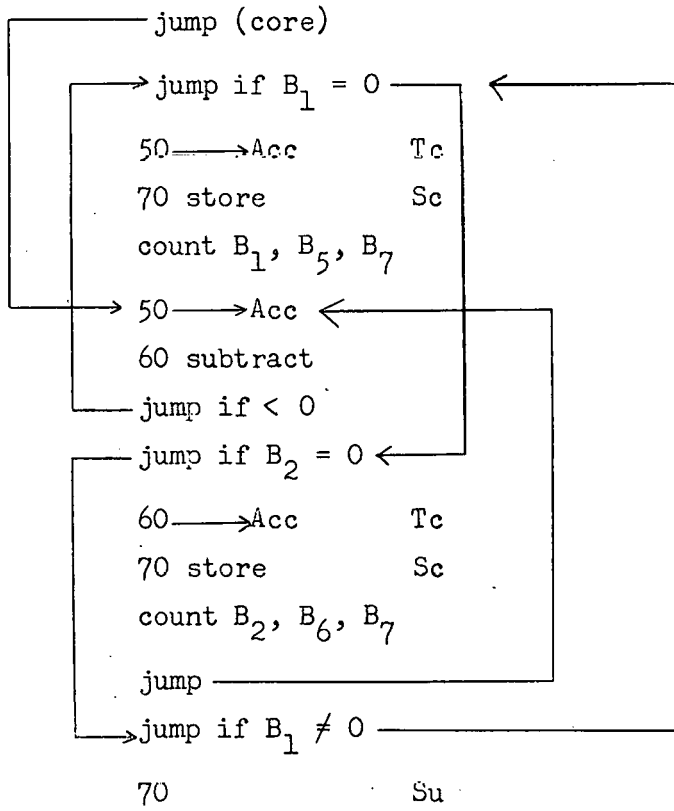
a → B<sub>5</sub> Tu

b → B<sub>6</sub> Tu

c → B<sub>7</sub>

k → B<sub>1</sub>

ℓ → B<sub>2</sub>



## CHAPTER 4

### BASIC CIRCUITS

#### 4.1 Introduction

Even assuming that only lumped constant circuits are considered, it is at once realized that the question, "Which are the best circuits for a computer doing a given job?" has no simple answer. Apart from the tremendous difficulty of comparing the diversity of circuits satisfying imposed conditions, it is not even clear what is meant by "best circuits". The decisive factor may be cost, or size and simplicity (low cost and simplicity do not necessarily go together); then the interest may be speed and most of all, perhaps, reliability. Ease of servicing and standardization, too, may be on the list. A preliminary statement of the position of the Digital Computer Laboratory would be to say that at least for the arithmetic unit and the control, the utmost in speed--compatible with a given design philosophy and rather severe standards of reliability--has been aimed at. This does not imply that complexity and cost were neglected, but here the limitation was the rather vague one of being "reasonable".

In other sections of this chapter the general philosophy of the Digital Computer Laboratory will be discussed, i.e., the reasons for choosing "direct-coupled, asynchronous non-saturating transistor circuits". It is, of course, true that the choice of circuits, design procedure, etc., is affected by the past experience of the group. There seem to be, however, good arguments in favor of our choice and these will be explained in some detail in the following sections.

Another introductory remark may not be out of place. It is well known that a complete computer can be designed using a single type of basic logical element, i.e., the AND-NOT (Sheffer-Stroke) element. Even storage elements can be produced from AND-NOT's. A more common and more flexible set is one starting with AND, OR, NOT and a symmetric flipflop of the Eccles-Jordan or some equivalent design. To the set of basic logical elements,

one invariably has to add purely electronic elements in the form of power amplifiers (drivers), level restorers, etc. In order to obtain maximum flexibility and speed, additional logical elements like Schmitt trigger flipflops (having only one output), EXCLUSIVE OR's, and others may be introduced. It is also quite possible to choose larger units of circuits, i.e., to design a separate circuit for every n-input, m-output box defined by a table of corresponding input and output combinations.

The set of logical elements should often be even more extensive than the augmented set just mentioned, containing such new designs as C-Elements. These are desirable in order to realize "speed-independent" circuits in the sense of Chapter 5, i.e., circuits in which information flow is not affected by race conditions. It seems especially desirable to use this kind of circuitry in the rather complex control proposed for the new machine.

Finally, mention should be made of the fact that the use of binary logic is not at all mandatory; some gates, to cite an example, are really 3-level devices, when we consider the states to be indicated by currents. One can pursue this idea of ternary logic and build elements of the flow-gating type described below. In this kind of circuitry the transmission of information (e.g., from one flipflop to another) is controlled by modifying the average potential of a whole flipflop. It turns out that this leads to a considerable economy in hardware. The speed of operation is, however, somewhat reduced.

#### 4.2 Asynchronous vs. Synchronous Operation

Computer circuits are frequently classified according to the type of control-sequencing used. In a synchronous computer the elementary operations like shifting, adding, etc., occur at fixed intervals in time, the moment of occurrence being controlled by a clock. In an asynchronous computer each operation produces an "end-signal" which in turn initiates the next operation in a list. If the production of end-signals is not too time-consuming,

an asynchronous machine will be faster. The duration of individual operations varies (even with the numbers processed) and a clocked synchronous machine must be adjusted for the maximum possible duration.

There are systems of various degrees of asynchronism ranging from those in which the times of action of a set of circuits are simulated in delay devices (i.e., in which the end-signal or reply-back signal is simply the previous end-signal delayed by a sufficient amount of time to allow the set of circuits to operate), to systems in which the operation of each set of circuits is examined by a checking circuit which gives end-signals if and only if the operation has really been performed.

A special type of completely asynchronous operation is obtained by using speed-independent circuits. In this type of circuit, information can continue to flow only when all previous elements in the chain have reacted to it. Chapter 5 gives the theory of interconnecting logical elements in such a way that the set is speed-independent. This means that individual logical elements too must be designed carefully to meet the conditions of speed-independence. In particular, last moving points (see section 4.7) should be provided in order to simplify the design procedures.

Although the absence of a clock has to be paid for in terms of circuitry, it is believed that the advantage in speed outweighs by far the additional expenditure. Furthermore, the combination of "asynchronous operation and 2-level dc-representation" explained in section 4.4 allows the machine to be stopped in any state for as long a period as is desired. Checking its operation is reduced to a simple steady-state voltage check of strategic points. Also, an asynchronous circuit is probably more reliable, since changing time constants cannot influence its operation.

Finally, it should be mentioned that in a very fast computer the time for information to travel from one part of the machine to another will compare to the binary operation times of the circuits. Signals are delayed by at least 3.3 nsec per meter. In asynchronous circuitry a knowledge of this information transit time is not required to assure correct operation.

### 4.3 Direct-Coupling vs. AC-Coupling

Computer circuits can be ac-coupled or dc-coupled. In the first case a capacitor or a transformer is used to connect one logical element to the next. In the second case only resistive networks are used as coupling elements. It should be noted that dc-coupling is only of interest if each element is also dc-coupled internally.

The Digital Computer Laboratory has followed the Institute for Advanced Study by working with circuits which are dc-coupled because it is believed that circuits of this type have advantages of reliability and serviceability without paying any direct price in speed. These circuits do require more engineering and possibly more hardware than ac-coupled circuits of comparable speed, especially because of drift problems. These, however, are by no means as serious in a switching circuit as in linear amplifiers.

Although dc-coupling does not necessarily imply the use of the two-level dc-representation explained in the next section, this representation certainly seems to be the natural complement of direct-coupling. This is especially true when we consider testing a single logical element. A given combination of dc-inputs then always corresponds to a predetermined combination of dc-outputs. Checking the operation of a logical element or a group of such elements therefore becomes quite simple.

### 4.4 Two-Level DC vs. Pulse Representation

The method of representation of binary states by electrical quantities (voltages or currents) is another important characteristic of circuits. In synchronous ac-coupled circuitry the states are usually represented by the presence or absence of a pulse at given times, these times being determined by a "clock". In magnetic recording this has been termed "return-to-zero" representation. In asynchronous dc-coupled circuitry it is usual to represent the binary states 0 and 1 by voltage or current levels, or to be

more precise, by voltage or current bands. This corresponds to the "non-return-to-zero" representation of magnetic recordings.

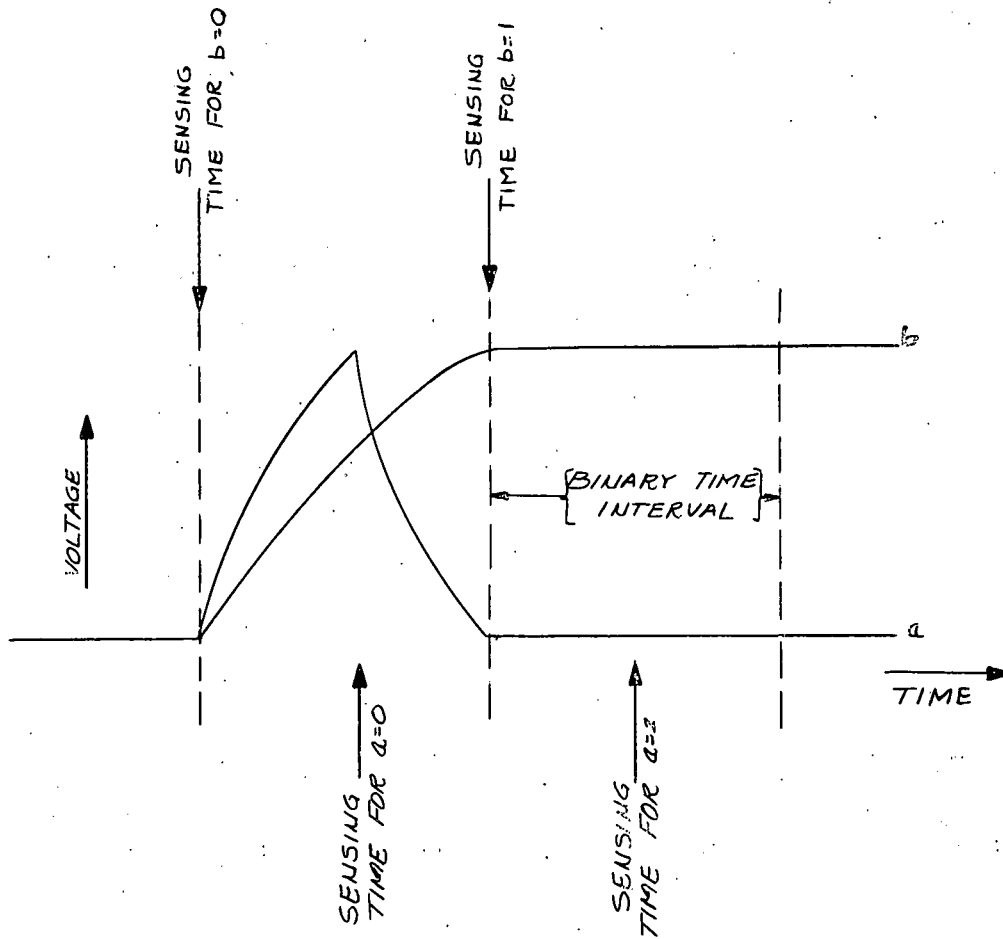


Figure 4.1

Representation of binary numbers by voltages:  
 a) pulse representation, b) two-voltage-level representation. This is drawn for the case of minimum sensing time, i.e., the time required for sensing the voltage state approaching zero.

The difference between the pulse representation and the two-voltage level representation may be illustrated with Figure 4.1. The pulse signal must be able to go up and down during a clock period. Thus, if a pulse represents a 0 and the absence of a pulse represents a 1 (the opposite convention being possible too), the waveform would be as shown and the



duty cycle of the switching element in the case of a sequence of 0's would be  $1/2$ . If the two-voltage representation is used, 0 corresponding to the lower voltage level, and 1 to the higher one, the signal as shown in b) changes from one state, 0, to the other state, 1, in the same time interval. Since the voltage excursion in amplitude is the same for the two cases, while the excursion time is twice as long for the two-voltage representation, the charging current and hence the power switched need be only half as great for the two-voltage representation. However, the switching element must be capable of being "on" with a duty cycle of 1. Thus, only  $1/2$  the power is available from the switching element in this case compared to the pulse system. These simple considerations show that the speed of a circuit is to a first order, the same for the two systems just described.

In practice, the pulse system requires a space between pulses to allow for timing tolerance. No space is shown in Figure 4.1 which is a limiting case of shortest possible times. In addition, some time is required to sense the existence of a signal and this must be added to the time tolerance. The two-voltage representation does not have a time tolerance requirement, but it does require a sensing time greater than zero. If the sensing time required for each system were the same, the time for a binary information period would be shorter for the two-voltage case by the amount of the time tolerance.

#### 4.5 Transistors vs. Tubes

As was indicated in the introduction, the aim of the Digital Computer Laboratory has been to design the fastest dc-coupled lumped-constant circuits, using presently available components. Both vacuum tubes and transistors were considered for the triode elements. Circuits using transistors appear to be faster than vacuum tube circuits, and the circuits to be presented later are based upon the use of graded-base transistors.

To see why transistor circuits are faster than tube circuits one can observe that in circuits the total time required to change a circuit element from one state to another is due to a circuit time and a device operation time. The circuit time will be considered first.

In any simple circuit the time of transition from one potential state to another is proportional to the load capacitor size and the potential difference between states and inversely proportional to the current charging the capacitor. Transistors and vacuum tubes may generally have load capacitors of about the same size in an operating circuit: usually 10 to 20  $\mu\text{f}$ . The voltage difference between states must be at least the value necessary to switch the device: about 0.5 volt for transistors and 5 volts for tubes. However, the switching signal must be generated at the collector or plate potential respectively, and usually an attenuation exists between the point of generation and the switching point. This attenuation and the tolerances of the units frequently, and perhaps usually, require the generation of a signal about 10 times the switching signal or about 5 volts for transistors and 50 volts for tubes. The currents switched by transistors and tubes are about the same: 5 ma.

One can use the parameters given above to obtain an idea of the time necessary for a circuit to change from one binary state to the other:

$$t = k \frac{CV}{i}$$

where

t = time

C = load capacity

V = voltage swing

i = charging or switching current

k = constant close to 1 but in the range  $1/2 \rightarrow 2$  depending on the complexities of the circuit.

Circuit time  
for transistors

$$C = 10 \cdot 10^{-12} \text{ farad}$$

$$V = 5 \text{ volts}$$

$$i = 5 \cdot 10^{-3} \text{ ampere}$$

$$t = \frac{10 \cdot 10^{-12} \cdot 5}{5 \cdot 10^{-3}} = 10^{-8}$$

$$t = 10 \text{ } \mu\text{s}$$

Circuit time  
for tubes

$$C = 10 \cdot 10^{-12} \text{ farad}$$

$$V = 50 \text{ volts}$$

$$i = 5 \cdot 10^{-3} \text{ ampere}$$

$$t = 100 \text{ } \mu\text{s}$$

The device switching speed must be added approximately to the above times. For vacuum tubes this is equal to the transit time of the electrons from grid to plate, a time of the order of 5  $\mu\text{s}$ . For transistors the time for switching is complicated by many factors. For small signal operation the reciprocal of the 3 db down- $\alpha$ -cutoff frequency may be used provided the transistor is not at any time in saturation<sup>(1)</sup>. For recent graded-base transistors this frequency is more than 200 mc per second and hence the time is less than 5  $\mu\text{s}$ . Thus, the total switching time for a transistor circuit would be about  $10 + 5 = 15 \mu\text{s}$ , while the corresponding time for a vacuum tube circuit is about 105  $\mu\text{s}$ .

The elimination of saturation, which is mandatory for high-speed operation, as mentioned above, often requires the addition of some circuitry as does the fact that the input impedance of transistors, even in the grounded emitter configuration, is low at all frequencies. The latter fact implies the presence of emitter followers after practically every dc-stepdown network. This, together with bumping diodes to re-standardize signals, makes for a somewhat greater complexity of the transistor logical elements as compared to those using tubes.

---

(1) S. R. Ray: "The Effects of Saturation on the Switching Response of Common-Emitter Amplifiers using Diffused-Base Transistors", Digital Computer Laboratory File No. 208, January 15, 1957.

Another problem should also be brought up. The lower power-handling capacity of transistors means that the ratio (noise-power)/(switched-power) could be higher than that of tubes. Luckily, the average impedance of transistor circuits is only about 10% of that for tubes and furthermore, the smaller physical size of the active elements makes for shorter leads. The noise-power (= average of square of noise voltage/impedance) is, therefore, probably smaller in the same ratio as the power-handling capacities.

#### 4.6 Reliability and Feasibility of a Transistorized Asynchronous DC-Coupled Machine

In order to investigate feasibility and reliability questions, a model computer using 752 transistors and 498 diodes was built by the Digital Computer Laboratory.<sup>(2)</sup> It operates using 4 words (2 bits for the instruction add, subtract, multiply, divide; 4 bits for the number to be operated on) taken from a nondestructive core memory and processing them in a six-register ( $\overline{AA}$ ,  $\overline{QQ}$ ,  $R^3R_3$ ) arithmetic unit. The circuitry used 1 mc transistors (GE 2N43 series and Ti 202 series) in npn-pnp combinations. The tolerances on supply voltages were 5% and on resistors 3%.

The model executed cycles of four orders, each one referring to the last contents of the accumulator and a number stored alongside the instruction. This procedure made it necessary to precede multiplication automatically by an  $A \rightarrow Q$  shift and to follow (again automatically) each division by a  $Q \rightarrow A$  shift. The 4 words stored in the memory were always chosen in such a way that the cycle of four orders brings the accumulator A back to its initial state.

This model machine had many successful runs of several hundred hours each. The average lifetime of transistors of the above type, extrapolated from the failure rate in these runs, turned out to be about 100,000 hours, as compared to less than 20,000 hours for tubes in ILLIAC.

---

(2) W. J. Poppelbaum, F. M. Lurie, G. A. Metze: "TRANCE, A Direct-Coupled, Asynchronous Computer-Model Using One Microsecond Transistor Circuits", Digital Computer Laboratory Report No. 73, December 3, 1956.

Two points should be mentioned: First, the fact that a given transistorized logical circuit may use as many as twice the number of active elements the corresponding tube circuit would use, makes a 1:2 ratio of tube lifetime / transistor lifetime necessary in order to break even. From the quoted figures it is seen that the model was only 2.5 times as reliable as a tube model doing the same job. Second, the cited figures do not refer to the presently used GF-45011 transistors (see section 4.12). Preliminary investigations show, however, that the latter type stands up well under actual operating conditions.

#### 4.7 Tolerances, Critical Levels and Discrimination Levels

Figure 4.2 shows a "single input-output" flipflop. The operation will be discussed for the strongly idealized case where both transistors have  $\alpha = 1$  and where there is no voltage drop between emitter and base when a transistor conducts. Furthermore it will be supposed that the output impedance  $R_o$  is very small compared to the input impedance.

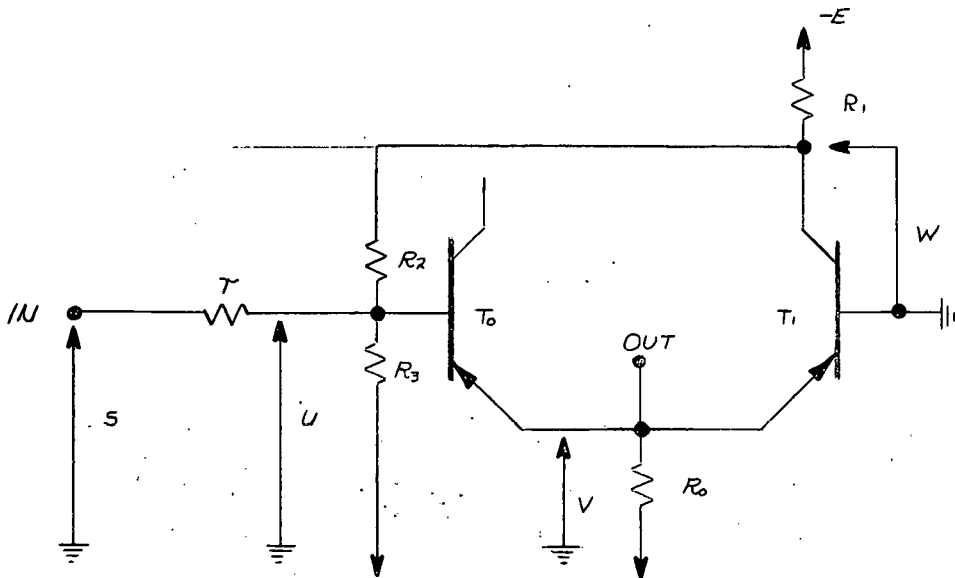


Figure 4.2  
Single Input-Output Flipflop

The figure shows the nomenclature used. By definition the flipflop is in the 0 state when transistor  $T_0$  conducts and in the 1 state when  $T_1$  conducts. As long as no input signals are impressed the circuit (for the two states respectively) can be redrawn as in Figure 4.3. It is very easy

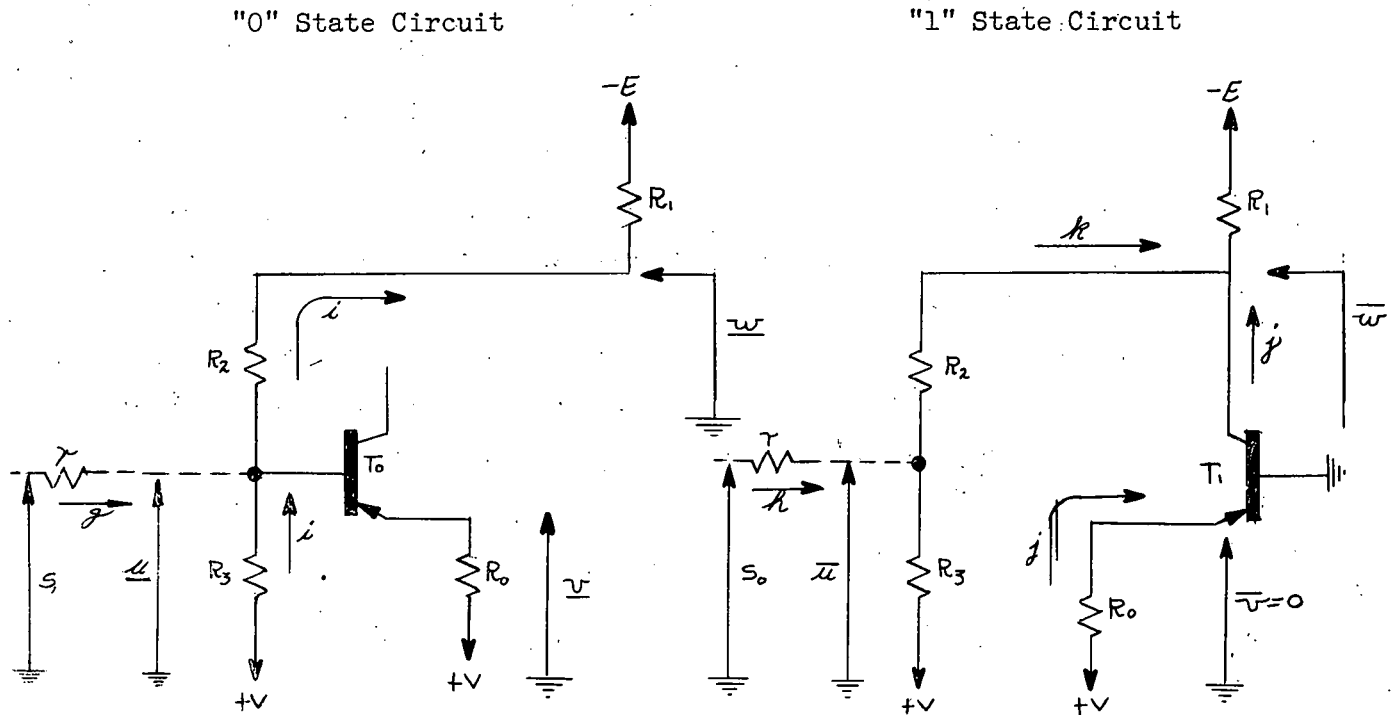


Figure 4.3  
Active Parts of a Single Input-Output Flipflop

to determine the voltage  $u$ ,  $v$ ,  $w$  at the three "cardinal points", base of  $T_0$ , OUT, and collector of  $T_1$  for both states:

$$\left. \begin{array}{l} \text{For the} \\ \text{0 state} \end{array} \right\} \begin{cases} \underline{u} = V \left( \frac{R_1 + R_2}{\Sigma} \right) - E \left( \frac{R_3}{\Sigma} \right) \\ \underline{v} = V \left( \frac{R_1 + R_2}{\Sigma} \right) - E \left( \frac{R_3}{\Sigma} \right) \\ \underline{w} = V \left( \frac{R_1}{\Sigma} \right) - E \left( \frac{R_2 + R_3}{\Sigma} \right) \end{cases}$$

For the  
1 state

$$\left\{ \begin{array}{l} \bar{u} = V \left( \frac{\Sigma + (\mu - 1)R_3}{\Sigma} \right) - E \left( \frac{R_3}{\Sigma} \right) \\ \bar{v} = 0 \\ \bar{w} = V \left( \frac{\mu(R_2 + R_3) + R_1}{\Sigma} \right) - E \left( \frac{R_2 + R_3}{\Sigma} \right) \end{array} \right.$$

where  $\Sigma$  is the sum  $R_1 + R_2 + R_3$  and  $\mu$  is the quotient  $R_1/R_0$ .

Consider the output voltage  $\bar{v}$ : in the 1 state it is perfectly fixed (at least in the idealized case treated here); for the 0 state however  $\bar{v}$  can take any value between two limits due to the fact that the power supplies and resistors vary - both from one unit to the next and in time: all circuit parameters have certain tolerances. If  $R$  is the "design center value" of a resistor with a fractional variation of  $x$ , in extreme cases  $R_{\max} = R(1 + x)$ ,  $R_{\min} = R(1 - x)$ . In the case  $E \gg V$  for instance

$$\bar{v}_{\min} = -E_{\max} \frac{R_{3\max}}{R_{1\min} + R_{2\min} + R_{3\max}}$$

$$\bar{v}_{\max} = -E_{\min} \frac{R_{3\min}}{R_{1\max} + R_{2\max} + R_{3\min}}$$

Suppose now that the flipflop is to be set to a new state by applying an input signal  $S$  through a resistance  $r$  (representing for instance the forward drop in a diode used to transfer the information in a gate). If it is in the 0 state, it will be necessary to apply a signal  $S_1$  which is sufficient to cut  $T_0$  off: if the circuit has sufficient amplification (i.e.  $a = \frac{R_1 R_3}{\Sigma r_e} > 1$  where  $r_e$  = emitter-base resistance), the flipflop will then proceed on its own toward the new state.  $S_1$  is determined by the condition that in the left-hand side of Figure 4.3,  $\bar{u} = 0$ . Similarly the signal  $S_0$  required to set the flipflop to zero is determined by the condition that in the right-hand side of Figure 4.3,  $\bar{u} = 0$ . This gives,

$$S_1 = -V \left( \frac{r}{R_3} \right) + E \left( \frac{r}{R_1 + R_2} \right)$$

$$S_0 = -V \left( \frac{r(R_1 + R_2 + \mu R_3)}{R_3(R_1 + R_2)} \right) + E \left( \frac{r}{R_1 + R_2} \right)$$

Visibly  $S_0 \neq S_1$  as long as  $r \neq 0$ : this means that the flipflop shows hysteresis. Again the fact that tolerances are not zero means that there are bands ( $S_1$  max,  $S_1$  min) and ( $S_0$  max,  $S_0$  min) in which the input signals necessary to trigger a flipflop must lie. Since  $S_1 > S_0$  under practical conditions, it will be sufficient to assure that  $S > S_1$  max to trigger the 1 state and that  $S < S_0$  min to trigger the 0 state.

To summarize: in order to assure triggering in a chain of flipflops, the output  $v$  of each one must swing under the worst conditions sufficiently far to trigger the next stage. This corresponds to the following inequalities:

$$\bar{v} \text{ min} \geq S_1 \text{ max} \quad (\text{here } \bar{v} \text{ min} = \bar{v} = 0)$$

$$\underline{v} \text{ max} \leq S_0 \text{ min.}$$

$S_1$  max will be called the upper critical level  $c_1$  and  $S_0$  min the lower critical level  $c_0$ . Triggering is produced by overswinging these critical levels.

By reasoning in a similar fashion discrimination levels  $d_1$  and  $d_0$  could be determined such that a signal in the band ( $d_0$   $d_1$ ) cannot possibly trigger any flipflop of the given set. Generalizing the arguments it is seen that there are output bands ( $\bar{v}$  max,  $\bar{v}$  min)  $\rightarrow$  "1" and ( $\underline{v}$  max,  $\underline{v}$  min)  $\rightarrow$  "0" as well as input bands ( $c_0$   $c_1$ ) outside of which triggering is achieved with probability one and ( $d_0$   $d_1$ ) inside of which no triggering can occur at all. If the circuit is well designed the general disposition of the bands is as in Figure 4.4.



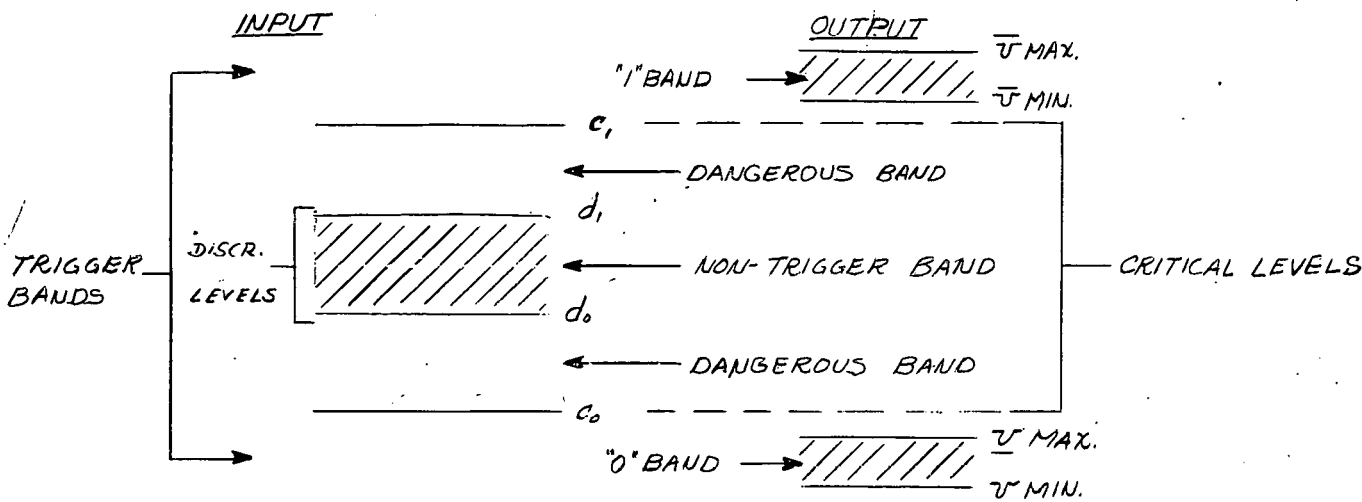


Figure 4.4  
 Input and Output Levels of a Single Input-Output Flipflop

4.8 The Last Moving Point Philosophy

Up to now only the static behavior of circuits has been considered; as soon as rapidly varying signals are taken into account, matters are complicated by the fact that the purely resistive divider networks described in the last section become combinations of resistances and capacitances, the latter being due to the stray capacitance between elements and between elements and ground. Consider for instance a divider of ratio  $k = R_0 / (R_0 + R_1)$  as in

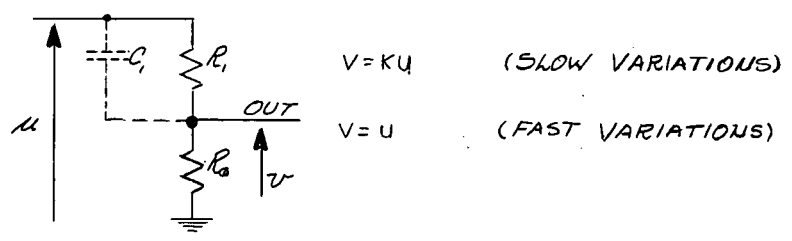


Figure 4.5  
 Voltage Divider with Stray Capacity

Figure 4.5, the resistor  $R_1$  having between its ends a capacity  $C_1$ . Then the output voltage becomes equal to the input voltage  $u$  whenever  $u$  varies fast enough. In other words: the dynamic behavior of a flipflop (or any other circuit) is different from its static behavior.

More trouble still is due to the fact that active elements like tubes and transistors behave in a way different from that predicted from static characteristics when they are used for rapidly varying signals. Take a transistor as in Figure 4.6 and examine what happens to the emitter and to the collector when the input changes suddenly from  $u_1$  to  $u_2$  <sup>(3)</sup> (both being in the conduction range). To do this three things must be noted:

- (1) As long as the emitter conducts, there is apparently present a large capacitance  $C_0$  between it and the base.
- (2) When a sudden change occurs in the emitter current (due to an impressed signal) this change is propagated through the base with a certain delay.
- (3) The stray capacitance  $C$  between collector and ground absorbs a large amount of current and the collector voltage varies far from instantaneously even when the signal has reached the collector junction.

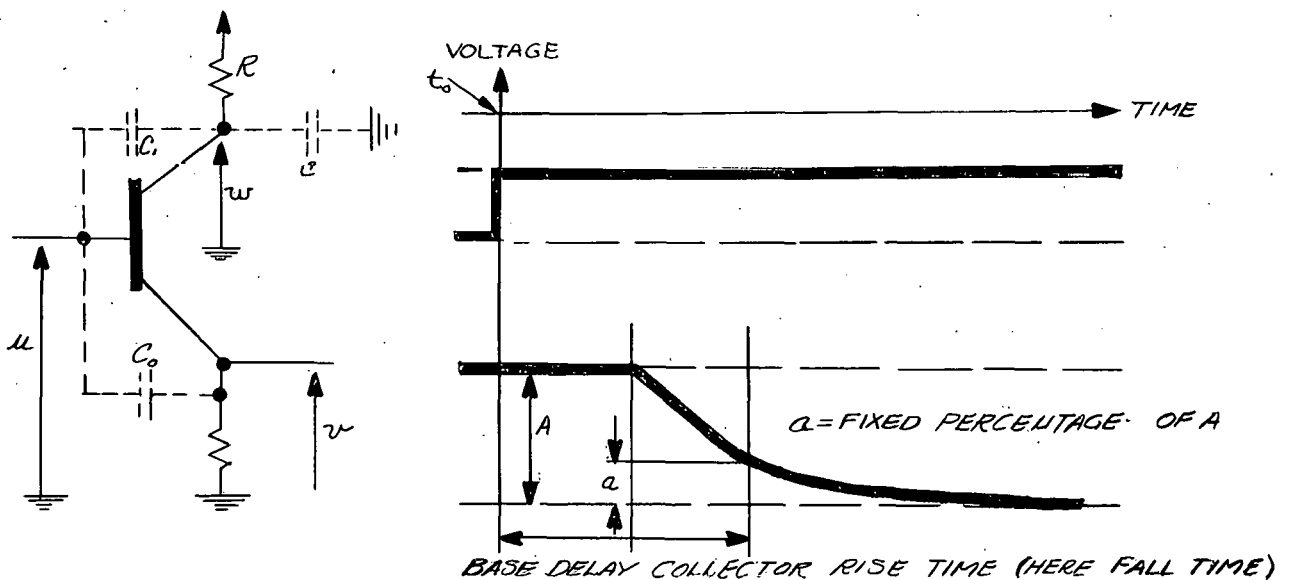


Figure 4.6

Base Delay and Collector Rise Time in a Transistor

- (3) This hypothesis eliminates difficulties due to the Miller effect caused by the capacitance  $C_1$  between collector and base.

The influence of these three factors is shown graphically in the figure on the preceding page. It is seen in particular that the emitter follows the base instantaneously. If the incoming signal varies so rapidly that the collector has no time to react before it goes back to its old value, the emitter will indicate that the transistor has reacted to the signal while in reality the collector has not had the time to follow.

Now re-examine the flipflop in the last section. Suppose that, while it is in the 1-state ( $T_1$  conducting)  $S$  is suddenly lowered. As soon as  $T_0$  starts conducting (i.e. without delay if  $S$  varies infinitely fast), the output will follow the input. If this flipflop is used to set another flipflop, it may do so before the amplifying action of  $T_1$  allows the first flipflop to reach stability, i.e. before it "holds": the new output thus does not guarantee the new state. This of course is serious in asynchronous circuitry because the outputs are used to initiate new operations.

The circuit in Figure 4.7 avoids (at least to a certain degree) the above difficulty by taking the output from a second voltage divider which indicates the new state of the collector of the amplifier rather than that of the emitter. Such a point, which by its new output gives

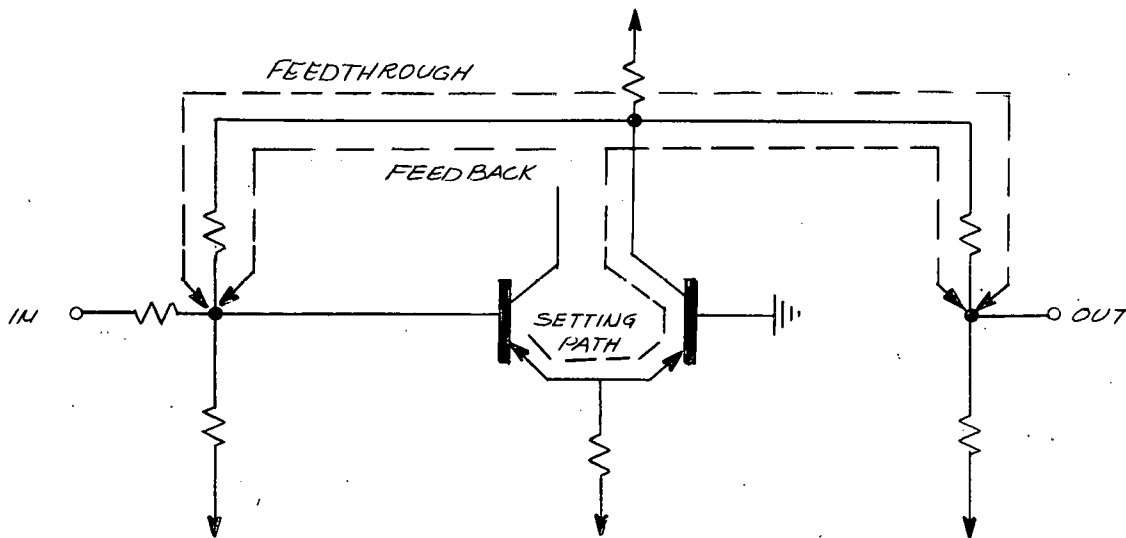


Figure 4.7  
Last Moving Point Flipflop

proof<sup>(4)</sup> of the new state of a circuit is termed a last moving point. Note however that even here there is a "feed through path" due to the fact that the input and the output voltage dividers are tied to the same collector: by designing these dividers correctly one can however "decouple" IN and OUT to such an extent that the output never reaches the dangerous bands of Figure 4.4 by feed-through action alone.

Note that in a symmetric flipflop, which has complementary sensing outputs and complementary inputs, last moving points exist only in a restricted sense. A given output terminal is generally a last moving point only for a given input and for one direction of signal change. This input and this direction of change has to be specified in each case.

#### 4.9 Characteristic Times: Switching Time and Operation Time

Synchronous circuits are often defined speedwise by specifying the frequency at which they operate. In multiphase systems this does not mean that the time in which information traverses a given logical element is equal to the reciprocal of this frequency.

In asynchronous circuits it is obviously impractical to talk about a clock frequency. The first idea is to introduce the rise or fall time of the signals, but as explained in the last section, this does not take account of the delay in the circuit before the signal begins to change. Also, and this is of the greatest importance, it is useless even to talk about times as long as no last moving points have been provided or determined. Therefore characteristic times will only be introduced for last moving point circuits. This does not mean that for a non-last moving point circuit one cannot, with caution, give a time after which it has reacted to an incoming signal. In this case it will be necessary to talk about the "estimated reaction time".

The first important characteristic time is the switching time of a circuit. This is the time which elapses between the moment when an

---

(4) It can be argued whether this proof is sufficient: it seems extremely difficult to design elements the last moving point of which indicates positively that the new state has been reached.

input signal in the form of a step function (and coming from a zero impedance generator) reaches the critical level,<sup>(5)</sup> and the moment when the output signal has gone as far as the critical level (of the following circuit) toward its new binary value. By the last section this guarantees switching of the next storage element.

The operation time is the second important characteristic time. This is the time which elapses between the moment when an input signal with the rise time characteristic of the type of circuit used (and coming from a generator of non-zero impedance in the form of precisely such a circuit) reaches the critical level, and the moment when the output signal has gone as far as the critical level toward its new binary value.

To clarify the important notion of operation time, the case of a particular circuit will be discussed. Assume that there is a set of absolutely identical last moving point flipflops, each one having identical output levels  $\bar{v}$  and  $\underline{v}$  and each one requiring triggering signals  $S_1$  and  $S_0$  respectively (this corresponds to making  $c_1 = d_1 = S_1$  and  $c_0 = d_0 = S_0$  in Figure 4.4; the discrimination levels will therefore be called  $S_1$  and  $S_0$ ). Consider how the output of one of these flipflops reacts when its input goes from  $\underline{v}$  to  $\bar{v}$ . Figure 4.8 gives the timing diagrams. It is seen that the operation time is the total time which elapses between the moment when the input starts to change (with a slope equal to that which a typical collector gives when the input has precisely this slope) and when the output reaches the value  $S_1$  necessary to trigger another stage. This time is composed of a "base delay" and a percentage of the collector rise time. Note that, thus defined the operation time varies from one set of identical circuits to another (different) set; it also can depend on the sense of the signal change, i.e. the  $0 \rightarrow 1$  operation time is not necessarily equal to the  $1 \rightarrow 0$  operation time. For a given circuit design involving tolerances it is usual to quote the average operation time for both ways, obtained by connecting a number  $n$  of flipflops in a loop closed by a NOT circuit; the period  $T$  of the oscillations setting in is then  $2n\tau + 2\tau'$  where  $\tau'$  is the average operation time of the NOT circuit. If the latter is unknown,

---

(5) The "critical level" has been defined for flipflops in the preceding section. For other elements it is defined as the level which turns over a flipflop following this element.

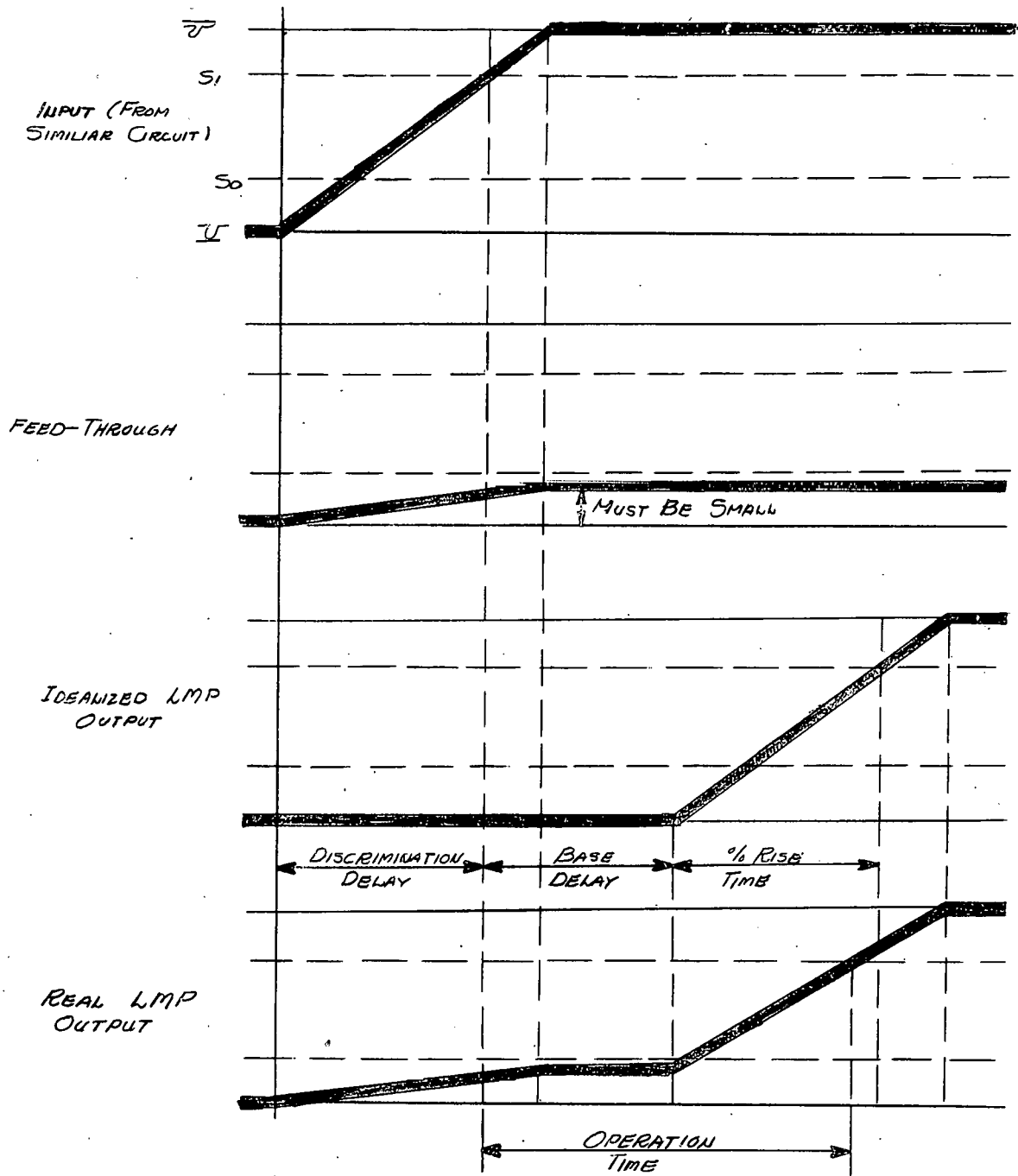


Figure 4.8

Behavior of LMP and Make-up of Operation Time

we can measure  $2(n+1)\tau + 2\tau'$  by adding one more flipflop and finding  $\tau$  by subtraction.

It should be remarked that the advantages of the average operation time are: (1) it refers to a circuit running under non-idealized conditions (i.e. not driven by a zero-impedance generator with zero rise-time), (2) it is an additive property for all circuits having the same output swing and the same discrimination levels (supposing that input impedances are high):

#### 4.10 Flow-Gating

In some of the newer circuit designs a new principle of information transfer is used which will be called "flow-gating" for short. The fundamental idea involved will be explained in a simple case. Consider a Schmitt trigger circuit using a single-supply voltage only. The input and output of such a circuit (which can be in either one of its states) can then exist at voltage levels which depend linearly on the supply voltage. In order to gate from one such circuit to another, a simple diode connection is used and the average potential of the two circuits is made different in such a way that information flows through the diodes. It turns out that the clearing, which has to precede the setting, can be accomplished automatically in the process of changing the average potential. Since information is gated by making it flow down a potential gradient (created by a gate signal which controls the supply voltage), the system is called "flow-gating".

Figure 4.9 gives the circuit diagram of the device.

The theory is as follows. For a fixed  $-E$ ,  $T_0$  acts like a grounded-base amplifier (base return voltage =  $-ER_5 / (R_4 + R_5)$  if  $\alpha = 1$ ) and  $T_1$  acts like an emitter follower. The fact that the output signal is taken from the collector of an emitter follower does not impair its function in the flip-flop. Note that OUT is in phase with the base of  $T_0$  which is used as a triggering point. It is seen therefore that  $T_0$  and  $T_1$  together act like a Schmitt trigger. The base of  $T_1$  is tied to a bias  $-u_0$  through a diode and the circuit values are chosen in such a way that OUT is above  $-u_0$  in the 1 state and below in the 0 state. (Note that the left-hand diode does not conduct as long as the supply voltage is  $-E$ .)

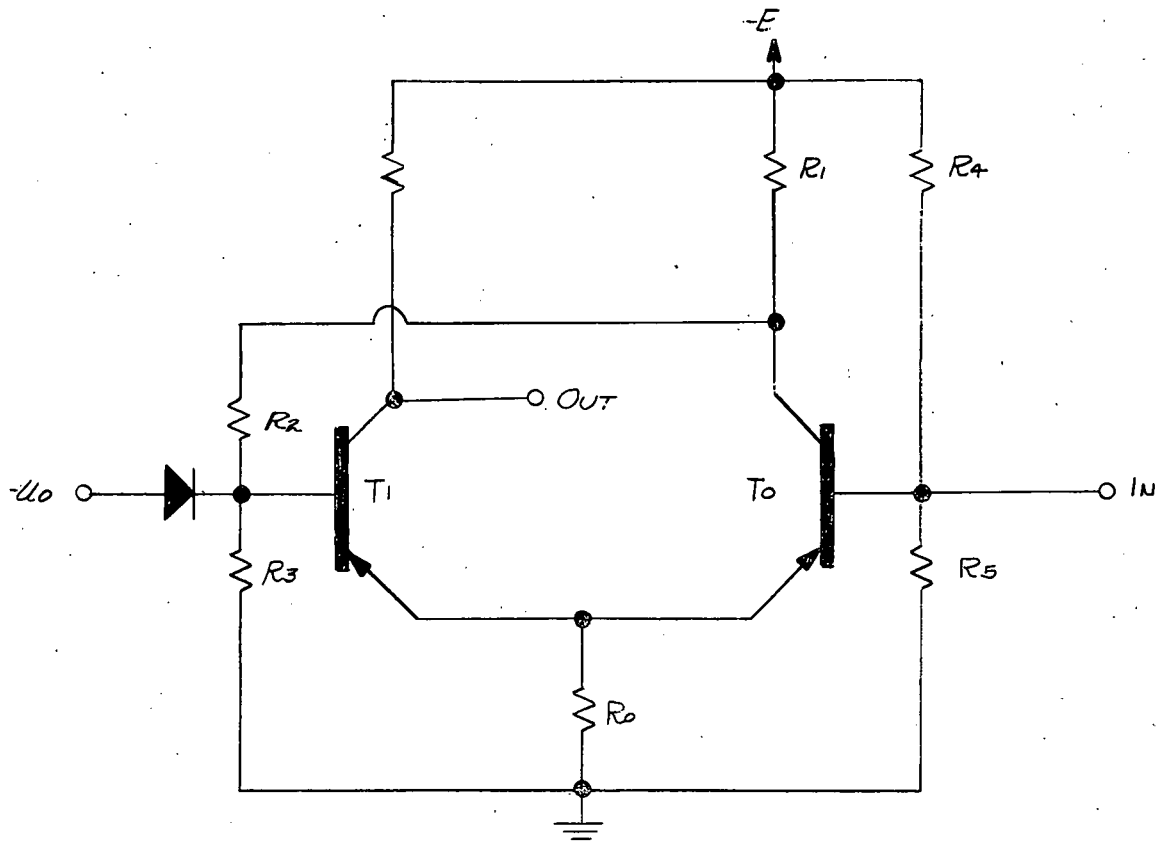


Figure 4.9

Layout of a Flow-Gating Flipflop

Suppose now that we connect several flipflops of the kind just described in a linear chain (see Figure 4.10), using diodes with their cathodes tied to IN. As long as the three supply voltages  $-E_1$ ,  $-E_2$  and

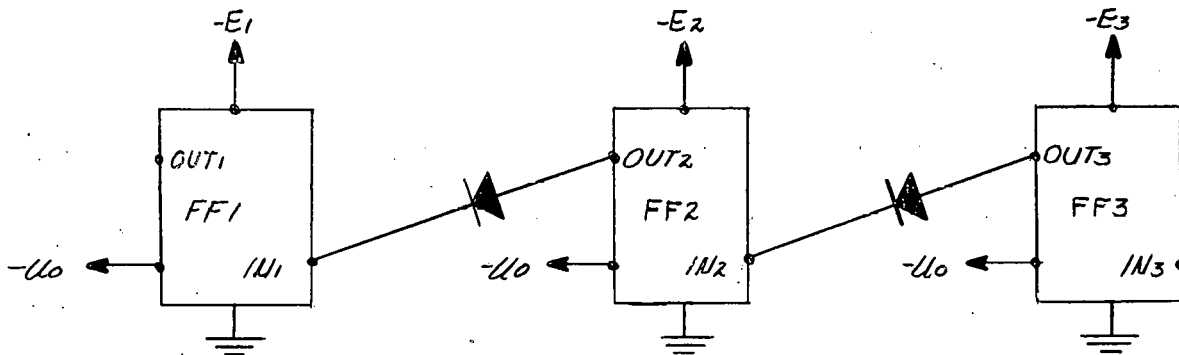


Figure 4.10

Gating with a Flow-Gating Flipflop



$-E_3$  are equal to their normal value  $-E$ , the output is so much more negative (both in the 0 state and in the 1 state) that all diodes are cut off allowing each flipflop to retain its state uninfluenced by neighboring units.

Now lower the supply of  $FF_2$  from  $-E$  to  $-E'$ . This is certainly not going to influence  $FF_1$  to the left since the connecting diode is cut off by an even greater margin than before.  $IN_2$ , however, will become more and more negative in this process and the moment will come (this depends on the judicious choice of  $-E'$ ) when its average value (determined by  $-E'$ ,  $R_4$  and  $R_5$ ) is equal to  $-u_0$ , the bias applied to the opposite base through a diode. As mentioned before, circuit values are chosen such that  $OUT$  is above  $-u_0$  in the 1 state, and below in the 0 state for a supply voltage  $-E$ . This means that if  $OUT_3$  indicates a 1,  $T_0$  is switched off, while if it indicates a 0, the bias diode switches  $T_1$  off (i.e.,  $T_0$  on). Once this operation is accomplished, the supply voltage is brought back to its normal value,  $-E$ . One can easily verify that during this transition, the state impressed at  $-E'$  is conserved and thus "trapped" when all supply voltages are equal again.

It turns out that the collector supply of  $T_1$  need not be tied to  $-E$ . This not only diminishes the current requirements for gating by a considerable factor, but also allows us to control the gating out of a flipflop without modifying the supply voltage of the flipflop to be gated into. It can also be shown that by introducing a bumping diode into the collector of  $T_1$ , we can have a constant output from a flipflop (whether it is in the 0 or the 1 state) independently of whether it is being gated into or whether it is in its normal supply-voltage range.

The property described in the last section permits the use of flow-gating flipflops in some interesting arrangements. Figure 4.11 shows a possible realization of a binary counter stage. It counts individual up-and-down sequences. The idea is to store in four flipflops, connected in a ring, the pattern 0011. The supplies of opposite flipflops are tied together, and a NOT circuit feeds one of the supply busses, the other one being connected directly

to the counter input (except for some amplification). For each 0 - 1 - 0 change at the input, the 0011 pattern is shifted cyclically. Looking at any one of the flipflop outputs, we obtain thus one 0 - 1 - 0 change for two 0 - 1 - 0 changes at the input to the counter.

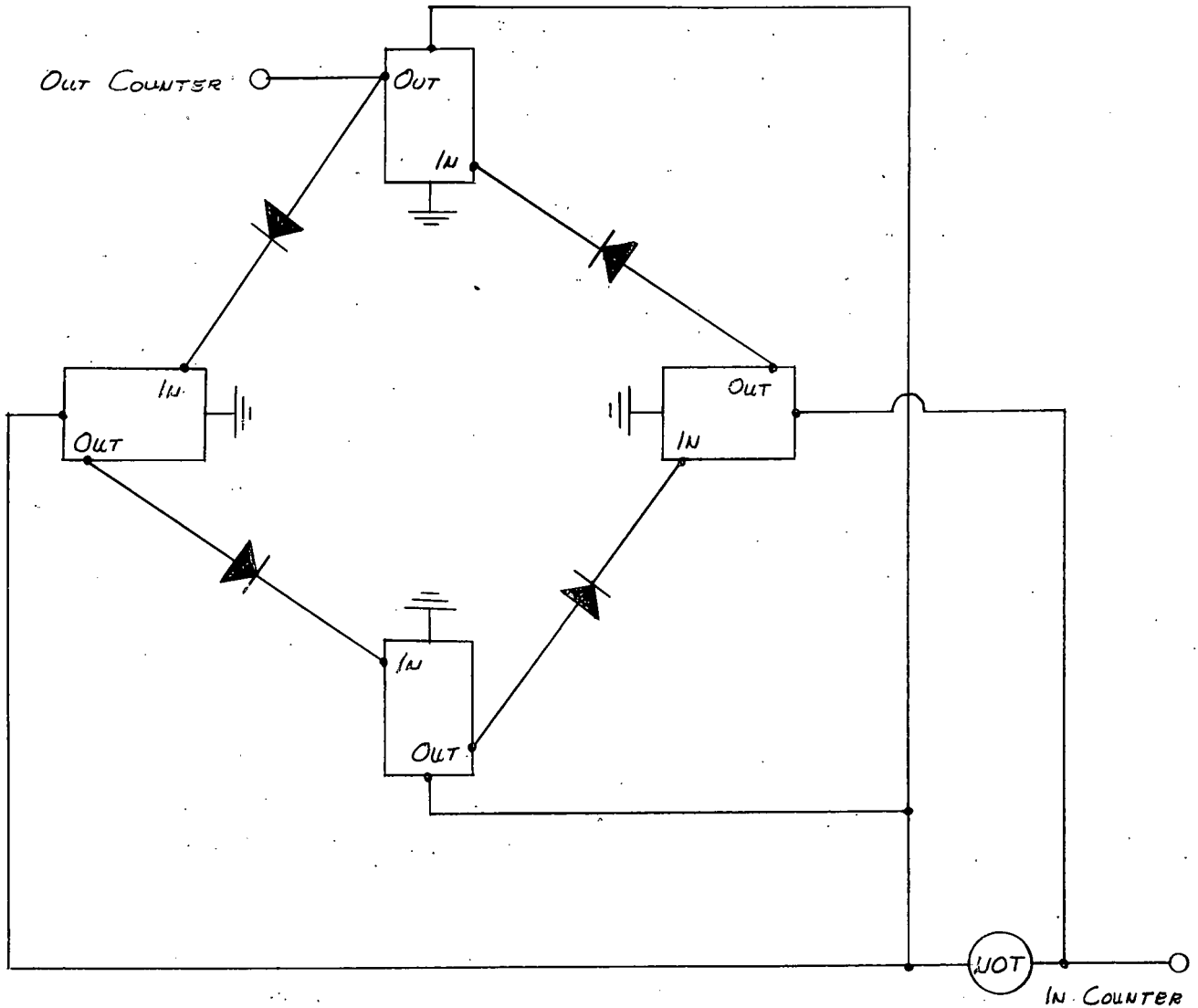


Figure 4.11  
A Flow-Gating Binary Counter Stage

#### 4.11 Principles of Circuit Analysis and Synthesis

In section 4.7 it was seen that satisfactory circuit operation demands that certain inequalities between functions of circuit parameters (supply voltages, resistors, transistor parameters) be satisfied under "worst case" conditions. In general, these inequalities are quite cumbersome to handle, especially when nonlinear relationships occur: emitter-base voltage as a function of emitter current, or  $\alpha$  as a function of emitter current (if this is high) are examples of nonlinear relationships. The procedure of finding a successful circuit requires that the calculation be made many times. Under these circumstances machine calculation by Illiac is most useful and a certain number of routines<sup>(6)</sup> have been written to this end (Problem Specifications 819, 982, 879, 928 and 1087 -- see Appendix).

The input parameters which the routines use in their calculations are those over which the designer has some control. Direct control is available over resistor values<sup>(7)</sup>, power supply voltages, and load current. Direct control is also available to the designer over transistor characteristics and power supply and resistor tolerances, but only over a limited range of variation, this range being determined by currently available hardware. Voltage drop values are also entered as input parameters since these are specified after the type of diode or transistor is decided upon. There must be enough input parameters to completely specify the circuit, however, since these routines are analytic programs.

It should be mentioned that all five programs calculate only the static dc conditions existing in a circuit. Going beyond this would mean a considerably longer program and would also necessitate a very detailed study of high-frequency transistor behavior. Therefore, the pulse response of a circuit is mostly studied on an experimental basis.

---

(6) G. H. Leichner: "Designing Computer Circuits with a Computer",  
Journal of the A.C.M., April, 1957.

(7) Except that sometimes the standard RETMA values have to be respected.

Evidently, the analytic programs mentioned should be supplemented by synthesizing programs. This would allow the reduction of the number of input parameters to the minimum dictated by the choice of hardware. There would then follow a search for an optimum solution within the remaining parameter space. The question, "What is the optimum solution?" is answered as follows: Let there be  $M$  circuit parameters  $p_i$  ( $i = 1, \dots, M$ ) representing resistances, transistor alphas etc. The satisfactory operation can then be expressed by inequalities between certain functions of the  $p_i$ 's.  $F_j$  being numerical constants, it is always possible to write these circuit inequalities in the form  $F_j \geq f_j(\dots p_i \dots)$  ( $F_j$  fixed,  $j = 1, \dots, N$ ). Note that  $N$  and  $M$  are generally of the same order of magnitude. To these inequalities one can add a small number of inequalities (identical in principle) with variable left-hand sides, the parameters being called  $G_k$  and the corresponding  $g_k$ 's having the property that they should be maximized (or minimized: this reduces to the former case). An example is the supply voltage swing in flow-gating. This gives the system  $G_k \geq g_k(\dots p_i \dots)$  ( $G_k$  variable,  $k = 1, \dots, n$ ). Note that in general  $N \gg n$ .

Let  $\pi_i$  be the best commercially-available tolerance on  $p_i$  and  $\lambda$  a "quality parameter" which should be maximized for the optimum circuit and which multiplies all tolerances. Operationally, this means that we search for the set  $p_i'$  which makes  $\lambda$  maximum in

$$F_j \geq f_j(\dots p_i' \pm p_i' \pi_i \lambda \dots)$$

$$G_k \geq g_k(\dots p_i' \pm p_i' \pi_i \lambda \dots),$$

the signs being chosen in each case in such a way that  $f_j$  or  $g_k$  increases.

It is then evident that  $\lambda_{\max}$  found in the above process is a function of all  $G_k$ 's. If  $\lambda_{\max} < 1$  over the whole range of admissible  $G_k$ 's, no solution exists: even the best components do not guarantee satisfactory operation under all drift conditions. Otherwise, in view of the small value of  $n$ , it is easy to decide at the end which set of the  $G_k$ 's (with  $\lambda_{\max} \geq 1$ ) satisfies best given engineering requirements.

Steps in the direction mentioned have been taken in some calculations, e.g., the "flow-gating" flipflop discussed in the last section. The above method was, however, only approximated by working with a relaxation net and by linearizing all inequalities.

#### 4.12 Component Specifications

The choice of components was made after considerable search, especially in regard to diodes and transistors. The most promising triode transistor appears to be the Western Electric GF-45011. This transistor, a drift type in which the necessary graded-base is obtained by a diffusion process, was designed to meet switching circuit requirements suggested in part by the Digital Computer Laboratory. All elements are considerably derated in the circuits.

Resistors: any commercially available 1% deposited carbon resistors offering good aging and temperature stability. Derating: 50% for power, a factor of 3 for tolerance (i.e., it is assumed that these resistors are at all times within 3% of their nominal value).

Power Supplies: short and long time stability better than 1% for all line and load variations. Derating: a factor of 3 for tolerance (i.e., the supply is assumed to be at all times within 3% of the nominal voltage).

Transistors: Western Electric, type GF-45011.

$V_{EB}$ forward ( $I_e = 10$ ma, $V_c = -4$ v)	$0.4 \pm 0.1$ v
$V_{EB}$ reverse ( $I_e = -100$ $\mu$ a, collector open)	greater than 4.0 volts
$V_{CB}$ reverse ( $I_c = -100$ $\mu$ a, emitter open)	greater than 25 volts
$h_{fe}$ ( $I_e = 10$ ma, $V_c = -10$ v, $f = 100$ mc), current gain, grounded emitter	greater than 7.2 db
$r'_b$ ( $I_e = 10$ ma, $V_c = -10$ v, $f = 250$ mc)	Less than 100 ohms

$\alpha_{DC}$ ( $I_e = 10 \text{ ma}$ , $V_c = -4\text{v}$ )	greater than 0.95
$C_c$ ( $V_c = -10\text{v}$ ) (measured without header)	less than 2.0 $\mu\text{mf}$
Collector dissipation, 25 <sup>o</sup> C	greater than 200 mw

Derating: Collector dissipation at 25<sup>o</sup> C assumed to be less than 150 mw  
 $\alpha_{DC}$  assumed between .93 and 1.00 for a first class of circuits,  
 $\alpha_{DC}$  assumed between .98 and 1.00 for a second class. The  
emitter-base junction dissipation is held to about 10 mw.

Diodes: Qutronics, types Q5-250 and Q10-600.

Specifications for type Q5-250:

1. Static characteristics:
  - a) For a current of 5 milliamperes in the forward direction, the voltage across the diode shall be  $0.35 \pm 0.02$  volt.
  - b) For a current of 10 milliamperes in the forward direction, the voltage across the diode shall be  $0.40 \pm 0.025$  volt.
  - c) For a current of 100 microamperes in the reverse direction, the voltage across the diode shall be greater than 5 volts.
2. Transient characteristics:
  - a) From being initially biased 5 volts in the reverse direction, each diode shall switch so as to conduct 10 milliamperes in the forward direction in not more than  $15 \cdot 10^{-9}$  second. The circuit resistance for this test shall be 100 ohms.
  - b) From initially conducting 10 milliamperes in the forward direction, the diode shall be switched so that it is biased 5 volts in the reverse direction. In not more than  $10 \cdot 10^{-9}$  seconds after switching to the specified reverse bias, the current in the reverse direction shall not exceed 1.5 milliamperes. Within  $80 \cdot 10^{-9}$  seconds after switching, the current shall be less than 250 microamperes, and within  $200 \cdot 10^{-9}$  seconds the current shall be less than 120 microamperes. The circuit resistance for this test shall be 200 ohms.

3. Each diode shall be capable of dissipating 100 milliwatts at an ambient temperature of  $25^{\circ}$  C.
4. The shunt capacity of each diode, when measured with a reverse bias of 3 volts, shall not exceed 0.5 micromicrofarad.
5. The physical dimensions of each diode shall be  $0.25 \pm 0.05$  inch in length, exclusive of leads, and shall be approximately 0.1 inch in diameter.
6. Each diode shall be hermetically sealed.

Derating: None.

Specifications for type Q10-600:

1. Static characteristics:
  - a) For a current of 5 milliamperes in the forward direction, the voltage across the diode shall be  $0.37 \pm 0.025$  volt.
  - b) For a current of 10 milliamperes in the forward direction, the voltage across the diode shall be  $0.42 \pm 0.03$  volt.
  - c) For a current of 100 microamperes in the reverse direction, the voltage across the diode shall be greater than 10 volts.
2. Transient characteristics:
  - a) From being initially biased 5 volts in the reverse direction, each diode shall switch so as to conduct 10 milliamperes in the forward direction in not more than  $15 \cdot 10^{-9}$  second. The circuit resistance for this test shall be 100 ohms.
  - b) From initially conducting 10 milliamperes in the forward direction, the diode shall be switched so that it is biased 5 volts in the reverse direction. In not more than  $10 \cdot 10^{-9}$  second after switching to the specified reverse bias, the current in the reverse direction shall not exceed 2 milliamperes. Within  $80 \cdot 10^{-9}$  second after switching, the current shall be less than 600 microamperes, and within  $200 \cdot 10^{-9}$  second the current shall be less than 120 microamperes. The circuit resistance for this test shall be 2000 ohms.

3. Each diode shall be capable of dissipating 100 milliwatts at an ambient temperature of  $25^{\circ}$  C.
4. The shunt capacity of each diode, when measured with a reverse bias of 3 volts, shall not exceed 0.5 micromicrofarad.
5. The physical dimensions of each diode shall be  $0.25 \pm 0.05$  inch in length, exclusive of leads, and shall be approximately 0.1 inch in diameter.
6. Each diode shall be hermetically sealed.

Derating: None.

#### 4.13 General Remarks about the Next Sections

In the next eleven sections some of the circuits which could be incorporated in a new very high-speed computer will be shown by way of example. It is by no means certain that these circuits will not be improved in the near future, either by using transistors satisfying tighter specifications or by incorporating into them certain features which will possibly improve their performance. In this line one could think of gating into flipflops of the Eccles-Jordan type by essentially flow-gating principles. Another possibility, successfully tried recently, is the replacement of the Eccles-Jordan flipflop by what amounts to the "back-to-back" combination of Schmitt trigger's (the emitters and bases of the amplifiers being cross-coupled).

It should be noted that efforts have been made to find resistor values available as RETMA standard values.

For convenience sake, the use of positive logic will be assumed; i.e. nominally

"1" corresponds to +1.6v

"0" corresponds to -1.6v.



Note that these voltages are measured at the output of emitter-followers in most cases. The bumps are -2.0v and +1.2v to take account of the (average) emitter-base drop of .4v.

The ordering principle for the rest of this chapter is the tolerance of circuits. If no complete tolerance analysis has been made, the circuit is automatically relegated to a later section. Four classes of circuits can be distinguished:

Class 1: Designed according to section 4.11 and using the component specifications of 4.12.

Class 2: Designed according to section 4.11, but using tighter specifications:  $.98 \leq \alpha_{DC} \leq 1.00$  and a resistor tolerance of 2% only. Non-standard output levels.

Class 3: Designed by procedures similar to those of section 4.11 but not quite as exact; in particular no table of emitter-base drops was stored with the program. Component specifications of 4.12, except for resistors, which have only 2% tolerance.

Class 4: Designed by diverse procedures of varying exactitude. The component specifications are those of Class 2.

#### 4.14 NOT Circuit (Class 1)

As Figure 4.12 shows, the NOT circuit is a constant-current-emitter inverter with a diode clamp holding the emitter down in order to allow switching. A bumped stepdown network drives an output emitter-follower. For the nominal signals, the input current does not exceed 1 ma; the maximum output current is 3 ma. A collector bump prevents saturation.

The figure also quotes the more exact relationship between voltages and currents for both the input and for the output, since the circuit can work satisfactorily with non-nominal inputs. This leads incidentally to the use of NOT circuits as LEVEL RESTORERS when inversion can be tolerated.

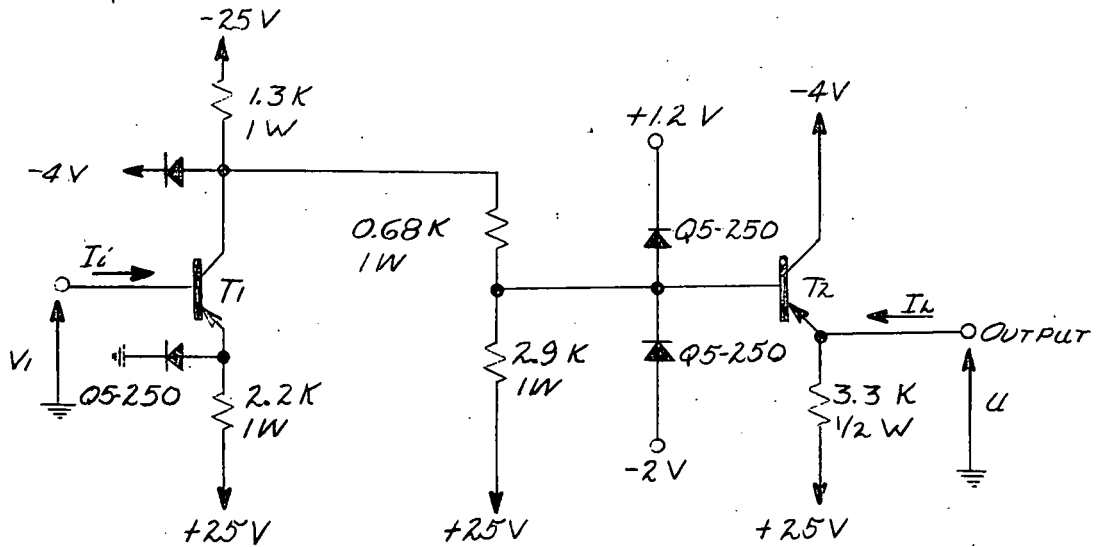


Figure 4.12

15 Millimicrosecond NOT Circuit

All resistors and power supply voltages within  $\pm 3\%$  of nominal

$$.93 \leq \alpha_{DC} \leq 1.0$$

Maximum Input Current:  $I_i = -.829 + .033 V_i$   
 $I_i$  Undefined  
 $I_i \approx 0$

$V_i \leq -.5v$   
 $-.5 \leq V_i \leq +.41v$   
 $V_i \geq +.41v$

Output Voltage:  $-2.08 \leq U \leq 1.63v$   
 $+1.51 \leq U \leq +2.02v$   
 $U \leq -1.5v$

Input "Zero" at  $I_L = 3 \text{ ma}$   
 Input "One"  
 Input Zero at  $I_L = 5 \text{ ma}$

Input Voltage:  $-2.5 \leq V_i \leq -.5v$   
 $+.41 \leq V_i \leq +2.5v$

"Zero" (Positive Logic)  
 "One" (Positive Logic)

Operation Time:  $< 15 \text{ } \mu\text{s}$

#### 4.15 Level Restorer (Class 1)

The circuit in Figure 4.13 is essentially the one described in the last section. Here, however, the inverter is replaced by a (non-inverting) grounded-base amplifier and this in turn necessitates the input emitter-follower. Note that no diode has to be used in the emitter of Transistor  $T_1$ .

The comments and notes of the last section concerning currents and voltages apply equally well to the circuit under consideration.

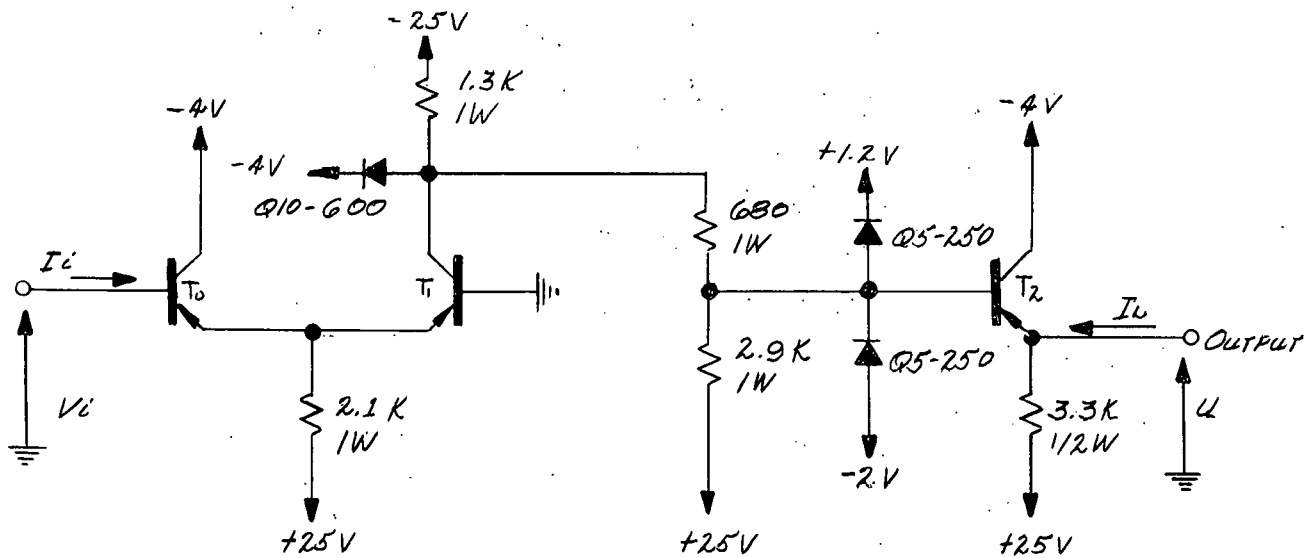


Figure 4.13

#### 15 Millimicrosecond Level Restorer

All resistors and power supply voltages within  $\pm 3\%$  of nominal.

$$.93 \leq \alpha_{DC} \leq 1.0$$

Maximum Input Current:  $I_i = -.868 + .0342 V_i$   
 $I_i$  Undefined  
 $I_i \approx 0$

Output Voltage:  $-2.08 \leq U \leq -1.63v$   
 $+1.51 \leq U \leq +1.97v$   
 $U \leq -1.5v$

Input Voltage:  $-2.5v \leq V_i \leq -.5v$   
 $+ .5 \leq V_i \leq + 2.5v$

Operation Time:  $< 15 \mu s$

$V_i \leq -.5v$   
 $-.5 \leq V_i \leq + .5v$   
 $V_i \leq + .5v$

Input "One" at  $I_L = 3 \text{ ma}$   
 Input "Zero"  
 Input "One" at  $I_L = 5 \text{ ma}$   
 "Zero" (Positive Logic)  
 "One" (Positive Logic)

#### 4.16 OR Circuit (Class 1)

The OR circuit of Figure 4.14 is a (triple) diode OR for positive logic, each diode being driven by a separate emitter-follower. Note that there is one "output-determining" transistor: the one corresponding to the most positive input.

In designing an OR circuit, it is important to consider the dc level change or "deviation" which a signal suffers in its transfer from input to output. The signal to be considered is precisely the one which determines the output voltage. A very meaningful measure of this deviation is the number of similar circuits allowed in cascade. The table below presents a compilation of input voltages, deviation and output voltage for each successive stage in a chain of OR circuits. The first stage is assumed to be driven by a slightly degenerated signal of  $-1.500\text{v}$  coming from a level-restorer or a NOT circuit. Since a NOT circuit can restore a negative signal which is only  $-0.55\text{v}$ , it is seen that 5 OR's can be cascaded before going into another NOT or a level-restorer. Note that the emitter-base drops and the diode drops cancel more or less under average conditions.

Table for Cascaded OR Circuits

Stage	Input	Max. Deviation	Output
1	-1.500	.176	-1.324
2	-1.324	.174	-1.150
3	-1.150	.171	-0.979
4	-0.979	.168	-0.811
5	-0.811	.165	-0.646

This shows that for negative swings, 5 OR's can be cascaded. For positive swings, conditions are similar.

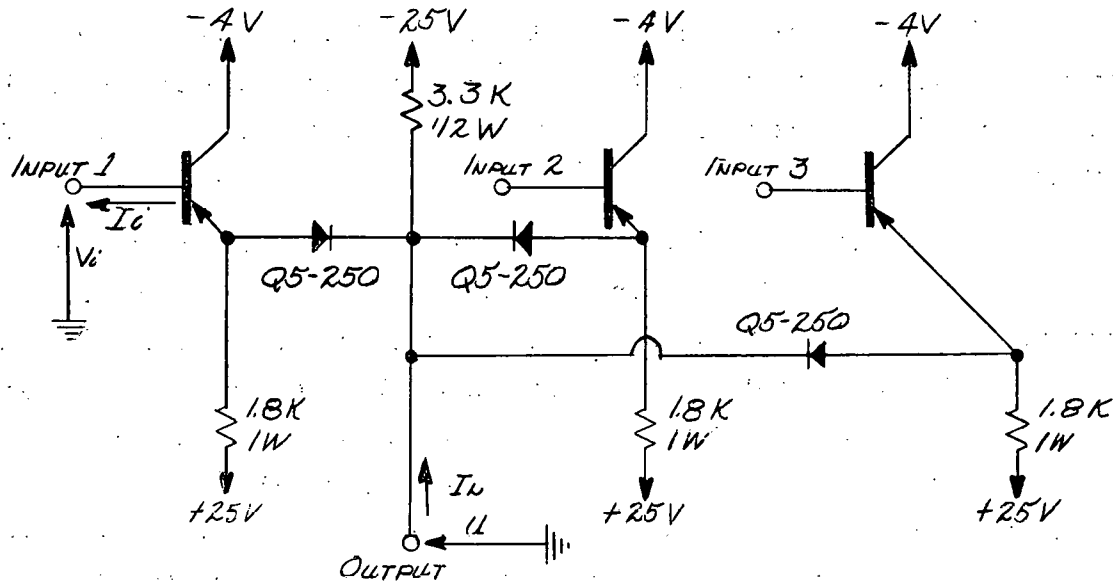


Figure 4.14

5 Millimicrosecond OR Circuit

All resistors and power supply voltages within 3% of nominal.

$$.93 \leq \alpha_{DC} \leq 1.0$$

Maximum Input Current to Output-Determining Transistor:

$$I_i = .634 + .064 I_L \quad V_i \geq -2v$$

Output Voltage: Max. Values:  $U = +.101 + .983 V_i + .026 I_L$

Min. Values:  $U = -.076 + .99 V_i + .013 I_L$

For  $-2 \leq V_i \leq 2v$  and  $0 \leq I_L \leq 3$  ma with  $I_L$  in milliamperes.

Maximum Input Current to Non-Output-Determining Transistor:

$$I_L = 1.1 \text{ ma} \quad V \geq -2v$$

Total number of circuits allowed in cascade (each driving +2 ma load, standardized initial input): 5

Operation Time: < 5  $\mu$ s

#### 4.17 AND Circuit (Class 1)

The AND circuit of Figure 4.15 is straightforward since pnp transistors are used. Note however the voltage divider in the output which compensates for the average emitter-base drop. Here again there is an "output-determining" transistor: the one corresponding to the most negative input. Cascading questions can be treated in a way similar to that of the last section.

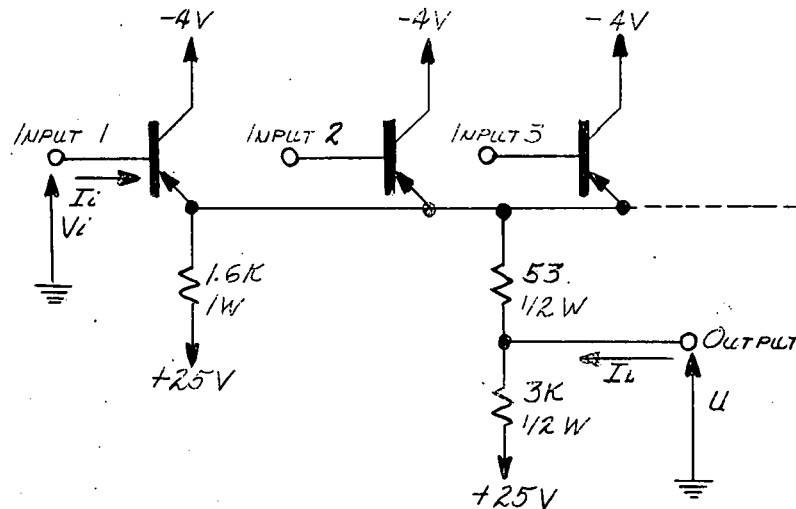


Figure 4.15

5 Millimicrosecond AND Circuit  
(Positive Logic)

All resistors and power supply voltages within  $\pm 3\%$  of nominal.

$$.93 \leq \alpha_{DC} \leq 1.0$$

Maximum Input Current to Output-Determining Transistor:

$$I_i = -.691 - .068 I_L \quad V_i \geq -2v$$

Output Voltage: Max. Values:  $U = +.031 + .969 V_i + .066 I_L$

Min. Values:  $U = -.173 + .978 V_i + .057 I_L$

For  $-2 \leq V_i \leq +2v$  and  $0 \leq I_L \leq 3$  ma with  $I_L$  in milliamperes

Total number of circuits allowed in cascade (each driving +2 ma load, standardized initial input): 5

Operation Time:  $< 5 \mu s$

#### 4.18 Flow-Gating Flipflop (Class 2)

The design principles of this type of circuit having been discussed at length in section 4.10, very few comments will suffice. The first one is to draw attention to the fact that resistors have to be of the 2% variety and that  $.98 \leq \alpha_{DC} \leq 1$ ; the second one is that two non-standard diodes are used, the type number being GA4-100 (Hughes). These diodes can withstand higher reverse voltages than the standard Qutronics types.

Note the presence in the output collector of a bumping diode to prevent saturation when the supply voltage is at its normal level and also to stabilize the output. The latter is -10v or -20v, depending on the state, i.e. as designed, the levels are not the standard  $\pm 1.6v$  levels.

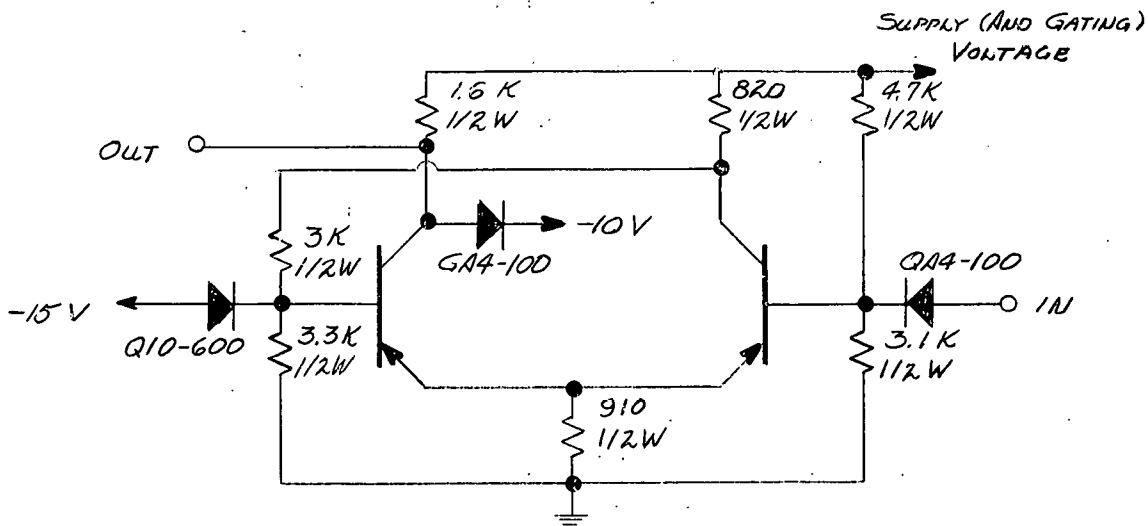


Figure 4.16

#### 50 Millimicrosecond Flow-Gating Flipflop

Power supplies within 3% of nominal value.

Resistors within 2% of nominal value.

$$.98 \leq \alpha_{DC} \leq 1$$

Maximum Input Current: 2.75 ma

Minimum Output Current: 3.75 ma

Output Voltages: -10v and -20v (nominal)

Supply and Gating Voltage: -20v (normal) , -45v (gate-in)

Operation Time: < 50 mps

#### 4.19 Schmitt Trigger (Class 3)

Figure 4.17 below gives the circuit values for a last moving point Schmitt trigger flipflop. Note the presence of collector bumps, tied to  $-4v$  and also the fact that the stepdown network in the regenerative loop has a much higher impedance than that going into the output emitter-follower. The input is tied to two separate diodes in order to be able to gate in both directions, the average gating current (for either case) being about 2.5 ma. With an output driving capacity of about 5 ma this gives a fan-out of 2 for the circuit. The operation time is approximately 15  $\mu$ s.

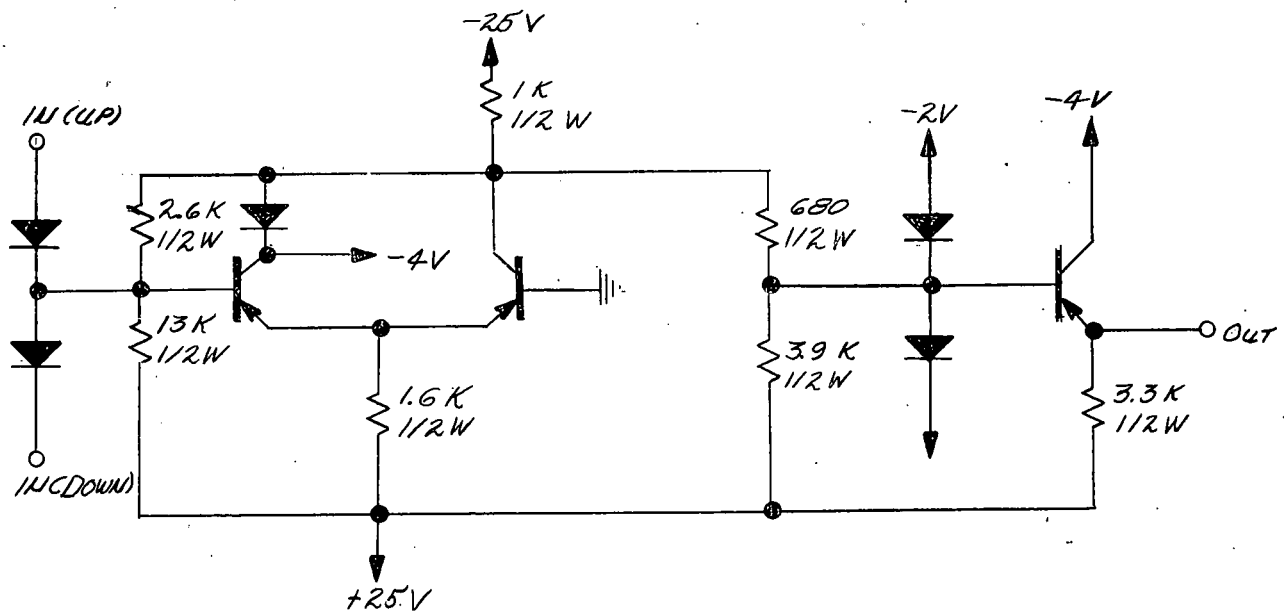


Figure 4.17

15 Millimicrosecond Schmitt Trigger Flipflop

(All diodes are Qutronics Q5-250)



#### 4.20 Eccles-Jordan (Class 3)

Figure 4.18 gives the circuit values for a last moving point Eccles-Jordan flipflop. Again collector bumps are used, but this time (since there is no longer a grounded-base amplifier in the circuit) supplementary bumps are used at the bases of the amplifiers and also from the common emitter to ground: This is necessary in order to respect the reverse specifications for the emitter-base junctions. Otherwise the circuit is much like two crosscoupled Schmitt-Triggers -- although the stepdown networks are much more similar in impedance; this is possible because double gating eliminates some of the awkward conditions encountered in a single-ended flipflop. Gating currents and output currents are, however, very similar to those mentioned in section 4.19 as is the fan-out of this circuit (2). The operation time with the direction of gating indicated is about 30  $\mu$ s.

#### 4.21 EXCLUSIVE OR (Class 4)

The principle of the EXCLUSIVE OR circuit of Figure 4.19 is well known: The input signals are applied to two transistors with a common collector load, the bases and emitters being crosscoupled. Then the common collector reacts only when unlike signals are received, at least as long as the swings applied to  $T_0$  and  $T_1$  are equal. The latter condition necessitates the re-standardization of the incoming signals: This is achieved by the collector bumps for  $T_2$  and  $T_3$ . Note that the output is taken from an emitter-follower, the dc stepdown being on the emitter side. The circuit operates with normal  $+1.6$ v signals and the operation time is about 50  $\mu$ s. The fact that one can obtain an EXCLUSIVE-OR operation by the common AND-NOT-AND-OR combination in about 20  $\mu$ s is less relevant when the number of transistors and diodes is compared: This circuit uses only 5 transistors vs. 8 for the combination. However, if the often required AND function has to be obtained too, the combination is probably more attractive: its use has therefore been assumed in this report.

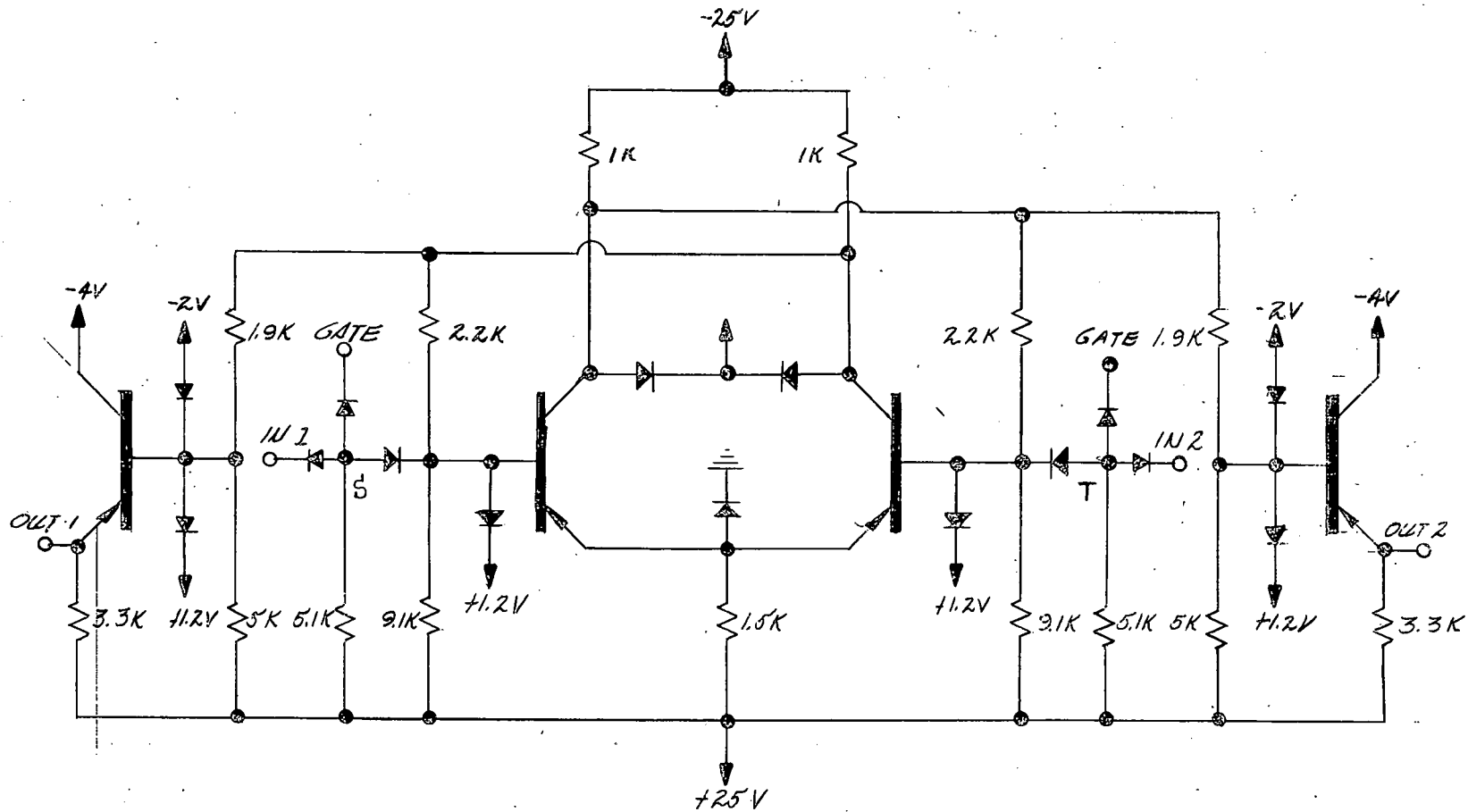


Figure 4.18

30 Millimicroseconds Eccles-Jordan Flipflop

(GATE = +1.2v gating, GATE = -2v non-gating,  
All diodes in collectors are Q10-600, all others Q5-250.  
All resistors are 1/2 w, 2%.)

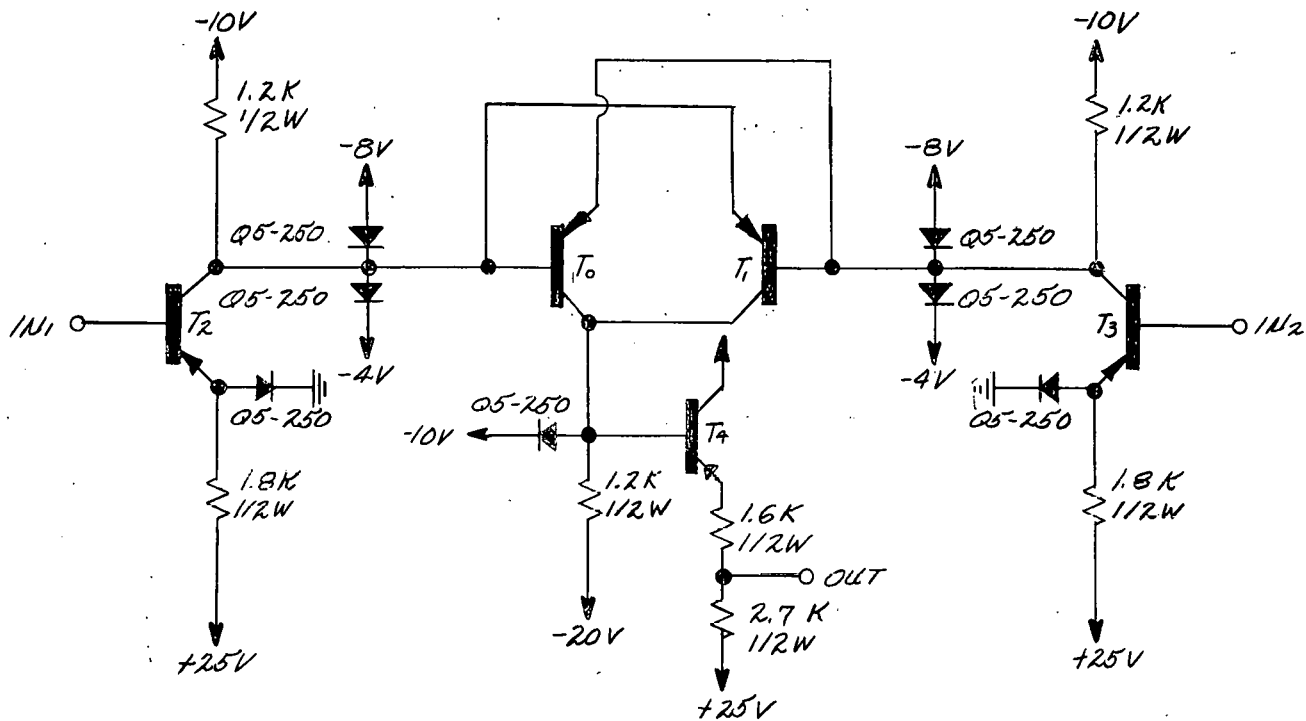


Figure 4.19  
 50 Millimicrosecond EXCLUSIVE OR  
 (Positive Logic)

#### 4.22 Driver (Class 4)

The diode gates described in the section on Eccles-Jordan flipflops necessitate large driving currents. Figure 4.20 indicates a driver capable of controlling 8 such gates. This driver is composed of a level-restorer type preamplifier and a number of emitter-followers, the last two being in parallel. Normal signals are used at both input and output, i.e. the gating voltages of section 4.20 are more than attained. This driver is capable of going up and down in less than 50  $\mu$ s: This corresponds to about 20  $\mu$ s operation time.

Figure 4.21 gives some of the waveforms observed in the gates and the flipflops of a shifting register (see section 4.24), the circuits being driven by a two-phase logical oscillator.

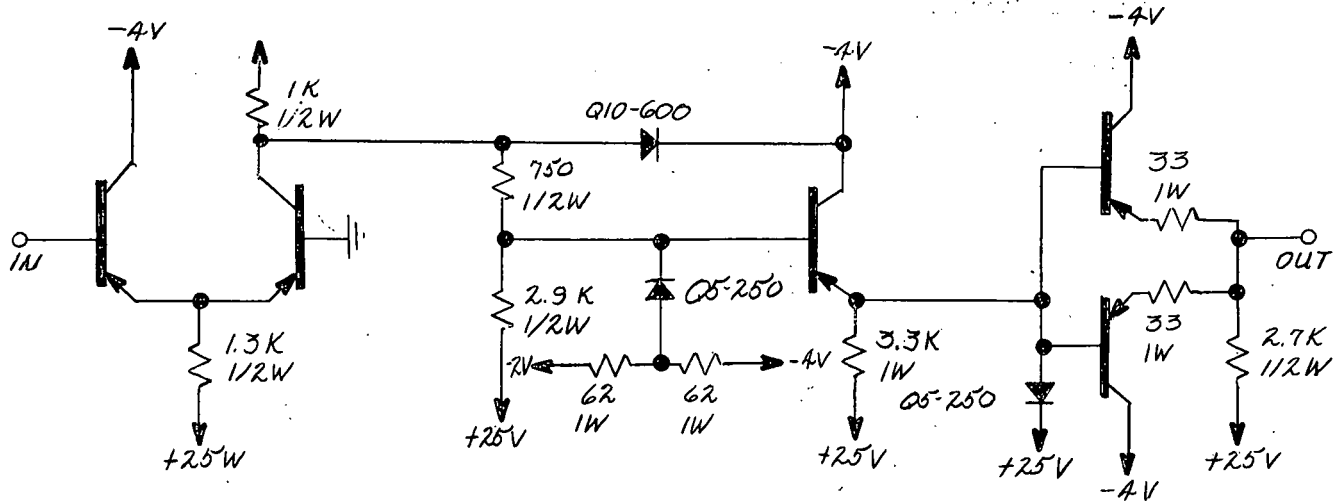


Figure 4.20 TRANSISTORS GF45011

20 Millimicrosecond Driver

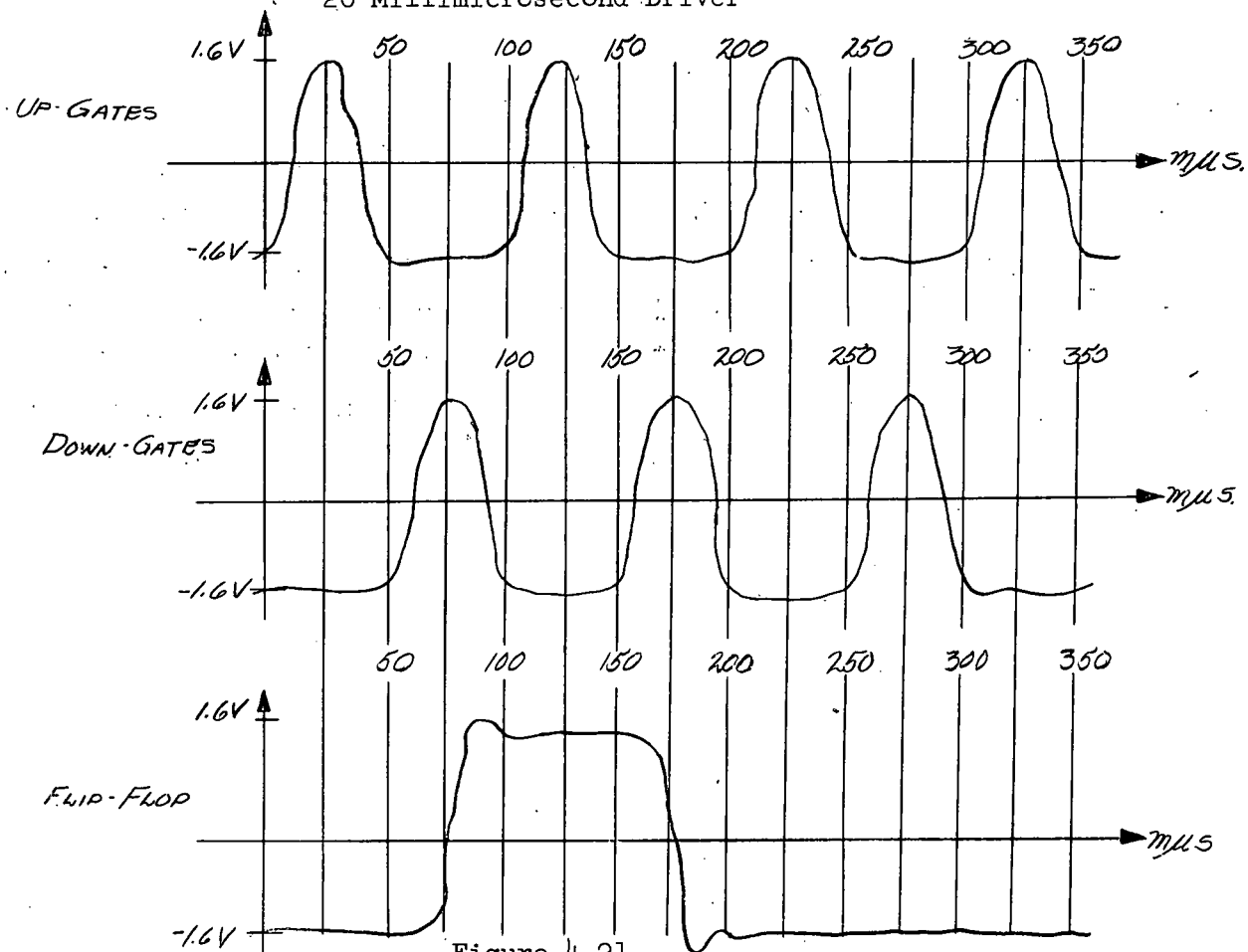
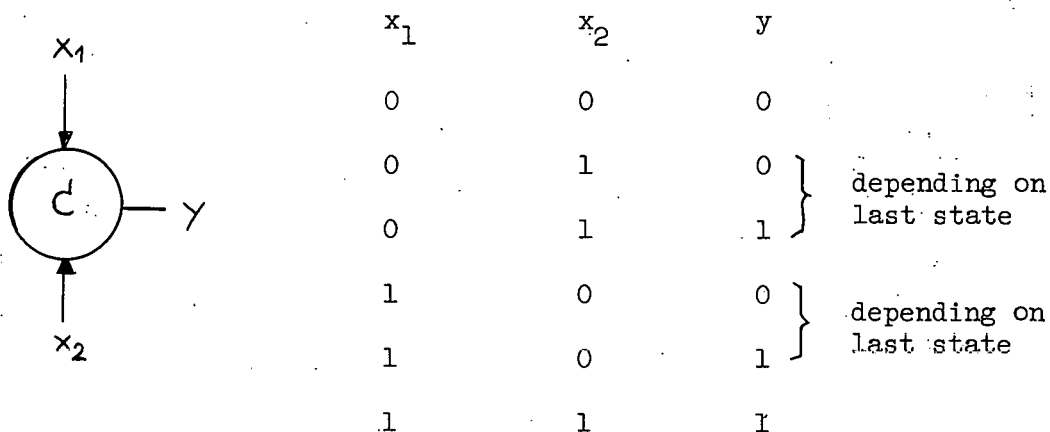


Figure 4.21

Timing Diagram for the 50 μs Shifting Register--Pattern 0001

#### 4.23 C-Element (Class 4)

Figure 4.22 shows a possible design for a C or  $\bar{C}$ -element. This is, by definition, an element described by the following equilibrium table:



The  $\bar{C}$ -element differs from a C-element only by having the complemented outputs.

As can be seen, the circuit consists essentially of a Schmitt-Trigger, controlled by a standardizing input amplifier. The center-point M between the collectors is "up" when both inputs are "down" and vice-versa. If the inputs differ, M stays at an intermediary voltage such that (especially because of the "hysteresis" diode combination H) the flipflop part does not react. The output shown corresponds to a  $\bar{C}$ -function, but the output could also be taken from P with a suitable emitter-follower and a dc-stepdown network: This would give the C-function.

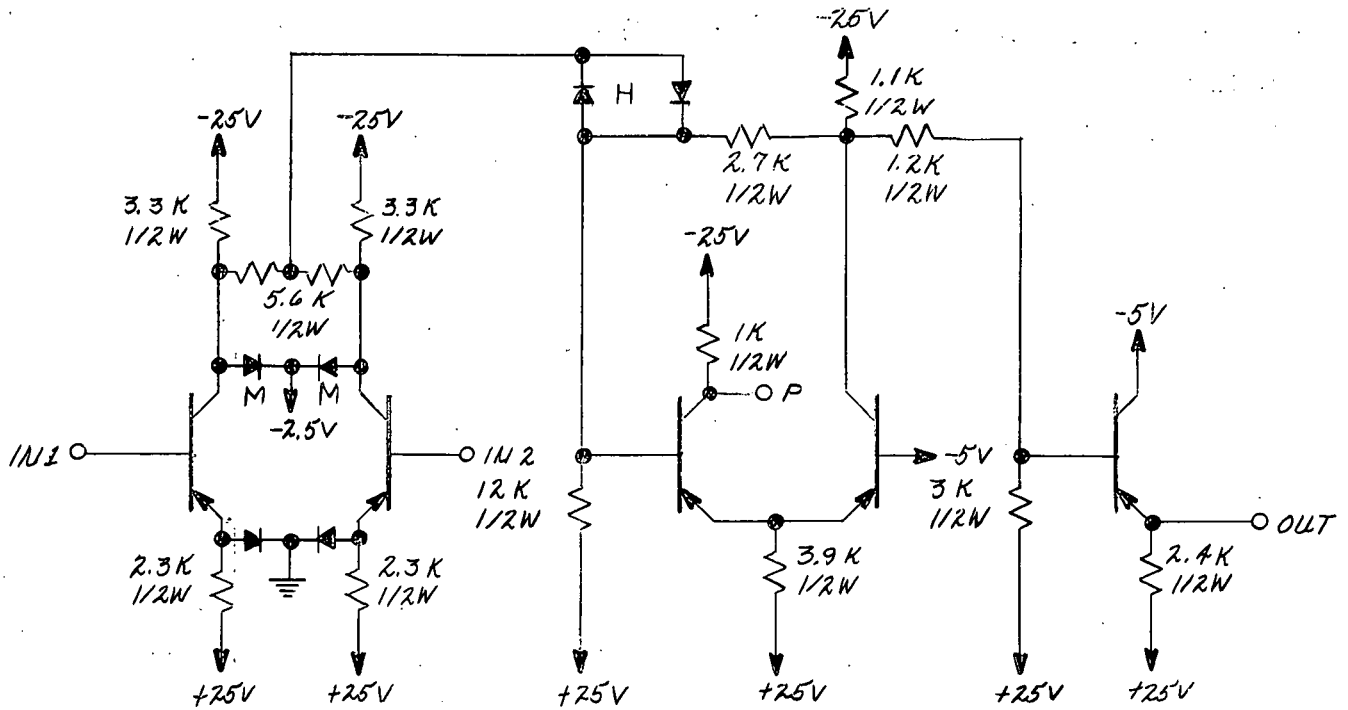


Figure 4.22  
C or  $\bar{C}$  Element

#### 4.24 Shifting Registers

For the flipflops discussed in this chapter, gating facilities have been indicated in all cases. However the full extent of the problem of transferring information between flipflops has not been covered. This section will bring up some aspects of transfer and shifting in view of obtaining a reasonable estimate of the number of parts involved. To simplify the discussion only the Eccles-Jordan flipflop of section 4.20 will be considered.

It is easily seen that the changes of state of the circuit are produced by providing a positive signal at the base of that one of the center transistors which happens to be conducting. For this purpose a

positive voltage is injected through a diode from either S or T (See Figure 4.18). Since positive logic is used, the properties of the flip-flop as seen from S and T as inputs are described by the following equilibrium table:

S	T	OUT <sub>1</sub>	OUT <sub>2</sub>	
0	0	0	1	} depending on last state
0	1	1	0	
0	1	0	1	
1	0	1	0	
1	1	not allowed		

The diodes going from IN<sub>1</sub> and GATE to S and from IN<sub>2</sub> and GATE to T are operationally equivalent to a (positive) AND circuit. Symbolizing the flipflop with S and T as inputs by the usual symbol, the transfer of information between two flipflops corresponds to the arrangement of Figure 4.23. For obvious reasons this procedure is called "double-gating": There are two possible paths for the transfer of information. Considerations will be limited to this system since it is inherently faster than the "clearing-and-gating" system in which flipflop 2 is set to the zero state (zero side output a 1) by a clearing signal which precedes the -- conditional -- transfer of a one.

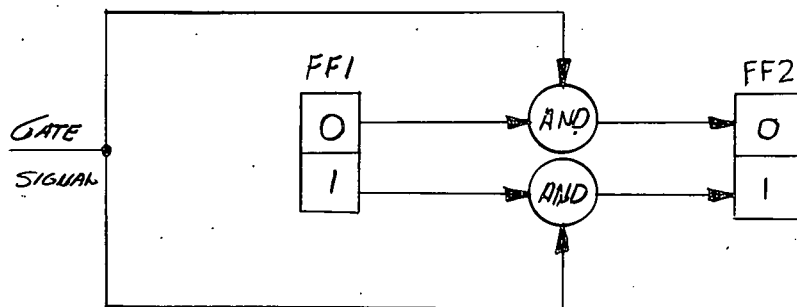


Figure 4.23

Double Gating

Shifting in an asynchronous machine is accomplished by adding to the principal register an auxiliary register. The latter is then connected back to the principal register by AND gates (not shown in Figure 4.24) leading to a flipflop one digit position to the right or one digit position to the left. Figure 4.24 shows a second input for the flipflops of the auxiliary register and a second output from the flipflops of the principal register. These connections are necessary if the double register is to perform any useful function. Each flipflop therefore necessitates two inputs and two outputs.

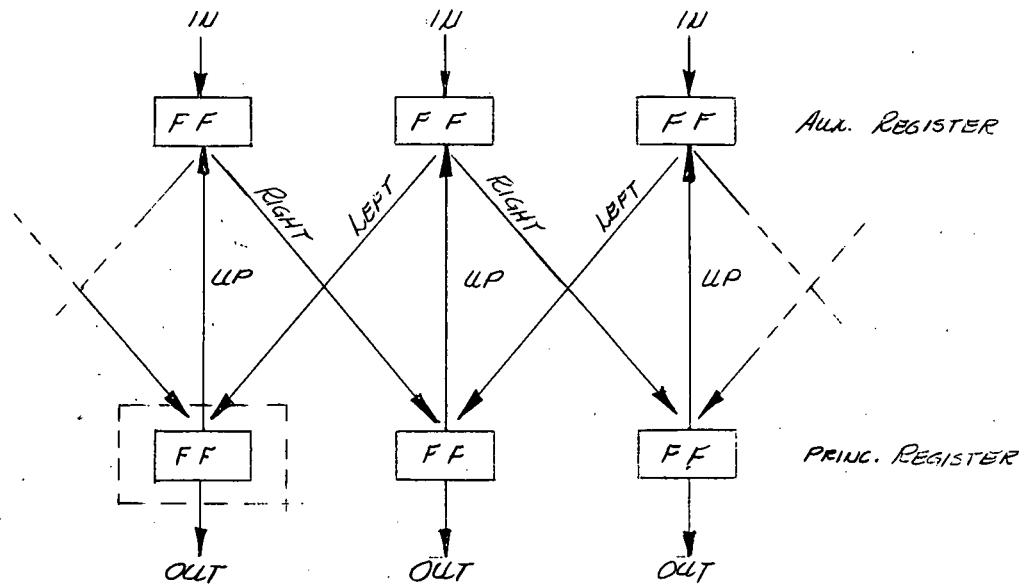


Figure 4.24

#### Shifting Register

Figure 4.25 shows more explicitly what is contained in the dotted box indicated in Figure 4.24. Since information is received from two sources, the flipflop must be preceded by two OR circuits. Practically this can be achieved by adding a second diode from the terminals of the second pair of inputs to the bases of the amplifying transistors of the Eccles-Jordan flipflop.



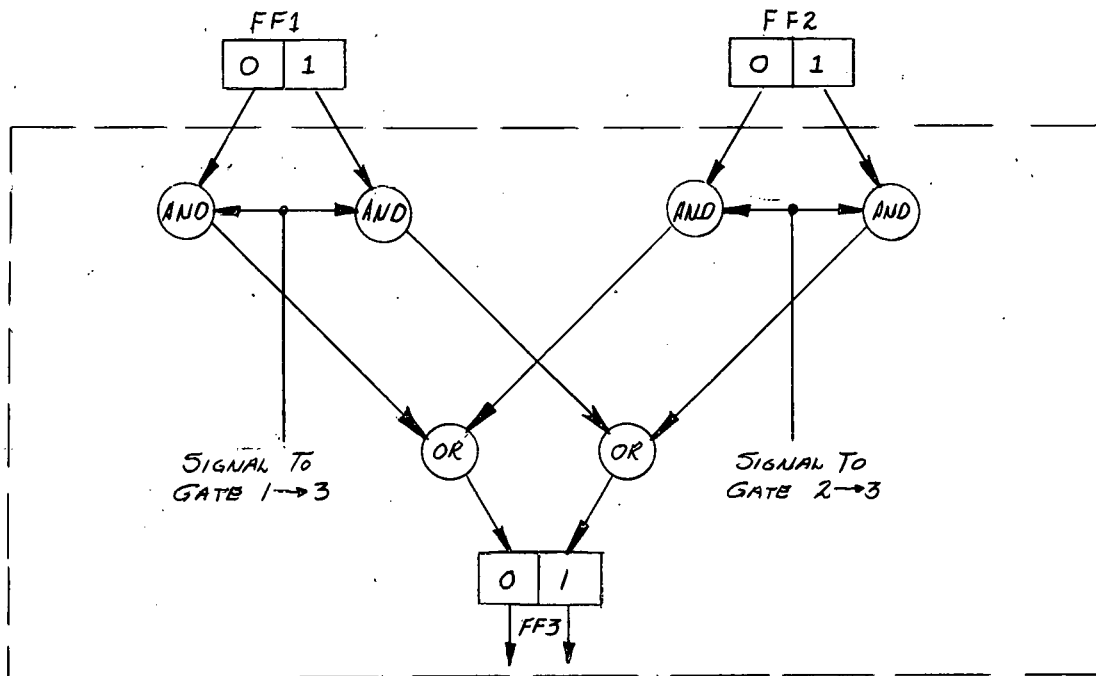


Figure 4.25

#### Circuitry Associated with One Flipflop

It should be noted that the simple diode AND gates of Figure 4.18 are sufficient if not more than two other flipflops are connected to any one flipflop. Otherwise the AND's of section 4.17 must be used; a "fan-out" of the order of 8 can then be obtained. The use of the standard transistor AND's shall be assumed.

The component count for all circuits associated with one flipflop using the standard AND's as gates and the simplified OR's described above is given below. For completeness' sake drivers as described in section 4.22 have been added: One such driver can take care of 32 AND's; i.e. per flipflop (involving 4 AND's), 0.125 drivers are needed.

	Transistors	Diodes	Resistors
1 flipflop without gates	4	11	13
4 AND's	8	0	12
OR diodes	0	2	0
0.125 driver	0.625	0.375	1.250
Total	12.625	13.375	26.250

For a 52-bit shifting register  $10^4$  flipflops with their associated circuitry are needed. This implies the use of 1313 transistors, 1381 diodes and 2730 resistors.

It should be mentioned that diode AND's can be used under the circumstances outlined above: This would save 832 transistors but the diode count would be increased by the same figure.

#### 4.25 Summary and Closing Remarks

This chapter shows conclusively that it is possible to design transistorized circuitry for asynchronous dc-coupled operation with operation times in the 5 - 50  $\mu$ s region. Most of these circuits also satisfy very strict tolerance requirements and therefore guarantee excellent reliability. Some life tests of units having up to 48 transistors have been very encouraging, as have some experiments concerning noise sensitivity.

The following table gives a synopsis of the circuits as regards the number of transistors, the number of diodes and the operation time.

Circuit	No. of Transistors	No. of Diodes	Operation Time (μs)
NOT	2	4	15
Level Restorer	3	3	15
AND	2	0	5
OR	2	2	5
Flow-Gating Flipflop	2	3	50
Schmitt Trigger	3	5	15
Eccles-Jordan	4	15	30
EXCLUSIVE OR	5	7	50
Driver	5	3	20
C-Element	4	6	-

## APPENDIX

As was mentioned in section 4.11, a number of programs have been written which analyze the operation of given circuits under given conditions. These programs correspond to the following problem specifications:

- Problem Specification 819 - Verify a non-last-moving-point Schmitt trigger
- Problem Specification 982 - Verify a non-last-moving-point Eccles-Jordan
- Problem Specification 879 - Verify NOT circuit of section 4.14
- Problem Specification 928 - Verify OR and AND circuits of sections 4.16 and 4.17
- Problem Specification 1087 - Verify Flow-Gating Flipflop of section 4.18.

As an example of the procedure, some details of Problem Specification 879 are given below:

The program requires that the values of the circuit components and the four values of the input voltage,  $v$ , (corresponding to the minimum and maximum values of both of the logical values 0 and 1) be used as program input parameters. The program then calculates the output voltage and other quantities under "worst case" conditions for each of the four circuit input voltages.

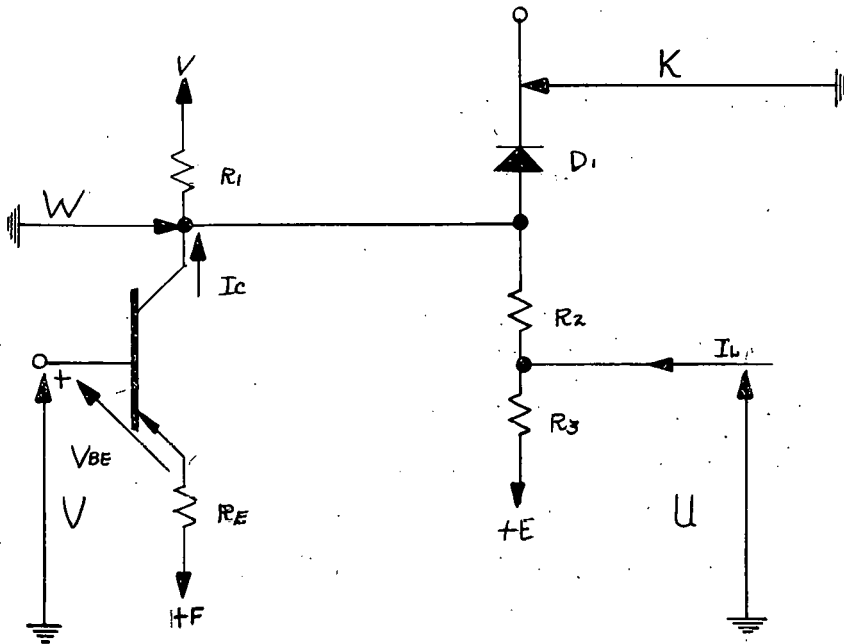


Figure 4.26

NOT Circuit

In addition to the above notation, the following applies:

$V_x^+$  = the maximum positive value of the input voltage

$V_m^+$  = the minimum positive value of the input voltage

$V_m^-$  = the minimum negative value of the input voltage

$V_x^-$  = the maximum negative value of the input voltage

$\alpha_m$  = the minimum value of  $\alpha = \frac{I_c}{I_e}$

$\alpha_x$  = the maximum value of  $\alpha$

$a$  = the fractional tolerance on all power supply voltages

$b$  = the fractional tolerance on all resistors

$I_{LX}$  = the maximum value of output load current

$I_{CO}$  = the maximum collector cut-off current

$I_{Lm}$  = the minimum value of output load current

P = - number implies  $D_1$  is used; + number implies  $D_1$  is not used

Parameter Tape: The following data may be placed on a separate tape and read in after the main tape:  $V_x^+$ ,  $V_m^+$ ,  $V_m^-$ ,  $V_x^-$ ,  $R_E$ ,  $R_1$ ,  $R_2$ ,  $R_3$ , F, -V, E,  $\alpha_m$ ,  $\alpha_x$ , a, b,  $I_{CO}$ ,  $I_{Lm}$ ,  $I_{LX}$ , P, K. These are to be entered in the order listed in standard floating decimal notation. (Note: K must be entered but may be anything if no diode is used.)

Operation: The program tape is read into the computer in the standard manner. At the end of this tape, a black switch stop will occur. The parameter tape is then placed in the reader and the stop is bypassed.

Output: The output information is (in addition to a reprint of the parameter tape):

$u(1)^-$  = most negative output of logical "1"

$u(1)^+$  = most positive output of logical "1"

$u(0)^-$  = most negative output of logical "0"

$u(0)^+$  = most positive output of logical "0"

VCX = maximum collector-to-base voltage

WCX = maximum collector power dissipation

IEX = maximum emitter current

DIODE = printed if diode is employed at collector.

Note: The diode is assumed to be perfect.

The equations to be solved are as follows:

$$1. \quad I_e = \frac{F - V - V_{be}}{H_{11}(I_e) + R_E}$$

$$2. \quad V_{be} = I_e \cdot H_{11}(I_e)$$

$$3. \quad w = \frac{\frac{1}{R_1} V \left( \frac{1}{R_2} + \frac{1}{R_3} \right) + \frac{1}{R_2} \left( I_L + \frac{E}{R_3} \right)}{\frac{1}{R_1 R_2} + \frac{1}{R_1 R_3} + \frac{1}{R_2 R_3}}$$

$$4. \quad u = \frac{\frac{1}{R_2} \left( \alpha I_e + \frac{V}{R_1} \right) + \left( \frac{1}{R_1} + \frac{1}{R_2} \right) \left( I_L + \frac{E}{R_3} \right)}{\frac{1}{R_1 R_2} + \frac{1}{R_1 R_3} + \frac{1}{R_2 R_3}} \quad (I_e = \alpha I_c)$$

Before the foregoing equations are solved in each of the four cases, the various parameters are multiplied by tolerance factors corresponding to "worst case" conditions.

An iterative procedure is employed to solve equations 1 and 2 since  $H_{11}(I_e)$  is a nonlinear function which is entered in tabular form.

## CHAPTER 5

### ASYNCHRONOUS CIRCUITS

#### 5.1 Introduction

In Chapter 4 some general remarks were made concerning the desirability of using asynchronous dc-coupled logic in the construction of the computer. Here we shall look more closely at the properties of asynchronous circuits and assess their advantages and disadvantages from various standpoints. Since it is proposed that the computer under discussion be built using the asynchronous philosophy, it is important to determine whether or not such a design is feasible and what its advantages and cost will be. It is further proposed that the behavior of the computer be made independent of the relative speeds of its elements whenever this can be done conveniently, and special scrutiny will be made of such circuits and their properties. As was pointed out in Chapter 4, one may attach special importance to a design which follows these principles when the circuits under consideration are so fast that the time required for information to flow from one part of the computer to another is comparable to the times required by the logical elements. By designing the computer so that its behavior is independent of the relative speeds of the elements, one may ignore the problem of matching speeds and synchronizing signals to achieve correct operation, thus separating the problems of logical design and physical layout.

#### 5.2 Speed and Complexity

A given computer may be designed from fewer logical elements if the synchronous philosophy is followed, since one need not include the additional elements which are required in asynchronous circuits to generate completion signals and hold information while it is in transit. This is true, in general, since any asynchronous computer could be made to work in the synchronous mode, but a synchronous computer may contain intolerable "races" if it is allowed to run asynchronously. Nevertheless, the situation is not entirely one-sided since a synchronous computer must contain the additional electronic equipment required by the clock and its gates.



It is also true that additional time is required by certain asynchronous circuits to form signals which indicate the completion of one operation and which initiate the next. As was mentioned in Chapter 4 such times must be balanced against the time which must be wasted by setting the clock period so as to be safely longer than the time taken by the slowest-acting combination in a synchronous system. The latter time, however, is likely to be longer since the clock frequency must be adjusted to the slowest of all the synchronized operations. Thus, depending on the method of synchronizing, a shift may take as long as an addition. In order to avoid such discrepancies it is necessary to use multiphase clocks and a correspondingly more complicated control. No matter how complex the synchronous system may be, it will only approximate the speed of the asynchronous system, discounting the time tolerances which are allowed in the synchronous system and the time to generate completion signals in the asynchronous system. Time tolerances in a synchronous system depend upon the margin of safety which one wishes for the system and upon the variation of response times which may be expected among the circuit elements. In the case of transistors the variation of characteristics is large, and for this reason the time tolerances must be taken as correspondingly large. An even greater variation may be expected among elements which have been aged for some time in the computer, and it is with respect to this latter variation that the time tolerances must be set for a synchronous system.

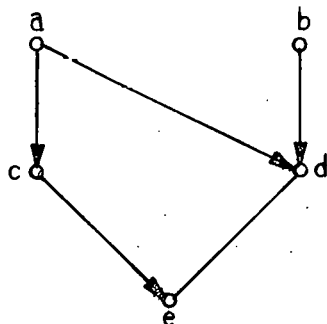
### 5.3 Design of Asynchronous Circuits

Very powerful methods are known for carrying out the logical design of synchronous circuits. These methods reduce a large part of the design to no more than an exercise in algebra, and they permit the design of significantly simpler computers than the more empirical methods which also require more time and effort on the part of the designer.

Until recently, no similarly elegant methods were known for designing asynchronous computers because the theory of asynchronous circuits was incompletely developed. Hence, some computer designers were tempted to resort to inferior engineering techniques so that the logical design could be carried out

in a straightforward manner. This situation no longer exists and systematic methods have now been invented in connection with the theory of asynchronous circuits which permit the application of logical design techniques to asynchronous design which were heretofore used only in the design of synchronous circuits. These methods will be illustrated in the latter part of this chapter. It is true that the processes are somewhat more complicated, but this seems a small price to pay for the engineering advantages which result from using asynchronous circuits. In any case, the formal logical design of the computer circuits represents a very small part of the work which must be done in the over-all design of the computer, and may be discounted now that known methods exist for carrying it out.

It was thought at one time that the signal changes taking place within an asynchronous computer must occur in a definite time sequence if the behavior of the computer was to be independent of the relative speeds of the elements. This requirement could only be satisfied by a completely serial machine, and the resulting slowing of the machine would be prohibitive. With the development of the theory of asynchronous circuits it was shown that parallel actions could indeed occur in asynchronous computers without giving rise to undesirable "race" conditions among the signals. Briefly, this may be accomplished by having logical elements within the circuit whose logical properties are such that they respond only when all of several incoming signals appear, thus yielding a completion signal indicating that all of several parallel operations have taken place. Not only is there no restriction upon the amount of paralleling which may be done in this way, but one is permitted to use much more complex paralleling schemes than are possible with synchronous circuits. For example, if five operations, a, b, c, d, and e, are required of a circuit, and if the completion of a and b are required for the initiation of d while only a is required for the initiation of c, and c and d are required for the initiation of e, we may represent the requirements by means of an ordering diagram of the type shown below.



Using recently-developed synthesis techniques for asynchronous circuits one can design a circuit which corresponds to this diagram, and which would permit c to begin even if the completion of b were delayed. A synchronous circuit could not easily be made to yield similar flexibility, for while a and b could be carried out simultaneously, the initiation of c would not occur until the next clock pulse following the completion of both a and b.

#### 5.4 Location of Malfunctions and their Repair

Many techniques are available for locating malfunctions in synchronous and asynchronous machines. The particular techniques employed usually depend upon the nature of the malfunction, and without knowing what sorts of malfunctions will appear most frequently in the system under consideration, one can hardly assess the problem. Experience with the Illiac has shown that a malfunction of the control often results in the stopping of the computer. When this occurs it is possible to trace the trouble by merely measuring voltage levels at critical points. It is hoped that if the asynchronous principles are adhered to even more strongly in the new machine, it will be correspondingly easier to service.

A further advantage in the location of malfunctions in an asynchronous computer results from the fact that information is held in flipflops. As a result, it is possible to slow down operations so that they may be observed individually and the individual gates may be checked. Such checks are impossible in those synchronous computers where the information is circulating at high speed through the logical elements and is lost if the clock frequency is reduced.

#### 5.5 Reliability

It seems likely that as a transistor develops faulty behavior it will exhibit this by going into saturation during otherwise normal operation. This would have the effect of slowing down the action of the logical element involved. In a synchronous circuit this type of malfunction would be likely to cause an

error in computation, but in an asynchronous circuit the effect would be merely to slow down the operation of the computer, provided it is built using the principles of speed independence which will be explained later. Such slowing of the computer could be detected by using checking routines and the fault could be located before it caused an incorrect calculation.

Some of the newer and more elaborate synchronous computers have been designed to operate correctly regardless of how slowly or rapidly the clock may run. Certainly, such computers possess most, if not all, of the advantages of an asynchronous machine. By the same token, however, they are correspondingly more complicated and tend to require as many additional logical circuits as an asynchronous computer. Hence, by eliminating the clock and resorting to completely asynchronous operation, one may actually achieve the same results, while obtaining a faster machine with fewer elements.

## 5.6 Asynchronous Principles

We now turn to a brief discussion of the principles of asynchronous circuit design, in which intuitive notions will be substituted for the more lengthy rigorous treatment appearing elsewhere.<sup>(1)</sup> In this theory one must make two important approximations. First, all signals are taken as assuming only discrete values. In the binary case this means that only the two signal values 0 and 1 are allowed. In physical circuits the signals do in fact, assume any of a continuum of values. Therefore one may make the approximation by splitting the continuum into two bands corresponding to 0 and 1, separated by a region of indeterminacy. The approximation now consists of saying that the logical behavior of a given element is independent of where the incoming signals lie in the bands.

---

1. D. E. Muller and W. S. Bartky: "A Theory of Asynchronous Circuits", Digital Computer Laboratory Reports Nos. 75 and 78, to be published in the Proceedings of a Symposium on the Theory of Switching, Harvard University Press.

Second, we must say that no variations occur in the position of the region of indeterminacy which are large enough to cause two elements to interpret the same incoming signal differently, that is, so that it appears in the 0 band for one element and in the 1 band for another element. All signals must also be assumed to be of sufficient amplitude to pass through the region of indeterminacy, as they change, and to enter the opposite band from which they came after reaching the region of indeterminacy.

Although voltage tolerances are important, no assumptions are made concerning speeds. Therefore, we must inspect all possible sequences of states which may occur. A circuit will be said to be speed-independent if the final condition of the circuit does not depend on the relative speeds of the logical elements which make it up. This assumes that the circuit was started in a given initial condition, and then allowed to run until it either stopped or entered some sort of never-ending cycle. In terms of computers this means that a speed-independent computer will always produce the same results and stop in the same way when fed a certain problem, regardless of how the speeds of its elements may vary. Speed-independent circuits have properties which are of particular importance in the design of reliable asynchronous circuits. For this reason we shall shift our attention from the more general problem of designing asynchronous circuits to the design of speed-independent circuits. The concept of speed-independence may be easily confused with checking. It is possible to have a speed-independent circuit which is not checked and a checked circuit which is not speed-independent. For example, one may check the operation of a given circuit by duplicating it, element for element, and comparing the resulting signals from the two circuits with a third circuit. If the signals disagree the third circuit will cause the computer to stop, indicating an "error" in operation. This "error" may be due to the unusually slow or fast action of a circuit element, or it may be due to the appearance of a spurious signal. If the former situation exists we see that the circuit is not acting in a speed-independent way by our previous definition but the "error" will nevertheless be detected so that the circuit is checked. On the other hand, if the original circuit has been made speed-independent it would be possible for a spurious signal from some element to give rise to an incorrect calculation but it could never result from a variation in the speeds of the elements. In other words, unusually fast or slow action of any of the elements is simply not regarded as an error-producing malfunction.

Three techniques are used in the design of speed-independent circuits. These techniques are generally not used independently but together, to give the most effective results. They permit the design of a class of circuits which, while speed-independent, is slightly more restricted in character than the class of speed-independent circuits. The circuits in this more restricted class, which we shall call semi-modular, have the property that no element in such a circuit which is excited (the signal which it produces is tending to change) is ever allowed to pass to equilibrium unless the signal produced by the element actually does change. It can be shown that if this condition is adhered to for all elements and all possible states of the circuit, then the circuit must be speed-independent. The restriction that circuits be semi-modular is not severe since it still permits the paralleling of operations taking place within the circuit. All of the essential circuits which are required in the control and arithmetic units of a computer have been designed by using the three techniques to be described here. Any parallelism which is possible in the operations may be retained in the semi-modular design.

An example of a violation of semi-modularity is shown in the familiar flipflop circuit of Figure 5.1. This flipflop, consisting of two AND NOT elements, is in an unstable condition in which all signals appear on all lines have the value 1.

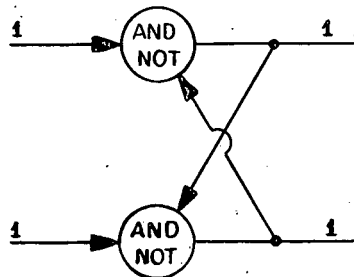


Figure 5.1  
Flipflop

The flipflop could have reached this condition if both incoming signals were initially 0 and if both were then changed to 1 simultaneously. The outputs will either go to the 1, 0 or 0, 1 configuration depending upon which of the AND NOT elements acts first. When such a change occurs, the AND NOT element which failed to act will be brought into equilibrium and the condition of semi-modularity will be violated. This circuit is also not speed-independent, since either of two final conditions is possible depending upon the relative speeds of the elements.

### 5.7 Method of Combining Blocks

In the first of the three design techniques one makes use of previously-designed circuits and connects them together to form more complex circuits. By following certain rules it is possible to retain the property of semi-modularity during the interconnection process.

This method will be illustrated by connecting two circuits, a shifting register and a counter, to form a circuit which will carry out any given number of shifts. Let us assume that the counter contains an element "a" whose signal changes from 0 to 1 to 0, n times before the circuit stops. The count n may be made to depend on the initial setting of the counter. The shifting register has an element "b" which changes from 0 to 1 to 0 every time a shift takes place. However, the shifting register is so constructed that the shifting will continue indefinitely provided the element "b" is able to continue acting. Both elements "a" and "b" may be required to feed their signals to the inputs of other elements in their respective circuits. A possible method for interconnecting the two circuits is shown in Figure 5.2. In this interconnection one of the elements, say "a", is made to feed a NOT element which in turn feeds all of the elements previously fed by "b". Element "b", on the other hand, is made to feed all of the elements previously fed by "a".

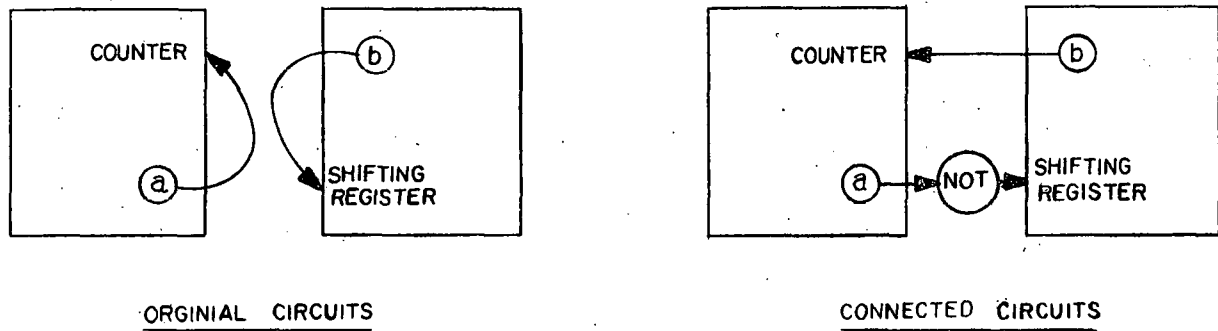


Figure 5.2  
Method for Connecting Two Circuits

With this interconnection the counter and the shifting register will act in turn until finally when the counter stops the action of the shifting register will also be arrested. That such alternate action does, in fact, take place may be verified by tracing the changes in the signals. Let us assume that initially the elements "a" and NOT have output 1 and that "b" has output 0. Thus, the NOT element is excited and tending to produce the output 0, while both the counter and shifting register are quiescent. When the NOT element acts, the shifting register will begin the process of shifting and upon completion of the first part of the operation, will cause "b" to go to 1. This initiates a count in the counter which responds by driving "a" to 0. The process may now be repeated, interchanging "0"s and "1"s, returning us to the initial configuration. The time required for the combination to come to rest will be equal to the time taken by the counter previously, plus the time for  $n$  shifts, plus  $2n$  operation times of the NOT element. If the NOT element is omitted when the circuits are interconnected it is impossible to achieve the alternate actions of the two circuits. In such a case either the signals from "a" and "b" are different and both circuits are quiescent indefinitely, or both signals are the same with the result that both circuits are tending to change in such a way as to lead to behavior which is not speed-independent.



A saving in time can be achieved by having the counter and the shifting register operate in parallel so that the times will not be strictly additive. This requires a different type of interconnection, and also that we postulate a new logical element. This element, labelled "c" in Figure 5.3 can actually be constructed from conventional AND, OR, and NOT elements, but we shall imagine it as a single element having two inputs and one output. Elements "a" and "b" are connected to the inputs of "c", and "c" is used to feed all the elements which were previously fed by "a" and "b".

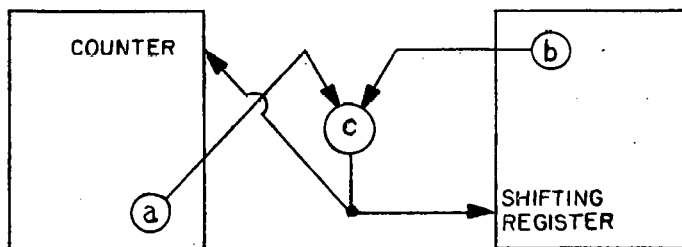


Figure 5.3  
Alternative Method of Connection

The logical properties of the "c" element are such that it will duplicate the signals at its inputs provided they agree, but if they disagree it retains its previous signal. In the latter case it has what may be called memory. The Boolean expression  $a b \vee a c \vee b c$  describes the "c" element's behavior. With the interconnection of Figure 5.3, the counter and shifting register execute their operations in parallel. If the counter acts more rapidly than the shifting register, then the initiation of the next count will be delayed until the signal from "b" appears and the signal from "c" is therefore caused to change. If the shifting register acts more rapidly, then the next shift will be delayed until the signal from "a" appears. The time taken by this combination is the greater of the times taken by the counter and shifting register plus the time taken for  $2n$  operations of the "c" element.

Even more complicated interconnection schemes than these can be devised involving two or more simple circuits and possessing more elaborate parallel-sequential relations between their operations, but the principles are well illustrated by the examples given here. All such techniques preserve the property of semi-modularity and hence are suitable for the design of circuits.

### 5.8 Method of Simulating Synchronous Circuits

The second technique permits one to design the basic simple circuits which carry out the fundamental operations required by a computer. These operations are first described in much the same way that one would describe them if one wished to design a synchronous computer, that is, by writing a set of Boolean equations to represent the operation in question.

Although this technique is most effective in the design of circuits such as those in the arithmetic unit which handle information in a parallel fashion we take as an illustrative example the two-stage binary counter circuit which cycles through the never ending sequence

(00), (01), (10), (11), (00), (01), etc.

Simpler speed-independent counting circuits may be designed by using other techniques. If we designate the first and second signals of the pair as  $Z_1$  and  $Z_2$  respectively, we may write

$$\begin{aligned} Z_1' &= Z_1 \oplus Z_2 \\ Z_2' &= \bar{Z}_2 \end{aligned} \tag{5.1}$$

to represent  $Z_1'$  and  $Z_2'$ , the pair of signals which immediately follows  $Z_1$  and  $Z_2$ . Here the symbol " $\oplus$ " indicates EXCLUSIVE OR and  $\bar{Z}_2$  indicates NOT  $Z_2$ . In a synchronous system, we may obtain the circuit directly from equations (5.1) by use of two elements each having unit time delay as shown in Figure 5.4.

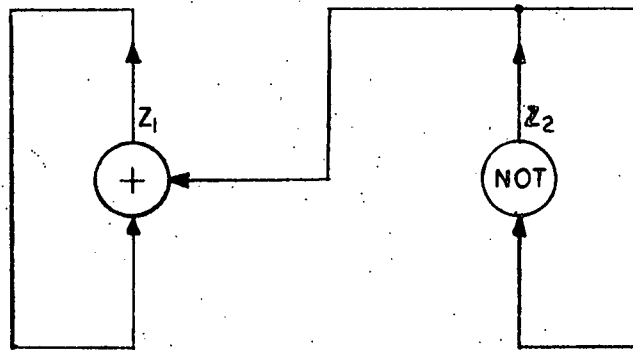


Figure 5.4. Synchronous Binary Counter

This circuit may be translated into a corresponding semi-modular circuit by means of the following systematic process. The result of this translation is shown in Figure 5.5.

First, introduce two flipflops for each signal present in the synchronous circuit. One flipflop of each pair will be assumed to have the properties of the flipflop of Figure 4.5. The other will be represented as its dual although by use of suitably placed NOT elements one may avoid using a dual flipflop. A dual flipflop is one in which the basic logical description of the flipflop is replaced by a dual description. Thus the dual of the flipflop shown in Figure 5.1 would be represented by two OR NOT elements. These flipflops serve the purpose of holding the information in the signals of the circuit of Figure 5.4. Two are required for each signal since, during the process of gating into one flipflop, the information is stored in the opposite flipflop. The flipflops corresponding to  $Z_1$  are shown on the left in Figure 5.5, while those corresponding to  $Z_2$  lie on the right.

Second, replace the logical elements of Figure 5.4 by a double wire system. Connect this system between the first and second flipflops of each pair and introduce direct connections from the second to the first. In the double wire system two lines are used to carry each signal so that during

transmission of information one line will always be 0 and the other 1. The presence of 0 or 1 on the first line determines a bit of information. In order to distinguish one such bit from the next, we intersperse transmission with clearing of the lines which occurs when both lines are given the same signal. It is this requirement of clearing which forces the use of a double wire system since three possible conditions may exist, 0, 1 or clearing. The double logic is shown in Figure 5.5 between the upper and lower flipflops. The two ANDS on the left and the ORS below then replace the EXCLUSIVE OR of Figure 5.4, while the NOT of Figure 5.4 is replaced by the crossed wires centered between the two flipflops in which the double wire signals of  $Z_2$  are interchanged.

Third, insert a completion circuit for sensing the completion of one gating operation and for causing the next. This circuit requires the use of a "c" element of the type used in the circuit of Figure 5.3. This element is fed from completion signals from the four flipflops. Such completion signals are formed from the three elements to the right of each flipflop in Figure 5.5. They indicate whether or not the state of the flipflop agrees with the sense of the incoming signal.

During the operation of this circuit the center or logical section is alternately cleared so that all signals assume the value 1, and used for generating new quantities  $Z_1$  and  $Z_2$ . In this example, the signals at the inputs to the upper flipflops are adequate to determine when the clearing operation has taken place, but when more complex logic is used it may be necessary to generate signals from additional AND elements within the logic which are also fed to the "c" element. In File Number 226<sup>(2)</sup> this technique is described and logical circuits are given for the various parts of an arithmetic unit.

---

2. James H. Shelly: "Design of Speed-Independent Circuits", Digital Computer Laboratory File No. 226, 7/19/57

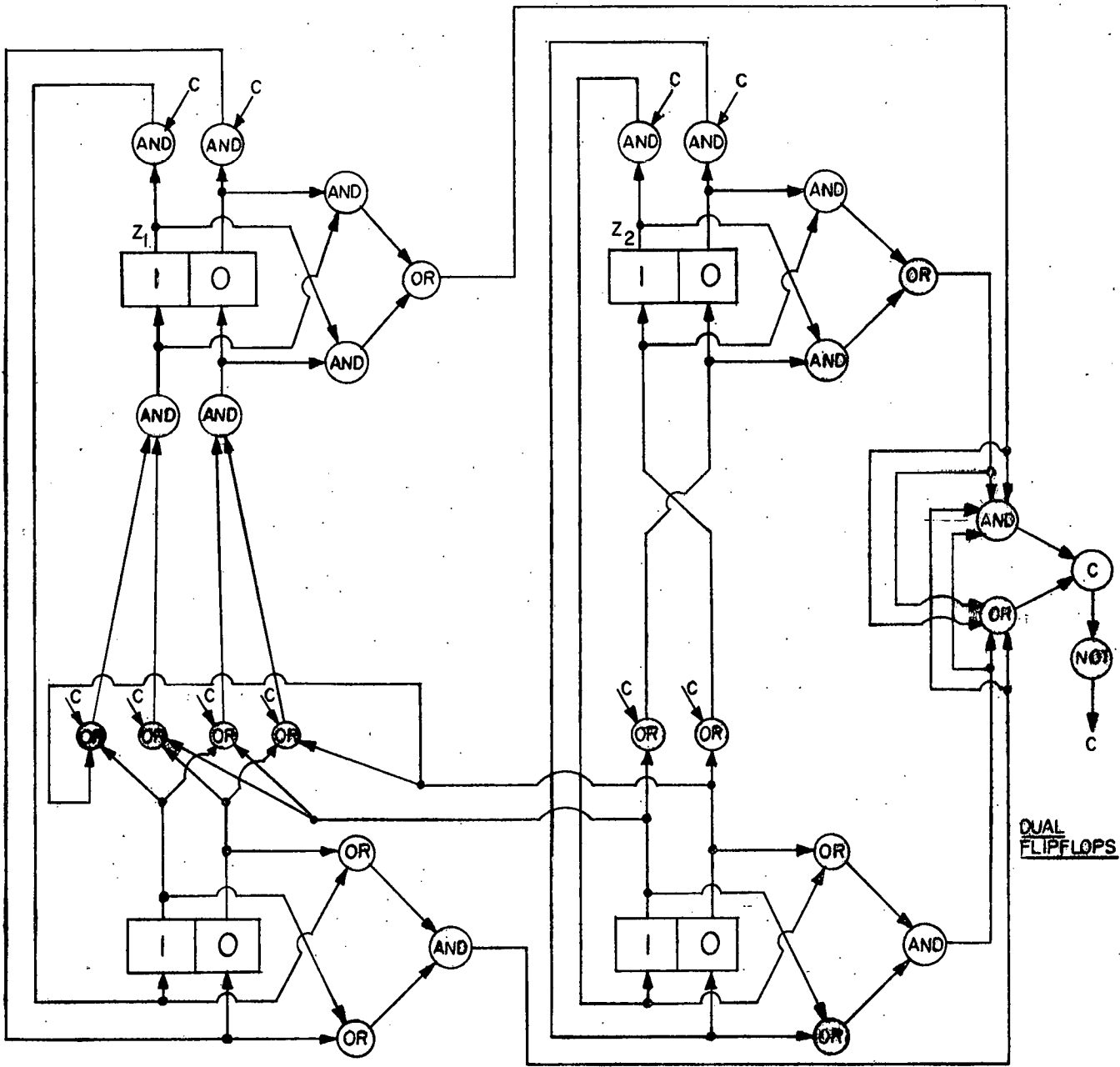


Figure 5.5

Asynchronous Binary Counter Simulating  
a Synchronous Binary Counter

## 5.9 Method of Design from Change Charts

The third technique which is used for speed-independent circuit design involves the use of what may be called change charts to design distributive circuits. Distributive circuits are slightly more restricted than semi-modular circuits but they still admit the possibility of parallel action. A change chart is simply a list of the signal changes which take place at the nodes of the circuit together with a statement of the chronological ordering of these changes. Since parallel changes are admitted, this ordering will be a partial ordering. Each change is written as a pair of positive integers  $(\alpha, i)$ . The integer  $i$  is the node number and the integer  $\alpha$  is the number of the change at that node.

The set  $\Sigma$  of changes is required to have the following properties:

1. The elements  $(\alpha, i)$  of  $\Sigma$  are partially ordered and satisfy the descending chain condition (i.e., that all descending chains are finite),
2. There exists an integer  $n$  such that  $i \leq n$  for all  $(\alpha, i)$  in  $\Sigma$ ,
3. If  $(\alpha, i)$  is in  $\Sigma$  and  $\alpha > 1$ , then  $(\alpha - 1, i)$  is also in  $\Sigma$  and  $(\alpha - 1, i) < (\alpha, i)$ .

The use of the set  $\Sigma$  to represent the behavior of a circuit is advantageous for three reasons. First, it represents a more natural way of describing the behavior of the circuit than does the conventional state diagram, since it deals with changes at individual nodes rather than states. Second, the change chart will contain many fewer elements than the state diagram when parallel changes occur. In such cases the number of changes is approximately equal to the logarithm of the number of states. Third, the circuit derived from the change chart design technique will be distributive and hence speed-independent.

The change chart  $\Sigma$  may be used to form a distributive lattice  $\mathcal{A}$  of  $n$ -dimensional vectors  $\underline{a}$ , whose components are non-negative integers, by the following rule.

A given vector  $\underline{a} = (\underline{a}_1, \underline{a}_2, \dots, \underline{a}_n)$  is in  $\mathcal{A}$  if and only if

1. for each  $\underline{a}_i \neq 0$  there is a change  $(\alpha, i)$  in  $\Sigma$  with  $\alpha = \underline{a}_i$ , and
2. if  $(\alpha, i) \geq (\beta, j)$  then  $\underline{a}_j \geq \beta$ .

It may be shown that  $\mathcal{A}$  is a distributive lattice with a zero where one defines the lattice ordering relation  $\underline{a} < \underline{b}$  to mean  $\underline{a}_i \leq \underline{b}_i$  for all  $i = 1, \dots, n$ . In this case the lattice operations of greatest lower bound and least upper bound correspond to numerical componentwise minima and maxima respectively.

The lattice  $\mathcal{A}$  is closely related to the state diagram corresponding to the change chart  $\Sigma$ . Each vector  $\underline{a}$  in  $\mathcal{A}$  may be taken as corresponding to a state of the circuit. This correspondence is a many-to-one correspondence since more than one vector  $\underline{a}$  may correspond to a given state. The correspondence is set up as follows: Let  $u = (u_1, u_2, \dots, u_n)$  be the initial state of the circuit. Here  $u_1, u_2, \dots, u_n$  are the binary signals on the nodes 1, 2,  $\dots$ ,  $n$ . Then the state  $v = (v_1, v_2, \dots, v_n)$  is formed by letting  $v_i = \text{residue}(\underline{a}_i + u_i) \text{ modulo } 2$ . Thus the component  $\underline{a}_i$  of  $\underline{a}$  may be regarded as the number of changes which have occurred at node  $i$  when going from state  $u$  to state  $v$ .

One may determine whether or not node  $i$  is excited or in equilibrium in any state  $v$  corresponding to a given vector  $\underline{a}$ . The rule which permits this determination may be stated as follows:

Node  $i$  is excited in state  $v$  corresponding to C-state  $\underline{a}$  if and only if  $(\underline{a}_i + 1, i)$  is in  $\Sigma$  and  $\underline{a}_j \geq \beta$  for all  $(\beta, j)$  in  $\Sigma$  for which  $(\beta, j) < (\underline{a}_i + 1, i)$  and  $j \neq i$ .

Using this rule we may determine whether or not any node  $i$  is excited for each vector  $\underline{a}$ . If it should happen that two such vectors, which give rise to the same state  $v$  disagree as to whether or not one or more nodes are excited, then we say that the change chart  $\Sigma$  is not realizable, while if no such disagreement occurs it is realizable. If the change chart is realizable we may write a set of Boolean functions which yield the equilibrium

and excitation conditions given. If these functions are used as elements in a circuit then it will have  $\Lambda$  for its state diagram and will follow the behavior described by  $\Sigma$  when placed in state  $u$ . This represents a completely systematic synthesis procedure, provided the original change chart is realizable. If it is not realizable we may always make it realizable by introducing additional nodes and corresponding changes on these nodes without altering the original partial ordering. Systematic methods have been devised for introducing such additional nodes.

One of the weaknesses of this method lies in the fact that the Boolean functions obtained in this way may not correspond to the relatively simple elements which have been discussed in Chapter 4. By systematic introduction of additional nodes, even if they are not required for realizability, one may also remove this objection.

The three methods summarized have proved workable in the design of logical circuits for the arithmetic unit and control. Whether or not the arithmetic unit should be designed in this fashion will be decided on the basis of speed and reliability as described in Chapter 3. It is felt that gains in speed and reliability will result if the control is made speed-independent. The third design technique can be used most effectively in this area to simplify the design and to allow a greater degree of parallelism than would otherwise be possible.

We expect the Illiac to be used effectively in carrying out the systematic part of the third design procedure. A use which has already been made of the Illiac is in the testing of circuit designs. Programs have been written for simulating the behavior of circuits with the Illiac and testing them for semi-modularity and for correct sequencing.<sup>(3)</sup> Without these programs the design of semi-modular circuits would be virtually impossible since the checking process is usually too tedious to be carried out by hand.

3. W. Scott Bartky: "Complete Circuit Analyzer", Library Routine Q3, Digital Computer Laboratory, 6/14/56.
- W. Scott Bartky: "Single Circuit Analyzer", Library Routine Q4, Digital Computer Laboratory, 11/20/56.
- James Shelly: "Complete Circuit Analyzer", Library Routine Q5, Digital Computer Laboratory, 7/22/57.



## CHAPTER 6

### MEMORY

#### 6.1 Introduction

The problems of organization discussed in Chapter 3 would have been greatly simplified if it had been found possible to increase the speed of memory devices by as great a factor as the speed of arithmetic circuits. A survey of the various forms of memory which have been used or proposed shows that very high speeds can indeed be obtained, for example, by using registers similar to those of the arithmetic unit, for storage purposes. The cost of such very fast memories is excessive, except possibly as small auxiliary parts of a memory hierarchy. For the main random-access memory something much less costly is required. Our efforts here have been largely devoted to a study of ferrite-core memories. Some preliminary study of electrostatic memories and diode-capacitor memories was also made.<sup>(1)(2)</sup> These tests did not determine definitely the limits of speed of either of these devices even when considered as small auxiliary memories. However, it became apparent that work on these lines was less likely to be rewarding than work on magnetic memories, so further effort was concentrated on the latter.

The speed of conventional ferrite-core memories, which use a three-dimensional coincident-current method of selection, is limited by the resulting 2:1 current selection ratio. For available cores this fixes the minimum cycle time for such memories at about 4  $\mu$ s. Other arrangements of ferrite cores have been studied, one of which, the word-arrangement memory described in section 6.3, may be several times faster.

1. G. H. Leichner: "Proposed High-Speed Williams Memory Tests", Digital Computer Laboratory File No. 219, May 28, 1957.
2. K. C. Smith: "Diode-Capacitor Memory", Digital Computer Laboratory File No. 218, May 10, 1957.

Memories using thin magnetic films give promise of high speeds but are not discussed in this report because much further research is needed before their potentialities can be properly assessed. Memories using ferrite cores with more than one hole are also omitted in this study because of lack of data concerning them.

## 6.2 Comparison of Some Types of Memory

In this section an attempt is made to show how increased speed is related to increased complexity for several types of memory. In order to form a useful basis for comparison of diverse forms of memory some over-simplification has been necessary. Since both reliability and cost of maintenance may be expected to be related to the number of tubes, transistors and diodes used, the comparison has been based on a count of these active elements. Initial cost would, of course, be influenced by other factors also.

To reduce the count of active elements to a single measure, a weighted total has been given which is equal to  $T + 2S + 5L + .5D$  where T, S, L and D refer to the numbers of transistors, small tubes, large tubes and diodes respectively. The weighting factor 5 for large tubes is based on expected lifetime as compared with transistors, and the factor 2 is based on the assumption that most of the small tubes (pentodes or double triodes) could be replaced by transistors, but that on the average twice as many transistors would be required.

It would be desirable to replace the high-current tubes also with transistors. If, as now seems likely, this is possible, some change in the weighted totals would result but the relative position of the various memory designs would not be significantly altered.

The comparison of speeds, in the last column of Table 6.2, is also somewhat qualitative because, with the exception of conventional core memories and registers, none of the memories compared in the table have been built with the speeds shown. The data given in the table were mainly calculated by extrapolation from experimental results obtained using models of parts of memories.

Considerable variety is possible in the design of any type of memory so that certain assumptions are necessary. For core memories we have assumed that

1. current drivers require 1 large and 2 small tubes,
2. signal amplifiers and related logical circuits require 9 small tubes per digit, and
3. decoding networks followed by level restorers to provide sufficient signal to actuate current drivers require 40 transistors for 8 drivers or 288 transistors for 64 drivers.

There may be some question whether better values may be obtained for the expenditure of a given number of tubes by making several separate memories, or by making a memory in which blocks of several words are read at once in place of a single memory with one-word access. The formulas below are made sufficiently general to allow such comparisons to be made. In addition, results for certain arrangements are given in tabular form.

Let  $A$  be the number of separate memories,  $B$  the number of  $n$ -bit words to be read simultaneously as a block, and  $C$  the number of blocks in each memory. The total number of words is then  $ABC$ . A conventional memory requires  $4C^{\frac{1}{4}}$  current drivers while a word-arrangement memory<sup>(3)</sup> requires  $2C^{\frac{1}{2}}$  current drivers. A conventional memory requires  $nB$  drivers for digit-inhibit windings, and a word-arrangement memory requires  $2nB$  such drivers.

On the basis of the above assumptions the following active components are required:

---

3. See section 6.3.

Table 6.1  
Total Number of Active Elements for ABC n-bit Words

Elements	Conventional Memory	Word-Arrangement Memory
Large Tubes	$A(nB + 4C^{\frac{1}{4}})$	$A(2nB + 2C^{\frac{1}{2}})$
Small Tubes	$A(11nB + 8C^{\frac{1}{4}})$	$A(13nB + 4C^{\frac{1}{2}})$
Transistors	$20AC^{\frac{1}{4}}$	$10AC^{\frac{1}{2}}$
Weighted Total	$A(27nB + 56C^{\frac{1}{4}})$	$A(36nB + 28C^{\frac{1}{2}})$

Table 6.2  
Comparison of Types of Memory

Type of Memory	Active Elements per Word (52-bit)					Operation Time $\mu$ s
	Large Tubes	Small Tubes	Transistors	Diodes	Weighted Total	
1. Conventional Core Memory 8192 words	.011	.080	.023		.24	6 <sup>(1)</sup>
2. Same, 4-word blocks	.029	.29	.016		.74	1.5 <sup>(2)</sup>
3. Word-Arrangement Core Memory 8192 words	.035	.12	.11		.54	1.5
4. Same, 2-word blocks	.040	.20	.08		.69	.75 <sup>(2)</sup>
5. Same, 4-word blocks	.063	.36	.06		1.1	.37 <sup>(2)</sup>
6. 8-word Delay Lines			50	50	75	.2 <sup>(3)</sup>
7. Registers			260	1200	860	.05

(1) Based on existing memories. Up to 50% greater speed may be possible.

(2) Time per word (assumes all words in block used).

(3) Average access time.

In Figure 6.1, the number of active elements required is plotted against total number of words, ABC. The different curves are labelled by the values A,B followed by c for conventional memories and w for word-arrangement memories.

Three systems might be considered roughly comparable since their (not necessarily random) access-time per word is  $1.5 \mu\text{s}$ : These are the word-arrangement memory reading one word at a time (1, lw), the conventional memory with four-word blocks (1, 4c) and four separate conventional memories (4, 1c). Of these the word-arrangement memory requires fewest active elements, although this advantage is lost for very large memories.

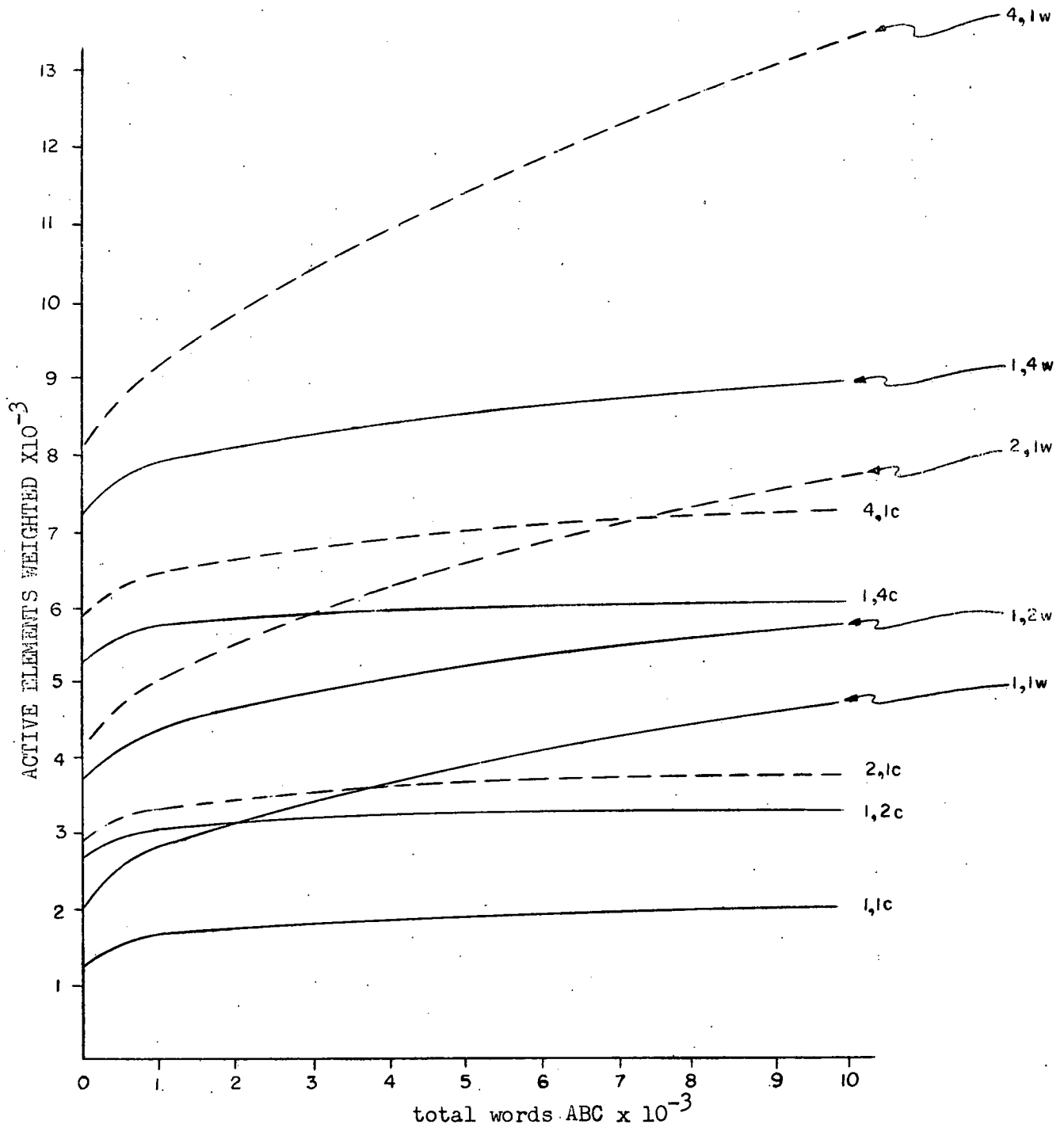
### 6.3 Word-Arrangement Memory

The word-arrangement memory first proposed by the National Bureau of Standards<sup>(4)</sup> may be made considerably faster than the conventional core memory. There is no limitation on current selection ratio for the read pulse, and a 3:1 selection ratio is possible for the write pulse, while conventional memories are limited to 2:1 for both. An access time of  $1.5 \mu\text{s}$  should be possible with the word-arrangement memory.

Depending on the choice of control system, the memory may be constructed to read out 4-word blocks or to read out separate words. The latter arrangement is illustrated in Figure 6.2. The method of operation can be seen from this diagram.

The memory core matrix consists of 8192 columns and  $2 \times 52$  rows of cores, divided in some convenient way into separate frames. Each column makes up one block of words, and a pair of adjacent cores represent each bit. One wire, labelled W, runs through all cores of one block of words. Three other wires perpendicular to this one run through each core. Two of these carry pulses from bit-selections drivers and the third is the sensing wire.

4. National Bureau of Standards: "Progress on Computer Components", October 1954 - March 1955.



NOTE: Curves are marked with values of A, B followed by c for conventional and w for word-arrangement.

Figure 6.1  
Comparison of Word-Arrangement  
and Conventional Core Memories

574 142

-156-

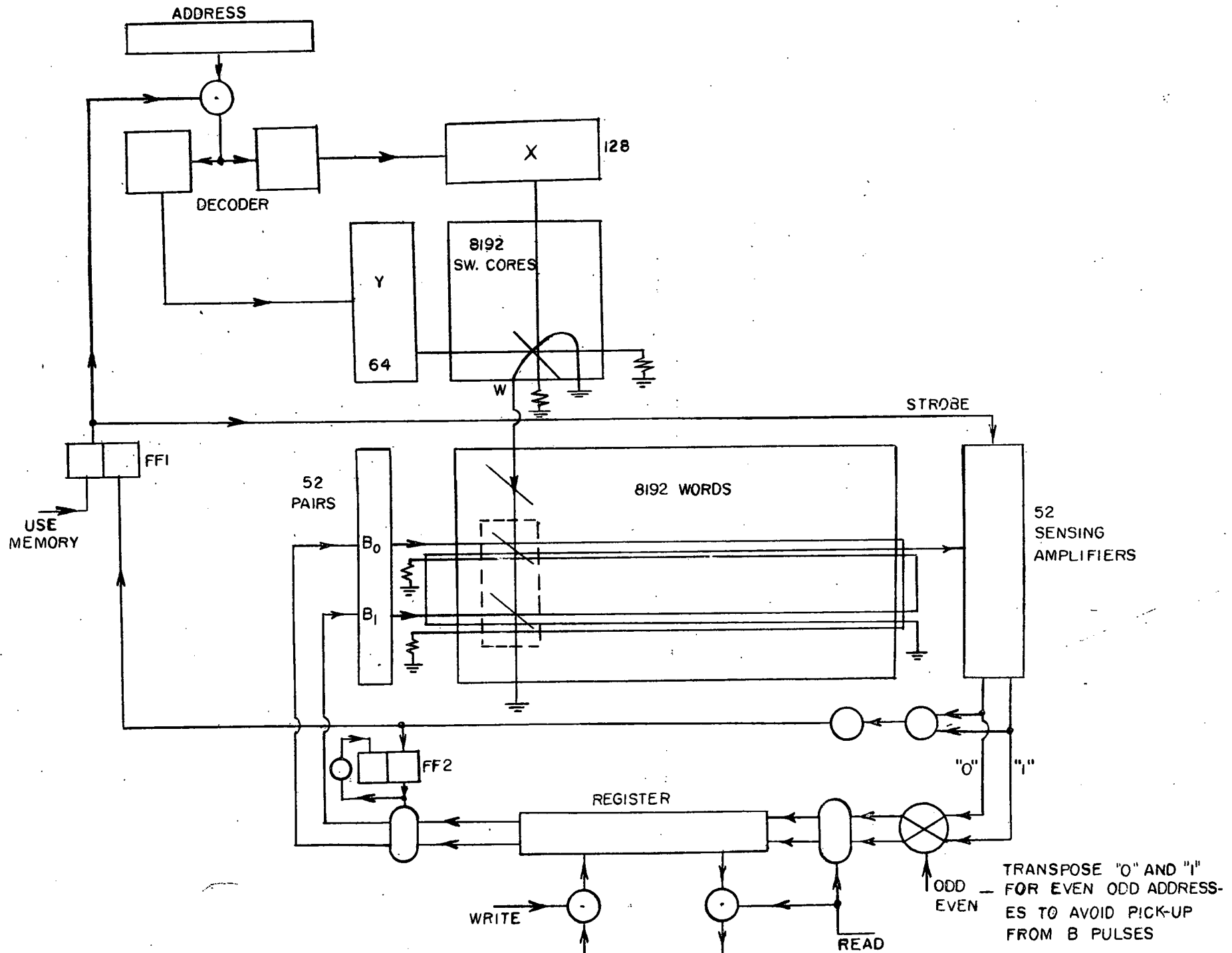


Figure 6.2  
Proposed Core Memory

TRANSPOSE "0" AND "1"  
FOR EVEN ODD ADDRESS-  
ES TO AVOID PICK-UP  
FROM B PULSES

The reason for using two cores to represent each bit is primarily to provide a constant load for the switch cores through the W wires. In the arrangement described there will always be 52 cores changing state on each pulse no matter what combination of 0's and 1's is stored. Some additional advantages of the two-core representation are mentioned later in this chapter.

The sensing process is destructive and must be followed by a "rewrite" process. In storing information, the "write" process must be preceded by a "clear" process which is the same as the sensing process. Thus both sensing and storing require essentially the same sequence of operations. This sequence is described in the next paragraph.

A signal indicating that the memory is to be used sets flipflop 1 allowing the address to be gated through the decoder to the X and Y drivers. The Y drivers are normally on, permitting the flow of current, biasing the switch cores in a given state. The current in the selected Y winding is reduced to zero, and the selected X driver is turned on resulting in the reversal of the selected switch core. This produces a pulse of 1.2 amperes in the wire marked W which passes through all cores of a given word. A pair of cores representing one bit, for example the cores in the dotted rectangle, are normally in opposite magnetic states. The W pulse brings whichever core was in the minus state to the plus state. The direction of the resulting induced pulse in the sensing winding depends on which core was switched, that is, on whether a "1" or a "0" was stored. The sensing amplifier produces a pulse on either the "1" or "0" output and sets the corresponding bit of the register in the appropriate sense. The output of the amplifier also sets flipflop 2 and resets flipflop 1, turning on either driver  $B_1$  or  $B_0$ , and returning the X and Y amplifiers to their normal states, thus producing a reverse pulse of 0.8 amperes in wire W. Assuming that driver  $B_0$  was the one turned on, the 0.4 ampere current produced by this driver opposes the 0.8 ampere current in the W wire in the top core, leaving a net current of 0.4 amperes which does not change the state of this core. In the lower core of this pair, however, the currents add, giving 1.2 amperes, and returning this core to the minus state. The 0.4 ampere current in the  $B_0$  winding passing through cores of unselected words is insufficient to alter their states.



The only difference in the operation cycle when information is stored instead of read is that the register is set by information from the arithmetic unit instead of from the sensing amplifier.

The X and Y lines through the switch cores and the  $B_0$  and  $B_1$  lines form transmission lines or low-pass filters, and must be terminated to avoid reflections. The W lines, on the other hand, pass through relatively few cores and half of these cores are reversed during each pulse so that these lines form a resistive load which varies in magnitude during the cycle but which does not depend on the number stored because of the use of compensating cores. Therefore no external resistance is needed to terminate these lines, greatly reducing the power which must be supplied by the switch cores.

In most memory applications, cores are completely switched from one maximum remanent state to the other. This is not necessary when compensating cores are used since reversible flux changes in the two cores cancel out and a good signal-to-noise ratio may be obtained even when the cores are only partly switched.

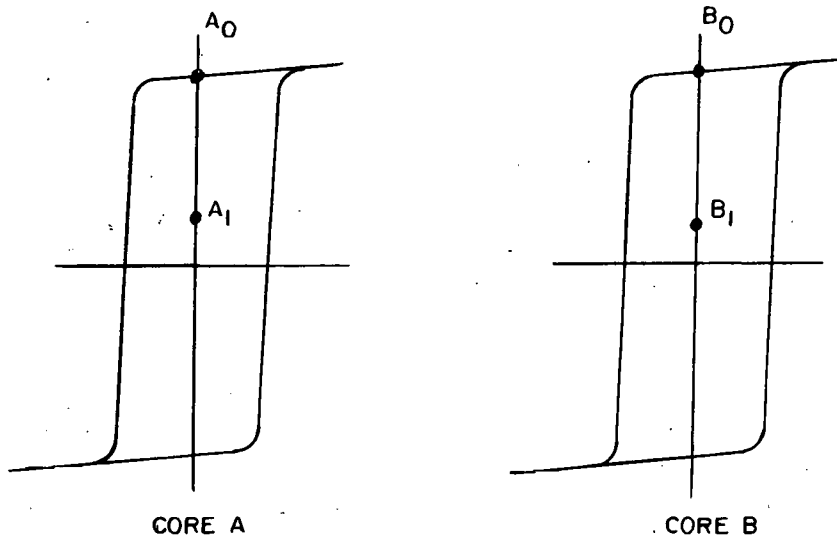


Figure 6.3  
Hysteresis Loops for a Pair  
of Cores Representing One Bit

The two hysteresis loops in Figure 6.3 represent the two cores of a given bit. Before the first  $W$  pulse starts suppose core a is in state  $A_0$  and core b in state  $B_1$ . The "read" pulse brings core b into state  $B_0$  also. During the "write" pulse either core a or b will be brought to state  $A_1$  or  $B_1$  depending on the digit to be written, the amount of flux change being limited by the available flux change of the switch core. Partial switching has several advantages. It reduces the power dissipated in the memory cores making the problem of heating less serious, and it reduces the size of the switch cores and the power required to drive them. Unfortunately, it does not reduce the value of the current required. A further advantage of partial switching lies in the possibility of non-destructive sensing which is discussed in the appendix to this chapter.

#### 6.4 Experimental Results

Experimental tests on a one-word model<sup>(5)</sup> were made to verify the predicted  $1.5 \mu s$  access-time, and to study problems related to current regulation, choice of switch cores, etc. The model memory consists of 100 cores (General Ceramics Type S1, Size F-394) strung on drive wires so that each pair of cores represents one bit of a 50-bit word. A block diagram is given in Figure 6.4. As only two bit-selecting amplifiers were constructed, these 50 bits cannot be selected independently. Only two combinations, 1010 ....10 and 0101 ....01, can be selected. Several operations are possible, reading and writing either word repeatedly, reading one word and writing the other alternately, or several simple sequences controlled by a counter. These operations should be sufficient to determine whether or not the first few cycles following a change in word stored are appreciably different from steady-state cycles.

Certain limits are placed on the values of current pulses by the fact that the model is intended to represent part of a larger memory.

1. Bit-selection pulses must not exceed the value of  $I/2$  for  $S_1$  cores (0.42 amp) so that cores of unselected words would not be affected.

---

5. R. W. McKay, N. N. Yu and C. Pottle: "A One-Word Model of a Word-Arrangement Memory", Digital Computer Laboratory Report No. 79, May 1957.

574 176

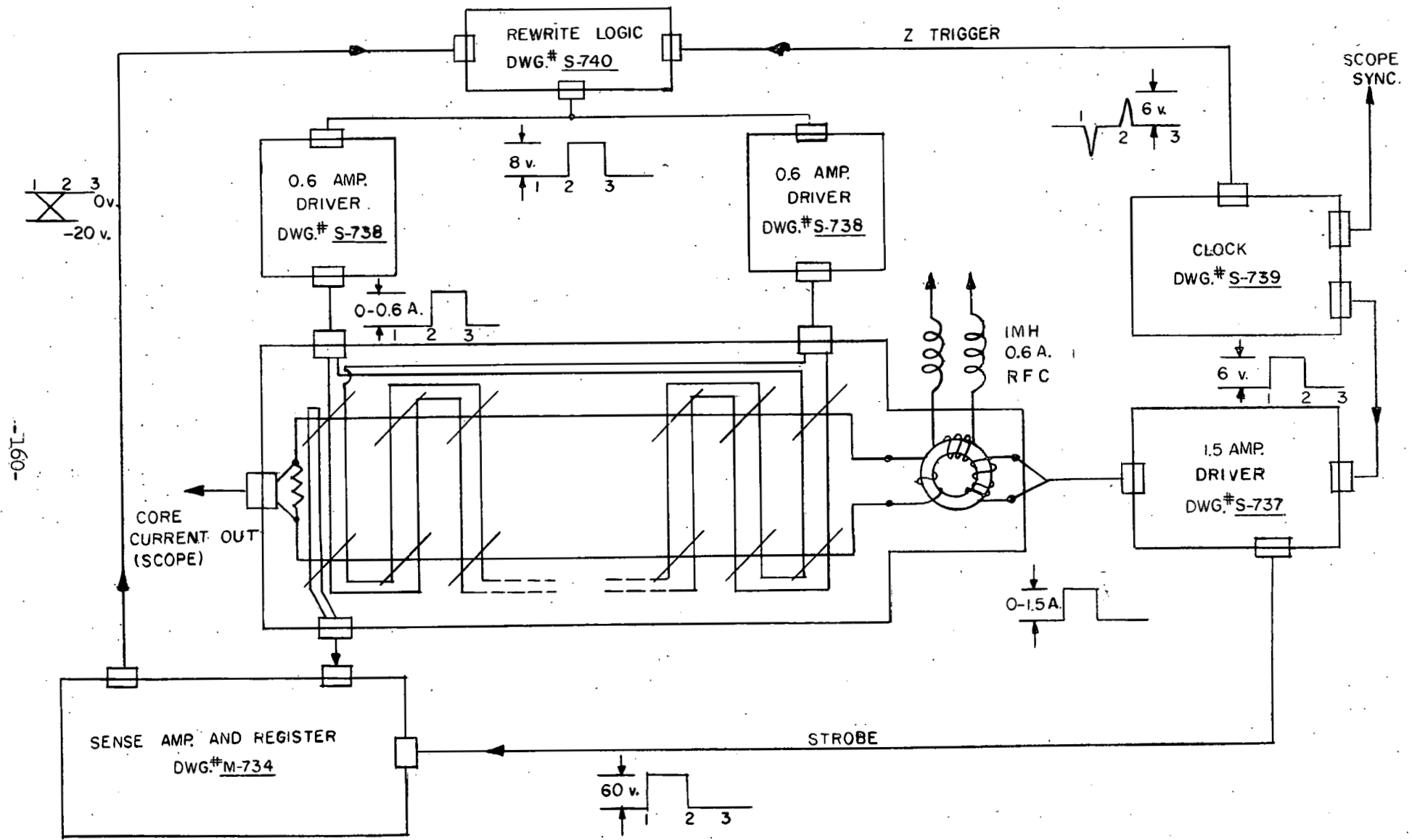


Figure 6.4. One-Word Core Memory

-160-

2. The output pulse from the switch cores on the write part of the cycle should not exceed the value of the bit-selection pulse plus  $I/2$  but this limit is not critical since the pulse is applied only to cores of the selected word, and a small change in the core which is supposed to be held fixed does not matter as long as the other core changes much more.
3. If the X and Y drivers are to be simple on-off switches the X pulse must not exceed the Y pulse by more than  $i_0$  for the switch core.

A switch core may be a very inefficient pulse transformer because the energy required to reverse the magnetization of the core itself may exceed the energy output. It is readily shown that the energy lost in the switch core is least when the radius is least. However, the area of cross-section must be great enough to supply the required flux change. Therefore long narrow cylinders would be ideal switch cores. A more practical arrangement having the same desirable features is provided by a switch made up of a number of 0.08 inch ferrite cores strung together. The heating effects should also be small in such a switch because of the large surface-to-volume ratio.

Switches made up of small ferrite cores in this way have been used in most of the tests on the one-word model. A few tests were made using permalloy-ribbon cores. Ferrite cores seemed to be preferable to permalloy cores for switches with the type of load provided by the W-lines in this model, because damped oscillation or "ringing" was produced by the permalloy cores. This effect could be eliminated by means of external resistances but at the cost of considerable wasted power.

Partial switching of the memory cores, as mentioned at the end of section 6.3 can be produced by decreasing the number of ferrite cores used in the switch and thus limiting the total flux change. Since there is no direct method of measuring the state of magnetization,  $\phi$ , of a core at any instant, this quantity must be determined by integrating the output voltage curve which is proportional to  $\frac{d\phi}{dt}$ . The output waveform from a memory core with partial switching is shown in Figure 6.5. The application of partial switching to produce a method of non-destructive sensing is described in the appendix.

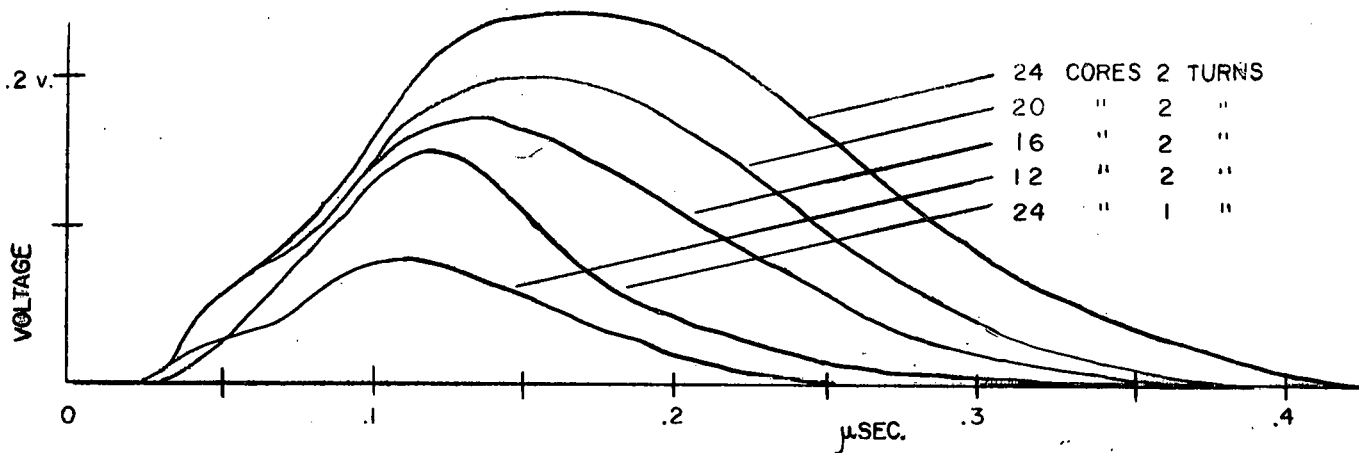


Figure 6.5  
Output Waveforms for Partial Switching

The complete read-rewrite cycle, in the model takes  $0.8 \mu\text{s}$ . However, in a full-scale memory, this basic cycle time may be increased due to variations in characteristics of tubes and other circuit components. The total random-access time of a large-scale core memory is dependent upon the capacity of the memory. One additional delay in a large core memory is the transmission time for pulses traveling through many cores. Since magnetic switches are used in the memory, transmission delay of pulses through these switches would also have to be accounted for. Estimates are available for the transmission time of pulses through memory cores ( $20 \mu\text{s}/1000$  cores). An approximate measure of the delay in switch cores of the type used in the model was found to be  $3 \mu\text{s}$  per switch.

To verify the cycle time of approximately  $1.5 \mu\text{s}$  for a full-scale memory, various delays would have to be accounted for. The following table shows the delay times which might be expected for such a memory.

Table 6.3  
 Estimated Time Delays in a Full-  
 Scale Word-Arrangement Core Memory

Basic cycle	0.85 $\mu$ s
Decoding	0.05 $\mu$ s
X or Y drivers	0.15 $\mu$ s
Bit-selection drivers	0.15 $\mu$ s
Readout Amplifier	0.05 $\mu$ s
Setting the Register	0.03 $\mu$ s
Transmission	T(n) $\mu$ s

The value of T(n) depends how the sensing wires and digit selection wires are arranged. The maximum value for 8192 words is approximately 3  $\mu$ s for each of 128 switch cores plus 20  $\mu$ s for each thousand cores which would give approximately .5  $\mu$ s. However, this time may be reduced considerably by dividing the long transmission lines into sections and by various other means. A value about .25  $\mu$ s seems quite possible.

## Appendix to Chapter 6

### Possibility of a Non-Destructive Readout

A possibility of non-destructive sensing arises from the fact that one of the cores representing a given bit may be only partially magnetized (or even practically demagnetized) whereas the other is completely magnetized. The slopes of the reversible magnetization curves for cores in these two states differ considerably. If the slope of the flux-current curve for the nearly-saturated core is represented by  $L$  henries and that of the partially-magnetized core by  $L'$  henries, the output voltage from the pair of cores when a current pulse passes through is  $+(L'-L)\frac{di}{dt}$ , where the sign depends on whether a "0" or "1" is stored. Provided the pulse is less than 0.4 amperes no permanent change in the state of the cores would occur. Some preliminary tests indicate that  $\frac{L'}{L}$  is approximately 1:9 and that this ratio is almost independent of current so that comparatively small currents could be used.

Preliminary experiments indicate that an extremely fast readout may be obtained by this system. In fact, the main delay may be that due to transmission of signals through the array of cores. To obtain the fastest readout times it might be necessary to apply this method only to a small section of the memory.

Problems related to signal-to-noise ratio exist which have not yet been thoroughly investigated. These problems do not appear to be insuperable, and if they can be overcome, a small memory with reading time of .2 to .3  $\mu$ s may be possible. Of course, the "write" time into this memory would be the same as in the destructive readout memory.

## CHAPTER 7

### INPUT, OUTPUT, AND AUXILIARY STORAGE

#### 7.1 Introduction

It is the purpose of this chapter to present a number of rather general considerations relating to the requirements for input, output and auxiliary storage devices necessary for a computer having the arithmetic and internal storage characteristics described in other parts of this report. The results are incomplete in that they are insufficient to indicate the detailed control structure and instruction code necessary.

The input-output equipment of a computer plays two somewhat distinct roles: (1) it serves as a means of communication between the computer and other automata or humans, and (2) it serves as an auxiliary storage medium for a computer. The distinction between auxiliary storage equipment and input-output equipment can sometimes be made in terms of the length of time information is stored on the equipment. Thus magnetic drums are quite often used as auxiliary storage for short-term storage of partial results of a calculation which are to be used fairly soon in a subsequent computation. Although magnetic tapes are used for this purpose they may also be used for storage of results away from the computer and for long periods of time. The use of the input-output equipment as a short-term auxiliary storage device is the most demanding one with respect to speed.

In this chapter we shall review the characteristics of input-output devices now commercially available and compare their data rates, the number of bits per second that can be read from or written on these devices, with the multiplication rate of arithmetic units and internal storage access rates. Comparison of data rates will also be made with the rates of generation and consumption of data by the human user.

For many scientific calculations the time spent in computation may be expressed as

$$T_c = FnNM$$



(cf. Chapter 1) where  $n$  is the number of multiplications per word stored in the internal memory,  $N$  is the number of words stored in the memory,  $M$  is the multiplication time and  $F$  a weighting factor which for a sequential machine is usually between one and ten and for the computer under consideration would probably be between one and five.

If the input-output equipment is such that a single word is read or written in  $W$  seconds then the time spent in reading and writing  $N$  words into the internal memory is

$$T_W = 2NW.$$

The adequacy of the input-output equipment as auxiliary storage would be measured by the ratio  $T_W/T_c$  which should be less than and at the most of the order of one. If this upper limit is used we obtain for a balanced computer (and problem)

$$\frac{T_W}{T_c} = \frac{2WN}{nFMN} = \frac{2W/M}{nF} = 1.$$

Thus for a "scientific" calculation a quantity which plays a role in estimating machine balance is  $W/M$ , the ratio of the read-write time per word to the multiplication time.

For problems dominated by memory access-time it is expected that  $T_c$  will be proportional to the product of the number of words in the internal storage and memory access-time. Thus the corresponding ratio is  $W/a_0$  where  $a_0$  is the memory access-time or  $(W/M)/a_0/M \approx 2W/M$ .

The reciprocals of these ratios, that is, the quantities  $M/W$  and  $a_0/W \approx \frac{1}{2} M/W$  are treated in the subsequent discussion and called input-output factors. We define

$$F_s = \frac{M}{W} \times 10^3$$

and

$$F_c = \frac{a_0}{W} \times 10^3.$$

The condition given above may be written as

$$F_s = \frac{2000}{nF}$$

It is extremely difficult to predict the manner in which the quantity  $nF$  varies over a range of problems. For many scientific problems whose data requirements exceed the storage capacity of the main memory and drum, such as problems in linear algebra, the value of  $nF$  may lie between 2 and 20. For other problems it may range as high as 100. However, it is frequently possible to rearrange the order of calculation in problems with small values of  $nF$  in such a way that  $nF$  becomes 50 or 100 corresponding to values of  $F_s$  of 20 or 40. The higher values of  $F_s$  demand faster input-output equipment. If  $F_s$  is 20 or 40 and  $nF$  is 100 or 50 respectively, then the time spent doing arithmetic is about the same as the time spent in using magnetic tape as an auxiliary memory. Therefore, although it would be very desirable to have magnetic tape units of the sort presently under development for which  $F_s$  is expected to be 200 or more, a very large class of problems may be put into a form suitable for tape units presently available for which  $F_s$  is about 20. The need for faster equipment would be lessened if internal memory capacity and the control were such that the input-output equipment could be used as auxiliary storage at the same time that the internal memory and arithmetic units were engaged in computation.

## 7.2 Data Rate Characteristics of Scientific and Data Processing Computers

The data rates for punched card, magnetic tape, and printing equipment are tabulated for a number of computers in Table 7.1. Data rates are reduced to units of binary digits per microsecond, under the assumptions that:

1. Binary punching is used for cards,
2. One alphanumeric character is equivalent to six binary digits; one decimal digit is equivalent to four binary digits.
3. Data rates are for one unit for any particular computer, even if several units of the same type can be operated in parallel.

SINGLE UNIT DATA RATES BITS/MS.

INPUT & OUTPUT FACTORS x 10<sup>3</sup>

"SCIENTIFIC" CALCULATION

COMPUTER	STOR- AGE	MULTI- PLY	PUNCHED CARDS		MAG. TAPE IN OR OUT  1 Unit	PRINT	PUNCHED CARDS		MAG. TAPE	PRINT
			In	Out			In	Out		
DATAMATIC 1000	4	0.048	0.0144	0.0016	0.24	0.0108	300	33.3	5000	225
IBM 709	3	0.23	0.0036	0.0014	0.09	0.006	15.7	6.2	391	26
UNIVAC II	2.4	0.028	0.00384	0.00192	0.084	0.0078	137	68.7	3000	279
IBM 704	3	0.15	0.0036	0.0014	0.09	0.0011	24.0	9.3	600	7.3
JOHNNIAC	2.67	0.104	0.00384	0.0016	--	0.0163	36.9	15.4	--	157
LARC	12	4.8	0.0096	0.0096	0.12	0.0156	2.0	2.0	25	3.25
UNIVAC SCIENTIFIC	4.5	0.151	0.00192	0.00192	0.0767	0.0078	12.4	12.4	507	51.6
ILLIAC	2.22	0.057	(PUNCHED TAPE)							
			0.0015	0.0003	--	0.000025	26.3	5.3	--	0.44
PROPOSED COMPUTER	33	13								

Table 7.1  
Data Rates for Input-Output Devices

574  
184

Data rates for storage and multiplication are found by dividing the number of binary digits per word by the access and multiplication times respectively.

Also shown in the table are Illiac rates with punched paper tape rather than punched cards. Experience with Illiac indicates that it is suitable for "scientific" calculation, useable for intermediate problems such as statistical work, and unsatisfactory for data processing applications. Illiac output is normally on paper tape with printing done on an off-line basis, although on-line operation of a printer is possible.

We shall employ the data of Table 7.1 in the following way:

1. For an indication of characteristics of card, magnetic tape, and printing devices currently available,
2. As a means of extrapolating input-output requirements for the computer whose design is discussed in this report,
3. As a means of determining internal storage access interruptions for input-output or auxiliary storage purposes.

### 7.3 Magnetic Tape Characteristics

The discussion in the introduction to this chapter indicated that an input-output factor  $F_s$  between 20 and 200 would enable the input-output equipment to be used as auxiliary storage for a large and useful class of problems. It is the purpose of this section to show that the requirements implied cannot be met by devices slower than magnetic tapes and to indicate to what extent the requirements can be met by magnetic tapes.

For the computer proposed, the data rate corresponding to a factor  $F_s = 20$  is  $20 \times 13 \times 10^{-3} = 0.26$  bits/ $\mu$ s; for  $F_s = 200$ , the rate is 2.6 bits/ $\mu$ s. The lower rate is approximately that of the magnetic tape unit with 31 parallel channels, 80 bits/inch, and 100 inches/second, which is used with the Datamatic 1000. The rate of 2.6 bits/ $\mu$ s corresponding to a factor  $F_s = 200$  cannot now be met by commercially available conventional tape units. However,

it seems quite likely that within the next few years commercial development will lead to magnetic tape units sufficiently fast to satisfy the requirements corresponding to factors  $F_s$  of 200 or more. In order to maintain a continuous data rate of 0.26 bits/ $\mu$ s it is necessary to use more than one tape unit so that rewinding can proceed in parallel with reading or writing. The simultaneous reading or writing on a number of tape units leads to a proportionate increase in the factor  $F_s$  and the data rate at a cost in complexity of control.

Inspection of Table 7.1 indicates that it would be impractical to provide punched card or punched paper tape equipment capable of a data rate of 0.26 bits/ $\mu$ s, since simultaneous operation of 18 of the fastest card readers, 100 paper tape readers, or 200 card punches would be required. The control problems and programming difficulties inherent in simultaneous operation, as well as the prohibitive equipment costs, preclude the use of paper tape or cards in this manner.

#### 7.4 Magnetic Drum Storage

It was noted in Chapter 1 that problems requiring 50,000 to 100,000 words of data are to be expected. Further discussion in Chapter 2 indicates that, with a relatively small increase in time, the bulk of the storage need not be supplied in the random-access core storage unit, but can be supplied in the form of a lower-speed non-random-access storage device. The characteristics of auxiliary storage devices which fulfill the requirements outlined in Chapter 2 are described here.

A magnetic storage drum consists of a rotating cylinder coated with magnetic material. Other design features being equal, the storage capacity is a function of the surface area of the cylinder, and is increased in direct proportion to either the length or diameter of the cylinder. The speed of rotation is limited by the mechanical difficulties in maintaining close tolerances between fixed reading and writing heads and the surface of the rotating cylinder. Roughly speaking, if the storage capacity is increased, the maximum allowable speed of rotation must be decreased.

A drum similar to that used with Illiac would provide a storage capacity of approximately 10,000 words, with the time for one revolution ( $\alpha$  of Chapter 2) equal to 17 milliseconds. The packing density is such that 2500 binary digits are recorded on one track around the periphery by one recording head; the time associated with each digit is then  $\frac{17000}{2500} = 6.8\mu\text{s}$ . . By reading and recording on 50 tracks in parallel, the minimum access time per word ( $\beta$  of Chapter 2) is  $6.8\mu\text{s}$ . . The length of the cylinder is such that 4 sets of 50 tracks can be used, so that the total storage capacity is  $4 \times 2500 = 10,000$  words of 50 bits each.

A drum storage capacity of 20,000 or 30,000 words can be achieved by use of 2 or 3 drum units of the type described. The cost is essentially the same for each unit, but this alternative has obvious advantages if one of the mechanical units should fail. The following characteristics are those of a commercially available unit with greater storage capacity:

6000 Bits per track

450 Tracks

$$\alpha = 51 \text{ ms}$$

$$\beta = 8.5 \mu\text{s}$$

Storage capacity = 54000 words.

### 7.5 Input-Output Requirements for Data Generated or Consumed by Human Users

The typing and reading rates of human users can be expressed in the units of bits/ $\mu\text{s}$ : employed for Table 7.1. Assuming each typist or reader is occupied 8 hours per day, the rates for 100 persons are: For typing at the rate of 60 words per minute

$$\frac{60 \times 5 \times 5 \times 100}{2 \times 60 \times 10^6} = 0.00125 \text{ bits}/\mu\text{s},$$

and for reading at the rate of 300 words per minute

$$\frac{300 \times 5 \times 5 \times 100}{2 \times 60 \times 10^6} = 0.00625 \text{ bits}/\mu\text{s}.$$

For both calculations, words of 5 letters are assumed, with 5 bits per letter and 50% redundancy assumed for English text.

Thus there are commercially-available punched card readers and fast printers which, if used 24 hours per day, would require on the order of 1000 typists and 250 human readers respectively. The point to be stressed here is that presently available equipment is more than adequate for handling data humanly generated or consumed.

#### 7.6 Other Modes of Data Generation or Consumption

It is to be expected that a wide variety of problems will be presented to the proposed computer, even though a relatively small variety of problems will occupy a majority of computing time. Special input-output facilities will be required for real time problems, and for handling data generated automatically, either in digital or analog form. For some problems digital punching or printing requirements would be reduced by a cathode ray tube output for analog or alphanumeric presentation of data. It should also be recognized that many calculations will be performed on data stored on punched cards or paper tapes.

In view of the requirements for magnetic tape units, it appears that with the exception of a direct data link for real time applications, the additional requirements discussed in the previous paragraph can be met by off-line magnetic tape conversion equipment.

#### 7.7 Other Aspects of Machine Balance

The characteristics of the drum storage unit impose requirements on the capacity of the core memory. For transfer of data to or from the drum, the time for transfer of a block of  $N$  words is, on the average,

$$\frac{\alpha}{2} + N$$

where  $\alpha$  and  $\beta$  are as defined in sections 2.1 and 7.4. If the average initial access-time  $\alpha/2$  is to be not more than half of the total block transfer time, then  $N$  must satisfy

$$N \geq \frac{\Delta}{2\beta} .$$

Thus the block size for  $\beta = 6.8 \mu\text{s}$  and  $\alpha = 17 \text{ ms}$  is at least 1250 words and for  $\beta = 8.5 \mu\text{s}$  and  $\alpha = 51 \text{ ms}$ , the block size is at least 3000 words. The capacity of the core memory must be sufficient for block transfers of the magnitudes indicated by drum characteristics as well as for storage of data and instructions otherwise required. The balance between core memory capacity and the characteristics of the drum unit represents an important consideration in the choice of the drum unit.

The data of Table 7.1 can be used to determine the relative number of storage accesses required for input, output and auxiliary storage purposes. The proposed computer is to have a storage rate of 33 bits/ $\mu\text{s}$ , and simultaneous use of a card reader, card punch, magnetic tape unit, and a fast printer would result in a combined data rate of 0.28 bits/ $\mu\text{s}$ . The ratio  $33/0.28 = 108$  indicates that less than 1% of the accesses to the core memory are required for data transfers to or from magnetic tape and input-output units.

### 7.8 Summary and Conclusions

The purpose of this chapter was to investigate the feasibility of meeting the input-output and auxiliary storage requirements with commercially available equipment. The bulk of the storage requirements can be met by one or more commercially available magnetic drums; additional auxiliary storage requirements can barely be met by the fastest of magnetic tape units now available.

For the purposes for which the proposed computer is intended, the input-output devices serve as a means of communication with human users and, less frequently, with other automatic devices. Presently available devices are more than adequate when human limitations are taken into account, and for the less frequent input-output uses, special equipment can be obtained.



CHAPTER 8  
ARITHMETIC UNIT

8.1 Introduction

The purpose of this chapter is to describe a design for a binary parallel arithmetic unit. The majority of the arithmetic units in binary digital computers now in existence are composed of an accumulator register A, a number register M, a quotient register Q, an adder, and one or more complementing circuits. For asynchronous operation, temporary accumulator and quotient registers ( $\bar{A}$  and  $\bar{Q}$ ) are provided. Figure 8.1 illustrates the interconnections and gates (G) for an arithmetic unit employing a complementary representation of negative numbers.

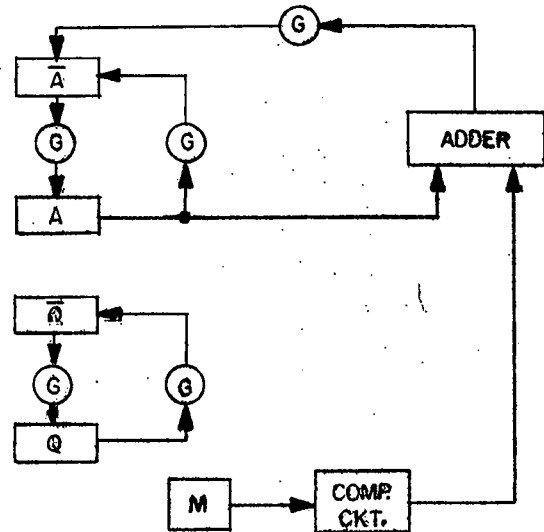


Figure 8.1  
Block Diagram of a  
Conventional Arithmetic Unit

Independent of the structural details, addition is a basic operation of most of the units now in existence, with multiplication performed as a sequence of conditional additions and shifts. The time allotted to addition is sufficiently long to allow for the worst case of a carry propagating from the least significant digit to the most significant digit of the adder. Multiplication requires, for a multiplier with  $n$  non-sign digits,  $n$  shifts and, on the average,  $n/2$  additions.

Methods have been proposed for increasing the speed and efficiency of the arithmetic unit. These include:

1. Use of circuitry to generate a "carry-completion" signal,<sup>(1)</sup> i.e., to indicate the completion of the (longest) carry sequence in each addition. The average longest carry sequence is on the order of

---

1. B. Gilchrist, J. H. Pomerene, and S. Y. Wong: "Fast Carry Logic for Digital Computers", IRE Trans. on Electronic Computers, vol. EC-4, no. 4, pp. 133-136, December 1955.

$\log_2 n$  digits for random addends, each with  $n$  non-sign digits, in contrast with the worst case of a carry sequence over  $n$  digital positions now provided for.

2. Use of separate carry storage during multiplication with assimilation of carries at the end of the multiplication operation. <sup>(2)(3)(4)</sup> Aside from the final assimilation, the time devoted to carry propagation is equivalent to a total of  $2n$  digits during the  $n$ -step process.
3. Recoding of the multiplier into digits which are  $-1$  as well as  $0$  and  $1$  in such a way that the number of non-zero digits, and consequently the number of uses of the adder, is reduced. <sup>(5)</sup> A reduction from on the average  $n/2$  to approximately  $n/3$  additions or subtractions, can be achieved. Furthermore, it is possible to shift two digital positions at each step, thereby halving the number of gating operations for shifts, without increasing the hardware requirements except for a conditional doubling circuit.

It is the goal in the design presented here to exploit the proposals listed above as efficiently as possible. In particular, the use of separate carry storage is extended to include a sequence of arithmetic operations; numerical results are represented with carries unassimilated whenever possible. Carry completion circuitry is used in two ways:

1. For the complete assimilation of carries when the conventional representation is required.
2. For a partial assimilation to the sign digit, when the sign of the unassimilated number must be known.

2. G. Estrin, B. Gilchrist, and J. H. Pomerene: "A Note on High-Speed Digital Multiplication", IRE Trans. on Electronic Computers, vol. EC-5, no. 3, p. 140, September 1956.
3. J. E. Robertson: "Preliminary Design of an Arithmetic Unit for Use with a Self-Checking Binary Parallel Digital Computer", Digital Computer Laboratory Report No. 19, June 1950.
4. Project Whirlwind: "Whirlwind I Computer Block Diagrams", Report R-127-1, vol. 1, p. 23, M.I.T., September 1947.
5. The method of multiplier recoding was described to the author by David J. Wheeler in 1951. It has recently been reinvestigated at the University of London, the National Bureau of Standards, and Aberdeen Proving Ground, Maryland. The method is similar to that described in Booth and Booth, Automatic Digital Calculators, Academic Press, Inc., New York; 1953, pp. 44-47 although the latter do not fully exploit the advantages of the method.

Hardware requirements for the proposed design are such that the accumulator (and temporary accumulator) are augmented by a carry register (and temporary carry register). The adder is replaced by a quasi-adder of equal complexity. A circuit for assimilation of stored carries in the carry register with the contents of the accumulator is also required for conversion to the conventional representation; it is proposed that circuitry also be provided for generating completion signals for carries propagating during the assimilation.

The ramifications of the requirement that numerical results be held in unassimilated form whenever possible are extensive. They include:

1. The best choice for representation of negative binary fractions is the two's complement representation.
2. An entirely new analysis of overflow is necessary.

These two statements can be partially justified by noting some features of a number represented in unassimilated form. First, the sign of the number in some cases is not known unless carries are assimilated, and second, with two registers used for representation of unassimilated numbers, the ranges of numbers represented in the two registers can be extended during some sequences of calculations. In contrast to conventional representations, there can be uncertainty either as to the sign or as to the range, but not both. As is discussed in detail in later sections, the sign uncertainty dictates that arithmetic methods be independent of sign insofar as possible. Thus, one of the complementary representations of negative numbers is preferable to the signed absolute value for addition and subtraction, since the latter representation requires an inspection of the sign of the difference of two absolute values. A detailed analysis of shifting between A and Q during multiplication indicates that inspection of signs is at times required for the one's complement representation, but not for the two's complement representation.

The range uncertainty for unassimilated numbers poses problems in overflow detection. In brief, three situations must be recognized.

1. It is apparent from the unassimilated digits that overflow has occurred.

2. Overflow cannot be detected from the unassimilated digits, but would be detected if carries were assimilated.
3. No overflow has occurred.

The two overflow cases are called unassimilated overflow (case 1) and assimilated overflow (case 2). An overflow analysis is a necessity for an arithmetic unit design, and is applicable to the following:

1. Detection of overflow in fixed point operations,
2. Automatic scaling for floating point operations,
3. Determining correct procedures following overflow occurring temporarily during one step of a multiplication or division.

It will be shown that unassimilated overflow detection is sufficient for the three requirements, and that assimilated overflow detection is necessary only when assimilation is required for some reason other than overflow detection. This approach is consistent with the goal that results should be left in unassimilated form whenever possible.

The multiplication method proposed incorporates the following features:

1. The multiplier is recoded to reduce the number of additions or subtractions. The rules for recoding the multiplier are such that no special attention need be given to the sign of the multiplier. A final step corresponding to the sensing of the sign digit of the multiplier is required, regardless of the multiplier sign.
2. No special attention need be given to the sign of the multiplicand if the accumulator can correctly accumulate either positive or negative partial products. Explicitly, successive partial products  $p_k$  (held in the accumulator) are related by the equation

$$p_{k+1} = 1/2 (p_k + y_{n-k} x)$$

where the recoded multiplier digit  $y_{n-k}$  is -1, 0, or +1. The sum  $(p_k + y_{n-k} x)$  may exceed range, but as a result of the shift

(i.e., multiplication by  $1/2$ ), the new partial product  $p_{k+1}$  is again within range. The problem is one of determining the correct sign digit to be inserted into the accumulator during the right shift, and is closely related to the problem of detection of overflow in addition and subtraction.

3. The use of a separate carry storage register in association with the accumulator poses a problem during the right shift. The least significant digit of the accumulator must be assimilated and transferred into the quotient register. Further difficulties would arise in shifting one digit from A to Q, particularly for the multiplier recoding proposed, if negative numbers are represented in any form other than two's complement.

The non-restoring division process seems most suitable for the arithmetic unit proposed. For a divisor  $y$  in M (Figure 8.1) and partial remainders  $r_k$  ( $k = 0, 1, \dots, n$ ) in  $\bar{A}$  and  $\bar{C}$ , the process is described by the recursion relationship

$$r_{k+1} = 2r_k \pm y$$

where the sign is chosen in such a way that the equation  $|r_k| \leq |y|$  is satisfied for each  $k$ , provided only that the dividend  $r_0$  satisfies  $|r_0| \leq |y|$ .

In order to choose the proper sign for the recursion relationship, the sign of  $r_k$  must be known. In the proposed design,  $r_k$  would be transferred with a left shift from  $\bar{A}$  and  $\bar{C}$  to form  $2r_k$  in A and C, in parallel with the partial assimilation of  $r_k$  to determine its sign. The fact that the partial assimilation and the shift can be paralleled for non-restoring division, in contrast to restoring division, dictates the choice of the division process. For a restoring division, the divisor is subtracted from (or added to) the partial remainder to form a tentative partial remainder. The sign of the tentative partial remainder is then sensed (by a carry assimilation to the sign digit) to determine whether

the tentative partial remainder is gated from the adder to  $\bar{A}$  and  $\bar{C}$  or the old partial remainder is gated from A and C to  $\bar{A}$  and  $\bar{C}$ . To summarize, the steps, in order of their occurrence, are:

- | Restoring division   | Non-restoring division   |
|--|--|
| 1. Subtract (or add) divisor in M from partial remainder in A, C.    | 1. Subtract or add divisor in M from A and C.  |
| 2. Assimilate to sign from adder.                                    | 2. Transfer to $\bar{A}$ and $\bar{C}$ .   |
| 3. Transfer to $\bar{A}$ and $\bar{C}$ .                             | 3. Transfer partial remainder from $\bar{A}$ and $\bar{C}$ to A and C with left shift. |
| 4. Transfer from $\bar{A}$ and $\bar{C}$ to A and C with left shift. | 4. Assimilate to sign digit.   |

None of the steps for restoring division can be paralleled; steps 3 and 4 of the non-restoring division can be paralleled. In restoring division, step 3 is conditional on the sign determined in step 2; in non-restoring division, step 1 is conditional on step 4 and can proceed after both steps 3 and 4 are complete. The non-restoring division method requires that the assimilator be connected to  $\bar{A}$  and  $\bar{C}$ , restoring division requires a connection to the adder; for circuit reasons the former seems preferable.

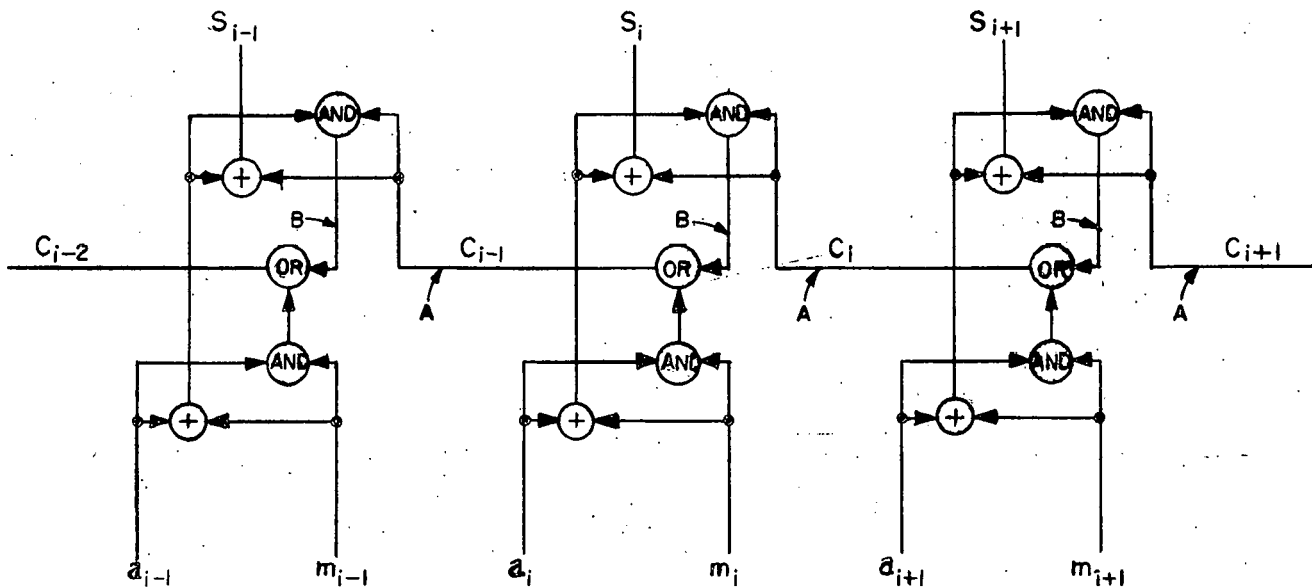
A reduction in division time can be achieved if the divisor  $y$  is standardized to lie in the range  $1/2 \leq |y| \leq 1$ , as would be the case for conventional floating point operation. In some instances, the partial remainders can be shifted left until the quantity in A and C is standardized; i.e., for non-restoring division, until  $1/2 \leq |2r_k| \leq 1$ . The number of uses of the adder and assimilator is thereby reduced, and if shifts of more than one digital position are available, the number of gating operations can be reduced, with a consequent reduction in shifting time.

The method is applicable to both restoring and non-restoring division, although special treatment of the quotient digits is required for the latter. In a conventional non-restoring division, the divisor is either subtracted or added, with +1's and -1's, respectively, inserted as quotient digits. A

relatively trivial conversion to the two's complement representation is then made. In the proposed method, the shift would require the inserting of 0 as a quotient digit. It is possible to devise a serial method for conversion of quotient digits, which are +1, 0, or -1, to conventional binary form. The method requires that one reversed ternary digit be held, and the conversion to binary is made one step later on the basis of the sign of the new partial remainder.

### 8.2 Separate Carry Storage in a Binary Arithmetic Unit

The structure of a conventional binary adder is indicated in block diagram form in Figure 8.2, in which the symbols represent circuits whose opera-



$$s_i = a_i \oplus m_i \oplus c_i$$

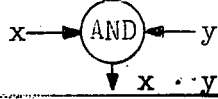
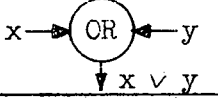
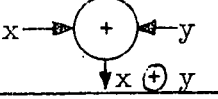
$$c_{i-1} = (a_i \oplus m_i) c_i \vee a_i m_i$$

$i = 0, 1, \dots, n$ , where  $n$  denotes the least significant digit

Figure 8.2. Conventional Adder

tions are summarized in Table 8.1.

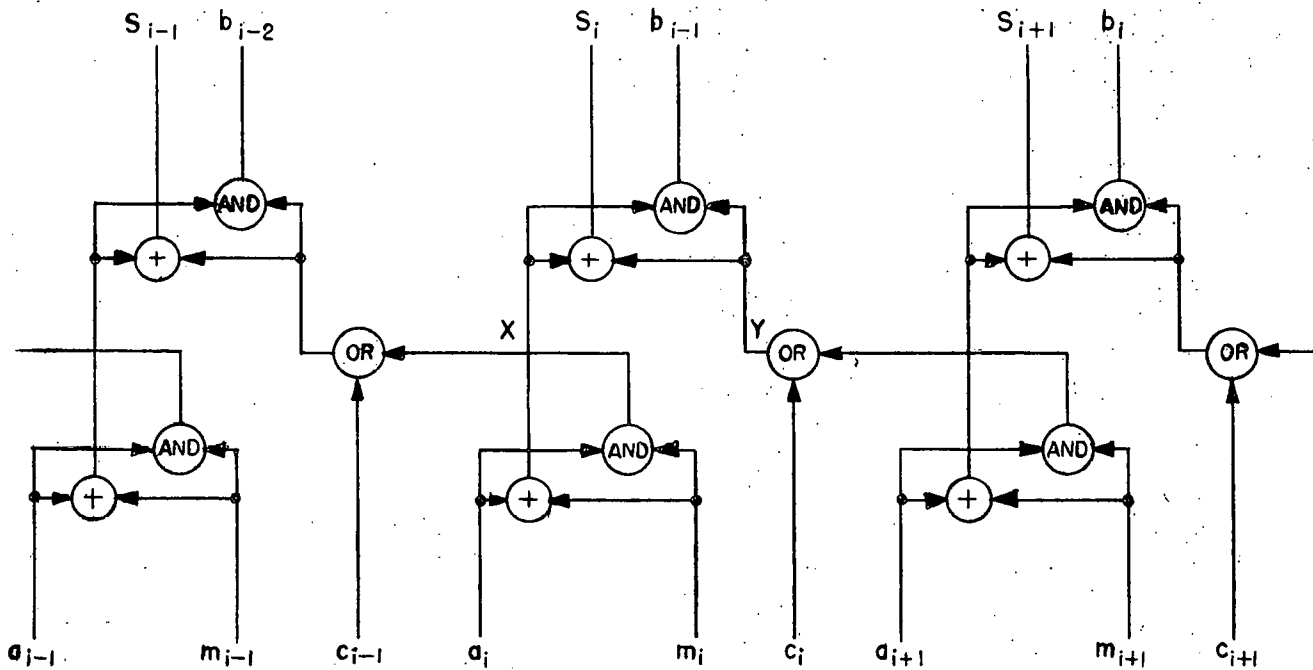
Table 8.1

x	y			
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

In a conventional parallel asynchronous arithmetic unit, the digits  $a_i$  ( $i = 0, \dots, n$ ) represent binary digits of an accumulator register A, the  $m_i$  are digits of a number register M, the  $s_i$  are sum digits to be gated to a temporary accumulator  $\bar{A}$ , and the  $c_i$  are carry signals internal to the adder. The index  $i$ , for each register, has the values  $0, 1, \dots, n$ , where  $n$  denotes the least significant digit. The ultimate speed of the conventional adder is limited by the fact that sufficient time must be allotted for the carry to propagate from the least significant to the most significant digital position.

If register storage for carries is provided, the carry chain can be broken either at points A or points B of Figure 8.2. For reasons given in following sections, breaks at point B are preferable, and lead to the logical structure of Figure 8.3. In Figure 8.3, the digits  $b_i$  would be gated to a temporary carry register  $\bar{C}$  and digits  $c_i$  are held in the carry register C. The logical structure of Figure 8.3 together with the registers A,  $\bar{A}$ , C,  $\bar{C}$ , and M and suitable interconnecting gates, is sufficient for sequences of additions and subtractions, including sequences required for multiplication. The true sum at any given instant of time would be found by assimilating the contents of A and C, or alternatively, of  $\bar{A}$  and  $\bar{C}$ .





$$s_i = a_i \oplus m_i \oplus [a_{i+1} \cdot m_{i+1} \vee c_i] \quad b_{i-1} = [a_i \oplus m_i] \cdot [a_{i+1} \cdot m_{i+1} \vee c_i]$$

Figure 8.3. Proposed "Adder" Structure

### 8.3 Separate Carry Storage for an Arbitrary Radix

The process of addition of an augend in unassimilated form and an addend in assimilated form for an arbitrary integral radix  $r$  can be viewed as consisting of two steps, as illustrated for the decimal system in Example 8.1. Each step involves, for each digital position, the formation of a digitwise sum, with the carry suitably displaced and stored separately from the sum (modulo  $r$ ) that may arise. During the first step, digitwise sums and carries are generated from corresponding addend and augend digits; during the second step, sums and carries are formed from the results of the first step. The final sum is repre-

sented as a set of digits modulo  $r$  with an associated set of binary carry digits. The formation of the conventional representation of the sum from the two sets of digits (carry assimilation) is equivalent to an addition of the digitwise sums and carries. The second step is required in order that the property, "if  $c_{i-1} = 1$  then  $a_i = 0$ " be preserved under addition.

$$\begin{array}{r}
 x = .157 \\
 y = \underline{.548} \\
 \phantom{y} = .695 \\
 \phantom{y} = \underline{.010} \\
 x + y \left\{ \begin{array}{l} .605 \quad a_i \\ .100 \quad c_i \end{array} \right. \\
 z = \underline{.199} \quad m_i \\
 \phantom{z} = .794 \quad u_i \\
 \phantom{z} = \underline{.110} \quad k_i \\
 x + y + z \left\{ \begin{array}{l} .804 \quad s_i \\ .100 \quad b_i \end{array} \right.
 \end{array}$$

Example 8.1

One feature of the addition with separate carry storage is that assimilation is not required during a sequence of additions. The unassimilated augend is represented by digits  $a_i$  ( $a_i = 0, 1, \dots, r-1$ ) and carries  $c_i$  ( $c_i = 0, 1$ ), the addend is represented by digits  $m_i$  ( $m_i = 0, 1, \dots, r-1$ ), where the index  $i$  increases with decreasing significance of the digital position. The  $a_i$  and  $c_i$  are restricted to values such that if  $c_{i-1} = 1$ , then  $a_i = 0$ . The first step in forming the sum is

$$u_i \equiv m_i + a_i \text{ modulo } r.$$

$$\text{If } c_{i-1} = 1 \quad k_{i-1} = c_{i-1} = 1.$$

$$\text{If } c_{i-1} = 0 \text{ and } m_i + a_i \geq r, \text{ then } k_{i-1} = 1.$$

$$\text{If } c_{i-1} = 0 \text{ and } m_i + a_i < r, \text{ then } k_{i-1} = 0.$$

It should be noted that the restriction  $c_{i-1} = 1 \implies a_i = 0$  insures that no carry can arise from the sum  $m_i + a_i$  to interfere with the unassimilated carry  $c_{i-1}$ . The second step is then:

$$s_i \equiv u_i + k_i \text{ modulo } r,$$

$$b_{i-1} = 1 \text{ if } u_i + k_i \geq r,$$

$$b_{i-1} = 0 \text{ if } u_i + k_i < r.$$

Since  $u_i = 0, 1, \dots, r-1$ , and  $k_i = 0$  or  $1$ , it follows that  $b_{i-1} = 1$  only if  $u_i = r-1, k_i = 1$ , and therefore  $b_{i-1} = 1 \implies s_i = 0$ . It is thus possible to use the  $b_i$  and  $s_i$  as augend (cf.  $c_i, a_i$ ) in a subsequent addition. For the binary system ( $r = 2$ ), the equations are:

$$u_i = a_i \oplus m_i$$

$$k_{i-1} = a_i m_i \vee c_{i-1}$$

$$s_i = u_i \oplus k_i = a_i \oplus m_i \oplus (a_{i+1} m_{i+1} \vee c_i)$$

$$b_{i-1} = u_i k_i = (a_i \oplus m_i) (a_{i+1} m_{i+1} \vee c_i)$$

where the binary digits are related by Boolean operations defined in Table 8.1.

It should be noted that the above equations agree with those found by the alternative method of considering modifications of a conventional binary adder.

#### 8.4 Binary Subtraction

In an arithmetic unit with negative numbers represented in two's complement form, subtraction is performed as an addition of the digitwise complement of the subtrahend (in M) to the minuend (in A and C) with a carry inserted into the least significant digit of the adder. For the quasi-adder with separate carry storage, the least significant digits of a sum are:

$$b_{n-2} = (a_{n-1} \oplus m_{n-1}) (a_n m_n \vee c_{n-1})$$

$$b_{n-1} = (a_n \oplus m_n) c_n$$

$$b_n = 0$$

$$s_{n-1} = a_{n-1} \oplus m_{n-1} \oplus (a_n m_n \vee c_{n-1})$$

$$s_n = a_n \oplus m_n \oplus c_n$$

By analogy with a conventional addition, the digit  $c_n$  is zero for addition and is one for subtraction. In either case,  $b_n$  is always zero. If  $c_n$  were to be interpreted as an unassimilated carry digit resulting from an addition or subtraction, it would follow that  $c_n = b_n = 0$ ; it is therefore possible to reinterpret  $c_n$  as a carry insertion signal generated by the control circuits.

### 8.5 Carry Assimilation

An important consequence of selecting points B of Figure 8.2 as the break points in the carry chain is that the digits  $s_i$  and  $b_{i-1}$  of the  $i^{\text{th}}$  digital position cannot simultaneously be ones. This readily follows if we take  $u_i$  and  $k_i$  as defined in section 8.3 for radix 2

and note that

$$s_i = u_i \oplus k_i, \quad b_{i-1} = u_i \cdot k_i,$$

and note further that

$$s_i \cdot b_{i-1} = (u_i \oplus k_i) \cdot (u_i \cdot k_i) = 0.$$

For carry assimilation, we first assume that a full adder of the structure of Figure 8.2 is required, with inputs  $s_i$  and  $b_i$ , with internal carry  $d_i$ , and with sum digits  $a_i$ . The equations for carry assimilation are then

$$a_i = s_i \oplus b_i \oplus d_i$$

$$d_{i-1} = (b_i \oplus d_i) s_i \vee b_i \cdot d_i.$$

We now show that  $b_i \cdot d_i = 0$  follows from  $s_i \cdot b_{i-1} = 0$ , by induction on  $i$ . For  $i = n$ ,  $d_n = 0$ , therefore  $b_n d_n = 0$ . The induction hypothesis is  $b_i \cdot d_i = 0$ . If  $d_{i-1} = 1$ , then  $s_i$  must equal 1, implying that  $b_{i-1} = 0$  and  $b_{i-1} d_{i-1} = 0$ . If  $b_{i-1} = 1$ , then  $s_i = 0$  and  $d_{i-1} = 0$ . The carry assimilation equations can therefore be simplified to

$$a_i = s_i \oplus (b_i \vee d_i)$$

$$d_{i-1} = s_i \cdot (b_i \vee d_i).$$

Thus the circuitry for carry assimilation is approximately half as complicated as the circuitry for a conventional adder.

The remarks of the preceding paragraph are based upon use of a conventional carry chain in the carry assimilation circuitry, rather than carry completion circuitry.

The completion of the longest carry sequence during an assimilation can be sensed by use of a zero's carry signal  $d_i^0$  as well as the usual one's carry signal  $d_i^1$ . The Boolean equations for assimilation with a carry completion signal are, for the  $i^{\text{th}}$  digital position:

$$a_i = s_i \oplus (b_i \vee d_i^1)$$

$$d_{i-1}^1 = g s_i (b_i \vee d_i^1)$$

$$d_{i-1}^0 = g (\bar{s}_i \vee \bar{b}_i d_i^0)$$

$$h = (d_0^0 \vee d_0^1) (d_1^0 \vee d_1^1) \dots (d_n^0 \vee d_n^1)$$

where  $g$  and  $h$  are signals initiating the assimilation and indicating the completion of carries, respectively.

In operation,  $g$  is initially 0, and  $d_i^0 = d_i^1 = 0$ , for each  $i$ . When assimilation is desired,  $g$  becomes 1, and at any one digital position, one of three operations will occur.

1. A zero's carry will arise if  $s_i = 0$ .
2. A one's carry will arise if  $s_i = b_i = 1$ .
3. If  $s_i = 1$  and  $b_i = 0$ , either a zero's carry  $d_{i-1}^0$  or a one's carry  $d_{i-1}^1$  will be propagated after the corresponding carry  $d_i^0$  or  $d_i^1$  is generated by the next digital position to the right.

When either a one's carry or zero's carry is generated at every digital position, completion of carries is signalled by  $h = 1$ .

Alternatively, carry assimilation could be performed by converting the structure of Figure 8.3 to that of Figure 8.2 by suitable switching circuits. It appears that switching circuits for such conversions are as complicated as those for separate carry assimilation described in the preceding paragraphs, and have the further disadvantage of increasing the number of circuits through which the carry must propagate.

For maximum speed of operation, carry assimilation should be performed only when absolutely necessary; namely, when the number in A is to be transferred elsewhere; or in parallel with some other operation; e.g., reading from core storage. For sign conditional operations, such as those which occur in division and sign conditional jumps, it is proposed that the sign digit carry completion signal be used to indicate that the sign has been correctly determined.

#### 8.6 Analysis of Overflow: Introduction

The study of the behavior of an accumulator with separate carry storage requires an explanation of the representation of numbers with carries separately stored. Overflow analysis is done largely by analogy with conventional overflow, but is complicated not only by the fact that the number representation is unfamiliar but also by the fact that carries propagate to the left and must be disposed of if the number representation is to be consistent.

Precisely what is meant by overflow must be redefined, since the fact that carries are unassimilated introduces uncertainty as to the range of numbers represented; the difficulties seem largely conceptual and require no exorbitant increase in equipment for mechanization.

An overflow analysis is applicable in two situations:

1. Detection of overflow during the fixed point operations of left shift and addition or subtraction and the equivalent problem of scaling for floating point.
2. Analysis of the repetitive steps performed during a multiplication or a division. Although overflow is temporarily permitted to occur, the product or quotient may be in range.

The latter situation is to be distinguished from the detection of overflow in a product or quotient; overflow detection in these cases is described, respectively, with the discussions of multiplication and division procedures.

### 8.7 Conventional Overflow Analysis

The analysis is restricted to binary fractions with negative numbers represented as complements with respect to two; i.e., a given number  $x$  lies in the range  $-1 \leq x < 1$  and is represented by binary digits  $x_0, x_1, \dots, x_n$  such that  $x = -x_0 + \sum_{i=1}^n 2^{-i} x_i$ . Overflow is said to occur if, as a result of some arithmetic operation, a result  $x$  lies outside the range  $-1 \leq x < 1$ .

Overflow can be detected if each result  $x$  which may exceed range is represented as a complement with respect to four; i.e., if the register holding the number  $x$  is extended one binary digit to the left. The two digits to the left of the binary point are designated as  $x_{-1}$  and  $x_0$ , and indicate the range of  $x$  as follows:

$x_{-1}$	$x_0$	range of $x$
0	0	$0 \leq x < 1$
0	1	$1 \leq x < 2$
1	0	$-2 \leq x < -1$
1	1	$-1 \leq x < 0$

For the table above to be correct, a result  $x$  which may exceed the range  $-1 \leq x < 1$  must nonetheless remain within the range  $-2 \leq x < 2$ . This condition is obviously satisfied by a sum or difference of two operands which are fractions, and by the result of the doubling of a fraction.

A somewhat more general viewpoint is useful for the discussions to follow. A number  $x$  can be represented as a complement with respect to  $2^m$  by extending the register in which  $x$  is held to the left. There are then  $m$  digits to the left of the binary point, designated as  $x_{-m+1}, x_{-m+2}, \dots, x_{-2}, x_{-1}, x_0$ .

Initially,  $m$  is chosen sufficiently large, and a particular value of an index  $k$  is found such that  $x_{-k}$  is a model of all digits to the left, i.e.,  $x_{-m+1} = x_{-m+2} = \dots = x_{-k-1} = x_{-k}$ . In this context  $x_0$  is a model of all digits to the left if the range  $-1 \leq x < 1$  is not exceeded, and  $x_{-1}$  is a model of all digits to the left if overflow occurs during one of the elementary operations of addition, subtraction, or left shift of one digital position.

### 8.8 Number Representation with Separate Carry Storage

In an arithmetic unit with separate carry storage, a number  $x$  is represented by two sets of binary digits  $a_{-1}, a_0, a_1, \dots, a_n$  and  $c_0, c_1, \dots, c_{n-1}$  with the  $a_i$  in the accumulator and the  $c_i$  in the associated carry register.  $a_{-1}$  and  $c_{-1} = 0$  are models of all digits to the left. Employing the relationship  $c_{i-1} a_i = 0$  for  $i = 1, \dots, n$  we can deduce from the digits  $a_{-1}, a_0, c_0$ , the information of Table 8.2 concerning the range of  $x$ .

Table 8.2

	$a_{-1}$	$a_0$	$c_0$	Range of $x$	
+	0	0	0	$0 \leq x < 1\frac{1}{2}$	} $x$ lies outside the range $-1 \leq x < 1$
+	0	0	1	$1 \leq x < 2$	
±	0	1	0	$1 \leq x < 2; -2 \leq x < -1\frac{1}{2}$	
-	0	1	1	$-2 \leq x < -1$	
±	1	0	0	$-2 \leq x < -\frac{1}{2}$	} $x$ is within the range $-1 \leq x < 1$
-	1	0	1	$-1 \leq x < 0$	
±	1	1	0	$-1 \leq x < \frac{1}{2}$	
+	1	1	1	$0 \leq x < 1$	



In essence, the effect of the carry assimilation is additive and can, if  $c_0 = 0$ , increase the sum  $2a_{-1} + (a_0 + c_0)$  by one unit. If  $c_0 = 1$ , the assimilation of carries will not affect the sum  $2a_{-1} + (a_0 + c_0)$  and the results of the conventional overflow analysis can be applied with  $2x_{-1} + x_0 = 2a_{-1} + (a_0 + c_0) \text{ [modulo 4]}$ .

From the table it is apparent that three cases arise:

- Class I 1. The unassimilated results indicate definitely that  $x$  lies outside the range  $-1 \leq x < 1$ . This unassimilated overflow occurs when  $a_{-1}$ ,  $a_0$ , and  $c_0$  have states 001, 010, or 011.
- Class II [ 2. The number  $x$  may or may not lie outside the range  $-1 \leq x < 1$  after assimilation. (States 000 and 100 of  $a_{-1}$ ,  $a_0$ , and  $c_0$ )
3. The number  $x$  is definitely in the range  $-1 \leq x < 1$ . (States 101, 110, 111 of  $a_{-1}$ ,  $a_0$ , and  $c_0$ ).

It is important to distinguish between unassimilated overflow (Case 1) and assimilated overflow which may occur in Case 2. For convenience in terminology, a number is said to be Class I if unassimilated overflow has occurred, and is said to be Class II otherwise.

It is necessary to consider two distinct representations of numbers; the unassimilated representation in the accumulator and carry registers and the conventional (assimilated) representation elsewhere. The function of the carry assimilator is to convert from the unassimilated representation to the conventional one; the conventional representation is a special case of the unassimilated representation, with the carry register zero.

### 8.9 Method of Analysis for Overflow

The representation of a number in the accumulator and carry registers comparable to the "in range" conventional representation is that of a Class II<sup>a</sup> number (no unassimilated overflow) with  $a_{-1}$  and  $c_{-1} = 0$  model digits for all

digits of greater significance. With a Class II number as an operand, the result of an operation will not, in general, be a Class II number, nor will the model digits of the result have the same significance as those of the operand. If the result is represented by digits  $s_i$  and  $b_i$ , then there is some value of  $k \geq 1$  such that  $s_{-k}$  and  $b_{-k} = 0$  are model digits for the result. A partial assimilation of carries to the left of  $s_0$  and  $b_0$  will ensure that  $b_{-1} = 0$  is a model digit for the carry register, however, in some special cases  $s_{-1}$  is not the model digit for the accumulator register. It can be shown that unassimilated overflow occurs if, after the partial assimilation, either

1.  $s_{-1}$  is not the model digit for the  $s_{-i}$  for  $i \geq 1$ , or
2.  $s_{-1}$  and  $b_{-1} = 0$  are model digits, and the result is a Class I number.

Thus, if the operands are initially Class II numbers, the result is also Class II unless unassimilated overflow has occurred.

For the analysis of multiplication, it is necessary to show that if a partial product  $p_k$  is a Class II number, then the next partial product  $p_{k+1}$  is also a Class II number, where  $p_{k+1}$  is formed by a right shift of the sum or difference  $p_k \pm y$ , where  $y$  is the assimilated multiplicand. For division a left shift of a partial remainder  $r_k$  is followed by the addition or subtraction of the divisor  $y$  in such a way that  $|r_k| \leq |y|$ , for every  $k$ . Thus, for the division analysis it is necessary to show that if  $r_k$  is Class II, then  $r_{k+1}$  is also a Class II number. Overflow may occur temporarily in either case.

The net result of the analysis is that it is possible to perform a sequence of arithmetic operations without carry assimilation. For overflow detection, it is sufficient to detect unassimilated overflow during the sequence of operations and to detect conventional overflow when the result of the sequence of operations is assimilated. Carry assimilation is not required for either overflow detection or for a multiplication step, but is required during some steps of a division in order that the proper choice of addition or subtraction of the divisor can be made.

### 8.10 Overflow-Detection: Left Shift of One Digital Position

For a left shift of one digital position we form the  $s_i$  and  $b_i$  to the left of the binary point ( $i \leq 0$ ) from the  $a_i$  and  $c_i$  as follows:

$$\begin{array}{ll}
 s_0 = a_1 & b_0 = c_1 \\
 s_{-1} = a_0 \oplus c_0 & b_{-1} = 0 \\
 s_{-2} = a_{-1} \oplus a_0 c_0 & b_{-2} = 0 \\
 s_{-3} = a_{-1} \oplus a_{-1} a_0 c_0 & \dots \\
 \dots &
 \end{array}$$

Note that a partial assimilation of carries to the left of  $b_0$  has been performed, yielding  $b_{-1} = 0$  and  $s_{-3}$  as model digits. If initially the operand represented by the  $a_i$  and  $c_i$  was Class II (no unassimilated overflow) then the resulting  $s_i$  and  $b_i$  of interest are (using  $c_0 a_1 = 0$ ):

$a_{-1}$	$a_0$	$c_0$		$s_{-3}$	$s_{-2}$	$s_{-1}$	$s_0$	$b_0$
0	0	0		0	0	0	$a_1$	$c_1$
1	0	0		1	1	0	$a_1$	$c_1$
1	0	1		1	1	1	0	$c_1$
1	1	0		1	1	1	$a_1$	$c_1$
1	1	1		0	0	0	0	$c_1$

The exceptional case for which  $s_{-1}$  is not the model digit for the  $s_{-i}$  for  $i \geq 1$  is that case in which  $a_{-1} = 1$ ,  $a_0 = 0$ , and  $c_0 = 0$ . For this case, inspection of Table 8.2 reveals that the range of the corresponding operand  $x$  before the shift is  $-2 \leq x < -1/2$ ; therefore the result  $y = 2x$  after the shift is in the range  $-4 \leq y < -1$ , and overflow occurs.

For the remaining cases,  $s_{-1}$  serves as the model digit for the  $s_{-i}$  ( $i \geq 1$ ). In these cases the digits  $s_{-1}$ ,  $s_0$ , and  $b_0$  can be inspected and the shifted result categorized as a Class I or Class II result in accordance with Table 8.2. It can easily be verified that, for each of the possible Class I results, the values of  $s_0 = a_1$  and  $b_0 = c_1$  are such that the corresponding range of the unshifted operand would indicate that overflow would occur during the left shift.

In short, overflow detection during left shift involves sensing the special case  $a_{-1} = 1$ ,  $a_0 = 0$ , and  $c_0 = 0$  before the shift, and detection of Class I numbers after the shift.

### 8.11 Overflow Detection: Addition or Subtraction

Since subtraction is executed as the addition of the complement of the subtrahend, it is sufficient without loss of generality to analyze the operation of addition only. Assuming that  $m_0$  is the model digit for the addend, and that  $c_{-1} = 0$  and  $a_{-1}$  are model digits for the augend, application of the "adder" equations (Figure 8.3):

$$s_i = a_i \oplus m_i + (a_{i+1} m_{i+1} \vee c_i)$$

$$b_{i-1} = (a_i \oplus m_i)(a_{i+1} m_{i+1} \vee c_i)$$

yields

$$s_0 = a_0 \oplus m_0 \oplus k$$

$$\text{where } k = a_1 m_1 \vee c_0$$

$$s_{-1} = a_{-1} \oplus \bar{a}_0 m_0$$

$$b_{-1} = (a_0 \oplus m_0)k$$

$$s_{-2} = a_{-1} \vee m_0$$

$$b_{-2} = \bar{a}_{-1} a_0 m_0$$

$$s_{-i} = s_{-2} \quad i \geq 2$$

$$b_{-i} = 0 \quad i \geq 3.$$

Assimilation to the left of  $b_0, s_0$  yields

$$s'_{-1} = a_{-1} \oplus (\bar{a}_0 m_0 \bar{k} \vee a_0 \bar{m}_0 k)$$

$$b'_{-1} = 0$$

$$s'_{-2} = a_{-1} (\bar{a}_0 \vee m_0 \vee \bar{k}) \vee \bar{a}_{-1} \bar{a}_0 m_0 \bar{k}$$

$$b'_{-i} = 0 \quad i \geq 1$$

$$s'_{-i} = s'_{-2} \quad i \geq 2$$

where

$$k = a_1 m_1 \vee c_0.$$

The values of  $s'_{-2}, s'_{-1}$  and  $s_0$  as functions of  $k, a_{-1}, a_0,$  and  $m_0$  are summarized in Table 8.3. In this table  $a_{-1} = 0, a_0 = 1$  indicate overflow in the initial augend, and need not be considered further. Of the remaining cases  $s'_{-1}$  serves as the model digit except for the one case for which  $k = a_0 = 0, a_{-1} = m_0 = 1$ . Since  $k = a_1 m_1 \vee c_0 = 0$ , then  $c_0 = 0$  and either  $m_1$  or  $a_1$  is 0. The two cases are:

1.  $a_1 = 0$  with  $a_{-1} = 1, a_0 = 0, c_0 = 0$  imply that  $x$  in  $A$  and  $C$  is in the range  $-2 \leq x < -1$  and  $m_0 = 1$  implies that  $m$  in  $M$  is in the range  $-1 \leq m < 0$ : therefore the sum  $s$  in  $\bar{A}$  and  $\bar{C}$  is in the range  $-3 \leq s < -1$ , and is outside the range  $-1 \leq s < 1$ .
2.  $m_1 = 0$  with  $m_0 = 1$  implies that  $m$  is in the range  $-1 \leq m < -1/2$  and  $a_{-1} = 1, a_0 = 0, c_0 = 0$  imply that  $x$  is in the range  $-2 \leq x < -1/2$ , therefore  $s$  is in the range  $-3 \leq s < -1$ , and is outside the range  $-1 \leq s < 1$ .

Thus, for addition, overflow is detected by the usual unassimilated overflow check on the sum (for Class I results) and by sensing the special case  $k = a_0 = 0, a_{-1} = m_0 = 1$ .

Table 8.3

k	a <sub>-1</sub>	a <sub>0</sub>	m <sub>0</sub>	s' <sub>-2</sub>	s' <sub>-1</sub>	s <sub>0</sub>	
0	0	0	0	0	0	0	
0	0	0	1	1	1	1	
0	0	1	0	0	0	1	] Overflow in A,C
0	0	1	1	0	0	0	
0	1	0	0	1	1	0	
0	1	0	1	1	0	1	See text
0	1	1	0	1	1	1	
0	1	1	1	1	1	0	
1	0	0	0	0	0	1	
1	0	0	1	0	0	0	
1	0	1	0	0	1	0	] Overflow in A,C
1	0	1	1	0	0	1	
1	1	0	0	1	1	1	
1	1	0	1	1	1	0	
1	1	1	0	0	0	0	
1	1	1	1	1	1	1	

8.12 Overflow Analysis: Addition or Subtraction Followed by a Right Shift

One step of a multiplication can be described by the formula

$$p_{k+1} = 1/2(p_k + y_{n-k}x)$$

where x is the multiplicand in M,  $y_{n-k}$  is a digit of the recoded multiplier having one of the values -1, 0, or 1, and  $p_k$  and  $p_{k+1}$  are successive partial products in unassimilated form in A and C. The quantity  $(p_k + y_{n-k}x)$  in  $\bar{A}$  and  $\bar{C}$  is permitted to exceed range, but  $p_{k+1}$  is correctly represented if the digit  $a'_{-1}$  inserted during the right shift is properly chosen. The operation will be correctly performed if, employing the notation of the previous section for the digits of  $\bar{A}$  and  $\bar{C}$  representing the sum,

$$\begin{array}{ll}
 a'_{-1} = s'_{-2} & c'_0 = 0 \\
 a'_0 = s'_{-1} & c'_1 = b_0 \\
 a'_1 = s_0 & \text{etc.,}
 \end{array}$$

where the  $a'_i$  and  $c'_i$  are digits of A and C representing  $p_{k+1}$ .

It should be noted that the digit  $a'_{-1}$  inserted during the right shift does not necessarily indicate the sign of  $p_{k+1}$ . However, if  $s'_{-2}$  would have been changed by assimilation of carries before the shift, then  $a'_{-1}$  would be similarly changed by assimilation after the shift.

The special case of  $k$ ,  $a_{-1}$ ,  $a_0$ , and  $m_0$  respectively 0, 1, 0, and 1 requires dissimilar treatment in detection of addition overflow from that required for a multiplication step. The sum digit  $s'_{-2}$  need not be generated for overflow detection, since  $s'_{-1} = 0$  and  $s_0 = 1$  by themselves indicate overflow. The digit  $s'_{-2}$  is in this exceptional case necessary for the right shift during multiplication for insertion as the digit  $a'_{-1}$ . Otherwise  $s'_{-2}$  need not be generated at all, since  $s'_{-1}$  is the model digit for  $\bar{A}$ , and  $a'_{-1} = a'_0 = s'_{-1}$  would be the correct digits in A for  $p_{k+1}$  after the right shift.

### 8.13 Overflow Analysis: Left Shift Followed by an Addition or Subtraction Such That the Result is in Range

Each step of a non-restoring division can be described by the formula

$$r_{k+1} = 2r_k \pm y$$

where  $y$  is the divisor in  $M$  with digits  $m_i$ , and  $r_k$  and  $r_{k+1}$  are successive partial remainders in  $\bar{A}$  and  $\bar{C}$  with digits  $s_i$  and  $b_i$  representing  $r_k$  and digits  $s'_i$  and  $b'_i$  representing  $r_{k+1}$ . The choice between addition and subtraction is made in such a way that each  $r_k$  satisfies  $|r_k| \leq |y| \leq 1$ , although  $2r_k$  may be outside the range  $-1 \leq 2r_k < 1$ . The sign of  $r_k$  is determined by a partial carry

assimilation to the sign digit, the divisor is subtracted from  $2r_k$  if signs of  $r_k$  and  $y$  agree and is added to  $2r_k$  if signs of  $r_k$  and  $y$  disagree. This arithmetic procedure guarantees that if  $|r_k| \leq |y|$ , then  $|r_{k+1}| \leq |y|$ . It also guarantees that  $r_{k+1}$  is within the range  $-1 \leq r_k < 1$ , since  $y$  is within this range and  $|r_k| \leq |y|$ . It remains to be shown that  $s'_{-1}$  and  $b'_{-1} = 0$  are correct model digits for  $r_{k+1}$ .

The analyses of previous sections for the left shift and addition and subtraction can be applied directly to show that  $s'_{-1}$  and  $b'_{-1} = 0$  are correct model digits for  $r_{k+1}$  if the digits  $m_i$  of the divisor  $y$  are chosen so that the relation  $|r_k| \leq |y| \leq 1$  is satisfied, except for the special case for which the digits  $s_{-1}$ ,  $s_0$ , and  $b_0$  of  $r_k$  are respectively 1, 0 and 0.

In the special case, the digits  $a_{-2}$  and  $a_{-1}$  of  $2r_k$  are unequal, so that the assumption that  $a_{-1}$  is the model digit for the register A used in the addition-subtraction analysis is violated. The relation  $|r_k| \leq |y| \leq 1$  imposes restrictions on the digits of  $r_k$  (and therefore on  $2r_k$ ) and  $y$  (or its complement) such that we need consider only the two cases:

$r_k$						$2r_k$					$\pm y$		
$s_{-1}$	$s_0$	$s_1$	$s_2$	$b_0$	$b_1$	$a_{-2}$	$a_{-1}$	$a_0$	$a_1$	$c_0$	$m_0$	$m_1$	$k = a_1 m_1 \vee c_0$
1	0	1	1	0	0	1	0	1	1	0	0	1	1
1	0	1	0	0	1	1	0	1	0	1	0	1	1

The values of  $s_1$ ,  $s_2$ , and  $b_1$  must be those shown if  $r_k \geq -1$ ; also, for the special case,  $r_k < -1/2$  and therefore  $m_0 = 0$ ,  $m_1 = 1$  if  $|y| \geq |r_k|$ . By an extension of the addition analysis, it is easily shown that, for  $r_{k+1}$ , the digits  $s'_{-2} = s'_{-1} = 1$ ,  $s'_0 = 0$ . Thus  $s'_{-1}$  is the correct model digit for  $\bar{A}$  in the special case.



8.14 Multiplication: Introduction

The multiplication procedure described here is the result of an investigation into methods of exploiting the relatively simple arithmetic unit structure of Figure 8.4, or some minor variant thereof. Comparison of Figure 8.4 with the Illiac arithmetic unit shown in Figure 8.1 indicates that only equipment required for separate carry storage has been added. Although some studies have been made of more complicated structures most easily described as binary versions of existing decimal arithmetic units, it is felt that the basic principles can most easily be presented in terms of the structure of Figure 8.4, and that these principles can be applied to multiple-adder or other more complicated arrangements should it prove desirable.

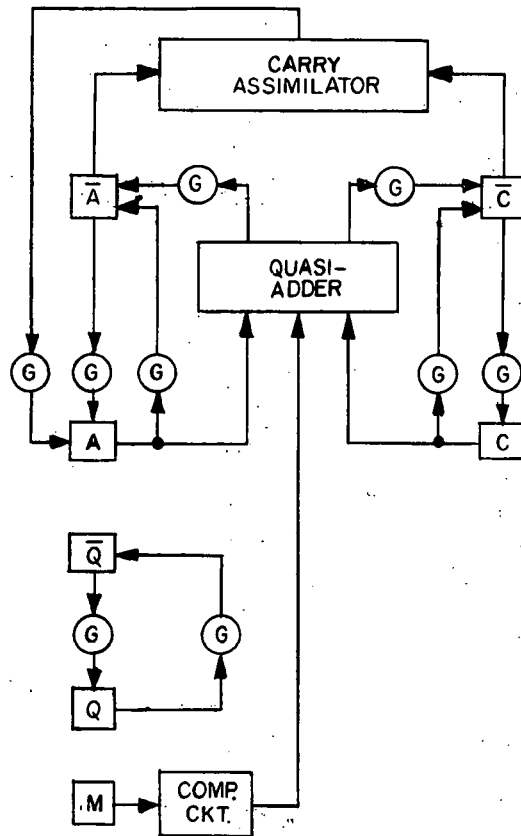


Figure 8.4. Block Diagram of the Proposed Arithmetic Unit

Without the equivalent of many adders available, multiplication necessarily involves a serial or serial-parallel sensing of the multiplier, and a sequence of conditional uses of the adder and shifts.

The sensing of the multiplier can begin with either the most or the least significant digit. In either case, the accumulator must be augmented by a second register, if all digits of the double-length product are represented. For initial sensing of the most significant digit, the accumulator is extended to the left, with the augmenting register holding the most significant half of the product; for initial sensing of the least significant digit, the accumulator is extended to the right, and the augmenting register holds the least significant half of the product. Factors affecting the choice of the method of sensing the multiplier include:

1. the desirability of providing facilities for shifting both to the right and to the left,
2. the possibility of holding in the augmenting register both product digits and multiplier digits,
3. the nature of auxiliary equipment required for the augmenting register.

In regard to shifting facilities, division requires the left shift and it seems desirable to provide right shifting facilities as well. If the right shift were not made available to the programmer, the choice of initially sensing the most significant digit of the multiplier could result in a decrease in hardware by the amount of equipment required for the right shift.

The decisive consideration is apparently the amount of equipment required for the augmenting or multiplier registers. If the choice is made to sense the most significant digit of the multiplier first, methods have not yet been devised for the multiplier recoding proposed such that the augmenting register can hold both multiplier and product digits, nor can complementing facilities for the augmenting register be avoided. The best choice thus seems

to be the sensing of the least significant digit of the multiplier first for which methods of holding product and multiplier digits in a single register exist, and for which very little additional equipment over and above that required for a single length shifting register is required. The latter comment applies only if the choice of two's complement representation of negative numbers is made, as will be shown.

### 8.15 The Recoding of the Multiplier

With a complementing circuit located between the multiplicand in register M and the adder, it is possible to both add and subtract the multiplicand from a partial product held in registers A and C. A third choice is to shift the partial product without use of the adder. Equivalently, it is possible to recode the multiplier into the digits -1, 0, and +1; the recoding is done in such a way that the number of uses of the adder is decreased, and requires the sensing of two multiplier digits and a mode digit  $w_{j-k+1}$ . For a multiplier with  $n+1$  digits  $y_0, y_1, \dots, y_n$ , the rules for the  $k^{\text{th}}$  step ( $k = 0, 1, \dots, n$ ) are:

	$w_{n-k+1}$	$y_{n-k-1}$	$y_{n-k}$	recoded multiplier digit	$w_{n-k}$	$t_{n-k}$
+ mode	0	0	0	0	0	0
	0	0	1	+1	0	1
	0	1	0	0	0	0
	0	1	1	-1, change mode to -	1	1
- mode	1	0	0	+1, change mode to +	0	1
	1	0	1	0	1	0
	1	1	0	-1	1	1
	1	1	1	0	1	0

where for  $k = 0$ ,  $w_{n+1} = 0$ , and for  $k = n$ ,  $y_{-1} = y_0$ . In addition to the equivalent recoded multiplier digit and the mode change rules, the table shows the binary values of the new mode digit  $w_{n-k}$ , which controls the setting of the

complementing circuit, during the  $k^{\text{th}}$  step, and a digit  $t_{n-k}$ , which, if 1, indicates that the adder is used during the  $k^{\text{th}}$  step. A negative multiplier poses no problems; it is only necessary that during step  $n$ ,  $y_{-1} = y_0$ , and the adder is used if  $w_1$  and  $y_{-1} = y_0$  are such that  $t_0 = 1$ .

It is relatively easy to show that the number of uses of the adder tends toward  $1/3$  the number of multiplier digits, as the number of multiplier digits increases, if the assumption is made that the binary digits of the multiplier are independent and each is equally likely to be 0 or 1. We fix our attention on one binary digit of the multiplier, say  $y_j$ , and inspect the binary digits of lesser significance to determine whether or not a use of the adder is indicated by the rules (specifically, by  $t_j$ ). If  $y_j = 0$ , with probability  $1/2$ , then it is equally likely that  $y_{j+1}$  is 0 or 1. If  $y_j = y_{j+1} = 0$ , with probability  $1/4$ , then  $t_j = 0$ . If  $y_j = 0$ ,  $y_{j+1} = 1$ , then  $y_{j+2}$  must be inspected. The values  $y_j = 0$ ,  $y_{j+1} = y_{j+2} = 1$  would yield  $t_j = 0$ ,  $t_{j+1} = 1$ , with probability  $1/8$ . By an obvious extension of the argument, the probability  $p_1$  that  $t_j = 1$  is,

$$p_1 = 2(1/8 + 1/32 + 1/128 + \dots) = \sum_{i=1}^m (1/4)^i.$$

Thus  $p_1$  tends toward  $1/3$  as the number of digits of the multiplier increases.

The multiplication time can be reduced if another feature of the recoding rules is utilized. If an addition or subtraction occurs during the  $j^{\text{th}}$  step of the process, i.e., if  $t_j = 1$ , then no use of the adder can occur on the following  $(j-1)^{\text{st}}$  step. This feature can be verified by inspection of the rules or by a Boolean proof that  $t_j \cdot t_{j-1} = 0$ , by direct substitution of the equations:

$$t_j = w_{j+1} \oplus y_j,$$

$$t_{j-1} = w_j \oplus y_{j-1},$$

and

$$w_j = w_{j+1}y_j \oplus w_{j+1}y_{j-1} \oplus y_jy_{j-1}.$$

It therefore follows that, for any pair of recoded multiplier digits, only one use of the adder is required. Thus, the number of gating operations for shifts can be reduced by a factor of two if a modified base 4 multiplication method is employed, with each shift displacing multiplier and partial product digits two digital positions to the right. Base 4 operation requires that the doubled multiplicand be available, that is, one base 4 digit of the multiplier is recoded as one of the reversed quinary digits -2, -1, 0, 1, or 2.

#### 8.16 The Multiplication Right Shift: Introduction

For the right shift, consideration must be given to the insertion of the most significant digits into A and C and to the transfer of the least significant digits of A and C into Q. The necessity for an explicit negative multiplicand correction is avoided if the most significant digits are inserted in such a way that the resulting partial product in A and C is correct, even if overflow has occurred. Correct insertion of the most significant digits is closely related to addition overflow, and is discussed in detail in section 8.12.

Two aspects of the transfer of digits from A and C into Q are described. The first aspect is of fundamental importance in the choice of the mode of representation of negative numbers and is concerned with the necessity for corrections if the quantities in A and C and in Q are of opposite sign. The second aspect is that carry assimilation is required for the digits transferred into Q, if no carry register is to be associated with Q.

#### 8.17 The Multiplication Right Shift: The Nature of Corrections if A and Q Are of Opposite Sign

The method of multiplication proposed implies that successive partial products may differ in sign. A partial product  $p_k$  is represented by  $n+k+1$  digital positions, occupying  $n+1$  digital positions of A and C, and  $k$  digits of Q. Since the adder has  $n+1$  digital positions which sense A and C, it is necessary

to consider the effect of an addition or subtraction of the  $n+1$  digit multiplicand to A and C in such a way that the signs of successive partial products differ. The difficulties that arise are illustrated by examples, one for each mode of representation of negative numbers, of the addition  $11/64 + (-5/8) = -29/64$  under the assumption that  $n = k = 3$ . In each example, the addition is performed correctly for the  $n+1$  most significant digital positions, and compared with the correct  $n+k+1$  digit representation of  $-29/64$ .

	Absolute Value	One's Complement	Two's Complement
11/64	0.001 011	0.001 011	0.001 011
-5/8	<u>1.101</u>	<u>1.010</u>	<u>1.011</u>
Sum	1.100 011	1.011 011	1.100 011
-29/64	1.011 101 [ ] [ ]	1.100 010 [ ] [ ]	1.100 011 [ ] [ ]
	A Q	A Q	A Q

The sum is correct in all examples provided that the digits in A are interpreted as  $-1/2$  and the digits in Q are interpreted as  $+2^{-3}(+3/8)$  according to the rules of the representation in use. It is only for the two's complement representation that the digits of Q are independent of the sign of A; otherwise it must be understood that the number in Q is positive and the number in A is negative.

For the multiplier recoding proposed, or for an unrestricted hold-multiply instruction with other multiplication methods, a change in sign in partial products is inescapable. For arithmetic units employing the absolute value or one's complement representation, the designer is then faced with the problem of either altering the digits of both A and Q so that the correct  $n+k+1$  digit representation is maintained, or shifting the least significant digit from A to Q where A and Q have opposite signs. Either alternative requires that the sign of the partial product be known but if the partial product is represented with carries separately stored, its sign in general can only be determined by a partial assimilation of carries. If the features of the multiplier recoding and separate carry storage are to be fully exploited, the conclusion is that the two's complement representation of negative numbers should be used.

8.18 The Multiplication Right Shift: Assimilation of Carries for Digits Transferred from A and C to Q

As noted in the discussion for binary subtraction, correct operation requires that  $c_n$ , the least significant digit of the carry register C, be zero, in order that the carry insertion for complementation will be correct. In a sequence of subtractions, the digit  $b_n$ , which would subsequently be gated into  $c_n$ , is always zero; however, during multiplication, a right shift of one digital position requires that  $b_{n-1}$ , which may not be zero, be transferred into a carry register R associated with Q, in order that  $c_n$  will be zero.

For purposes of analysis, we extend the discussion of carry assimilation to an  $n+k+1$  digit representation of the partial product, assuming temporarily that A and C are  $n+1$  digit registers augmented by  $k$  digits of Q with a carry register R associated with Q. During a right shift of one digital position, the digit  $s_n$  of  $\bar{A}$  is transferred to become  $q_1$  of Q, while  $b_{n-1}$  of  $\bar{C}$  becomes  $r_0$  of R. Thus the relationship  $s_n b_{n-1} = 0$  becomes  $q_1 r_0 = 0$ , and after  $k$  steps of a multiplication, Q and R each contain  $k$  digits,  $q_j, r_{j-1}$ , with  $j = 1, 2, \dots, k$ , and the relationship  $q_j r_{j-1} = 0$  holds for each  $j$ .

The existence of R is a temporary fiction, since assimilation of Q and R can be performed serially as the shifting occurs. Assimilation of  $k$  digits of Q yields an assimilator carry

$$e_0 = q_1(r_1 \vee e_1)$$

where the relationship  $q_j r_{j-1} = 0$  guarantees that  $e_0 r_0 = 0$ . One step later,  $e_0$  is shifted right to become  $e_1$ ,  $r_0$  becomes  $r_1$ , digits  $b_{n-1}$  and  $s'_n$  are transferred from  $\bar{C}$  and  $\bar{A}$  into R and Q, and a single step of the assimilation yields

$$r'_0 = b'_{n-1}$$

$$e'_0 = s'_n(r'_1 \vee e'_1) = s'_n(r_0 \vee e_0)$$

$$q'_1 = s'_n \oplus (r_0 \vee e_0)$$

Thus, a single step of the assimilation can be performed with each right shift, and a carry register associated with  $Q$  is not required. The assimilation of  $\bar{A}$  and  $\bar{C}$  following a multiplication proceeds in accordance with that following any other arithmetic operation, except that  $d_n = e'_0$ , and  $b_n = r'_0$ , with  $b_n d_n = 0$ .

### 8.19 Multiplication Overflow

Overflow can occur in multiplication when the multiplicand  $x$  and the multiplier  $y$  lie in the range  $-1 \leq x < 1$ ,  $-1 \leq y < 1$ , if  $x = y = -1$ . Even if a hold-multiply is executed, overflow is indicated correctly if signs of multiplier, multiplicand, and product are simultaneously negative.

### 8.20 Division: Introduction<sup>(6)</sup>

The choice of restoring or non-restoring division requires consideration of

1. the requirements for mechanization of the method; with particular attention to the mechanization of one step, and
2. the nature of the quotient and remainder resulting from the process.

During each step, each division process requires a permutation of the same four operations. The paralleling of two operations is possible for non-restoring division, serial sequencing of the four operations is necessary for restoring division. With the exception of special cases such that divisor and dividend are equal in absolute value, the quotients resulting from the two processes are the same, and the corresponding remainders can be determined with equal facility. Thus, the choice of the non-restoring division process can be based on the timesaving made possible by paralleling of operations during each step.

A further reduction in division time can be achieved by initially shifting divisor and dividend left until the divisor  $y$  is standardized to lie in the range  $1/2 \leq |y| \leq 1$ , and during the division, standardizing the quantities held in  $A$  and  $C$ . The method can be applied to both restoring and non-restoring

6. A new quaternary division method, which in many respects is the inverse of the multiplication described in section 8.15, is discussed in Digital Computer Laboratory report no. 82, "A New Class of Digital Division Methods", by James E. Robertson, March 5, 1958. The new division method supersedes the method described in section 8.22.



division, but has the disadvantage that the remainder is relative to the standardized divisor, rather than the unstandardized one. The method is particularly attractive if the register M holding the divisor is a shifting register, as would be convenient for floating point operation.

The programmer is interested in the characteristics of the division instructions available. It is proposed that two division instructions representing a compromise between ease of programming and ease of mechanization be available. The sorts of difficulties that arise can most easily be discussed in connection with the formula

$$q \cdot y + 2^{-n} r_n = r_0,$$

where  $q$  is the quotient formed in register Q,  $y$  is the divisor in M,  $r_n$  is the remainder in  $\bar{A}$  and  $\bar{C}$ , and  $r_0$  is the dividend initially in  $\bar{A}$  and  $\bar{C}$ . In theory,  $q$  and  $r_n$  are functions of one another; for example  $q$  may be so adjusted that any one of the following restrictions applies to  $r_n$ :  $0 \leq r_n < |y|$ ,  $0 \leq |r_n| \leq 1/2 |y|$ ,  $0 \leq |r_n| < |y|$  such that  $r_n$  has the sign of the dividend, etc. Unfortunately  $q$  is formed in the quotient register Q where addition facilities are unavailable, and the nature of  $q$  as formed by a mechanized process is such that the mathematically attractive choices of restrictions on  $r_n$  cannot be achieved without an additive or subtractive change of more than the least significant digit of  $q$ .

With these considerations in mind, the two division instructions proposed are then:

1. Correctly rounded quotient in A with no remainder. This instruction requires that  $n+1$  non-sign digits of a quotient  $q$  be determined, that  $q$  be transferred to A, destroying the remainder, and that  $2^{-n}$  be added to  $q$  if the  $(n+1)^{\text{st}}$  digit of  $q$  is 1.
2. Quotient in Q with remainder in A and C. For this instruction, the restrictions on  $r_n$  are such that  $q$  as formed in Q by a mechanized process need not be modified in more than the least significant digit.

The first instruction should suffice for the bulk of fractional single-precision calculations requiring division; the second will be useful for multiple precision or integer divisions.

### 8.21 Non-Restoring Division

The recursion relationship for the partial remainders for non-restoring division is

$$r_{k+1} = 2r_k - (-1)^{p_k + y_0} y$$

where  $r_k$  and  $y$  are partial remainder and divisor, with sign digits  $p_k$  and  $y_0$ , respectively, and  $k = 0, 1, \dots, n$  is the recursion index. For  $k = 0$ ,  $r_0$  is the dividend; and for  $k = n$ ,  $r_n$  is the remainder. The  $r_k$  are held in  $\bar{A}$  and  $\bar{C}$ ,  $y$  is held in  $M$ . From the recursion relationship, it can be shown that

$$2^{-n} r_n = r_0 - y \sum_{i=1}^n 2^{-i} (-1)^{y_0 + p_{i-1}}$$

The quotient digit  $Z_k = +1$  or  $-1$  formed at each step is

$$Z_k = (-1)^{p_{k-1} + y_0}$$

and the quotient  $q$  is then

$$q = \sum_{i=1}^n 2^{-i} Z_i = \sum_{i=1}^n 2^{-i} (-1)^{p_{i-1} + y_0}$$

from which it readily follows that  $qy + 2^{-n} r_n = r_0$ .

It can also be established by induction on  $k$  that  $-|y| \leq r_k < |y|$ , provided  $-|y| \leq r_0 < |y|$ . For the proof, consider two cases:

$$-|y| \leq r_k < 0$$

$$0 \leq r_k < |y|$$

Then  $-2 |y| \leq 2r_k < 0$  or  $0 \leq 2r_k < 2 |y|$ .

Noting that  $(-1)^{y_0} y = |y|$  and that  $-(-1)^{p_k} = +1$  for  $r_k < 0$

and  $-(-1)^{p_k} = -1$  for  $r_k \geq 0$ , it follows that, for the two cases:

$$- |y| \leq 2r_k + |y| < |y| \quad - |y| \leq 2r_k - |y| < |y|$$

which shows that

$$- |y| \leq r_{k+1} < |y|.$$

The quotient  $q$  with digits  $Z_i = \pm 1$ , can easily be converted to the conventional binary representation; the conversion requires that the least significant digit  $q_n$  of the converted quotient be 1. If  $q_n$  is set to 0, the analysis above must be modified as indicated by

$$(q-2^{-n})y + 2^{-n}(r_n + y) = r_0,$$

i.e., a decrease in  $q$  by  $2^{-n}$  corresponds to adding  $y$  to  $r_n$ .

The range restrictions on  $r_k$  apply in particular to the remainder  $r_n$ , and may be rewritten directly if  $q_n = 1$  or modified by addition of  $y$  to  $r_n$  if  $q_n = 0$ .

	$q_n = 1$	$q_n = 0$
$r_n < 0 \quad y < 0$	$y \leq r_n < 0$	$2y \leq r_n + y < y$
$r_n < 0 \quad y \geq 0$	$-y \leq r_n < 0$	$0 \leq r_n + y < y$
$r_n \geq 0 \quad y < 0$	$0 \leq r_n < -y$	$y \leq r_n + y < 0$
$r_n \geq 0 \quad y \geq 0$	$0 \leq r_n < y$	$y \leq r_n + y < 2y$

The above may be regarded as a list of all possible ranges of remainders available without modification of more than the least significant digit  $q_n$  of  $q$ . Two possibilities are:

1.  $q_n = 1$ ,
2.  $q_n = 1$  or  $0$  in accordance with agreement or disagreement, respectively, of signs of  $r_n$  and  $y$ .

In either case, the undesirable situation  $|r_n| = |y|$  cannot be avoided. The second choice seems preferable in that the remainder is less than  $y$  if  $y \geq 0$ ; furthermore the remainder and the divisor agree in sign. It should be noted that either the remainder  $r_n$  or the conditionally modified remainder  $r_n + (1 - q_n)y$  can be formed with relative simplicity. However, if it is required that a remainder  $r'_n$  be found such that  $0 \leq |r'_n| \leq 1/2 |y|$ , the mechanization would be difficult, since

1. An additional step of the division would be required, involving destruction of  $r_n$ .
2. If  $r_n$  and  $y$  agree in sign, addition facilities may be required to increase  $q$ .

The conclusion is that the division with remainder instruction be the second of the two possibilities listed above, and that a distinct division instruction resulting in a correctly rounded quotient without a remainder should also be available.

## 8.22 The Standardized Division

If the dividend and divisor  $y$  are initially shifted left so that  $y$  lies in the range  $1/2 \leq |y| \leq 1$ , the quantities  $2r_k$  in non-restoring division can be similarly standardized with a reduction in division time, whenever  $0 \leq |r_k| \leq 1/4$ . If  $m$  binary shifts are initially performed, the quotient and remainder satisfy

$$q(2^m y) + 2^{-n} r_n = 2^m r_0,$$

indicating that  $q$  is unchanged, and that  $r_n'' = 2^m r_n$ . Thus the correct remainder  $r_n$  could only be obtained by a right shift of  $r_n''$  by  $m$  digital positions. The method can therefore be applied most easily to a floating point division or to a fixed point division which does not require a remainder.

The quotient digit insertion for non-restoring division must be modified, as follows:

$$Z_k = 1 \text{ if the subtraction } r_{k-1} - y \text{ is performed,}$$

$$Z_k = 0 \text{ if a shift of } r_{k-1} \text{ is executed,}$$

$$Z_k = -1 \text{ if the addition } r_{k-1} + y \text{ is performed.}$$

The conversion of the reversed ternary representation to the conventional binary representation can be performed serially if one reversed ternary digit, say  $Z_k$ , is held for one step and modified on the basis of the sign  $p_k$  of the next partial remainder  $r_k$ . During standardization, of course, the signs of successive partial remainders do not change; therefore, the partial remainder signs need be determined only for those  $r_k$  immediately following an addition or subtraction. The method of conversion of a single reversed ternary digit  $Z_k$  to the corresponding binary digit  $q_{k-1}$  is:

$Z_k$	$(-1)^{p_k+y_0}$	$q_{k-1}$
+1	-1	0
+1	1	1
0	-1	1
0	1	0
-1	-1	0
-1	1	1

Some idea of the reduction in division time can be gained from the following calculation. If it is assumed that  $y$  is equally likely to be anywhere in the range  $1/2 \leq |y| \leq 1$  and that each  $r_k$  is equally likely to be

within the range  $0 \leq |r_k| \leq y$ , then the range of  $r_k$  such that at least one use of the adder is eliminated is  $0 \leq |r_k| \leq 1/4$  with a probability of  $\frac{1}{4|y|}$ . The fractional reduction  $R$  in uses of the adder can be calculated by averaging this probability over the interval  $[1/2, 1]$  of  $|y|$ :

$$R = \frac{\int_{1/2}^1 \frac{d|y|}{4|y|}}{\int_{1/2}^1 d|y|} = \frac{1}{2} \left[ \ln |y| \right]_{1/2}^1 = \frac{1}{2} \ln 2 = 0.346$$

### 8.23 Division Overflow

If the division process is begun by forming  $r_0 - (-1)^{p_0+y_0} y$ , the result is a numerical comparison of absolute values of divisor  $y$  and dividend  $r_0$ . The sign of this result can then be used to determine the sign of the quotient. The quotient sign can be independently determined by inspection of signs of divisor and dividend. Overflow is indicated if the results of the two methods of sign generation differ.

### 8.24 Quaternary Operation

The theory of the arithmetic unit presented thus far has been expressed in terms of binary operation. There appear to be a number of advantages to quaternary operation of the arithmetic unit. Among these are:

1. The equipment required for the carry register would be halved. Only base 4 carries would be stored, and shifts would involve a displacement of carries over two binary digital positions.
2. Carry assimilation could be completed more quickly. With proper attention to circuit details, the carry through a quaternary digital position can be completed as quickly as a binary carry propagation.

3. Gating time for shifts could be reduced in multiplication and in the standardized division. The characteristics of the multiplier recoding are such that at most one use of the adder is required for each quaternary multiplier digit. During division, the number of gating operations would be decreased in those instances where standardization is possible.

With one exception, the theory of binary operation can be readily extended for quaternary operation. For storage of carries, the general theory for arbitrary radix  $r$  can be applied with the result that the existence of a carry implies that the associated quaternary sum digit is 0. On the other hand, a modified quasi-adder can be designed in such a way that the existence of a carry implies that the most significant binary digit of the associated quaternary sum digit is 0. The latter design is slightly simpler and faster.

Quaternary operation in multiplication and division requires that, if  $m$  is the multiplier or divisor in  $M$ , the quantities  $m$ ,  $-m$ ,  $2m$ , and  $-2m$  be available for addition. Such a doubling and complementing circuit is approximately twice as complicated as the complementing circuit required for binary operation.

For arithmetic operations, it would be sufficient to provide gating circuits for quaternary shifting only, since the lack of binary shifts can in part be compensated for by use of the doubling circuit attached to  $M$ . On the other hand, binary shifts are preferable for logical operations. Further investigation is necessary to determine whether or not both binary and quaternary shifting facilities should be provided.

#### 8.25 Estimates of Operation Times and Hardware Requirements

In the absence of large-scale experimental data, estimates of operation times and hardware requirements are at best approximate. It is nonetheless possible, if assumptions are consistent, to compare the relative merits of various

proposals for the arithmetic unit structure. The tabular material which follows is sufficiently detailed so that the calculations of time and hardware estimates can easily be repeated for design modifications necessitated by experimental results.

The data on which the estimates are based are these:

Circuit	Transistors	Diodes	T + 1/2D	Operation Time
FLIPFLOP (f)	4	11	9.5	
HALF ADDER (h)	8	6	11	25 $\mu$ s
SINGLE GATE (s)	2.3	0.7	2.7	100 $\mu$ s
DOUBLE GATE (d)	4.3	1.2	4.9	50 $\mu$ s
AND CIRCUIT (a)	2	0	2	5 $\mu$ s
OR CIRCUIT (o)	2	2	3	5 $\mu$ s
NOT CIRCUIT (n)	2	4	4	15 $\mu$ s
LEVEL RESTORER (r)	3	3	4.5	15 $\mu$ s
COMPLEMENTING CIRCUIT (c)	6	2	7	10 $\mu$ s

NOTE 1 The half adder is composed of 2 ANDS, 1 OR, and 1 NOT circuit and yields a sum s and carry c from two binary inputs x and y according to the Boolean equations  $c = x \cdot y$   $s = \overline{(x \cdot y)} \cdot (x \vee y)$ .

NOTE 2 The operation time of the flipflop is included in the gating times. The time given for single gating includes an estimate of clearing time as well.

NOTE 3 Equipment estimates for gates include equipment for gate driver circuits.

The variations in design result from a number of choices which include:

1. Method of gating during transfers or shifts,
2. Inclusion of storage facilities for carries,
3. Use of a carry completion signal, during either conventional addition or carry assimilation,
4. Recoding of multiplier,



5. Quaternary operation,
6. Shifting number register.

In order to study the merits of the proposals, we determine the hardware and speed characteristics of various arithmetic units, the first (unit A) being a transistorized version of the Illiac, the remainder embodying one or more proposals as follows:

Unit	Method of Gating	Separate Carry Storage	Carry Completion	Multiplier Sensing	Base Used	Number Register
A	Single	No	No	Binary	Binary	Fixed
B	Double	No	No	Binary	Binary	"
C	"	Yes	Yes	Binary	Binary	"
D	"	No	No	Reversed Ternary	Quaternary	"
E	"	Yes	No	Reversed Ternary	Quaternary	"
F	"	Yes	Yes	Reversed Ternary	Binary	"
G	"	Yes	Yes	Reversed Ternary	Quaternary	"
H	"	Yes	Yes	Reversed Ternary	Quaternary	Shifting

It should be noted that unit G is the unit proposed for fixed point operation; units E and F are modifications of unit G, such that the merits of carry completion circuitry and quaternary operation can be determined. The equipment costs per bit for registers and gates are then

Register	A	Q	M	C	Totals	Trans.	Diodes	T + 1/2D
Unit								
A	2f+4s	2f+3s	f	-	5f+7s	36.2	59.8	66.1
B	2f+4d	2f+3d	f	-	5f+7d	50.2	63.3	81.9
C	"	"	f	2f+4d	7f+11d	75.4	90.0	120.4
D	"	"	f	-	5f+7d	50.2	63.3	81.9
E	"	"	f	f+2d	6f+9d	62.8	76.7	101.2
F	"	"	f	2f+4d	7f+11d	75.4	90.0	120.4
G	"	"	f	f+2d	6f+9d	62.8	76.7	101.2
H	"	"	2f+3d	f+2d	7f+12d	79.7	91.2	125.3

and the equipment costs for adding, complementing, doubling or assimilating, as the design requires, are:

Unit	Adder or Quasi-adder	Assimilator	Comp. or Doubling	Totals	Trans.	Diodes	T+1/2D
A	$2h+o+0.2r$	-	c	$2h+o+0.2r+c$	24.6	16.6	32.9
B	"	-	c	$2h+o+0.2r+c$	24.6	16.6	32.9
C	$2h+o$	$h+4a+2o+0.3r$	c	$3h+4a+3o+0.3r+c$	44.9	26.9	58.4
D	$2h+o+0.2r$	-	2c	$2h+o+0.2r+2c$	30.6	18.6	39.9
E	$2h+o$	$h+o+0.2r$	2c	$3h+2o+0.2r+2c$	40.6	26.6	53.9
F	"	$h+4a+2o+0.3r$	c	$3h+4a+3o+0.3r+c$	44.9	26.9	58.4
G	"	"	2c	$3h+4a+3o+0.3r+2c$	50.9	28.9	65.4
H	"	"	2c	"	"	"	"

For our speed estimates of arithmetic operations, it is convenient to calculate the time required for the basic operations, for an n bit arithmetic unit.

Unit	Shift ( $t_s$ )	Adder or Quasi-adder with Complementing and Doubling, ( $t_a$ )	Assimilation ( $t_c$ )
A	$2s = 0.2$	$c+2h+n(a+o+0.2r) = 0.06+0.013n$	-
B	$2d = 0.1$	"	-
C	"	$c+2h = 0.06$	$h+a+\log_2 n(2a+o+0.3r) = 0.03+0.02 \log_2 n$
D	"	$2c+2h+n(a+o+0.2r) = 0.07+0.013n$	-
E	"	$2c+2h = 0.07$	$h+0.5n(a+o+0.2r) = 0.025 + 0.007n$
F	"	$c+2h = 0.06$	$h+a+\log_2 n(2a+o+0.3r) = 0.03+0.02 \log_2 n$
G	"	$2c+2h = 0.07$	$h+a+0.5 \log_2 n(2a+o+0.3r) = .03+.01 \log_2 n$
H	"	"	"

NOTE: For units C,F,G, and H,  $t_c$  is the average assimilate time. For the maximum  $t_c$ , replace  $\log_2 n$  by  $n$ . Otherwise, maximum and average times are the same.

Approximate formulas for operation time are then:

	Addition or Subtraction	Assimi- lation	Multiplication		Division	
			Avg.	Max.	Avg.	Max.
A	$t_s + t_a$	--	$n(t_s + 1/2t_a)$	$n(t_s + t_a)$	$n(t_s + t_a)$	$n(t_s + t_a)$
B	"	--	$n(t_s + 1/2t_a)$	$n(t_s + t_a)$	$n(t_s + t_a)$	$n(t_s + t_a)$
C	"	$t_c$	$n(t_s + 1/2t_a)$	$n(t_s + t_a)$	Note 1	Note 1
D	"	--	$n(1/2t_s + 1/3t_a)$	$n(1/2t_s + 1/2t_a)$	$n(t_s + t_a)$	$n(t_s + t_a)$
E	"	$t_c$	$n(1/2t_s + 1/3t_a)$	$n(1/2t_s + 1/2t_a)$	Note 2	Note 2
F	"	$t_c$	$n(t_s + 1/3t_a)$	$n(t_s + 1/2t_a)$	Note 1	Note 1
G	"	$t_c$	$n(1/2t_s + 1/3t_a)$	$n(1/2t_s + 1/2t_a)$	Note 1	Note 1
H	"	$t_c$	$n(1/2t_s + 1/3t_a)$	$n(1/2t_s + 1/2t_a)$	Note 3	Note 3

NOTE 1 For units C, F, and G, the division time is  $n[1/2 t_s + t_a + \max(1/2 t_s, t'_c)]$ , where  $t'_c$  is the time for assimilation to the sign digit. Average and maximum division time estimates are determined from average and maximum values of  $t'_c$ .

NOTE 2 For unit E, the division time is determined by the formula of NOTE 1, except that  $t'_c$  is the assimilation time for all carries.

NOTE 3 For the average division time of unit H, multiply the formula of NOTE 1 by  $(1 - 0.5/n)$ . For the maximum add  $1/2 n t_s$  to the formula of NOTE 1.

The speed and hardware requirements are, then, for  $n = 52$

Unit	$t_s$	$t_a$	$t_c$		Add	Multiply		Divide		Total Equipment		
			Avg.	Max.		Avg.	Max.	Avg.	Max.	Trans.	Diodes	T + 1/2D
A	0.2	0.736	-	-	0.936	29.6	48.7	48.7	48.7	3160	3980	5150
B	0.1	0.736	-	-	0.836	24.4	43.5	43.5	43.5	3890	4160	5970
C	0.1	0.06	0.15	1.07	0.16	6.76	8.32	10-20*	61.5 <sup>+</sup>	6250	6090	9300
D	0.1	0.746	-	-	0.756	15.5	22.0	44.0	44.0	4200	4260	6330
E	0.1	0.07	0.39	0.39	0.17	3.81	4.42	26.5	26.5	5370	5370	8060
F	0.1	0.06	0.15	1.07	0.16	6.24	6.76	10-20*	61.5 <sup>+</sup>	6250	6090	9300
G	0.1	0.07	0.09	0.55	0.17	3.81	4.42	10-20*	35 <sup>+</sup>	5910	5490	8660
H	0.1	0.07	0.09	0.55	0.17	3.81	4.42	7-14*	35 <sup>+</sup>	6790	6250	9900

\* These figures are guesses since the characteristics of numerical data are not sufficiently well known for the calculation of  $t_c$ .

+ It is highly unlikely that these maxima would be obtained in any practical situation since they assume a maximum  $t_c$  at each step.

## 8.26 Interpretation of Speed and Hardware Estimates

In sections 3.2 and 3.3, a criterion for the choice of one of several alternate designs is presented. If  $T$  is a measure of the faultless computer time required for solution of a problem, and  $n$  is the number of equally reliable switching elements in the computer, then the criterion is that a 1% increase in  $n$  should yield a 1.05% increase in speed. The criterion is used here to evaluate the arithmetic unit proposals, with  $T$  interpreted as the average multiplication time, and  $n$  interpreted as a measure of the hardware requirements for the arithmetic unit. We may consider  $T$  to be a measure of the time for solution of many problems, however,  $n$  should properly be a measure of the number of switching elements in the entire computer rather than in the arithmetic unit alone. However, if a proposal satisfies the criterion with  $n$  interpreted as the amount of hardware in the arithmetic unit, it will certainly satisfy the criterion with  $n$  a measure of the total computer hardware, since a 1% increase in arithmetic hardware is less than a 1% increase in total computer hardware.

Using the results of section 8.25, we compute, relative to Unit A,

Units	$\Delta T$	$T$	$-\frac{\Delta T}{T}$	$\Delta n$	$n$	$\frac{\Delta n}{n}$	$-\frac{n\Delta T}{T\Delta n}$
B/A	- 5.2	24.4	0.213	820	5150	0.159	1.34
C/A	-22.84	6.76	3.38	4150	"	0.806	4.19
D/A	-14.1	15.5	0.909	1180	"	0.229	3.97
F/A	-23.36	6.24	3.74	4150	"	0.806	4.64
G/A	-25.59	3.81	6.72	3510	"	0.682	9.85

Thus, Unit G, which results in a six-fold increase in the multiplication rate for a 2/3 increase in hardware, is best by the criterion. It may be noted that units F and H are not included in the above table since the effects of carry assimilation time and division time are refinements outside the realm of the relatively crude approach of this section. Furthermore, the feasibility of the fast division for unit H is dependent on shifting facilities for register M; the decision as to whether or not M should be a shifting register is properly related to whether or not floating point arithmetic facilities are available.

## 8.27 Floating Point Arithmetic

Although the details of the mechanization of floating point operations are subject to the results of future investigations, it is nonetheless true that certain requirements are imposed on the arithmetic unit structure independent of the details. A number in floating point consists of a fractional part and an exponent. Two such floating point numbers are, as far as their fractional parts are concerned, treated in much the same way as fixed point quantities, except that values of exponents may require preliminary operations for addition and subtraction, and that addition or subtraction facilities are required for exponents in multiplication and division. Furthermore, standardization of results may be necessary.

For floating point operation, it is proposed that:

1. Fractional parts of floating point numbers should be expanded to fixed point precision in the arithmetic unit. This feature tends to reduce round-off errors if, as is proposed in Chapter 3, extra registers are provided for the arithmetic unit such that relatively complicated operations are performed in the arithmetic unit with a minimum of storage references.
2. Shifting facilities should be provided for the memory register M. For floating point addition, for example, either the addend in M or the augend in A must be shifted, depending on the relative size of the exponents. If M should be shifted, the alternatives are to
  - a. provide shifting facilities in M, or
  - b. transfer the number in M to Q, thereby destroying the quantity in Q, shift in Q, and transfer the shifted result to M.

Since the gates for transfers between M and Q are as complex as gates for shifting M, and since further advantages in division and in retaining the contents of Q accrue if the former choice is made, we propose that M be constructed as a shifting register.

With these choices in mind, we conclude that a floating point arithmetic unit requires the following hardware over and above that required for a fixed point unit:

1. Addition and subtraction facilities for exponents. It may be possible to time-share these facilities with the address modification registers (B-lines).
2. Shifting facilities for memory register M.
3. Suitable control facilities for standardization, etc.
4. Storage facilities for exponents associated with operands. In some cases counters are also required for exponents.

From the arithmetic unit designer's viewpoint, floating point operation utilizes the equipment required for fixed point operation, plus equipment for facilities described in the previous paragraph. Thus, a complete theory of fixed point operation including overflow analysis determines the major portion of the characteristics of a floating point unit.

