THE DESIGN AND CONSTRUCTION OF

THE DIGITAL COMPUTERS SNOCOM, NIMBUS AND ARCTURUS

D.G. WONG

A thesis presented in support of an application for the degree of Doctor of Philosophy in the School of Electrical Engineering of the University of Sydney.

December, 1966.
This thesis contains an account of the development of the three digital computers SNOCOM, NIMBUS and ARCTURUS within the School of Electrical Engineering of the University of Sydney.

SNOCOM was the first semiconductor, general-purpose digital computer constructed and installed in Australia. It was built for the Snowy Mountains Hydro Electric Authority and was commissioned in August, 1960.

NIMBUS is the first educational digital computer of its kind to be built in Australia (and possibly in the world). It was commissioned in December, 1961.

ARCTURUS is a parallel, general-purpose digital computer which utilises advanced techniques of design and construction, and which contains some novel and original features. Its internal processing speed is greater than that of other computers constructed in Australia. It was commissioned in May, 1966, and has been used for both teaching and research purposes.

Chapter 1 of this thesis deals with SNOCOM. A brief description of this computer is presented, and then the design of the "Auto-input" (an original feature due to the Author) is described in detail. Conclusions drawn from the SNOCOM project terminate the chapter.

Chapter 2 deals with NIMBUS, and contains a brief description, a summary of the design of a very simple computer and conclusions drawn from the NIMBUS project.

Chapters 3-7 deal with the various stages of the ARCTURUS project, viz : - system design (Chapter 3), functional and logical design (Chapter 4), circuit design (Chapter 5), constructional techniques and commissioning (Chapter 6) and programming (Chapter 7). Although many logical design considerations are not even mentioned, Chapter 4 is, by far, the longest chapter. The approach taken with this chapter was to describe designs for selected sections of the computer so that a reasonably accurate representation of the design techniques used in ARCTURUS is produced.

Chapter 8 contains a review of computer developments and an appraisal of the SNOCOM, NIMBUS and ARCTURUS projects.

The ARCTURUS project was carried out with extremely limited man-power and financial support. By salvaging semiconductors and connectors from old equipment, and by constructing several peripheral units in the laboratory, this computer was produced with a component cost of less than $A 10,000. This sum was obtained over a period of about five years.
The Author's publications relating to the work described in this thesis are as follows:-


Reformatted versions of Papers (i), (iii) and (iv) appear in the Appendix. Paper (ii) is not included as it is similar to Paper (iii).

Papers (i), (ii) and (iv) were read at Computer/Engineering Conferences. The Author received the 1963 Award of the J. R. Bainton Prize from the Institution of Engineers, Australia for Paper (i).

The Author was in charge of the final stages of the development of SNOCOM and of all stages of the development of NIMBUS and ARCTURUS. Mr. K. R. Rosolen (the Author's Design Assistant) played a major role in all three projects, and was mainly responsible for circuit design and constructional techniques. System design, functional design, logical design and programming were the main responsibilities of the Author.

The merit of the work described in this thesis lies in (i) the novel and original features incorporated in the three computers SNOCOM, NIMBUS and ARCTURUS, (ii) the fact that, under the Author's direction, a small but effective research group capable of establishing advanced concepts and techniques has been formed and (iii) the fact that the SNOCOM, NIMBUS and ARCTURUS projects represent a significant contribution to the development of computers in Australia.
PHOTOGRAPHS

SNOCOM

NIMBUS

ARCTURUS
The construction of SNOCOM was made possible by the generous support given by the Snowy Mountains Hydro Electric Authority to computer development within the School of Electrical Engineering of the University of Sydney.

Dr. M.W. Allen, formerly of the Mathematical Instruments Section of the Commonwealth Scientific and Industrial Research Organization (now Professor of Electronic Computation at the University of New South Wales), was responsible for the circuit design and constructional techniques utilized in SNOCOM. The initial stages of development of SNOCOM were under his direct supervision, and he is to be given much of the credit for its success.

The assistance given by Messrs. W.S. Lamond, L.G. Bellamy, J.E. Todd, R.B. White, K.R. Rosolen, P.T. Brown and D. Linsten during various stages of the SNOCOM project is also gratefully acknowledged.

The author is particularly indebted to Mr. K.R. Rosolen who capably assisted in all stages of the NIMEUS and ARCTURUS projects. Mr. Rosolen is to be given much of the credit for the circuit design and constructional techniques used in these projects. He was directly responsible for the construction of the tape reader, the modifications to the printer, the wiring of a large number of packages, all the base wiring of ARCTURUS and all the mechanical construction.

The author would like to thank Mr. I.M. Cowell for wiring some of the packages for ARCTURUS and for his assistance during the latter stages of the project.

The author would particularly like to thank Professor W.N. Christiansen, Head of the School of Electrical Engineering of the University of Sydney for his encouragement and support of all the projects described in this thesis.

Finally, the author would like to thank his supervisor, Dr. H.K. Messerle,* for his guidance and sound advice on numerous occasions during the running of the projects and also during the production of this thesis.

* (now Professor of Electrical Engineering at the University of Sydney)
CONTENTS

INTRODUCTION

CHAPTER 1 - THE DIGITAL COMPUTER SNOCOM
  1.0 List of symbols used in this chapter
  1.1 Introduction
  1.2 General description
  1.3 Functional & logical design
  1.4 The auto-input
  1.5 Conclusions

CHAPTER 2 - THE EDUCATIONAL DIGITAL COMPUTER NIMBUS
  2.0 List of symbols used in this chapter
  2.1 Introduction
  2.2 Brief description of NIMBUS
  2.3 The design of a simple computer
  2.4 Conclusions

CHAPTER 3 - ARCTURUS - SYSTEM DESIGN
  3.0 List of symbols
  3.1 Approach to system design
  3.2 General description of Arcturus
  3.3 Implementation of system design

CHAPTER 4 - ARCTURUS - FUNCTIONAL & LOGICAL DESIGN
  4.0 List of symbols
  4.1 Approach to functional & logical Design
  4.2 Timing unit
  4.3 Machine phases
  4.4 Control
  4.5 Carry lookahead adder
  4.6 Machine registers
  4.7 Multiplication procedure
  4.8 Division procedure
  4.9 Programmed operators
  4.10 Peripheral units
  4.11 Comments on functional & logical design

CHAPTER 5 - ARCTURUS - CIRCUIT DESIGN
  5.0 Design approach
  5.1 Circuit types & characteristics
CHAPTER 6 - ARCTURUS - CONSTRUCTIONAL TECHNIQUES  & COMMISSIONING

6.0 Packaging
6.1 Base-wiring
6.2 Engineering construction
6.3 Commissioning procedure

CHAPTER 7 - ARCTURUS - PROGRAMMING

7.0 The assembly program API
7.1 Programmed-operators
7.2 Subroutine hierarchy
7.3 Software development

CHAPTER 8 - REVIEW & APPRAISAL

8.0 Preface to final chapter
8.1 Review of computer developments
8.2 Appraisal of SNOCOM, NIMBUS & ARCTURUS projects

REFERENCES

APPENDIX
INTRODUCTION

The development of the digital differential analyser ADA (Ref. 1) was carried out as a joint project of the School of Electrical Engineering of the University of Sydney and the Mathematical Instruments Section of the Commonwealth Scientific and Industrial Research Organization. As this project was supported by the Snowy Mountains Hydro Electric Authority, one of its aims was to produce a second computer for the Authority.

In 1957 investigations carried out by the author (Ref. 2) led to the conclusion that the Authority's computing requirements could be better met by a general-purpose digital computer than by a digital differential analyser. After ADA was completed early in 1958, the development of the computer, which was to be known as SNOCOM, was commenced. The availability of Frankel's design of the LGP-30 (Ref. 3) and the realization that essentially the same circuit and constructional techniques developed for ADA could be used with the second machine were the main reasons for basing SNOCOM (Ref. 4) on the logical structure of the LGP-30. SNOCOM was commissioned in August, 1960, and has given many years of good service.

In 1961 the educational digital computer NIMBUS (Refs. 5 and 6) was constructed and since 1962 it has been used very effectively in final-year courses on computer design. NIMBUS consists essentially of a large number of logic and storage elements which may be readily patched together. The number, type and arrangement of these elements have been selected so that it is possible to synthesize a very simple computer.

NIMBUS has also been used to test logic circuit configurations for carrying out high-speed arithmetic and to bread-board equipment for testing computer peripherals.

During the years 1961 to 1963 the peripheral units for ARCTURUS (including a paper tape reader of novel design (Ref. 7) were constructed; the circuitry associated with the ferrite-core memory was constructed and a new range of 4 MHz logical elements was developed.

The specification and design of the central processing unit for ARCTURUS were completed by mid-1964. Construction of this unit was carried out in the years 1964 to 1965. Commissioning of the computer commenced in late 1965, and by early 1966 the computer was in good operating order.
The development of ARCTURUS was carried out with very limited resources. This has meant that much time was spent in developing peripherals rather than purchasing commercial units, in salvaging transistors and diodes from old equipment and in manufacturing in the laboratory almost every logical package of the computer. Items of major expenditure include a Teletype punch, a 32x32x20 ferrite-core stack, about 2000 transistors and about 5000 diodes. The justification for the purchase of these units still had to include the usefulness of the final product for teaching purposes. Hence the specifications of ARCTURUS had to be made so that the computer could be constructed in a reasonable time with the available resources and so that the final product would be useful both for teaching and research.

In spite of the difficult conditions under which the ARCTURUS project was carried out, a reasonably fast computer with some novel and original features has been constructed and is now in good operating order.

Because ARCTURUS is significantly different from all computers previously constructed in Australia, a new approach to design had to be established. This has resulted in a powerful central processing unit which was relatively simple to implement.

No relaxation of the specifications of ARCTURUS was found to be necessary during commissioning. In fact, not only were the design and constructional techniques found to be quite adequate, but also they suggested methods which could be used in faster and more powerful computers.
CHAPTER 1
THE DIGITAL COMPUTER SNOCOM

1.0 LIST OF SYMBOLS USED IN THIS CHAPTER
1.0.0 Symbols used in the text
1.0.1 Symbols appearing in the diagrams only

1.1 INTRODUCTION

1.2 GENERAL DESCRIPTION

1.3 FUNCTIONAL AND LOGICAL DESIGN

1.4 THE AUTO INPUT
1.4.0 A bootstrap routine
1.4.1 The Auto Input
1.4.2 Logical design of the Auto Input

1.5 CONCLUSIONS
1.5.0 Usefulness of the Auto Input feature
1.5.1 Design approach to minimize hardware
1.5.2 Flexibility of a serial arithmetic and control unit.
1.0 LIST OF SYMBOLS USED IN THIS CHAPTER

1.0.0 Symbols used in the text

- \( S_0 - S_5 \) Segment timing waveforms (Fig. 1.3)
- \( D_0 - D_5 \) Digit timing waveforms (Fig. 1.3)
- \( T_0, T_2 \) Interleaved clock pulses (Fig. 1.3)
- \( R_T \) Once-per-revolution marker (Fig. 1.3)
- \( V \) Store playback flip-flop
- \( V_b \) Auto input store playback signal
- \( C \) Sequence counter
- \( 0 - 9 + - N J F L \) Sexadecimal characters
- / \# Control characters used in input instruction
- \( w_1 - w_{16} \) Binary counter outputs of the sector number generator
- \( B \) Auto input flip-flop
- \( B' \) Signal which sets \( B \) to ONE
- \( B'' \) Signal which resets \( B \) to ZERO
- \( \phi_1 \) Phase 1
- \( t_w \) Waste digit (Fig. 1.3)
- \( A \) Accumulator playback flip-flop

1.0.1 Symbols appearing in the diagrams only

- \( F \) Phase flip-flops
- \( G \) Sector search, lock-out and augmentation flip-flop
- \( H \) Carry-borrow flip-flop
- \( Q_1 - Q_4 \) Order flip-flops
- \( P_1 - P_6 \) Track selection flip-flops
- \( A'' \) Accumulator input (i.e. signal to accumulator record amplifier)
- \( A_* \) Accumulator playback flip-flop
- \( A^* \) Double length accumulator playback flip-flop
- \( C \) Counter playback flip-flop
- \( R'' \) Instruction register input
- \( R^* \) Instruction register playback flip-flop
- \( V'' \) Digits to be recorded (in the main store)
- \( f \) Duration of recording (in the main store)
- \( t_s \) Sign digit
- \( w \) Full address waveform
- \( w_e \) Early full address waveform
- \( v \) Sector number generator output.
1.1 INTRODUCTION

In the early 1950's engineers of the Snowy Mountains Hydro-Electric Authority had used the C.S.I.R.O. Mechanical Differential Analyser (Ref. 8) with some success for the solution of flood-routing problems (Ref. 9).

In the mid-1950's the development of the digital differential analyser ADA (Ref. 1) was commenced as a joint project of the School of Electrical Engineering of the University of Sydney and the Mathematical Instruments Section of the Commonwealth Scientific and Industrial Research Organization.

Because the digital differential analyser had been accredited with many of the outstanding advantages of both analogue and digital computers, and because of its previous success with a mechanical differential analyser, the Snowy Mountains Hydro-Electric Authority supported the ADA project with the understanding that a second computer would be built for its own use.

The Authority's computer was to be specifically designed for the solution of a system-operational problem which had previously taken many man-years of manual calculations for its solution. It was shown by the author that a digital differential analyser solution was possible, but about four hundred integrators would have been required and the computation time might have been prohibitively long (Ref. 2). The main reason for this was that the problem had not been formulated as a set of ordinary non-linear differential equations, and although the differential analyser approach (e.g. for the generation of analytic functions and for the evaluation of integrals) could be applied to advantage in some sections of the problem, there were some sequences of arithmetic and logical operations on whole numbers which could not be avoided. As the digital differential analyser is an incremental computer, operations on whole numbers are extremely inefficient.

In 1957 a system-operational problem typical of those which the Authority wanted solved was programmed for its solution on SILLIAC, the general-purpose digital computer within the Basser Computing Department of the University of Sydney. The SILLIAC studies carried out by the author led to the conclusion that the Authority's computing requirements could be better met by a general-purpose digital computer than by a digital differential analyser.
After ADA was completed early in 1958, the development of the computer which was to be known as SNOCOM was commenced. The availability of Frankel's design of the LGP-30 (Ref. 3) and the realization that essentially the same circuit and constructional techniques developed for ADA could be used with the second machine were the main reasons for basing SNOCOM (Ref. 4) on the logical structure of the LGP-30.

SNOCOM was constructed within the School of Electrical Engineering of the University of Sydney, and was then installed in August, 1960 within the offices of the Snowy Mountains Hydro-Electric Authority in Cooma - about 260 miles from Sydney. Since 1960 SNOCOM has given many years of good service.

A paper on SNOCOM (Ref. 4) was presented at the Annual Conference of the Institution of Engineers, Australia in March, 1962 in Cooma. The author received the 1963 Award of the J.R. Bainton Prize for this paper. A reformatted version of this paper is presented in the appendix. A general description of SNOCOM will now be presented. This is followed by an outline of the functional and logical design and by details of the "Auto Input". Conclusions drawn from the SNOCOM project terminate the chapter.
1.2 GENERAL DESCRIPTION

SNOCOM is a fixed-point, binary, serial, stored-program, single-address, general-purpose digital computer using transistor circuits and magnetic drum storage. The storage capacity of the computer is 2048 words, each of 32 binary digits. These are arranged on 64 tracks of the drum with 32 words per track. The Model 512-A Bryant magnetic drum (5" dia., 12" long) runs at 6,000 R.P.M., giving a clock rate of 102.4 kHz, a word time of 312 microseconds and a mean access time of 5 milliseconds. The accumulator, instruction register and instruction counter are in the form of recirculating registers, with the spacing between record and playback heads corresponding to one word period. The addition of a second playback head on the accumulator recirculating register enables double length numbers to be stored for use in the multiplication and division processes. The computer logic is synchronized by clock pulses derived from a clock track and a once-per-revolution marker permanently recorded on the drum.

An interlaced sector number system enables simple instructions to be obeyed within nine word periods. Another sixty-four word periods are required for the multiplication and division instructions. Hence the addition and multiplication times are 2.8 milliseconds and 22.8 milliseconds, respectively. The bench-mark time (i.e. the time to carry out ten additions and one multiplication) is about 51 milliseconds.

The peripheral units consist of (1) a Ferranti TR 5 paper tape reader which operates at 300 characters per second, (2) a Teletype BRPE paper tape punch which operates at 50 characters per second and (3) a modified IBM electric typewriter which operates at 10 characters per second. Both output units are computer controlled and all units may operate simultaneously.

A SNOCOM number word consists of a sign bit, thirty magnitude bits and a spacer bit as shown in Figure 1.0. The binary point is assumed to be between bit 0 and bit 1, and the normal two's complement system for the representation of negative numbers is used.

A SNOCOM instruction word consists of four order bits, six track bits and five sector bits as shown also in Fig. 1.0. The track and sector bits together constitute the full address of the instruction. Apart from the sign bit which is used in conjunction with the conditional transfer of
control order for external program control, the remaining bits have no significance. Corresponding to the different combinations of the four order bits, there are sixteen basic instructions which SNOCOM is capable of executing.

The order code for SNOCOM's sixteen basic instructions is shown in Table 1.0. These instructions are defined in detail in the Appendix.

<table>
<thead>
<tr>
<th>CODE</th>
<th>SEXADEcimal</th>
<th>Binary</th>
<th>ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0001</td>
<td>Bring</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>1110</td>
<td>Add</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>1111</td>
<td>Subtract</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>Multiply fractions</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>Multiply integers</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>Divide</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>Extract</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1010</td>
<td>Unconditional transfer</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>1011</td>
<td>Conditional transfer</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>1100</td>
<td>Store</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>1101</td>
<td>Store and clear</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>Store address</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>Return address</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>Stop</td>
<td></td>
</tr>
</tbody>
</table>

* See Appendix J for details.
Figure 1.0 - FORMAT OF SNOCOM NUMBER & INSTRUCTION WORDS
1.3 FUNCTIONAL AND LOGICAL DESIGN

The following brief description refers to the simplified diagram of Fig. 1.1. More details appear in the SNOCOM paper.

The execution of each instruction stored on the magnetic drum takes place in four to eight phases with each phase consisting of one or more word periods. During computation information is held on the drum, in the recirculating registers and in flip-flops. The definition of the operations, which are to be executed in each phase, such as the transfer of information from drum to recirculating register or from recirculating register to flip-flop, constitutes the functional design of the computer.

Logical design consists of deriving Boolean equations from which logic circuits may be readily obtained to implement the functional design.

The logical design of the LGP-30 as described by Frankel (Ref. 3) consists primarily of (1) the specification of the conditions under which each of the main fifteen flip-flops is set and reset, (2) the specification of the digits to be recorded in the recirculating registers in each digit period and (3) the specification of the exact location on the magnetic drum in which specified digits are to be recorded.

SNOCOM is a synchronous computer as all operations carried out in all possible machine phases are synchronized by timing waveforms generated from clock tracks on the drum. These timing waveforms are shown in Fig. 1.2.

To maintain complete flexibility so that any digit within a word may be specified, a word period has been divided into six segments with each segment containing either four, five or six digits. The segment waveforms $S_0 - S_5$ and the digit waveforms $D_0 - D_5$ of Fig. 1.2 are generated by two ring counters and interconnecting logic. Any digit within a word is specified in terms of these waveforms. For example, the waste digit and the sign digit are the first and last digits in the word and may be specified as $S_0D_2$ and $S_0D_1$, respectively.

The interleaved clocks $T_0$ and $T_2$ are used in the two-phase logic arrangement. The once-per-revolution marker synchronizes the segment and digit rings.

In SNOCOM consecutive addresses do not correspond to consecutive locations on a track of the drum, but to locations which are spaced nine word periods apart. In this way an instruction, which refers to an operand which is appropriately
located between itself and the next instruction, is obeyed in nine word periods rather than the minimum time of thirty-two word periods (corresponding to one drum revolution) which would otherwise be required. To make the above interlaced system possible, a sector number must be formed by subtracting seven from the number of the preceding sector (modulo 32). The resulting sector number sequence is shown in the Appendix (in the SNOCOM paper).

Two three-stage binary counters are used in the generation of the sector number sequence. The first counter corresponds to the three least significant digits of the sector number and is pulsed to count forwards every word period. The second counter corresponds to the two most significant digits of the sector number plus an additional (more significant) digit which effectively counts odd and even drum revolutions. This second counter is pulsed to count backwards on every word period except every eighth when the first counter changes from the state 111 to 000. With the above arrangement one is added to the weight one position of the sector number and one is subtracted from the weight eight position; this results in a sector number being formed which is seven less than the preceding one. The second counter is inhibited from changing when the first has changed from 111 to 000 as this already represents a subtraction of seven.

The binary counters are pulsed early in the word period and their states are gated by the digit waveform \(D_0 - D_5\) so that their representation is converted into the serial form required by the machine logic.

In the LGP-30 the sector number pattern is recorded permanently on one track of the drum. This is not the case for SNOCOM, and the parallel representation of the sector which is next to be presented by the store playback amplifier is utilized in the auto input logic.
Figure 1.1 - SIMPLIFIED BLOCK DIAGRAM OF SNOCOM
INTERLEAVED CLOCK PULSES

\[ T_0 \]
\[ T_2 \]
\[ R_T \]

REVOLUTION MARKER

\[ D_0 \]
\[ D_1 \]
\[ D_2 \]
\[ D_3 \]
\[ D_4 \]
\[ D_5 \]

WASTE

\[ S_0 \]
\[ S_1(u) \]
\[ S_2(z) \]
\[ S_3 \]
\[ S_4(x) \]
\[ S_5(y) \]

SECTOR NUMBER

TRACK NUMBER

ORDER

SECOND SECTOR NUMBER

ONE WORD PERIOD

WASTE DIGIT \( S_0D_2 \)

SIGN DIGIT \( S_0D_1 \)

\( t_w \)

\( t_s \)

\( w \)

\( w_e \)

\( v \)

FULL ADDRESS (LATE)

FULL ADDRESS (EARLY)

SECTOR NUMBER GENERATOR (TYPICAL SECTION OF WAVEFORM)

Figure 1.2 - SNOCOM TIMING WAVEFORMS
1.4.0 A bootstrap routine

With SNOCOM's order code the minimum number of instructions required for a bootstrap or short input routine is three. In sexadecimal these instructions are as follows:

<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00400000</td>
<td>Input</td>
</tr>
<tr>
<td>001</td>
<td>00N00030</td>
<td>Store in location 3</td>
</tr>
<tr>
<td>002</td>
<td>00400000</td>
<td>Input</td>
</tr>
</tbody>
</table>

The first two words on the input tape must be:

/00N00040# /00+00000# ,

where / and # are the control characters detected by the input logic to enable a complete word to be read into the accumulator by the input instruction.

The routine must be started at location 000 by pressing the "clear C" button on the control console. The latter word on the tape is stored in location 004 on the execution of the former (instruction) word which had previously been stored in location 003 by the three orders of the initial bootstrap routine. On the execution of the latter (instruction) word in location 004 control is transferred to location 000, and the following program on the input tape is then effectively under the control of the following routine:

<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00400000</td>
<td>Input</td>
</tr>
<tr>
<td>001</td>
<td>00N00030</td>
<td>Store in location 3</td>
</tr>
<tr>
<td>002</td>
<td>00400000</td>
<td>Input</td>
</tr>
<tr>
<td>003</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>004</td>
<td>00+00000</td>
<td>Transfer to location 0</td>
</tr>
</tbody>
</table>

This routine may be used to fill a program into the store when each word of the program is preceded by a "store" or "store and clear" instruction whose operand address corresponds to the location in which the word is to be stored.

The execution of the bootstrap routine can be terminated if the first order of an order pair is an unconditional transfer of control order.
1.4.1 The Auto Input

After the auto input button has been pressed, the normal store playback (V) from sectors 0, 1 and 2 is inhibited and replaced by a pattern of digits generated from the timing unit. This pattern of digits is equivalent to the following three orders:

\[
\begin{align*}
00400030 & \quad \text{(from sector 0)} \\
00N00030 & \quad (\quad \quad \text{1}) \\
00400030 & \quad (\quad \quad \text{2}).
\end{align*}
\]

If the first two words on a program tape are:

\[
/00N00040# /00+00000#
\]

and the computer is started after pressing the "clear C" button, the word 00+00000 will be stored in 004 and the program will then be effectively under the control of the following bootstrap routine:

\[
\begin{array}{cccc}
\text{Location} & \text{Instruction} \\
000 & 00400030 \quad \text{From timing unit} \\
001 & 00N00030 \\
002 & 00400030 \\
003 & (\quad \quad \text{From store}) \\
004 & 00+00000
\end{array}
\]

The normal store playback from sectors 0, 1 and 2 (track 0) is not inhibited permanently, conditions being returned to normal immediately a ONE is sensed in the waste digit position of the accumulator in phase 1.

It is to be noted that, because of a simplification of logical design, the normal store playback is inhibited whenever the sector number generator announces a number whose binary equivalent contains a ZERO in the weight 4 position, and this applies not only to track 0 but to all tracks. To eliminate these abnormal conditions as soon as possible, it is recommended that all program tapes should begin with the following eight words:

\[
\begin{align*}
/00N00040# & /00+00000# \\
/00N00000# & /00400000# \\
/00N00010# & /00N00030# \\
/00N00020# & /00400001#
\end{align*}
\]

When this is done, the bootstrap routine discussed above will be actually written into the store, and conditions will be returned to normal by the ONE in the waste digit position of the
eighth word.

After the program tape has been placed in the tape reader, the normal procedure for entering a program using the auto input is as follows:

1. Press the initial set button.
2. Press the clear C button.
3. Press the auto input button.
4. Press the operate button.

It is to be noted that without the auto input feature many times this number of operations would have been required for entering programs into the computer.

1.4.2 Logical design of the Auto Input

The sector number generator announces the sector number of the next word to be presented to the read-write heads of the main memory. Hence, when sectors 0, 1 and 2 are being played back, the sector number announces 25, 26 and 27, respectively. If the binary counter outputs of the sector number generator are \( w_1, w_2, w_4, w_8 \) and \( w_{16} \), the relevant sectors are represented by the following:

<table>
<thead>
<tr>
<th>Sector played back</th>
<th>Sector number announcement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( w_1 \bar{w}_2 \bar{w}_4 \bar{w}<em>8\bar{w}</em>{16} ) (sector 25)</td>
</tr>
<tr>
<td>1</td>
<td>( \bar{w}_1w_2 \bar{w}<em>4w_8 \bar{w}</em>{16} ) (&quot; 26)</td>
</tr>
<tr>
<td>2</td>
<td>( w_1w_2 \bar{w}<em>4w_8 \bar{w}</em>{16} ) (&quot; 27)</td>
</tr>
<tr>
<td>3</td>
<td>( \bar{w}_1 \bar{w}<em>2w_4w_8 \bar{w}</em>{16} ) (&quot; 28)</td>
</tr>
<tr>
<td>4</td>
<td>( w_1 \bar{w}<em>2w_4w_8 \bar{w}</em>{16} ) (&quot; 29)</td>
</tr>
</tbody>
</table>

It can readily be seen that \( w_4 \) alone distinguishes sectors 0, 1 and 2 from sectors 3 and 4.

After the auto input button has been pressed, the B flip-flop is set to the ONE state, and while this flip-flop is in this state, the normal V playback from sectors 0, 1 and 2 (and other sectors but not sectors 3 and 4) is inhibited. The conditions to be satisfied while \( V \) is inhibited are therefore represented by \( \bar{w}_4 \cdot B = 1 \).

The normal V playback is replaced by a pattern of digits generated from the timing unit. For example, an input order \( 00400000 \) is represented by a single digit \( S_4D_2 \), and the order \( 00N00030 \) is represented by the 4 digits:

\[ S_4D_3 + S_4D_2 + S_{1D_1} + S_{1D_0} \]
Using $w_1$ and $w_2$ to distinguish between sectors 0, 1 and 2; and $w_4$ to distinguish sectors 0, 1 and 2 from sectors 3 and 4, the pattern of digits to be generated from the timing unit may be represented by:

$$S_4D_2w_1w_2w_4 + (S_4D_3 + S_4D_2 + S_1D_1 + S_1D_0) \overline{w}_1w_2w_4 + S_4D_2w_1w_2w_4.$$

The digits from sectors 0, 1 and 2 have $S_4D_2$ in common. Also, as the address portion of an input order has no significance, the digits $S_1D_1$ and $S_1D_0$ may be added to the input orders without harm. The pattern therefore may be simplified to:

$$(S_4D_2 + S_1D_1 + S_1D_0) \overline{w}_4 + S_4D_3\overline{w}_1w_2w_4.$$

The auto input logic is introduced into the main machine logic by replacing the normal $V$ playback by a new variable $V_b$ which is equal to $V$ under normal conditions, but equal to the digits generated from the timing unit under the conditions of Auto Input (i.e. when $\overline{w}_4B = 1$). Hence

$$V_b = V \cdot \overline{w}_4 + \overline{w}_4 \cdot B(S_4D_2 + S_1D_1 + S_1D_0 + S_4D_3 \overline{w}_1w_4).$$

The $B$ flip-flop is set to ONE when the auto input button is depressed, and is reset to ZERO when a ONE is detected in the waste digit of the accumulator in $\phi_1$. It is also reset to ZERO when the initial set button is depressed so that the computer may be used without the Auto Input operation. Hence

$$B' = \text{auto input button}$$
$$\overline{B}' = \phi_1tA + \text{initial set button}.$$

A logic diagram corresponding to the above equations is shown in Fig. 1.3.
Figure 1.3 - SNOCOM's AUTO INPUT LOGIC
1.5 CONCLUSIONS

1.5.0 Usefulness of the Auto Input feature

Some simple magnetic drum computers (such as the LGP-30) require an undesirably long sequence of console operations to enter programs into the main store. One possible solution to this problem would be to have a bootstrap routine or an assembly routine permanently recorded on one track of the drum. As this track cannot be used for any other purpose, one track of the drum is the price paid for the simpler loading procedure. The most desirable arrangement would be one (like SNOCOM's Auto Input) which requires very little additional hardware and which does not in any way diminish the capabilities of the computer.

Since its installation the Auto Input feature of SNOCOM has been found to be an extremely useful feature and almost all programs are entered this way.

1.5.1 Design approach to minimize hardware

The approach taken by Frankel (Ref. 3) in the design of the LGP-30 had as one of its main objectives the minimization of hardware. One design approach which significantly contributed to the attainment of this objective was the allocation, to computing elements (especially flip-flops), of active functions in as many of the machine phases as possible. For example, the $K$ flip-flop is used to search for the correct sector in phases one and three, to increment the sequence counter in phase two and to lock out the computer when peripheral units are not ready. Another example is the use of the $Q_2$ flip-flop as a means of stopping the computer as well as one of four order flip-flops which determine the type of instruction which is to be obeyed.

Another useful design approach was the appropriate grouping of instructions so that complete decoding of the order flip-flops would not be required as the flip-flop outputs themselves would provide effective control signals.

A close examination of the design of the LGP-30 and SNOCOM would reveal some of the subtleties which enabled the designers to produce remarkably economical designs.

For a commercial computer like the LGP-30 the economies gained by an ingenious design would be extremely desirable because of the saving of cost. For a custom-built computer like SNOCOM (of which only one was built) some of these economies may well prove to be false economies. A case in point is the multiple use of the $Q_2$ flip-flop. When
the computer stops (e.g. on single-shot operation), the Q indicators only reveal that the last instruction obeyed was one of two possible instructions. The additional complexity in interpreting the console indicators would provide a strong case for another flip-flop.

It appears that as computing elements become less expensive, the designer's objective should not be a minimal hardware design but rather an efficient (near minimal) design which considers simplicity of understanding and use as well. Other considerations, of course, include simplicity of construction, commissioning and maintenance procedures.

1.5.3 Flexibility of a serial arithmetic and control unit

Starting from the general framework of the LGP-30 design, it has been found relatively simple to introduce a number of desirable features into SNOCOM. These features include (i) the allowable selection of the number of characters entered by an input instruction, (ii) the auto input feature, (iii) the reduction of the operand search phase (phase three) to one word period for some instructions and (iv) the buffering of all peripheral units and the modification of the phase logic to enable simultaneous operation of all peripheral units and the central processing unit.

As SNOCOM is a serial computer, it would be expected that relatively small amounts of hardware would be necessary to introduce new features, as in many cases only a single signal line is affected. It would, for example, be relatively simple to extend the order code of the computer by providing a large number of variants of some of the instructions.

By making use of recent advances in semiconductor devices, it would be possible to construct a central processing unit similar to SNOCOM's which was significantly more powerful and several orders of magnitude faster. However, as SNOCOM is already "memory-limited", there would be little point in doing this without changing the organization of the computer and the form of the memory. These observations would suggest that a good design could be obtained by combining a parallel, ferrite-core store with a serial central processing unit. This approach has, in fact, been taken by designers of some commercial computers (Ref. 10).
CHAPTER 2

THE EDUCATIONAL DIGITAL COMPUTER NIMBUS

2.0 LIST OF SYMBOLS USED IN THIS CHAPTER

2.1 INTRODUCTION

2.2 BRIEF DESCRIPTION OF NIMBUS

2.3 THE DESIGN OF A SIMPLE COMPUTER

2.4 CONCLUSIONS

2.4.0 Usefulness of machines like NIMBUS for teaching
2.4.1 Usefulness of machines like NIMBUS for research

2.4.1.0 NOR configurations
2.4.1.1 High-speed circuit configurations
2.4.1.2 Multi-function computer elements
2.4.1.3 Testing computer peripherals
LIST OF SYMBOLS USED IN THIS CHAPTER

- Clock pulse (Figs. 2.0 and 2.1)
- Binary counter drive pulse (Figs. 2.0 and 2.1)
- Binary counter outputs (Figs. 2.0 and 2.1)
- Waveform differentiated (Figs. 2.0 and 2.1)
- Waveform differentiated (Figs. 2.0 and 2.1)
- Initial set (Fig. 2.0)
- Signals which stop the timing unit (Fig. 2.0)
- Machine phases 0-3 (Fig. 2.1)
- Function toggle outputs
- Sequence counter outputs
- Sequence counter input
- Sequence counter drive input
- Address register outputs
- Address register input
- Address register drive input
- Fixed store output
- Word 15 (= P_3P_2P_1P_0)
- Instruction register (address) outputs
- Instruction register (address) input
- Instruction register (address) drive input
- Instruction register (order) outputs
- Instruction register (order) input
- Instruction register (order) drive input
- Accumulator outputs
- Accumulator input
- Accumulator drive input
- Store register outputs
- Store register input
- Store register drive input
- Signal which sets the K flip-flop (to 1)
- Signal which resets the K flip-flop (to 0)
- Counter augmentation flip-flop output
- Carry/borrow flip-flop output
- Sum/difference digit
- Store output
- Counter augmentation and carry/borrow flip-flop
2.1 INTRODUCTION

As a result of the rapidly increasing use of digital equipment both for research and industrial applications, there is great demand for personnel trained in digital systems engineering. Training in this field can be carried out quite effectively with the use of "educational digital computers" (Refs. 5 and 6). These are experimental units which have been specifically designed with the specification of a complete computer in mind. The number, type and arrangement of logical circuits provided in the experimental unit enable the synthesis not only of various combinational and sequential switching circuits, but also of a complete stored-program computer.

The first digital-logic training devices were marketed in 1960, and now a large range is commercially available (Refs. 11 and 12). The function of these devices is of course to provide training in the fundamentals of digital circuits. Educational digital computers must also be able to do this, but they go one step further. The synthesis of a complete digital computer would be extremely useful to teach engineers computer design techniques and to help programmers visualize the internal organization of a computer.

Some of the desirable features of a digital-logic training device or an educational digital computer are as follows:- the equipment should contain standard transistorized digital circuits (or integrated circuits) for good simulation of actual design problems; the circuit modules may be either fixed or removable, but the arrangement must be flexible so that many circuits may be synthesized; logic panels must be clearly engraved and extensive monitoring facilities must be provided to facilitate the understanding and testing of circuit configurations; the main control unit must enable the monitoring of memory states after each state-change (i.e. after each clock pulse); and finally the device must be "student-proof" so that any interconnection may be made without damage to the equipment.

Most of the above features have been incorporated into the educational digital computer NIMBUS. This machine has been used since 1962 by final-year students of electrical engineering of the University of Sydney. A brief description of NIMBUS is presented in section 2.2; an outline of the design of an extremely simple computer is presented in section 2.3; and finally conclusions concerning the usefulness of machines like NIMBUS for teaching and research are presented in section 2.4.
2.2 BRIEF DESCRIPTION OF NIMBUS

A photograph of NIMBUS appears on page iv, and a reproduction of a paper (Ref. 6) containing a description of this machine appears in the Appendix.

In essence, the machine consists of several sub-assemblies and a large number of logical and storage elements which may be readily patched together. The sub-assemblies provided in NIMBUS enable experiments on serial, synchronous, digital systems to be carried out.

One of the main sub-assemblies is the control and timing unit. A logic diagram of this unit is shown in Figure 2.0, and timing waveforms are shown in Fig. 2.1. When the START button is depressed, the timing unit generates one pulse, eight pulses, thirty-two pulses or a train of pulses depending on whether the MODE-SELECTION switch is in the BIT, PHASE, ORDER or NORMAL positions, respectively. When this switch is in the MONITOR position, the unit runs continuously and all waveforms may be monitored. The ability to change the mode of operation in this way is an extremely useful and almost indispensable feature of a training device.

To demonstrate the principles of operation of a stored-program computer, some storage device must be provided in the educational machine.

In NIMBUS a sixteen word pinboard-store is provided, and this is supplemented by a very limited amount of erasable storage in the form of a one-word flip-flop register. The outputs of the four-stage address register are decoded into sixteen lines which drive the WORD-busses of the pinboard-store. Component plugs containing sub-miniature diodes may be inserted to bridge the WORD-busses with the eight BIT-busses at appropriate places. The pinboard, together with component plugs, forms a diode-matrix which produces (in parallel) the eight bits of the word specified by the contents of the address register. The diode-matrix outputs are combined with the timing waveforms $B_1$, $B_2$ and $B_4$ in parallel-to-serial conversion circuits to produce the serial store output $V_O$. The selection of either the pinboard-store output or the flip-flop register output may be readily carried out using the outputs of the address register as the gating signals.

Other sub-assemblies take the form of self-contained shift registers. The arrangement of these sub-assemblies were chosen with the organization of a simple computer in mind; for example, a four-stage register for the (sequence) counter is provided and the panel for this sub-assembly is accordingly engraved with this function in mind. Other registers are
provided for the address register, the instruction register and the accumulator.

The sloping panels of the machine contain storage and logical elements. Each of the existing panels contains two flip-flops, three invert circuits and nineteen NOR circuits. Other circuits may be used as the panels are self-contained and readily interchangeable.

The inputs and outputs of all elements in the machine are brought to sockets on the panels so they may be interconnected with flexible leads.
Figure 2.0: NIMBUS CONTROL & TIMING UNIT

Figure 2.1: NIMBUS TIMING WAVEFORMS
2.3 THE DESIGN OF A SIMPLE DIGITAL COMPUTER

The sub-assemblies and logic circuits briefly described in the preceding section may be interconnected to form a simple digital computer. Computers with different order codes may be constructed, the only restriction being the amount of hardware which is necessary for their implementation. An example of a computer which may be constructed is a fixed-point, binary, serial, single-address digital computer with a storage capacity of sixteen eight-bit words of which fifteen are fixed and one is erasable. The computer is capable of carrying out only eleven basic instructions, although four stages in the order part of the instruction register make provision for sixteen. The order code is shown in Table 2.0. A complete description of the functional and logical design of this computer appears in the paper which is reproduced in Appendix K.

The Boolean equations of Table 2.1 represent a summary of this design. The symbols used in these equations are listed in section 2.0.

Table 2.0: Order Code of a Simple Computer

<table>
<thead>
<tr>
<th>Code</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>INPUT — Illuminate the Z₁ monitor light and stop the computer so that eight digits may be set into the accumulator.</td>
</tr>
<tr>
<td>0001</td>
<td>OUTPUT — Illuminate the Z₂ monitor light and stop the computer so that the accumulator may be monitored.</td>
</tr>
<tr>
<td>0010</td>
<td>TRANSFER IF NEGATIVE — Transfer control to the address specified by the instruction if the accumulator is negative.</td>
</tr>
<tr>
<td>0011</td>
<td>UNCONDITIONAL TRANSFER — Transfer control to the address specified by the instruction.</td>
</tr>
<tr>
<td>0100</td>
<td>STORE — Store the contents of the accumulator into the store register (address 15).</td>
</tr>
<tr>
<td>0101</td>
<td>BRING — Bring the word specified by the address of the instruction into the accumulator.</td>
</tr>
<tr>
<td>0110</td>
<td>ADD — Add the word specified by the address of the instruction to the contents of the accumulator retaining the result in the accumulator.</td>
</tr>
<tr>
<td>0111</td>
<td>SUBTRACT — Subtract the word specified by the address of the instruction from the contents of the accumulator retaining the result in the accumulator.</td>
</tr>
<tr>
<td>1010</td>
<td>TRANSFER IF ZERO — Transfer control to the address specified by the instruction if the accumulator is zero.</td>
</tr>
<tr>
<td>1100</td>
<td>ACCUMULATE IN STORE — Add the contents of the accumulator to the contents of the store register (address 15) retaining the result in the accumulator.</td>
</tr>
<tr>
<td>1110</td>
<td>EXTRACT — Obtain the logical product of the word specified by the address of the instruction and the contents of the accumulator retaining the result in the accumulator.</td>
</tr>
</tbody>
</table>

Table 2.1: Boolean Equations for a Simple Computer

<table>
<thead>
<tr>
<th>C₀</th>
<th>B₀T₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₀’</td>
<td>B₀C₁</td>
</tr>
<tr>
<td>B₀B₁(R₆C₀ + R₆C₀’) + R₆R₅R₄ + R₅R₄R₃</td>
<td></td>
</tr>
<tr>
<td>R₅’</td>
<td>B₀B₄</td>
</tr>
<tr>
<td>x = A₄A₃A₂A₁A₀A₉</td>
<td></td>
</tr>
<tr>
<td>A₄’ = B₄B₅T₆</td>
<td></td>
</tr>
<tr>
<td>B₅ = B₄T₆</td>
<td></td>
</tr>
<tr>
<td>b = A₅V + A₆V’ + A₇V’ + A₈V’ + A₉V</td>
<td></td>
</tr>
<tr>
<td>L’ = B₇ + (R₇A₄V + R₇A₄V’) B₅T₆</td>
<td></td>
</tr>
<tr>
<td>Sₗ = W₈B₅T₆</td>
<td></td>
</tr>
<tr>
<td>S’ = B₇S₇ + B₈B₉R₁E₈S₈ + B₉R₁R₉R₇(R₉A₇ + bR₇)</td>
<td></td>
</tr>
</tbody>
</table>

Logic configurations using NOR circuits and flip-flops with transient storage gates may be readily derived from the above Boolean equations. From the logic circuit diagrams, wiring lists may be obtained if they are felt to be necessary. The computer may then be patched-up and "commissioned". Simple programs may be run by inserting the appropriate pins in the pinboard.
2.4 CONCLUSIONS

2.4.0 Usefulness of machines like NIMBUS for teaching

The development of a new computer is in almost every case a group effort. As members of the group may have responsibilities in restricted (and often mutually exclusive) areas (such as integrated circuit manufacture and compiler writing), it is necessary to bring the group's understanding of all computer fundamentals and techniques to a level where every member of the group can contribute to the overall project and also where there is a strong likelihood of some cross-fertilization of ideas. Here machines like NIMBUS have an important role.

In the university environment, training is carried out at various levels. At the undergraduate level, NIMBUS has been used very successfully to give students some insight into computer design techniques. At the honours student level, NIMBUS has been used as part of a more complex digital control system (Ref. 13). At the research assistant level, all personnel associated with the current research project have made use of NIMBUS as part of their training. Finally, at the professional engineer level, NIMBUS provided preliminary training for engineers who were to be responsible for a computer data-logging system in a modern power station.

Since 1962, when it was put into operation, NIMBUS has proved to be very effective for teaching computer fundamentals, and there is a strong case for the continuation and expansion of this educational approach.

2.4.1 Usefulness of machines like NIMBUS for research

2.4.1.0 NOR configurations

Many papers have been written on the design and application of NOR and NAND circuits (Refs. 14-19). Reference 14 contains an annotated bibliography. Reference 15 is an example of a paper on the design of diode-transistor NOR circuits; this is the type of circuit used in NIMBUS.

The dual polarity logic suggested by Kintner (Ref. 16) has been found to be extremely useful. As its name implies, this type of logic represents binary signals using both positive and negative logic. A change in representation occurs when signals pass through a NOR circuit, and the NOR circuit performs the OR function when the input signals are represented in positive logic, while the application of De Morgan's theorem shows that the NOR circuit performs the AND function when the input signals are represented in negative logic.
The implementation of NOR and NAND logic is described in Maley and Earle's book (Ref. 17) and this has been found to be very helpful. Hellerman's catalogue of three-variable NOR and NAND circuits (Ref. 18) and Smith's table of minimal three-variable NOR and NAND circuits (Ref. 19) have been noted for future reference. Variations of these circuits such as the NAND-AND configuration suggested by Burke and Bosse (Ref. 20) could be quite useful.

Logic circuits composed of NOR and NAND elements are now being used extensively in digital systems. As many integrated circuits carry out the NOR or NAND function, and are being used as the major logical element in many (if not most) digital research projects, machines like NIMBUS, which enable configurations of circuit elements with these logical functions to be readily constructed, must become useful research tools. At the time of writing a second educational computer (NIMBUS II) containing integrated circuits (instead of discrete component circuits) is in its course of construction.

2.4.1.1 High-speed circuit configurations

NIMBUS was used for testing circuit configurations for the computer ARCTURUS (Ref. 21). These included circuits for a rapid multiplication procedure and a parallel, reversible counter.

2.4.1.2 Multi-function computer elements

The minimization of hardware is one of the designer's objectives. One way of doing this is to time-share computer elements so that they carry out many functions. An example of what can be done is provided by the design of section 2.3.

The K flip-flop is used for incrementing the sequence counter in phase 1 and is idle at other times. The L flip-flop is used to hold the carry/borrow digit in the addition/subtraction process in phase 3 and is idle at other times. One flip-flop (say J) can carry out both functions. The setting and resetting equations for K and L (Table 2.1) are now replaced by the setting and resetting equations for J viz.:-

\[
J' = B_8 T_O + (\overline{R_4 A_O V} + R_4 A_O V) \overline{B_16 B_8 T_O}
\]

\[
J' = B_8 \overline{C_T} T_O + (\overline{R_4 A_O V} + R_4 A_O V) B_16 B_8 T_O
\]

From the above equations it can be seen that one flip-flop has been saved at the cost of the additional \( B_16 \) and \( \overline{B_16} \) gating. Examples like this provide the designer with a clear understanding of the advantages and additional complexities of logical systems in which computer elements are time-shared. Armed with this knowledge, he will be in a...
better position to decide when to apply this approach in more complex research projects.

2.4.1.3 Testing computer peripherals

NIMBUS was used to test a high-speed paper tape reader (Ref. 7). One of the tests carried out consisted of placing a tape loop of pairs of complementary-characters in the reader and arranging the logic in NIMBUS to read a pair of characters from the tape into two flip-flop registers (formed by cross-coupling NOR circuits) and to test if they were complementary. If they were complementary, the test would be repeated automatically; if not, the test would stop. The reader was tested for periods of many hours in this way.

In the context of time-sharing systems much research effort has been directed at special peripheral devices for improving man-machine communication (Ref. 22), and it is very likely that research along these lines will continue for some time (Ref. 23).

Towards the broad objective of developing a time-shared, digital process controller, a remote-console is being constructed for ARCTURUS (Ref. 24). The logic for the remote console is treated as a complex sequential circuit and a state-flow-diagram approach is being taken for its design. Some of the sequential circuits (consisting of NOR circuits with some feedback connections) which are described in Maley and Earle's book (Ref. 17) have already been synthesized on NIMBUS, and it is planned to test the remote console logic (which utilizes integrated circuits) in the same way.

Based on experience with NIMBUS, it seems very likely that machines like NIMBUS (but perhaps with a little more computing power) would be extremely useful for testing special peripheral devices for computers, and they could become standard (and almost indispensable) items of test equipment for digital research laboratories.
CHAPTER 3

ARCTURUS - SYSTEM DESIGN

3.0 LIST OF SYMBOLS

3.0.0 Symbols used in the diagrams.

3.1 APPROACH TO SYSTEM DESIGN

3.1.0 Background and objectives
3.1.1 Design considerations
3.1.2 Chronology

3.2 GENERAL DESCRIPTION OF ARCTURUS

3.2.0 Word formats
3.2.1 Register configuration
3.2.2 Arithmetic unit
3.2.3 Memory
3.2.4 Peripherals
3.2.5 Instruction Code
3.2.6 Programmed operators and indirect addressing
3.2.7 Machine phases
3.2.8 Control console
3.2.9 Summary of machine specification

3.3 IMPLEMENTATION OF SYSTEM SYSTEM

3.3.0 An introduction to following chapters
3.0 LIST OF SYMBOLS

3.0.0 Symbols used in the diagrams

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>Memory register</td>
</tr>
<tr>
<td>K</td>
<td>Accumulator</td>
</tr>
<tr>
<td>L</td>
<td>Multiplier - quotient register</td>
</tr>
<tr>
<td>R</td>
<td>Instruction register</td>
</tr>
<tr>
<td>S</td>
<td>Sequence counter</td>
</tr>
<tr>
<td>U</td>
<td>Operand counter</td>
</tr>
<tr>
<td>V</td>
<td>Number of times counter</td>
</tr>
<tr>
<td>DFS</td>
<td>Distribution function selector</td>
</tr>
<tr>
<td>AS</td>
<td>Address selector</td>
</tr>
<tr>
<td>T</td>
<td>Timing signals</td>
</tr>
<tr>
<td>C</td>
<td>Control signals</td>
</tr>
<tr>
<td>Ø</td>
<td>Phase signals</td>
</tr>
<tr>
<td>X,Y</td>
<td>Selection lines of coincident-current memory system</td>
</tr>
<tr>
<td>Z</td>
<td>Inhibit lines of coincident-current memory system</td>
</tr>
</tbody>
</table>
3.1 APPROACH TO SYSTEM DESIGN

3.1.0 Background and objectives

The ARCTURUS project was initiated with the background of many years of computer research and development within the School of Electrical Engineering of the University of Sydney. With the Mathematical Instruments Section of the Commonwealth Scientific and Industrial Research Organisation playing a major role, this School took part in the development of the C.S.I.R.O. mechanical differential analyser (Ref. 8), the digital differential analyser ADA (Ref. 1) and the digital computer SNOCOM (Ref. 4).

Towards the end of 1959 the Mathematical Instruments Section of C.S.I.R.O. closed down, and further computer work was carried out by a very small group of University personnel. SNOCOM was commissioned in August 1960; early in 1961, it was decided to retire ADA. At this time the objective was to utilise ADA's electronic components by building a SNOCOM-type computer. A later objective was the connection of this computer to the analogue computer EEDAC.

Very limited resources were available to the computer group. This was responsible for the objectives of the group to be changed several times, and affected the sequence in which the various stages of the project were carried out.

The peripheral units of ARCTURUS were the first units to receive attention. A relay system for a monitor printer and a high-speed paper tape reader (Ref. 7) were developed leaving the Teletype BRPE paper tape punch as the only item of major expenditure.

Former experience with the peripheral units of SNOCOM suggested that the peripheral units (being electro-mechanical) were the least reliable units of a computer and hence some built-in checking procedure was desirable. This was carried out by arranging the peripheral units in such a way that they could operate off-line as a comparator-reperforator-printer.

The decision to change the form of storage from a magnetic drum to ferrite-cores was responsible for abandoning the plans based on a SNOCOM-type computer. At this stage the digital computer CIRRUS (Ref. 25, 26), which was being constructed at the University of Adelaide, stimulated interest in "micro-programming" (Ref. 27-32) and its logical extension "stored-logic" (Ref. 33-36). Hence the objective changed to one of building a stored-logic computer.
With a full appreciation of the capabilities of an extremely small computer group with very limited resources, the group's objective was finally changed to one of building a reasonably powerful general-purpose digital computer with some novel and new features. The usefulness of the computer for teaching and research purposes supported this final decision concerning main objectives.

The approach taken with the design of the arithmetic and control units of ARCTURUS was first to examine the specifications of a number of recent, commercial computers and to select some of the useful features. To these were added features suggested by the computer group's experience with former projects and by information in journals.

With the performance characteristics of the peripheral units, the memory and the logical elements in mind, several forms of machine organisation were considered, and some idea of the best attainable specification for a computer using these units was obtained. Because of the very limited resources available to the group, the construction of such a computer proved to be quite a challenge. However no relaxation of this specification was found to be necessary and the ARCTURUS computer was subsequently completed and commissioned successfully.

Some features not incorporated into ARCTURUS but which received some consideration are listed in section 3.1.1, and a chronology of the various phases of the project are listed in section 3.1.2. A general description of ARCTURUS appears in section 3.2.

3.1.1 Design considerations

Some of the computer systems and computer features which were considered but which were not incorporated into ARCTURUS are indicated below. This list is not exhaustive as many other features received cursory consideration.

Hybrid computer: Papers on hybrid computers (Refs. 37-40) stimulated an interest in the possibility of connecting the new digital computer (ARCTURUS) with the analogue computer EEDAC, and the hardware requirements of such a hybrid computer system were considered.
Control computer: An interest in adaptive control systems and the knowledge that digital computers were being used extensively for process control resulted in some attention being given to the requirements of a control computer.

Stored-logic: The conception of the micro-programmed control unit (Refs. 27-32) has led to the development of a number of computers using this method of control. A logical extension of this method of control utilises a modifiable control memory, and results in a "program-modifiable" or "stored-logic" computer (Refs. 33-36). Such a method of control was considered for ARCTURUS.

Minimal computer: Papers on the minimal logical complexity for a computer (Ref. 41) resulted in consideration being given to a minimal parallel computer containing only three registers.

Parallel-serial system: Papers on high-speed logic circuits capable of operating at pulse repetition rates of 160 MHz (Ref. 42) stimulated some interest in a system containing a parallel (ferrite-core) store and a serial arithmetic unit.

Asynchronous adder: An appreciation of the advantages of the asynchronous adder over the synchronous (ripple-carry) adder resulted in a proposed system containing both asynchronous and synchronous elements. The asynchronous adder was later replaced by a carry-lookahead adder.

Small fast memory: The use of a small fast memory either as a nesting store or as a scratch-pad store was considered.

Fast program loops: The obvious advantages of being able to hold complete programs in flip-flop registers resulted in an investigation of a proposed "micro-mode" (Ref. 36). Micro-mode instructions were addressless and hence several could be stored in each word-length register. It was hoped that the micro-mode feature would enable small program loops to be obeyed very rapidly.
Number formats: Features which would facilitate double-length, floating-point and complex number operations received some attention.

High-level languages: List-processing operations (Refs. 43-44), recursive operations and other features which would facilitate the development of high-level languages were also given some consideration.

3.1.2 Chronology

The following is a chronology of events leading to the commissioning of ARCTURUS. The dates shown are completion dates, and are only approximate as there was much overlapping of the various stages of the project.

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Late-1960</td>
<td>SNOCOM was installed. The next major computer project was considered.</td>
</tr>
<tr>
<td>Mid-1961</td>
<td>The relay system for the monitor printer was developed. The printer logic was designed and constructed. The printer unit was tested.</td>
</tr>
<tr>
<td>Late-1961</td>
<td>The punch logic was designed and constructed. The punch unit was tested.</td>
</tr>
<tr>
<td>Early-1962</td>
<td>NIMBUS was designed, constructed and commissioned.</td>
</tr>
<tr>
<td>Late-1962</td>
<td>The high-speed paper tape reader was developed. The reader logic was designed and constructed. The reader unit was tested.</td>
</tr>
<tr>
<td>Mid-1963</td>
<td>A range of 4 MHz logic circuits was developed.</td>
</tr>
<tr>
<td>Late-1963</td>
<td>The input-output console for ARCTURUS was designed, constructed and tested. This console was arranged so that the input-output units could operate off-line as a punched paper tape editing set.</td>
</tr>
<tr>
<td>Early-1964</td>
<td>The circuitry for the ferrite-core store was designed, constructed and tested.</td>
</tr>
<tr>
<td>Mid-1964</td>
<td>The specification of the arithmetic and control units of ARCTURUS was completed.</td>
</tr>
<tr>
<td>Early-1965</td>
<td>The design of the arithmetic and control units of ARCTURUS was completed.</td>
</tr>
<tr>
<td>Late-1965</td>
<td>The construction of the arithmetic and control units of ARCTURUS was completed.</td>
</tr>
<tr>
<td>Early-1966</td>
<td>ARCTURUS was commissioned.</td>
</tr>
</tbody>
</table>
3.2 GENERAL DESCRIPTION OF ARCTURUS

ARCTURUS is a fixed-point binary, parallel, single-address, general-purpose digital computer using packaged diode-transistor circuits, ferrite-core storage and paper-tape peripheral units.

3.2.0 Word formats

The instruction and number formats, which use a word length of 20 bits, are shown in Figure 3.0. An operation code of 5 bits provides 32 distinct instructions. Some of these have a large number of variants specified by the bits normally used as an address. A 13 bit address enables 8192 words to be directly addressed, although at present only 1024 words are available. Bit 6 of an instruction word specifies indirect addressing while bit 5 specifies a programmed-operator (see Section 3.2.6). A single-length number consists of a sign bit and 19 magnitude bits; a double-length number (such as the product formed by the multiplication of two single-length numbers) consists of a sign bit and 38 magnitude bits as shown in the figure. A two's complement representation of negative numbers is used.

3.2.1 Register configuration

A block diagram of ARCTURUS is shown in Figure 3.1. The memory-output register (J), the accumulator (K), the multiplier-quotient register (L) and the instruction register (R) are all full-length registers, while the sequence counter (S) and the operand counter (U) are only address-length.

The ADDRESS SELECTOR (AS) selects the memory address, from U, S or R. The contents of K or L, or the output of the ADDRESS SELECTOR may be set into J prior to an arithmetic or logical operation being carried out on the contents of J and K. The results of this operation may be set into any register via the DISTRIBUTOR.

3.2.2 Arithmetic unit

The arithmetic unit makes extensive use of a high-speed carry-lookahead-adder which can be used to accumulate numbers in 1/2 μs. By appropriate selection of its inputs, the adder can be used for instruction modification as well as for arithmetic operations. The DISTRIBUTOR FUNCTION SELECTOR (DFS) selects arithmetic or logical functions of register outputs and distributes these signals to all registers. This produces an arithmetic unit which is quite powerful (in terms of arithmetic and logical functions which are possible), and it makes good utilization of the fast adder.
Figure 3.0 - INSTRUCTION & NUMBER FORMATS
Figure 3.1 - BLOCK DIAGRAM OF ARCTURUS
Multiplier recoding produces a multiplication time between 5µs and 7½ µs depending on the multiplier, and a non-restoring procedure for division produces a divide time of 10 µs. The contents of the two arithmetic registers K and L may shift in either direction at a rate of 4 MHz.

3.2.3 Memory

A ferrite-core memory using coincident-current-selection is used. The memory cycle may be either a normal (READ-WRITE) or a split (READ-EXECUTE-WRITE) cycle. The latter is used by instructions with execution phases which modify storage locations in one memory cycle. The memory cycle time is about 3 µs and the access time 1 µs. The execution of some instructions may be carried out simultaneously with the regeneration of information in the store.

The replacement of the 1024 word store by a 4096 word (2 µs) store is planned, and a recently acquired (7 million word) disc file will be added at a later stage.

3.2.4 Peripherals

The peripheral units consist of a 1000 characters-per-second paper-tape reader, a 100 characters-per-second paper-tape punch and a 10 characters-per-second type-writer printer. Each of these units has a one-character buffer, and hence all units and the central processing unit may operate simultaneously. All peripheral units are connected to the computer via the K register.

3.2.5 Instruction code

The ARCTURUS instruction code is shown in Table 3.0. A detailed definition of these instructions appear in Appendix A.
### TABLE 3.0
**ARCTURUS Instruction Code**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Address (A)</th>
<th>Variants (V)</th>
<th>Instruction Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>V</td>
<td>V</td>
<td>Stop</td>
</tr>
<tr>
<td>00001</td>
<td>V</td>
<td>V</td>
<td>Input-Output</td>
</tr>
<tr>
<td>00100</td>
<td>A</td>
<td>A</td>
<td>Transfer Unconditional</td>
</tr>
<tr>
<td>00101</td>
<td>A</td>
<td>A</td>
<td>Transfer if Negative</td>
</tr>
<tr>
<td>00110</td>
<td>V</td>
<td>V</td>
<td>Skip</td>
</tr>
<tr>
<td>00111</td>
<td>A</td>
<td>A</td>
<td>Register Transfer</td>
</tr>
<tr>
<td>01000</td>
<td>A</td>
<td>A</td>
<td>Add Index</td>
</tr>
<tr>
<td>01001</td>
<td>A</td>
<td>A</td>
<td>Subtract Index</td>
</tr>
<tr>
<td>01010</td>
<td>A</td>
<td>A</td>
<td>Index Skip</td>
</tr>
<tr>
<td>01011</td>
<td>A</td>
<td>A</td>
<td>Jump to Subroutine</td>
</tr>
<tr>
<td>01100</td>
<td>A</td>
<td>A</td>
<td>Compare Skip</td>
</tr>
<tr>
<td>01101</td>
<td>A</td>
<td>A</td>
<td>Load K</td>
</tr>
<tr>
<td>01110</td>
<td>A</td>
<td>A</td>
<td>Load L</td>
</tr>
<tr>
<td>01111</td>
<td>A</td>
<td>A</td>
<td>Add</td>
</tr>
<tr>
<td>10000</td>
<td>A</td>
<td>A</td>
<td>Subtract</td>
</tr>
<tr>
<td>10001</td>
<td>A</td>
<td>A</td>
<td>And</td>
</tr>
<tr>
<td>10010</td>
<td>A</td>
<td>A</td>
<td>Multiply</td>
</tr>
<tr>
<td>10011</td>
<td>A</td>
<td>A</td>
<td>Divide</td>
</tr>
<tr>
<td>10101</td>
<td>A</td>
<td>A</td>
<td>Store K</td>
</tr>
<tr>
<td>10110</td>
<td>A</td>
<td>A</td>
<td>Store L</td>
</tr>
<tr>
<td>10111</td>
<td>A</td>
<td>A</td>
<td>Store K Address</td>
</tr>
<tr>
<td>11000</td>
<td>A</td>
<td>A</td>
<td>Accumulate</td>
</tr>
<tr>
<td>11001</td>
<td>A</td>
<td>A</td>
<td>(Spare)</td>
</tr>
<tr>
<td>11011</td>
<td>A</td>
<td>A</td>
<td>Push</td>
</tr>
<tr>
<td>11100</td>
<td>A</td>
<td>A</td>
<td>Decrement Hierarchy</td>
</tr>
<tr>
<td>11101</td>
<td>A</td>
<td>A</td>
<td>Increment Hierarchy</td>
</tr>
<tr>
<td>11110</td>
<td>V</td>
<td>V</td>
<td>Return Hierarchy</td>
</tr>
<tr>
<td>11111</td>
<td>A</td>
<td>A</td>
<td>Execute</td>
</tr>
</tbody>
</table>

The majority of instructions use bits 7-19 to specify a memory address. When these bits are not used in this way but rather to specify variants of the instruction, a large number of variants is possible. For example, during the execution of a REGISTER TRANSFER instruction, the variant bits determine the main control signals of the DISTRIBUTOR-FUNCTION-SELECTOR. As these signals, produce all the arithmetic and logical operations required during all phases of machine operation, all these operations (together with some additional ones, such as absolute-value operations) may be executed as variants of a single instruction. Tables of instruction variants are presented in Appendix E.

Several variants of instructions have been suggested by programming experience. For example, the SKIP IF NORMALISED variant of the SKIP instruction was found to be very useful for floating-point subroutines and was very simple to implement.

Most of the arithmetic instructions are quite standard. Some utilize the READ-EXECUTE WRITE feature of the memory.
For example, during the execution of the ACCUMULATE instruction, the operand is read from memory into the memory output register; the contents of the accumulator are added to the operand and the result is written back into memory. These operations require only a single memory cycle.

Sub-routine entry is simplified using the JUMP TO SUB-ROUTINE instruction. This instruction plants the return link in the location specified by the address in the instruction and then transfers control to the next location (following the link).

An instruction may be indexed by preceding the instruction with an ADD INDEX/SUBTRACT INDEX instruction. In this way any location in memory may be used as an index register, but of course an additional instruction is required for every indexed instruction.

The programming of loops associated with an index is facilitated using the INDEX SKIP instruction. The programming of loops associated with a memory-search is facilitated using the COMPARE-SKIP instruction.

Nesting operations may be carried out using the PUSH and POP instructions.

A hierarchical structure of sub-routines in which a sub-routine may return to any higher or lower level sub-routine may be programmed using the INCREMENT HIERARCHY, DECREMENT HIERARCHY and RETURN HIERARCHY instructions.

The EXECUTE instruction causes a specified instruction to be executed without altering the normal sequencing of the sequence counter.

3.2.6 Programmed operators and indirect addressing

Programmed operators are a feature of computers marketed by Scientific Data Systems (S.D.S.) (Ref. 10). This feature enables a single instruction to specify the address of an operand, to store the sequence counter (return link) and to transfer control to a sub-routine determined by the operation code of the instruction.
Using a simple assembly program (see Chapter 7), such an instruction could be coded like a normal machine language instruction but in fact could cause quite a complex subroutine to be executed. This feature provides the programmer with a modifiable extension of the order code.

The programmed operator feature in ARCTURUS is slightly superior to the feature in S.D.S. computers as programmed operator instructions in ARCTURUS may be indirectly addressed. In ARCTURUS the indirect-addressing bit (bit 6) is sensed before the programmed operator bit (bit 5). If bit 6 is ONE the computer enters an indirect addressing phase in which a new operand address is read from memory and set into the instruction register (the R register). Bit 6 is again sensed and the above procedure may be repeated until bit 6 of the operand address is ZERO. When this condition is satisfied, the final (effective) operand address is set into the U register, (as well as into the R register) where it may be found at a later stage by the programmed operator sub-routine.

When the programmed operator bit (bit 5) is a ONE, the incremented sequence counter is stored in location 0, and control is transferred to location 32 plus the binary number represented by the operation code (bits 0-4). Locations 32 to 63 would contain a directory of starting addresses for the programmed operator sub-routines. This may be stored in the form of a list of UNCONDITIONAL TRANSFER OF CONTROL instructions.

The programmed operator sub-routine finds the operand address in the U register and normally returns to the main program via the link in location 0. If programmed operator sub-routines are nested, each sub-routine must provide internal storage for its link to the next higher level.

Due provision is made for programmed operators by the simple assembly program described in chapter 7.
3.2.7 Machine phases

There are eleven possible machine phases viz: -

(i) fetch instruction
(ii) indirect addressing
(iii) programmed operator
(iv) indexing
(v) execute
(vi) execution - 0
(vii) execution - 1
(viii) execution - 2
(ix) execution - 3
(x) clear store
(xi) auto input

Phases (i) - (ix) in the above list are each associated with a single memory cycle. The CLEAR STORE phase and AUTO INPUT phase may be associated with many memory cycles in which locations are cleared (for the former phase) and in which information read from tape is stored (for the latter phase).

Every machine cycle must contain a FETCH INSTRUCTION phase. In this phase the sequence counter specifies the location from which the instruction is to be read and to be set into the instruction register. If this instruction does not require a memory cycle for its execution (eg. SHIFT, SKIP instruction), the machine remains in the FETCH INSTRUCTION phase during the execution of the instruction. If the instruction requires a single memory cycle for its execution (eg. ADD, STORE instructions), the machine enters the EXECUTION-O phase. On the completion of a machine cycle, the machine returns to the FETCH INSTRUCTION phase for the start of another cycle. The phases EXECUTION - 1, EXECUTION - 2 and EXECUTION - 3 are required by some instructions (eg. PUSH, RETURN HIERARCHY instructions) which require three or four memory cycles for their execution.

Before the operation code of an instruction is used to initiate the execution of the instruction, bit 6 (which specifies indirect addressing) and bit 5 (which specifies a programmed operator) are sensed. When these bits are non-ZERO, the machine may enter the INDIRECT ADDRESSING or PROGRAMMED OPERATOR phases during which the procedure already described in section 3.2.6 is carried out.
When the operation code specifies either an ADD INDEX or SUBTRACT INDEX instruction, the machine enters the INDEXING phase during which the index is read from memory for use in the modification of the following instruction.

When the operation code of the instruction specifies an EXECUTE instruction, the machine enters the EXECUTE phase to read an instruction into the instruction register without any further change being made to the sequence counter.

3.2.8 Control console

The control console for ARCTURUS has been divided into two sections; one section contains all the controls such as SENSE-SWITCHES, BREAK-POINT SWITCHES, START BUTTON, STOP BUTTON etc. which are frequently used by the machine operator; the other section contains MONITOR LIGHTS for all registers and important flip-flops as well as all controls required by the maintenance engineer for carrying out tests on the machine.

More details and a photograph of the control console are presented in a later chapter.

3.2.9 Summary of machine specification

Parallel
Fixed-point, binary
Word-length : 20 bits
Single-address
Number of basic instructions : 32
Addressless instructions have a large number of variants
Indirect addressing
Programmed operators
Ferrite-core memory
Present memory capacity : 1024 words
Designed memory capacity : 8192 words
Memory cycle time : 3μs
Memory access time : 1μs
Accumulate time : 1/2μs
Multiply time : 5 - 7 1/2μs (depending on multiplier)
Divide time : 10μs
Shift time : 1/4μs
Simultaneous instruction execution and memory regeneration
Buffered input and output
Reader speed : 1000 char./sec
Punch speed : 100 char./sec
Printer speed : 10 char./sec
Packaged diode-transistor circuits
Auto input.
CHAPTER 4
ARCTURUS - FUNCTIONAL AND LOGICAL DESIGN

4.0 LIST OF SYMBOLS
4.0.0 Symbols for logical elements
4.0.1 Symbols used in diagrams and text

4.1 APPROACH TO FUNCTIONAL AND LOGICAL DESIGN

4.2 TIMING UNIT
4.2.0 Memory timing
4.2.1 Main timing unit
4.2.2 Repeated operations loop

4.3 MACHINE PHASES
4.3.0 Flow diagram
4.3.1 Implementation

4.4 CONTROL
4.4.0 Arithmetic unit control
4.4.1 Memory control
4.4.2 Miscellaneous control signals

4.5 CARRY LOOKAHEAD ADDER
4.5.0 Brief description
4.5.1 Selection of adder inputs
4.5.2 Alternative configurations
4.5.3 Final adder configuration

4.6 MACHINE REGISTERS
4.6.0 Distributor function selector
4.6.1 Memory input output register
4.6.2 Main arithmetic registers
4.6.3 Counters

4.7 MULTIPLICATION PROCEDURE
4.7.0 General principles
4.7.1 Implementation
4.7.2 Multiplication time

4.8 DIVISION PROCEDURE
4.8.0 General principles
4.8.1 Implementation
4.8.2 Division time
4.9 PROGRAMMED OPERATORS
   4.9.0 Implementation
   4.9.1 Extensions

4.10 PERIPHERAL UNITS
   4.10.0 Block diagrams
   4.10.1 Off-line tape editing
   4.10.2 Input-output logic
   4.10.3 The ARCTURUS Auto Input

4.11 COMMENTS ON FUNCTIONAL AND LOGICAL DESIGN
4.0 LIST OF SYMBOLS

4.0.0 Symbols for logical elements

AND gate

OR gate

INVERT circuit

NOR gate

NAND gate

\[ F' \text{ sets flip-flop to ONE} \]
\[ \bar{F}' \text{ resets flip-flop to ZERO} \]

FLIP-FLOP

\[ F \text{ ONE output of flip-flop} \]
\[ \bar{F} \text{ ZERO output of flip-flop} \]

DIFFERENTIATE circuit

\[ T'_{\text{RD}} \text{ trigger input (ZERO} \rightarrow \text{ONE)} \]

(POSITIVE-TRIGGER)

MONOSTABLE-MULTI

\[ T_{\text{RD}} \text{ ONE output} \]
\[ \bar{T}_{\text{RD}} \text{ ZERO output} \]

NEGATIVE-TRIGGER

MONOSTABLE-MULTI

\[ T'_{0} \text{ trigger input (ONE} \rightarrow \text{ZERO)} \]
\[ T_{0} \text{ ONE output} \]

STORAGE GATE

\[ g' \text{ gate input} \]
\[ T_{\text{OC}} \text{ clock} \]
\[ g \text{ gate output (to flip-flop)} \]
4.0.1 Symbols used in diagrams and text

Symbols used in Fig. 4.0 follow:

\( T \)'s \): All subscripted \( T \)'s are timing waveforms and/or clock pulses

\( T_{RD} \): (Memory) X-Y Read timing waveform

\( T_{EP} \): Execute period timing waveform

\( T_D \): Delay timing waveform

\( T_{WR} \): (Memory) X-Y Write timing waveform

\( T_{PWD} \): Post-write-disturb timing waveform

\( T_{EX} \): Execute clock

\( T_{EM} \): End-of-memory-cycle

\( T_{SSS} \): Start-strobe-spacing timing waveform

\( T_O \): (Memory) Strobe

\( T_1, T_2, T_3 \): Clocks

\( J \)'s \): All subscripted \( J \)'s are control signals

\( J_{REW} \): Read-execute-write control signal

Additional symbols used in Fig. 4.2 follows:

\( J_{RO} \): Repeated operations control signal

\( J_T \): Transfer control signal

\( J_{IA} \): Instruction assembled control signal

\( R_i \) \((i = 0 \text{ to } 19)\): Flip-flop outputs of the instruction register (the R register)

\( R_{0-R_4} \): Operation code

\( \mu_i \) \((i = 0 \text{ to } 31)\): Decoded operation code signals

\( T_{CRT} \): Change phase after register transfer pulse

\( T_{DE} \): Delayed end of memory pulse

\( F \)'s \): All subscripted \( F \)'s are flip-flops

\( F_{ERO} \): End of repeated operations flip-flop

\( F_{EM} \): End of memory flip-flop

\( S \)'s \): All subscripted \( S \)'s are switches

\( S_{MCI} \): Memory cycle inhibit switch

\( T_{ST} \): Start pulse

\( F_{ASP} \): Abnormal stop flip-flop
FrSPS Stop start flip-flop
SBS Break start switch
TPG Pulse generator clock pulse
SPGS Pulse generator start switch
SBPC Break phase clock switch
\(\phi\)'s Subscripted \(\phi\)'s are phase flip-flops
\(\phi\) Phase
T\(\phi\)C Phase clock
T\(\phi\)S Start phase pulse
\(\phi\)SA Store using address selector control signal
\(\phi\)SD Store using distributor control signal
\(\phi\)F Fetch control signal
\(\phi\)AS Address selector settling timing waveform
\(\phi\)AS2 Second address selector settling timing waveform
TERO End of repeated operations pulse
TADS Address selector or distributor settling timing waveform
TDS Distributor settling timing waveform

Additional symbols used in Fig. 4.3 follow:

\(V_i\) \((i = 0--4)\) Flip-flop outputs of the number-of-times counter (the V register)

\(V=0\) Signal which senses condition \(V = 0\)

\(V\neq0\) Signal which senses condition \(V \neq 0\)

TOP Operate pulse of repeated operations loop

T\(125\) Pulse which follows 125 ns after TOP

G Control flip-flop whose main function is to
determine whether a T\(250\) or T\(500\) pulse follows TOP

T\(250\) Pulse which follows 250 ns after TOP

TD5 Delay pulse for T\(500\)

T\(500\) Pulse which follows 500 ns after TOP

Additional symbols used in Fig. 4.4 follow:

B 's All subscripted B 's are push-buttons
BIS Initial set push-button
BAI Auto-input push-button
\( \phi_{\text{FAI}} \) Auto-input phase
\( \phi_{\text{CS}} \) Clear store phase
\( \lambda_{\text{NMC}} \) No memory cycle (required for execution of instruction) control signal
\( \lambda_{\text{H}} \) Hierarchy etc. control signal (i.e. instruction requires more than one memory cycle for its execution)

\( F_{\text{EEB}} \) End of bootstrap (auto-input) flip-flop

\( U_{\hat{i}i} \) (i = 10 -- 19) Flip-flop outputs of the operand register (the U register)

\( \Pi U \) Signal which senses condition \( \Pi U_i = \text{CNE} \)

\( \phi_{\text{FFI}} \) Fetch instruction phase
\( \phi_{\text{E0}} \) Execution-0 phase
\( \phi_{\text{E1}} \) Execution-1 phase
\( \phi_{\text{E2}} \) Execution-2 phase
\( \phi_{\text{E3}} \) Execution-3 phase
\( \phi_{\text{PO}} \) Programmed operator phase
\( \phi_{\text{ID}} \) Indirect addressing phase
\( \phi_{\text{IX}} \) Indexing phase
\( \phi_{\text{EX}} \) Execute phase

\( \mu_{88-9} \) Either \( \mu_8 \) or \( \mu_9 \) signal

Additional symbols used in Fig. 4.6 follow:

\( \mu_{28-31} \) \( \mu_{28} \) to \( \mu_{31} \) (inclusive) signal (similarly for \( \mu_{12-15} \) and \( \mu_{16-23} \))

\( \mu_{26-7} \) Either \( \mu_{26} \) or \( \mu_{27} \) signal

\( A_{\hat{i}i} \) (i = 10 -- 19) Address selector outputs

\( S_{\hat{i}i} \) (i = 10 -- 19) Flip-flop outputs of the sequence counter (the S register)

\( A_{SS} \) Control signal which selects S for memory address

\( A_{UU} \) Control signal which selects U for memory address

\( A_{RR} \) Control signal which selects R for memory address

\( A_{SA\Delta} \) Control signal which restricts the selection of S for memory address to the period \( \Delta_{AS} \)

\( A_{UA\Delta} \) Control signal which restricts the selection of U for memory address to the period \( \Delta_{AS} \)
$A_{UA}^I$ Control signal which restricts the selection of $U$ for memory address to the period $\bar{A}_{AS}$

$A_{RA}^I$ Control signal which restricts the selection of $R$ for memory address to the period $\bar{A}_{AS}$

$Y_i$ ($i = 10 -- 19$) Control signals for specifying an absolute address (only $Y_{19}$ is used)

Additional symbols used in Table 4.0 follow:

- $a, s, r, k$: Control signals for selection of adder inputs (see Section 4.5.1)
- $a_1$: Control signal which selects adder output
- $C_{20}$: Carry digit into least significant stage of adder
- $F_{T2}$: Flip-flop for changing control signals at $T_2$ time
- $F$: Add/subtract flip-flop for multiplication and division
- $F_{EX}$: Flip-flop for changing control signals at $T_{EX}$ time
- $J_i$ ($i = 0 -- 19$): Flip-flop outputs of the memory input-output register (the $J$ register)

Additional symbols used in Fig. 4.7 follow:

- $K_i$ ($i = 0 -- 19$): Flip-flop outputs of the accumulator (the $K$ register)
- $\Pi K$: Signal which detects condition $\prod K_i = \text{ONE}$ (i.e. zero accumulator)
- $O/F$: Overflow signal (to be specified)
- $SS1, SS2, SS3$: Sense switches
- $\eta_{SCS}$: Skip condition satisfied signal

Additional symbols used in Fig. 4.8 follow:

- $A_i, B_i$ ($i = 0 -- 19$): Adder inputs
- $P_i$ ($i = 0 -- 19$): Propagate function
- $G_i$ ($i = 0 -- 19$): Generate function
- $P^I_n$ ($n = 1, 2, 3$): First-level generate function
- $C_i$ ($i = 0 -- 19$): Carry digit
$S_i \ (i = 0 -- 19) \text{ Sum digit}$

Additional symbols used in Fig. 4.9 follow:

- $T$ Terminate function
- $\beta$ Control signal equal to $\overline{k} + \overline{T}$
- $\alpha$ Control signal equal to $\overline{a} + \overline{kT}$

Additional symbols used in Fig. 4.10 follow:

- $T_i \ (i = 0 -- 19) \text{ Terminate function}$
- $D_i \ (i = 0 -- 19) \text{ Output of distributor-function-selector}$

Additional symbols used in Fig. 4.11 follow:

- $J_m$ Memory strobe pulse
- $J_{d1}$ Distributor to $J$ clock for stages 0 - 4
- $J_{d2}$ " " " " 5
- $J_{d3}$ " " " 6 - 19
- $J_{as}$ Address selector to $J$ clock
- $J_r$ Reset $J$ clock

Additional symbols used in Fig. 4.12 follow:

- $r_{in} \ (n = 1 -- 5) \text{ Buffer flip-flops of reader 1}$
- $K_{15}$ Fifth bit input clock
- $K_i$ Input clock
- $D_{-1}$ Digit shifted into $K_0$ during right shift
- $K_r$ Right shift $K$ clock
- $K_d$ Distributor to $K$ clock
- $K_l$ Left shift $K$ clock
- $D_{20}$ Digit shifted into $K_{19}$ during left shift

Additional symbols used in Fig. 4.13 follow:

- $F_{OP}$ Flip-flop set by $T_{OP}$
- $V_{4M}$ Special $V_4$ stage used for multiplication
- $L_{-1}$ Digit shifted into $L_0$ during right shift
- $L_{20}$ Digit shifted into $L_{19}$ during left shift
- $L_r$ Right shift $L$ clock
- $L_l$ Left shift $L$ clock
Additional symbols used in Fig. 4.14 follow:

The following three symbols are applicable only to the description of the binary counter.

- $G_n$: Binary counter gate signal
- $B_n$: Binary counter output
- $F$: Forward-backward control signal

Additional symbols used in Fig. 4.15 follow:

- $T_{sro}$: Start repeated operation pulse
- $F_{sro}$: Flip-flop set by $T_{sro}$

Additional symbols used in Fig. 4.16 follow:

- $T_5$: Clock triggered by $T_4$
- $V$: State of $V$ register

Additional symbols used in Table 4.1 follow:

- DR: Divisor
- DD: Dividend
- FR: First remainder

Additional symbols used in Fig. 4.17 follow:

These symbols are applicable only to this figure.

- $\alpha$: Inhibit brake signal
- SCR: Silicon Controlled Rectifier

Additional symbols used in Fig. 4.21 follow:

- $\Delta$: A delay
- W: Bit counter
4.1 APPROACH TO FUNCTIONAL AND LOGICAL DESIGN

Functional design consists of the specification in words, timing diagrams, tables or any other convenient form, of all the functions which are to be performed by the computer's hardware in every distinct time period of the computer's operation. Logical design consists of implementing the functional design using configurations of logical elements.

The approach which must be taken to the functional and logical design of a complex digital system like a computer is firstly to break up the system into a number of distinct sections with well-defined links between sections. The separate requirements and design of each section are co-ordinated and these evolve into a general framework for the whole system. Redesign of each section must then be carried out to fit in with the framework as much as possible.

Examples of sections of the computer for which a first design can be carried out independently of the remainder of the computer are the memory unit and the input-output unit. With the former, the main links with other sections would be in the form of the memory address signals, the START add END pulses of the memory cycle and the memory input-output register. With the latter, the links would be the information busses and the START and END pulses of the input-output cycle.

Examples of sections of the framework which carry out many functions include the REPEATED OPERATIONS LOOP and the CARRY-LOOKAHEAD-ADDER. The former is used to count the number of SHIFT operations, to count the number of characters read from tape and to terminate the multiplication and division processes. The latter is used not only for the arithmetic processes but for indexing instructions and for incrementing the contents of a memory location. The first design for the last mentioned operation required additional logic to enable the memory output register to operate as a parallel binary counter. After the decision to build a fast CARRY-LOOKAHEAD-ADDER was made, a more economical design was found.
The TIMING and CONTROL sections form a major part of the main framework of the computer. The design aim taken with the TIMING unit was to produce as fast a computer as possible. This meant that operations could not be tied rigidly to a clock common to all instructions as this inevitably would result in some wasted time periods. Instead, a minimum time was allowed for each elementary operation, and the computer's operation would consist of time periods during which only useful operations were carried out. The timing unit was realized using a system of gated monostable multivibrators. The system produced is still synchronous as a specific amount of time is allowed for each elementary operation but the system is similar to an asynchronous system as an END OF OPERATION PULSE is produced to trigger the next useful operation. The design aim taken with the CONTROL unit was to produce a computer with as much flexibility as possible. As the computer would be useful for teaching and research purposes, future modifications to the order code and other design changes were likely. To facilitate this, complete decoding of the OPERATION CODE and the EXECUTION PHASES was carried out and the control signals are formed by diode ENCODE MATRICES. Changes to these matrices can be readily carried out.

Design examples are presented in the following sections.
4.2 TIMING UNIT

4.2.0 Memory timing

A logic diagram of the memory timing unit is shown in Fig. 4.0. The unit consists essentially of a chain of gated monostable multivibrators. Two types of monostable multivibrators are used. The first type triggers off the front edge of signals (i.e., when the input changes from ZERO to ONE). With this type, both the normal and inverted outputs are available, and a timing chain is obtained by arranging each monostable multi to be triggered by the inverted output of the preceding stage. With this arrangement, the READ, DELAY, WRITE and POST-WRITE-DISTURB waveforms (shown in Fig. 4.1) which are necessary for memory timing may be obtained.

Alternative chains of timing waveforms may be obtained by gating the trigger signals of monostable multivibrators by control signals. An example of this is shown in Fig. 4.0. When the READ-EXECUTE-WRITE signal \( \text{REW} \) is a ONE, specifying a split memory cycle, an additional period, the EXECUTE PERIOD is inserted in the timing chain as shown in Fig. 4.1.

The second type of monostable multi triggers off the back edge of signals (i.e. when the input changes from ONE to ZERO) and is very convenient for constructing timing chains. An example of the application of this type of monostable multi is the \( T_0-T_1-T_2-T_3 \) timing chain of Fig. 4.0.

4.2.1 Main timing unit

From an examination of the timing requirements of each machine phase, it was decided that the main timing unit should be built on the framework of the memory timing unit. In all cases except one (the auto-input) this has resulted in a very satisfactory arrangement. For the auto-input (to be described later), the arrangement is still quite acceptable, although it is not ideal.

A logic diagram of the main timing unit is shown in Fig. 4.2. In this unit, the READ pulse \( T_{RD} \) and the POST-WRITE-DISTURB pulse \( T_{PWD} \) are the first and last pulses produced by the memory timing unit. At the end of \( T_{PWD} \), an END-OF-MEMORY pulse \( T_{EM} \) is produced. In many cases, this pulse initiates the next phase by firstly producing a PHASE-CLOCK pulse \( T_{PL} \) and a little later a START-PHASE pulse \( T_{SP} \).
The delay $\Delta_{AS2}$ allows the memory address circuits to settle before the memory cycle is initiated.

At the end of the memory cycle, clock pulses $T_{ADS}, T_{DS}$ and $T_{CRT}$ may be produced for the REGISTER TRANSFER instruction.

Before a memory cycle, an additional delay $\Delta_{AS}$ may be inserted if the ADDRESS-SELECTOR is used to transfer information to the MEMORY-INPUT-OUTPUT register (the J register) before it is changed to the memory address where the information is to be stored.

If the REPEATED OPERATIONS LOOP (described in the next section) is initiated (e.g. for a SHIFT instruction) the timing unit must wait for an END-OF-REPEATED-OPERATIONS pulse ($T_{ERO}$) before initiating the next phase. As shifting can occur simultaneously with memory regeneration, the initiation of the next phase must be inhibited until after both of the $T_{ERO}$ and $T_{EM}$ pulses have been produced. This is the function of the $FERO$ and $FEM$ flip-flops.

### 4.2.2 Repeated operations loop

A logic diagram of the REPEATED OPERATIONS LOOP is shown in Fig. 4.3. This circuit operates in conjunction with the five-stage V counter which is used to terminate the loop.

One of the applications of this circuit is to generate the clock pulses for the multiplication process. This process involves "multiplier-recoding" and is described in a later section. In this application, the circuit generates a specified number of pulses with each pulse following either 250 ns or 500 ns after the preceding one, depending on the setting of a control flip-flop $G$. The state of $G$ may change during the generation of pulses.

The circuit utilises monostable multis which trigger off the back edge of signals. As flip-flop start switching on the front edge of signals, a careful examination of the times allowed for circuit switching must be made.
G is clocked by the OPERATE pulse $T_{OP}$, and it is used to
gate $T_{125}$ pulses. The important point to note is that
the time which is allowed for the G flip-flop to settle
to its new state and to carry out the gating of $T_{125}$
successfully is the period of the $T_{OP}$ pulses (not $T_{125}$).
A similar comment applies to the V counter. This
counter is initially set to the number of pulses which is
to be generated. The counter is decremented by $T_{125}$
pulses and it is used to gate $T_{250}$ to $T_{500}$ pulses.
As the front edge of $T_{125}$ changes V to its new state
and $T_{250}$ is triggered by the back edge of $T_{125}$, it is
the period of $T_{125}$ which is allowed for the settling
of V and its associated circuits.
Figure 4.0 - LOGIC DIAGRAM OF MEMORY TIMING UNIT
Figure 4.2 - LOGIC DIAGRAM OF TIMING UNIT
Figure 4.3 - REPEATED OPERATIONS LOOP
4.3 MACHINE PHASES

4.3.0 Flow diagram

A flow diagram of machine phases is shown in Fig. 4.4. Every machine cycle begins with a FETCH INSTRUCTION phase ($0_{FI}$). During this phase the instruction is read from memory into the instruction register (the R register). If the INSTRUCTION ASSEMBLED signal ($\overline{2}_{IA}$) is ONE and the instruction does not require a memory cycle for its execution (i.e. $\overline{2}_{NMC} = ONE$), the machine will remain in $0_{FI}$ during the execution of the instruction and hence will be in this phase at the beginning of the next machine cycle. If the instruction does require a memory cycle for its execution (e.g. $\overline{2}_{NMC} = ZERO$ for an ADD instruction), the machine will enter the EXECUTION-0 phase ($0_{EO}$) from where it will return to $0_{FI}$ at the beginning of the next machine cycle.

If an instruction requires more than one memory cycle for its execution (i.e. if $\overline{2}_{H} = ONE$), the EXECUTION-1 phase ($0_{E1}$) will follow $0_{EO}$, and from $0_{E1}$ the machine proceeds to $0_{E2}$. If the instruction is a RETURN HIERARCHY instruction (i.e. if $\overline{\mu}_{30} = ONE$), the machine enters a further execution phase $0_{E3}$ from where it returns to $0_{FI}$. If $\overline{\mu}_{30} = ZERO$, the machine returns to $0_{FI}$ from $0_{E2}$.

The INSTRUCTION ASSEMBLED signal ($\overline{2}_{IA}$) may be ZERO for a number of reasons viz:

(i) the instruction uses INDIRECT ADDRESSING (i.e. $R_6 = ONE$)
(ii) the instruction specifies a PROGRAMMED OPERATOR (i.e. $R_5 = ONE$)
(iii) the instruction is an ADD-INDEX or SUBTRACT-INDEX instruction (i.e. $\overline{\mu}_{8-9} = ONE$)
(iv) the instruction is an EXECUTE instruction (i.e. $\overline{\mu}_{31} = ONE$)
(v) the machine is in the AUTO-INPUT phase (i.e. $0_{AI} = ONE$)
(vi) the machine is in the CLEAR STORE phase (i.e. $0_{CS} = ONE$)
The conditions (v) and (vi) above are included so that the AUTO-INPUT and CLEAR STORE logic may operate quite independently of the contents of the instruction register.

The remaining conditions (i) - (iv) are sensed in a definite sequence. Condition (i) has the highest priority and is sensed first; this is followed by condition (ii) and finally the mutually exclusive conditions (iii) or (iv).

If condition (i) is satisfied, the machine will enter the INDIRECT ADDRESSING phase in which a new address (including the INDIRECT ADDRESSING bit - (bit 6) - is read into the instruction register. If condition (i) is again satisfied, this procedure is repeated.

If condition (ii) is satisfied, the machine will enter the PROGRAMMED OPERATOR phase \(\phi_{PO}\) in which the incremented sequence counter is stored in location 0 and is then filled with the binary number represented by the operation code of the instruction plus thirty-two. From \(\phi_{PO}\) the machine returns to \(\phi_{FI}\).

If condition (iii) is satisfied, the machine will enter the INDEXING phase \(\phi_{IX}\) in which the index or negated index is read into the instruction register depending on whether the instruction was ADD-INDEX or SUBTRACT-INDEX respectively. A flip-flop is set so that the following instruction is added to the contents of the instruction register instead of being set into this register. From \(\phi_{IX}\) the machine returns to \(\phi_{FI}\).

If condition (iv) is satisfied, the machine will enter the EXECUTE phase \(\phi_{EX}\) in which a new instruction is read into the instruction register. Interpretation of this instruction is then carried out just as if it had been read from memory on a FETCH INSTRUCTION phase.
4.3.1 Implementation

A diagram of the phase logic is shown in Fig. 4.5. The eleven machine phases are represented by eleven flip-flops, only one of which can be in the ONE state at any one time. With only minor variations for $\varphi_{AI}$ and $\varphi_{CS}$, all flip-flops have transient storage gates in their setting (ONE-input) and resetting (ZERO-input) lines, and these are clocked by the PHASE CLOCK ($T\varphi_C$).

The transient storage gates require signals which determine the next phase into which the machine will proceed when the next PHASE CLOCK is produced. The signals are established at sometime in the current phase. For example during the FETCH INSTRUCTION phase the instruction is clocked into the instruction register (the R register) by the $T_2$ clock. Several levels of logic circuits are required to form the storage gate signals in terms of the outputs of the R flip-flops and other signals. Hence the time between the $T_2$ clock and the next PHASE-CLOCK (which is produced after the current memory cycle) is the time allowed for the above circuits to settle and to charge up the appropriate storage gates in readiness for the PHASE CLOCK.

Storage gates are used to advantage. For example the connection of the ONE-output to the ZERO-input line of a flip-flop ensures that the flip-flop cannot be in the ONE state for two consecutive memory cycles. This is the requirement for $\varphi_{D0}$, $\varphi_{IX}$, $\varphi_{EO}$, $\varphi_{E1}$, $\varphi_{E2}$ and $\varphi_{E3}$. The derivation of the ZERO-input signals for the $\varphi_{FI}$, $\varphi_{ID}$ and $\varphi_{EX}$ flip-flops using inverters enables these flip-flops to remain in their ONE states for any number of consecutive memory cycles.

The connection of the ONE-output of one flip-flop to the ONE-input of another flip-flop ensures that one phase is followed by another (e.g. $\varphi_{E1}$ is followed by $\varphi_{E2}$). The cascading of several flip-flops in this way produces a chain of consecutive phases. Such an arrangement is used for instructions which require more than one memory cycle for their execution.

The passage from one phase to two alternative phases depending on some machine condition can be easily arranged with simple gating.
For example the two AND gates with $\lambda_H$ and $\lambda_H$ as inputs ensure that after leaving $\phi_{EO}$, the machine enters $\phi_{FI}$ or $\phi_{EI}$ depending on whether $\lambda_H$ is ZERO or ONE respectively.

Conditions which can exist simultaneously can be sensed in a definite sequence by arranging each condition to be inhibited by all others of higher priority. For example the conditions $R_6 = \text{ONE}$, $R_5 = \text{ONE}$ and $\mu_{8-9} = \text{ONE}$ can all exist simultaneously. The condition $R_6 = \text{ONE}$ which requires the machine to enter the INDIRECT ADDRESSING phase ($\phi_{ID}$) has priority over the other two, and the condition $R_5 = \text{ONE}$ which requires the machine to enter the PROGRAMMED OPERATOR phase ($\phi_{PO}$) has priority over the last. This can be arranged by using the signals $R_6, R_6R_5$ and $R_6R_5\mu_{8-9}$ (together with signals necessary for other reasons) as setting inputs for the $\phi_{ID}$, $\phi_{PO}$ and the $\phi_{IX}$ flip-flops respectively.

An alternative arrangement for the phase logic which used only four phase flip-flops was considered, but was not used because it required complete decoding of the flip-flop outputs and the logic for changing the states of the flip-flops was quite complex. The arrangement used was found to be very flexible as it enabled quite a fundamental change to the phase sequencing to be carried out by a relatively small change in hardware. It was also found to be quite a powerful method for phase sequencing as many complex sequences are possible, and yet the logic is relatively simple to understand and to implement.
Figure 4.4 - FLOW DIAGRAM OF MACHINE PHASES
4.4 CONTROL

4.4.0 Arithmetic unit control

The control signals of the arithmetic unit are functions of the PHASE signals (φ's), the OPERATION CODE section of the instruction register (R₀ - R₄) and other machine signals. Not all the thirty-two signals μ₀ - μ₃₁ are produced by a complete decoding of R₀ - R₄; instead, most are combined with some of the φ's to produce signals which specify every distinct machine phase. For example the signal μ₁₅φ₁₀ is produced to specify the EXECUTION-0 phase of the ADD instruction. Signals such as μ₁₆-₂₃ φ₁₀ are also produced to specify the EXECUTION-0 phase of a group of instructions.

With every distinct machine phase specified by the μφ signals, machine control signals may be formed by ENCODE circuits. Tables, like Table 4.0, in which the ONES indicate the phases in which the control signals (a₁, a₂, a₃ etc.) are equal to ONE, can be used to specify these circuits. Entries such as F or F indicate additional restraints and hence represent AND gates. Hence Table 4.0 represents a circuit with at most two levels of logic.

The generation of the control signals for the REGISTER TRANSFER instruction requires some explanation. At the end of the FETCH INSTRUCTION memory cycle of the REGISTER TRANSFER instruction, the φ₁ flip-flop is reset and additional timing waveforms TADS, TDS and TERT are produced. As all phase flip-flops are in their ZERO-states all the φ and μφ signals are ineffective, and hence control signals and clocks may be defined in terms of the above timing waveforms. This is the reason for the TDS entry in the last row of Table 4.0.

4.4.1 Memory control

Control signals are required to specify the different types of memory cycles. For example μ₇ specifies a FETCH cycle, μ₁₀ a STORE-USING-DISTRIBUTOR cycle, μ₁₃ a STORE-USING-ADDRESS-SELECTOR cycle and μ₂₀ a (split) READ-EXECUTE-WRITE cycle. As these signals are functions of the φ and μφ signals, they are produced in the same way as the control signals of the arithmetic unit.
The memory address may be specified by the SEQUENCE COUNTER (the S register), the OPERAND COUNTER (the U register) or the address section of the INSTRUCTION REGISTER (the R register). The ADDRESS SELECTOR shown in Fig. 4.6 determines this address.

During a STORE-USING-ADDRESS-SELECTOR cycle, the ADDRESS SELECTOR is used to transfer information to the MEMORY-INPUT-OUTPUT register (the J register) before it is used to specify the memory address where this information is to be stored. The timing signal which determines which of these two functions the ADDRESS SELECTOR has during the cycle is $\Delta_{AS}$ (see Fig. 4.2). A general expression for the outputs of the ADDRESS SELECTOR would then be:

$$A_n = (f_1\Delta_{AS} + f_2\bar{\Delta}_{AS}) S_n + (f_3\Delta_{AS} + f_4\bar{\Delta}_{AS}) U_n + (f_5\Delta_{AS} + f_6\bar{\Delta}_{AS}) R_n + Y_n$$

In the above equation $f_1 - f_6$ are functions of the $\phi$ and $\mu\phi$ signals. It is to be noted that a direct implementation of the above equation would require the $\Delta_{AS}$ (and $\bar{\Delta}_{AS}$) signals to pass through four levels of logic before reaching the output. A slight improvement in circuit configuration can be obtained by utilising the fact that in the machine design, the conditions $f_1 = \text{ONE}$, $f_2 = \text{ONE}$ are mutually exclusive, and this is also so for the pairs of conditions $f_3 = \text{ONE}$, $f_4 = \text{ONE}$ and $f_5 = \text{ONE}$, $f_6 = \text{ONE}$. An equivalent expression for $A_n$ may now be derived:

$$A_n = (f_1 + f_2)(\bar{f}_1 + \Delta_{AS})(\bar{f}_2 + \bar{\Delta}_{AS}) S_n + (f_3 + f_4)(\bar{f}_3 + \Delta_{AS})(\bar{f}_4 + \bar{\Delta}_{AS}) U_n + (f_5 + f_6)(\bar{f}_5 + \Delta_{AS})(\bar{f}_6 + \bar{\Delta}_{AS}) R_n + Y_n$$

It is now to be noted that the $\Delta_{AS}$ (and $\bar{\Delta}_{AS}$) signals must pass through only three levels of logic before reaching the output. The inverters necessary for the generation of the inverse functions $\bar{f}_1$ etc. do not come into consideration here, as the $\mu\phi$ and $f$ signals are assumed to settle well before the $\Delta_{AS}$ signal is produced.
The circuit configuration suggested by the last equation is only slightly superior to the one suggested by the first $A_n$ equation as there are other much larger delays associated with the memory address circuits. However only a slight increase in hardware was involved and hence the slightly superior configuration for the ADDRESS SELECTOR was used. This is shown in Fig.4.6.

4.4.2 Miscellaneous control signals

The computer utilises a large number of miscellaneous control signals. Examples of these are signals which determine when binary counters are to be pulsed, signals which determine whether a counter runs forwards or backwards, signals which determine whether a clock pulse is to be produced and a signal which determines when the peripheral units are ready.

A logic diagram showing how one of these signals is generated is shown in Fig. 4.7. The SKIP CONDITIONS SATISFIED signal ($\text{SCS}$) is used by the SKIP instruction logic to determine whether or not the SEQUENCE COUNTER is to be incremented. Bits 7-19 of the SKIP instruction do not specify an operand address but rather the machine conditions which are to be sensed. Complete decoding of these bits would produce an enormous number of possible conditions, but of course is out of the question because of the amount of hardware involved. As in many engineering designs a compromise is made, and decoding for pairs of bits is provided to increase the number of conditions sensed without an appreciable increase in hardware. The problems of circuit fan-in is solved by using a NOR-NAND configuration with the outputs of the NOR gates represented in negative logic. As these outputs are transmitted some distance to another package via the base-wiring, the signal regenerative property of the NOR gate is used to advantage.
### TABLE 4.0

**SPECIFICATION OF SOME CONTROL SIGNALS**

<table>
<thead>
<tr>
<th>( \psi \phi )</th>
<th>( a_1 )</th>
<th>( a )</th>
<th>( s )</th>
<th>( C_{20} )</th>
<th>( r )</th>
<th>( k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi_{FI} )</td>
<td>( \bar{F}_{T2} )</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \phi_{ID} )</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \phi_{IX} )</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_8 \phi_{IX} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_9 \phi_{IX} )</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \phi_{EX} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{10} \phi_{EO} )</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{11} \phi_{EO} )</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{12} \phi_{EO} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( \mu_{15} \phi_{EO} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( \mu_{12-15} \phi_{EO} )</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{16} \phi_{EO} )</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{17} \phi_{EO} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( \mu_{18-19} \phi_{EO} )</td>
<td>( G )</td>
<td>( \bar{F} )</td>
<td>( \bar{F} )</td>
<td>( \bar{F} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{20} \phi_{EO} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{22} \phi_{EO} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{23} \phi_{EO} )</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{16-23} \phi_{EO} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( \mu_{26-27} \phi_{EO} )</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{28-31} \phi_{EO} )</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \mu_{27} \phi_{E1} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( \mu_{26} \phi_{E2} )</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( \mu_{27} \phi_{E2} )</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( \mu_{28} \phi_{E2} )</td>
<td></td>
<td></td>
<td>( F_{EX} )</td>
<td></td>
<td>1</td>
<td>( F_{EX} )</td>
</tr>
<tr>
<td>( \mu_{29} \phi_{E2} )</td>
<td></td>
<td></td>
<td>( F_{EX} )</td>
<td>( F_{EX} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \phi_{E3} )</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_{DS} )</td>
<td></td>
<td>( R_g + R_{12} J_0 )</td>
<td>( R_{10} + R_{12} J_0 )</td>
<td>( R + R J_0 )</td>
<td></td>
<td>( R_{13} ) ( R_{14} )</td>
</tr>
</tbody>
</table>
Figure 4.6 - ADDRESS SELECTOR LOGIC
Figure 4.7 - SKIP LOGIC

SENSE SWITCH /SSI / SS2 / SS3

OTHER SKIP CONDITIONS TO BE SPECIFIED

OFF

ON

Figure 4.7 - SKIP LOGIC
4.5 CARRY LOOKAHEAD ADDER

4.5.0 Brief description

A discussion of the CARRY LOOKAHEAD principle and a more detailed description of the adder used in ARCTURUS appears in the Appendix L. A brief description follows.

A block diagram of the adder is shown in Fig. 4.8. The adder stages are numbered 0 to 19, with stage 0 being the most significant and stage 19 the least significant. The adder inputs are \( A_i \) and \( B_i \) (\( i = 0 \ldots 19 \)), and the carry digit into the least-significant stage is designated \( C_{20} \). The SUM and CARRY digits are designated \( S_i \) and \( C_i \) respectively.

PROPAGATE and GENERATE functions (\( P_i \) and \( G_i \)) are first formed for each stage. The adder is then divided into four groups each of 5 stages; and first-level PROPAGATE and GENERATE functions are formed for the three least significant groups. These functions are designated \( P_n \) and \( G_n \) (\( n = 1, 2, 3 \)) with \( n = 3 \) corresponding to the least significant group.

By using FIRST-LEVEL-LOOKAHEAD, i.e. by applying the lookahead principle to groups of stages, the CARRY digits \( C_{15} \), \( C_{10} \) and \( C_5 \) may be formed. Then with the CARRY digits \( C_{20} \), \( C_{15} \), \( C_{10} \) and \( C_5 \) either known or established, the remaining CARRY signals may be formed using ZERO-LEVEL-LOOKAHEAD. Finally, the SUM digits are formed in terms of the CARRY digits and PROPAGATE functions.

The advantage of the CARRY-LOOKAHEAD-ADDER over the conventional RIPPLE-CARRY-ADDER is that in the latter, the CARRY and SUM digits of each full-adder stage are formed in succession as CARRY digits of less significant stages ripple through to more significant stages, whereas in the former, the CARRY and SUM digits of almost all stages are formed simultaneously. Hence, the CARRY-LOOKAHEAD-ADDER is much faster but requires more hardware.

4.5.1 Selection of adder inputs

The adder is used for carrying out arithmetic operations on instructions (e.g. INDEXING instructions) as well as on numbers held in machine registers.
A large number of different operations (such as CLEAR, SUBTRACT, DECREMENT, etc.) are required. Both these requirements may be satisfied by the appropriate selection of adder inputs.

Control signals, \(a, s, r\) and \(k\) are used to select \(J, \overline{J}, 0\) or 1 as the first adder input and \(R, \overline{K}, 0\) or 1 as the second adder input. The defining equations are

\[
A = aJ + s\overline{J} \\
B = rR + kK + rk
\]

Different combinations of the control signals \(a, s, r, k\) and \(C_{20}\) produce many useful functions as the adder output. For example, the combination ZERO, ONE, ZERO, ONE, ONE respectively for these signals produces the function \(K - J\) which is required by the SUBTRACT instruction; the combination ONE, ZERO, ZERO, ZERO, ONE produces the function \(J + 2^{-19}\) which is required by the INCREMENT SKIP instruction; and the combination ZERO, ZERO, ONE, ZERO, ONE produces the function \(R + 2^{-19}\) which is required by the JUMP TO SUB-Routine instruction. A table containing a list of these functions would have been produced by the designer soon after the adder inputs were defined, and the machine design was carried out to take full advantage of the flexibility and speed of the adder.

4.5.2 Alternative configurations

A number of logic circuit configurations for the adder were considered; alternative configurations for obtaining the PROPAGATE and GENERATE functions are shown in Fig. 4.9.

Configuration I represents the most direct approach. The adder inputs \(A\) and \(B\) are formed; PROPAGATE (P) and GENERATE (G) signals are then formed in terms of the adder inputs, and are completely regenerated using double invert circuits.

Configuration II is a simplification of configuration I but allows an AND-OR circuit for forming \(P\).

Configuration III is derived from the Boolean equations for the inverses of the TERMINATE (T) and GENERATE (G) functions written in SUM-OF-PRODUCTS form.
This configuration was chosen because the number of cascaded INVERT circuits was smaller than the number in other configurations. Hence the use of Configuration III would result in a faster adder. The smaller number of INVERT circuits would also compensate for the increase in the number of AND gates over that of Configuration II.

A slight relaxation of the speed specification of the computer would possibly make Configuration II very attractive. The selection of adder inputs is more straightforward and hence more flexible than in Configuration III. If Configuration II had been chosen, a wider choice of adder inputs might have been provided.

4.5.3 Final adder configuration

A skeletal logic diagram of the adder configuration is shown in Fig. 4.10. This diagram is intended to give some indication of the signal attenuation and time delays which are involved.

Time delays introduced by INVERT circuits are much greater than those introduced by AND or OR circuits. However, INVERT circuits do regenerate signals while diode AND and OR gates cause signal deterioration in the form of attenuation and level-shift.

Circuit tests showed that the configuration AND-OR-EF-AND-OR-EF (where EF represents an emitter follower) was quite satisfactory, but that further cascading of logic circuits without regeneration was unwise.

Hence the aim was to obtain a circuit arrangement for obtaining the SUM digits with the minimum number of INVERT circuits in cascade and with the restriction that the AND-OR-EF-AND-OR-EF-circuit configuration is the longest chain of logic circuits through which signals must pass before they are regenerated.

The choice of the circuit configuration for the P and G functions was discussed in the last section. This is followed (in Fig. 4.10) by configurations for the FIRST-LEVEL PROPAGATE and GENERATE functions \( P_n^1 \) and \( C_n^1 \), the FIRST-LEVEL-LOOKAHEAD circuits which produce \( C_{15} \), \( C_{10} \) and \( C_5 \), and by the ZERO-LEVEL-LOOKAHEAD circuits which produce the remaining CARRY signals.
The SUM digits which are functions of the CARRY signals and the PROPAGATE functions may be selected by the DISTRIBUTOR-FUNCTION-SELECTOR (DFS) and distributed (i.e. transmitted) to all machine registers.

From Fig. 4.10, it can be seen that from the effective selection of adder inputs to the complete regeneration of the DFS output, the maximum number of INVERT circuits in cascade is seven. Allowing a delay of 40 ns for an AND-OR-EF-INVERT-EF circuit, a reasonable estimate for the total delay of the adder circuits would be 280 ns. Allowing an additional 200 ns for charging transient storage gates, an estimate for the "accumulate time" (i.e. the time required to add the contents of J to K and to place the answer in K) would be about $\frac{1}{2} \mu s$. This figure was achieved in practice.
Figure 4.8 - BLOCK DIAGRAM OF CARRY-LOOKAHEAD-ADDER
Figure 4.10 - SKELETAL LOGIC DIAGRAM OF ADDER
4.6  MACHINE REGISTERS

4.6.0  Distributor Function Selector

The DISTRIBUTOR FUNCTION SELECTOR (DFS) was introduced in the last section. Its function is to select arithmetic and logical functions of register outputs and to distribute these functions to all registers.

The DFS may select the ADDER output in which case the nominal time of $\frac{1}{2} \mu s$ is allowed. However the ADDER may be by-passed (as required by the high-speed multiplication process) in which case the DFS selects a register output and the nominal time of $\frac{1}{4} \mu s$ is allowed.

The DFS output is fully regenerated and is used to feed all types of logical elements including transient storage gates, AND gates, R-S flip-flops and binary counters.

4.6.1  Memory Input Output Register

A logic diagram of the MEMORY INPUT OUTPUT REGISTER (the J register) is shown in Fig. 4.11.

A RESET-SET cycle using the CLOCKS $J_r$ and $J_m$ is used for sensing the memory output.

Two pairs of transient storage gates are connected to the inputs of each stage of the register. To these gates are connected the outputs of the DISTRIBUTOR FUNCTION SELECTOR and the ADDRESS SELECTOR. In this way the contents of all the main machine registers, as well as arithmetic and logical functions of these contents may be set into J and subsequently stored in the main memory.

An earlier machine configuration required J to be a very powerful register which could count and shift in either direction and which had the peripheral units connected to it. After the high-speed ADDER was designed, it was decided to use this unit for incrementing and decrementing. The design of the other arithmetic sections of the computer suggested that it was better to carry out shifting in the main arithmetic registers (K and L); this then led to the decision to connect the peripheral units to the K register.
Figure 4.11 - LOGIC DIAGRAM OF THE J REGISTER
4.6.2 Main Arithmetic Registers

The main arithmetic registers to which programmers have access are the ACCUMULATOR (the K register) and the MULTIPLIER-QUOTIENT-REGISTER (the L register). The OPERAND-REGISTER (the U register) is also available for limited use.

A logic diagram of the K register is shown in Fig. 4.12. Three pairs of transient storage gates are connected to each stage of the K register. To these gates are connected the output of the corresponding DISTRIBUTOR-FUNCTION-SELECTOR (DFS) and the outputs of the two adjacent DFS's. In this way, all the arithmetic and logical functions produced by the DFS may be set into K in one of three positions: the UNSHIFTED position, the LEFT SHIFT position and the RIGHT SHIFT position. This is controlled by three separate CLOCK pulses. Hence the ADD-SHIFT or SUBTRACT-SHIFT operations required by the multiplication and division procedures may be carried out by the generation of a single CLOCK pulse.

When the ADDER is by-passed, both the K and L registers may shift (in either direction) at the rate of 4 MHz. All the variants of the SHIFT instruction and all the SHIFT-modes required by the multiplication, division and input-output procedures may be obtained by supplying the appropriate signals to the transient storage gates at the extremities of the K and L registers and by supplying the appropriate CLOCK. The former section of the SHIFT logic is shown in Fig. 4.13.

A relatively simple modification to the logic represented by Fig. 4.13 was carried out quite late in the project to incorporate an ILLOGICAL SHIFT variant which was suggested by programming experience. This variant shifts ONES (compared with ZEROS for the LOGICAL SHIFT) into the extremities of the registers, and has proved to be very useful on a number of occasions.

The logic diagram of the K register (Fig. 4.12) shows AND gates connected to the ONE-sides of flip-flops. These are for the connection of the input buffers to the K register. A LOGICAL SHIFT operation, which clears the appropriate stages of K, precedes a pulse which clocks the buffers into K. The outputs of the least
Figure 4.12 - LOGIC DIAGRAM OF THE K REGISTER
Figure 4.13 - SHIFT LOGIC
4.6.3 Counters

Parallel binary counters are used for the ten-stage SEQUENCE COUNTER (the S register), the ten-stage OPERAND COUNTER (the U register), the five-stage NUMBER-OF-TIMES-COUNTER (the V counter) and the three-stage BIT COUNTER (the W counter). The SEQUENCE COUNTER runs forwards; the OPERAND COUNTER is reversible and the remaining two run backwards.

In the design of the counters, it was decided that the same types of packages should be used in all cases, and that the only difference should be in their method of interconnection. The logic circuit configuration for a parallel, reversible binary counter is shown in Fig. 4.14. The COMPLEMENTER is omitted if the counter is not required to be reversible, and the ONE-outputs or ZERO-outputs of the BINARY COUNTERS are connected to the NOR gates of the COUNTER GATING depending on whether the counter is to run backwards or forwards respectively.

The counter configuration of Fig. 4.14 (of which a slightly more detailed description appears in Appendix L) was found to be very flexible and produced compact standard packages. The complete circuit easily satisfied the design requirement of reliable operation at 4 MHz.
Figure 4.14 - Logic Diagram of Parallel, Reversible Binary Counter
4.7 MULTIPLICATION PROCEDURE

4.7.0 General Principles

Extremely fast multipliers using only combinational circuits have been proposed (Ref. 45), but are not common because of the amount of hardware involved. Most multiplication procedures involve a sequence of ADDITIONS and SHIFTS in which the product of the numbers is formed as the accumulated sum of a number of partial products.

A common technique is to utilise a double-length ACCUMULATED-MQ REGISTER configuration which holds the growing ACCUMULATED PARTIAL PRODUCT and the diminishing MULTIPLIER. The MULTIPLIER digits are sensed one at a time, and used to determine whether or not the MULTIPLICAND is to be added to the ACCUMULATED PARTIAL PRODUCT. After this the double-length register is shifted, and the procedure repeated until all MULTIPLIER digits have been sensed.

The arrangement of the inputs to the K register in ARCTURUS (described in Section 4.6) enables the ADD-SHIFT and PASS-SHIFT operations to be carried out by a single CLOCK pulse. This speeds up the multiplication procedure.

In most arithmetic units the SHIFT operation requires less time than the ADD operation. Hence the multiplication procedure may be again speeded up by "by-passing" the adder and consequently allowing less time for the (PASS)-SHIFT operation.

The number of SHIFT operations may be increased and the number of ADD operations reduced by MULTIPLIER RECODING. This produces a further speeding up of the multiplication procedure.

The procedure used in ARCTURUS is to sense MULTIPLIER digits two at a time. These are used to determine which of three operations are to be performed during each stage of the multiplication procedure. The three possible operations are (i) ADD-SHIFT (ii) SUBTRACT-SHIFT and (iii) PASS-SHIFT. The last of these does not require the use of the ADDER, and hence if adder-bypassing is available, this operation requires less time than either of the other two.
As the procedure allows three different operations rather than two, it corresponds to a recoding of the MULTIPLIER from a BINARY to TERNARY system. This recoding is carried out so that (as far as possible) the number of PASS-SHIFT operations is maximised. This results in a significant reduction of the average multiplication time.

4.7.1 Implementation

The implementation of the multiplier recoding procedure in ARCTURUS is fully described in Appendix L. A very brief summary follows.

The multiplication logic makes use of the REPEATED OPERATIONS LOOP (Fig. 4.3). Typical timing waveforms are shown in Fig. 4.15.

When the REPEATED OPERATIONS LOOP is started by a \( T_{SRO} \) pulse, it generates several pulse trains. Passage through the loop always produces a \( T_{OP} \) pulse, while the generation of a \( T_{250} \) or \( T_{500} \) pulse (250 ns or 500 ns respectively after \( T_{OP} \)) depends on the setting of a control flip-flop (G). Appropriate delays are introduced by the \( T_{125} \) and \( T_{D5} \) pulses.

The CLOCK pulses required by the multiplication procedure (e.g. \( L_r \), \( K_r \) and \( L_d \)) may be generated from the pulses produced by the REPEATED OPERATIONS LOOP and from various control signals.

Some of the problems involved in the design of the multiplication logic were:

(i) the critical timing of the CONTROL signals and CLOCK pulses required by the MULTIPLIER RECODING and ADDER BY-PASSING operations

(ii) the correct treatment of positive and negative operands

and

(iii) the correct positioning of the result with the specified DOUBLE LENGTH NUMBER format.

All of these problems were of course solved; and the design has been verified by tests and operating experience.
Figure 4.15 - Timing Waveforms for Multiplication
4.7.2 Multiplication time

The nominal times allowed for an ADD/SUBTRACT-SHIFT operation and a PASS-SHIFT operation are 500 ns and 250 ns respectively. The multiplication time depends on the multiplier and varies between the time required for twenty PASS-SHIFT operations (i.e. 5 μs) and the time required for ten PASS-SHIFT operations plus ten ADD/SUBTRACT-SHIFT operations (i.e. 7.5 μs).

The above times exclude one memory cycle time required to fetch the instruction and one memory access time required to fetch the multiplicand. The multiplication process may proceed while the multiplicand is being regenerated in the memory.
4.8 DIVISION PROCEDURE

4.8.0 General Principles

Many early computers and even some second
generation computers such as the IBM 1620 (Ref. 46 )
required a program to carry out division. The
incorporation of the division function into the
hardware of the computer can be carried out in a
number of ways (Ref. 47 ). NON-RESTORING procedures
are invariably faster than RESTORING procedures
and hence they are used unless the register configuration
or other reasons strongly favour some other procedure.
In a parallel, binary computer, the basic requirements
for hardware division are a double-length ACCUMULATOR-MQ
REGISTER which can shift left (i.e. in the opposite
direction to that required for multiplication) and a
control unit which enables the DIVISOR to be added to
or substracted from the REMAINDER held in the ACCUMULATOR.

A number of methods (Ref. 48 ) have been used
for speeding up the division procedure in large-scale,
scientific computers. Some methods (Ref. 49 ) are
analogous to those used for multiplier recoding in that
they utilise adder bypassing; however these methods assume
normalised operands and hence are only applicable to
floating-point arithmetic units.

A high-speed procedure for division is not as
important as it is for multiplication as most programs
do not use division instructions as often as multiplication
instructions.

Although some attention was given to the possibility
of incorporating a high-speed division unit into the
ARCTURUS structure, it was finally decided to use a
conventional NON-RESTORING technique.

4.8.1 Implementation

A very brief outline of the implementation of
the division procedure in ARCTURUS follows.
This procedure makes use of the REPEATED OPERATIONS LOOP
(Fig. 4.3); timing waveforms are shown in Fig. 4.16.
For division, the G flip-flop is always in the ZERO state; hence the REPEATED OPERATIONS LOOP produces a train of equally spaced pulses with a pulse spacing of 500 ns. These are used to generate the CLOCK pulses for the K - L registers and for the F flip-flop. Except for the final CLOCK pulse for K, all CLOCK pulses effectively shift the new contents of the double-length K - L register one place to the left. Hence each pair of ADD/SUBTRACT and SHIFT operations required by the NON-RESTORING division procedure is carried out by a single CLOCK pulse. The F flip-flop determines whether the DIVISOR is added to or subtracted from the REMAINDER held in the K register.

Some of the problems involved in the design of the division logic were:

(i) the detection of an incorrect result (by the DIVISION HANG-UP logic)
(ii) the correct timing of CONTROL signals and CLOCK pulses
and
(iii) the correct positioning of REMAINDER and QUOTIENT in the K - L registers.

The DIVISION HANG-UP conditions warrant further comment. These are shown in Table 4.1. The fourth column of this table shows whether the NON-RESTORING division procedure produces the correct result for different relative magnitudes and different signs of the DIVISOR and DIVIDEND. For the detection of an incorrect result, three machine conditions were considered. These were:

(i) DIVIDEND and FIRST REMAINDER have same sign (column 5)
(ii) DIVISOR and DIVIDEND are equal (column 6)
(iii) FIRST REMAINDER is zero (column 7)

The entries in each of the columns 5, 6 or 7 do not correspond exactly to those in column 4. A composite condition consisting of either condition (i) or condition (iii) (above) was considered preferable to any one condition alone. This composite condition is represented by the last column of the table. It shows that the DIVISION HANG-UP condition is satisfied (with the consequent stopping of the computer) if the absolute
value of the DIVISOR is less than or equal to the absolute value of the DIVIDEND. The logic configuration of that part of the computer required for division requires that the two conditions (i) and (iii) above be sensed at two different times. This is shown in Fig. 4.16. The latter condition makes use of the $\overline{WR}$ gate which is used for sensing when the ACCUMULATOR is ZERO. It was found that only a small amount of additional logic was necessary for the DIVISION HANG-UP logic but extreme care was necessary for the interpretation of machine signals at various stages of the division procedure.

4.8.2 Division time

The nominal time allowed for an ADD/SUBTRACT-SHIFT operation is 500 ns. The division procedure requires twenty of these operations. Hence the division time is 10 $\mu$s. This time excludes one memory cycle time required to fetch the instruction and one memory access time required to fetch the divisor. The division procedure may proceed while the divisor is being regenerated in the memory.
CLOCK DIVISOR INTO J
RESET F

T_0

SENSE SIGNS OF DIVIDEND & 1st REMAINDER: 1st K-L CLOCK: CLOCK F

T_1

T_2

T_3

SENSE 1st REMAINDER = 0: 2nd K-L CLOCK: CLOCK F

T_4

T_5 : T_{SRQ}

T_{TOP}

T_{125}

T_{500}

T_{ERO}

19th K-L CLOCK: CLOCK F

K_d CLOCK: 20th L CLOCK

END PULSE

V | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0

125 ns INTERVALS

Figure 4.16 - TIMING WAVEFORMS FOR DIVISION
<table>
<thead>
<tr>
<th>RELATIVE MAGNITUDES</th>
<th>SIGNS DR DD</th>
<th>INCORRECT RESULT</th>
<th>SAME SIGNS DD &amp; FR</th>
<th>EQUAL DR &amp; DD</th>
<th>ZERO FR</th>
<th>DIVISION HANGUP: SAME SIGNS DD &amp; FR OR ZERO FR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+ +</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>- +</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>- -</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>DR &gt; DD</td>
<td>+ +</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>- +</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>- -</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>DR &lt; DD</td>
<td>+ +</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>- +</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>- -</td>
<td>YES (CLOSE)</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>DR = DD</td>
<td>+ +</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>- +</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>- -</td>
<td>YES (CLOSE)</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
4.9 PROGRAMMED OPERATORS

4.9.0 Implementation

The PROGRAMMED OPERATOR feature of ARCTURUS was described in section 3.2.6. A brief description of the hardware necessary for the feature follows.

At the end of a FETCH INSTRUCTION phase, the sequence counter is incremented, and the (incremented) sequence counter is then stored in location 0 during the PROGRAMMED OPERATOR phase. This is carried out by firstly transferring the sequence counter via the ADDRESS SELECTOR to the memory input-output register (the J register), and then initiating a STORE cycle in which the ADDRESS SELECTOR outputs are all ZERO. This last condition is readily obtained by ensuring that all the control signals associated with the ADDRESS SELECTOR are ZERO.

Transferring control to the address determined by the OPERATION CODE of the instruction plus 32 is carried out by SELECTION CIRCUITS connected to the setting lines of the sequence counter (the S register). During the PROGRAMMED OPERATOR phase, these SELECTION CIRCUITS set \( R_0 - R_4 \) into \( S_{15} - S_{19} \); \( S_{14} \) is forced to ONE and all other stages of S are forced to ZERO.

Setting the effective operand address into the operand register (the U register) is readily accomplished by clocking the address portion of the instruction register (the R register) via the DISTRIBUTOR into the U register during the PROGRAMMED OPERATOR phase.

4.9.1 Extensions

The amount of hardware required for the PROGRAMMED-OPERATOR feature (in relation to its usefulness) is very small. In particular, the extension of the feature in ARCTURUS which enabled indirect addressing to be used with PROGRAMMED-OPERATOR routines which referred to multi-word-length operands required an insignificant amount of hardware, as the only operation involved was the setting of the effective operand address into the U register.
A further improvement, which incidentally is utilised in later Scientific Data Systems computers (Ref. 60), is to use a hardware register instead of location 0 to hold the return link. In terms of the ARCTURUS structure, this would completely eliminate the PROGRAMMED OPERATOR phase and of course speed up the computer.

An even better arrangement would be one which allowed lists of return links to be stored either in flip-flop registers or some (fast) scratch-pad memory. This arrangement would allow PROGRAMMED OPERATOR routines to be nested without the necessity to store the return links within the routines. With costs of computer components (including scratch-pad memories) continuously decreasing, this arrangement may become quite attractive.
4.10 PERIPHERAL UNITS

4.10.0 Block Diagrams

Block diagrams of the logic required for the reader, printer and punch are shown in Figs. 4.17, 4.19 and 4.18 respectively. Brief descriptions follow.

The reader logic has been designed to assemble the last character read from tape into the flip-flop BUFFER where it is ready for use by the computer. Having sensed the state of the BUFFER, the computer produces a START pulse. This resets the READY flip-flop and puts the CLUTCH-BRAKE flip-flop into the state which removes the brake and applies power to the clutch. As the tape moves forward, the amplified LOCATION PHOTO-CELL output is used to clock the BUFFER, to set the READY flip-flop and (normally) to apply the brake. Improved performance can be obtained (Ref. 7) by requiring the computer to produce an INHIBIT BRAKE SIGNAL (α) when it requires and can handle the information assembled into the BUFFER at the maximum rate of the reader.

The CLOCK & START pulse of the punch logic (Fig. 4.18) is used firstly to clock the signal lines representing the CHARACTER TO BE PUNCHED into the flip-flop BUFFER, and secondly to start the punch cycle. The READY flip-flop is reset by the CLOCK & START pulse and set at the end of the timing waveform produced by the TIMING MULTI. The SYNCHRONISING flip-flop is used to trigger the TIMING MULTI at the correct instant in the punch cycle. Under normal conditions, the outputs of the BUFFER are gated by the timing waveform (produced by the TIMING MULTI), and the gated outputs are used by the POWER DRIVERS for the FEED & DIGIT SOLENOIDS. When the RUN-OUT button is depressed, a pre-determined character (e.g. a DELAY character for 5-channel tape or a DELETE character for 8-channel tape) is punched continuously. The TEST SWITCH represents hardware for carrying out simple tests on the punch and its associated logic.
The printer logic of Fig. 4.19 assumes that the computer produces a CLOCK BUFFER pulse before a START pulse. The states of the BUFFER which represent the CHARACTER TO BE PRINTED are decoded into 32 lines by 32 x 5-input NOR gates. These lines are then gated by TIMING WAVEFORMS which specify either a FIGURE-SHIFT character, a LETTER-SHIFT character or a character common to both shifts. The outputs of the SHIFT GATING are then used by the POWER DRIVERS of the SOLENOIDS (and relays) attached to the IBM ELECTRIC TYPEWRITER. FIGURE-SHIFT/LETTER-SHIFT operation is provided (electronically) by the use of the SHIFT flip-flop, and UPPER-CASE/LOWER-CASE operation is provided (mechanically) by the use of the mechanical lock on the typewriter together with a sensing micro-switch. The function of the START INHIBIT logic is to inhibit the START pulse if the BUFFER contains an UPPER-CASE character or a LOWER-CASE character, and the printer is in the same case. The TIMING MULTI produces the (65 ms) signal required by the SOLENOIDS. The LOCKOUT MULTIS cause the printer to be locked out (i.e. for the READY flip-flop to be in the ZERO state) for 100 ms for most characters, and for 1 sec for the CARRIAGE-RETURN/LINE-FEED character, the UPPER-CASE character and the LOWER-CASE character. The printer logic also contains test circuits (not represented in Fig. 4.19) which enables every SOLENOID to be operated under single-shot or dynamic conditions.

The approach taken with the reader, punch and printer has been to design self-contained units with autonomous timing and control. This minimises the problems associated with the design of the interface between the computer and the peripheral units. With the examples given, it appears that there are four basic signals (or groups of signals) which the designer of such an interface must consider. These are:

(i) the START signal from computer to peripheral unit,
(ii) the READY signal from peripheral unit to computer,
(iii) the DATA signals between the computer and the BUFFER of the peripheral unit,
and
(iv) the SENSE signal or CLOCK signal associated with the BUFFER.
A brief account of how these signals are treated in the design of an off-line tape editing set and in the design of the input-output logic for ARCTURUS appear in following sections.

4.10.1 Off line tape editing

A block diagram of the logic which enables the computer's peripheral units to operate as an off-line tape editing set is shown in Fig. 2.20. The basic interface signals described in the last section are switched by a RELAY SELECTION system. This consists of a relay network containing many change-over contacts under the control of a single-toggle switch. The EDITING SET LOGIC receives READY signals from all units, and generates CLOCK & START signals of those units selected by the operator. This logic is arranged to operate the units as fast as possible. For example, comparison of two tapes takes place at reader speed, reperforation of a tape at punch speed, reperforation and printing at printer speed and so on.

Since ARCTURUS was commissioned, the off-line tape editing unit has been found to be extremely useful. As the amount of hardware required for this unit was extremely small, it is considered that designers of computers similar to ARCTURUS should seriously consider this worth-while feature.
Figure 4.18 - BLOCK DIAGRAM OF PUNCH LOGIC
Figure 4.19 - BLOCK DIAGRAM OF PRINTER LOGIC
Figure 4.20 - BLOCK DIAGRAM OF INPUT-OUTPUT UNIT
4.10.2 Input-Output Logic

A block diagram of the input-output logic is shown in Fig. 4.21. A brief description follows.

Reference to the "description of instructions" (Appendix C) shows that, for an input-output instruction, bit 9 specifies a 4-bit or 5-bit character, bits 12, 13 and 14 specify reader 1, printer and punch respectively, and bits 15-19 specify the number of times the input-output operation is carried out.

The NUMBER OF TIMES COUNTER (V) is used to count the number of input-output operations. This is set initially to correspond to bits 15-19 of the instruction; V is decremented during the input-output operation; and the condition $V = 0$ is used to terminate the execution of the instruction.

The I-O READY LOGIC produces a composite READY signal which depends only on those input-output units specified by the instruction. This is the feature which enables all input-output units to operate simultaneously.

The presence of an UNWANTED CHARACTER in the reader buffer (e.g. a 5th - bit character when a 4 - bit character is specified) is treated as a NOT READY condition for the reader. When this condition exists, the reader is started to assemble the next character.

A cyclic left shift of the accumulator (K) precedes each operation. The number of places shifted is 4 or 5 depending on whether bit 9 is 0 or 1 respectively. This shift operation is under the control of the W counter.

After the contents of K have been correctly positioned, the appropriate input-output cycles are started. If the reader is involved, the reader buffer is clocked into the least significant stages of K, and the reader is then started to assemble the next character. If the printer and/or punch are involved, the logic generates the appropriate CLOCK & START pulses to output the character represented by the least significant 4 or 5 stages of K.
The independent specification of input-output units by individual bits of the instruction is a most flexible arrangement, and produces many useful variants of the instruction. Other bits of the instruction have been reserved for reader 2, a comparison of reader buffers and 8-bit operation. These extensions will require little or no change to the existing logic.
Figure 4.21 - BLOCK DIAGRAM OF INPUT-OUTPUT LOGIC
4.10.3 The ARCTURUS Auto Input

The ARCTURUS Auto Input is a special mode of operation which causes information on tape to be read and stored in consecutive memory locations. Such a feature is essential for loading an assembly program into the computer. The hardware for the Auto Input may be represented by the following itemised steps:

*START  Press AUTO INPUT push-button to put the computer into the AUTO INPUT phase. Press START push-button.

*STORE  Store K in location specified by contents of U.

*INCREMENT  Increment U.

*READ  Read five non-fifth bit characters from tape into K.

*TEST  If terminating character # has been read, go to * SET ; if not, go to * STORE.

*SET  Set K into U and S.

*EXIT  Leave the AUTO INPUT phase; enter the FETCH INSTRUCTION phase.

*STOP.  Stop.

The OPERAND COUNTER (the U register) is used to specify the memory address. The operations "within the loop" are (i) store K, (ii) increment U and (iii) read a word into K. This is repeated until the terminating character # (a fifth-bit character) has been sensed while reading into K. When this happens, K is stored into U as well as into the SEQUENCE COUNTER (the S register). Before stopping, the computer leaves the AUTO INPUT phase and enters the FETCH INSTRUCTION phase. This is the normal "idling" position and represents the beginning of a normal machine cycle.
The setting of $K$ into $U$ after $#$ is sensed enables the programmer to specify where the assembly program is to be stored. The setting of $K$ into $S$ enables him to specify the starting address.

Other "bootstrap" methods which have been used in various computers include (i) the setting of an instruction pair into the instruction register by a bootstrap push-button (Ref. 51) and (ii) the use of the input typewriter for loading the assembly program (Ref. 46). The use of a pin-board store for the bootstrap routine also appears to be a satisfactory arrangement.

The ARCTURUS Auto Input feature has made operating procedures for the computer extremely simple. It did not require an appreciable amount of hardware for its implementation, but is as good as, if not superior to, any of the schemes mentioned above.
4.11 COMMENTS ON FUNCTIONAL & LOGICAL DESIGN

A number of examples of the functional and logical design of various sections of ARCTURUS have been given. These designs have not been described in detail but rather an outline of the design and a discussion of some of the salient points have been presented. Because of space limitations, descriptions of the design of other sections of ARCTURUS must be omitted.

From the experience gained with the functional and logical design of ARCTURUS, several general comments and conclusions can be made.

A good design attempts to make good utilisation of all the electronic circuits in the machine. While these circuits have adequate margins for reliable operation, they must be operated in such a way that the computing power of the over-all machine is maximised as far as possible. Towards this objective, the time delays associated with every elementary operation, such as the time between the setting and sensing of a flip-flop, must be known; nominal times must be allocated to these operations to ensure reliable operation, and then the design is carried out so that as far as possible operating times equal to the nominal values are allowed for every computer operation irrespective of how elementary this may be. There is no point in allowing more time than the nominal value as this slows down the computer, and the hopeful expectation, that the use of an operating time less than the nominal value for perhaps only one of several hundred similar circuit configurations will not depreciate the reliability of the over-all system, is completely unfounded, as we cannot assume that the worst-case conditions will not be met by the circuit configuration in question.

There should be as much concurrency of operations as possible. This is related to the aim of eliminating unnecessary delays or the aim of allocating active functions to all computer elements in as many computer phases as possible. The degree to which this principle can be applied, of course, depends on the structure of the computer (in particular, the autonomy of component units) and on the amount of buffering between the units.
In ARCTURUS, the multiplication and division procedures can proceed concurrently with the regeneration of the operand in memory, and the buffering of all peripheral units enables the concurrent operation of all peripheral units and the central processing unit.

A computer should contain well-matched component units. There is, for example, little point in providing extensive hardware for speeding up the arithmetic unit if the computer is already severely limited by the speed of its memory. A possible mis-match between the peripheral units and the central processing unit should also be considered.

In ARCTURUS, the arithmetic unit has an accumulate time of $\frac{1}{2}\mu s$ and a multiply time of $5 - 7\frac{1}{2}\mu s$; these speeds reasonably match the $3\mu s$ cycle time of the memory, and the unit can well accommodate the $2\mu s$ memory which is planned for the near future. The peripheral units of ARCTURUS (see Section 4.10) are adequate for a scientific computer of its class.

The designer of digital equipment must of course give due consideration to the problems of circuit delays and signal deterioration. As circuit speed is an important design criterion, the problems of timing often become critical, and a sound knowledge of all circuit delays involved becomes very important. This was particularly so for the design of the timing and control units of ARCTURUS. In systems which use diode-transistor logic, diode gates are found to be fast and cheap, and the over-all speed of a circuit configuration often depends to a great extent on the number of transistor invert circuits in cascade, (see Section 4.5). Hence configurations are chosen to take full advantage of the speed of the diode gates and the regenerative property of transistor invert circuits. The regeneration problem does not exist with systems which use integrated circuits as regeneration does in fact take place in every circuit, and the main design criterion for maximum speed is a minimum number of circuits in cascade.
A thorough understanding of both the algorithms for carrying out machine functions and the hardware required to implement these algorithms is extremely important for the machine designer. The situation is not a static one as the availability of new logical elements provides the designer with a challenge to make full utilisation of these elements in the production of faster and more powerful digital computers.
CHAPTER 5
ARCTURUS - CIRCUIT DESIGN

5.0 DESIGN APPROACH
5.0.0 General considerations
5.0.1 ARCTURUS circuit design

5.1 CIRCUIT TYPES & CHARACTERISTICS
5.1.0 Characteristics of selected circuits
5.1.1 Summary of types & characteristics
5.0 DESIGN APPROACH

5.0.0 General Considerations

There is a wealth of information in the literature on the design of transistorized circuits for digital computers. This information is in the form of books, papers, correspondence, design handbooks and manufacturer's application notes.

"Proper functioning" of the circuit is, of course, the primary requirement to be satisfied by any design procedure. This requirement may narrow the choice of circuit types and may introduce constraints on the parameters of the circuit, but it does not lead to a specific set of parameters. Some form of "optimisation" must be applied or additional design criteria must be satisfied to produce a specific set of parameters from which an actual circuit may be constructed.

Optimisation in computers is that mechanization of equipment which results in maximum performance at minimum cost. To achieve such an objective, designers of computer circuits not only must be proficient in electronic circuitry, but must understand logical design, and must be familiar with manufacturing techniques - both for the basic electronic components and for the synthesis of the complete computer system. This is clearly so as the minimisation of over-all costs does not depend solely on the cost minimisation of isolated electronic circuits. Costs can be reduced, for example, by a simplification of packaging techniques or by a relaxation of specifications and tolerances of power supplies, or again by an increase in reliability as this reduces commissioning and servicing costs.

The first step in the design of logical circuits for a computer is the derivation of circuit specifications. For example, the objectives for the delay per gate, fan-in and fan-out must be specified. These specifications can be obtained only by an examination of the logical design of those sections of the computer which demands the severest circuit requirements. In a conventional computer configuration, the adder could well be expected to determine these requirements.
The next two steps involve an evaluation of components and types of circuits. Simple procedures for checking transistor parameters, and possibly a computer program to correlate these parameters with the delay produced when inserted in circuit would be extremely useful. For the development of a large-scale system, the evaluation of types of circuits could even involve the construction of small prototype systems using different circuits.

With the type of circuit selected, the final step in the design is an optimization of this circuit. This could be carried out using a "worst case design" procedure (Ref. 52). The design of the basic logical gate for the UNIVAC-LARC (Ref. 53) is an example of a worst-case design. Worst-case considerations produced 6 equations in 14 independent unknowns. Some of these unknowns were selected so that a high percentage of diodes and transistors from the manufacturers could be used; others were selected to minimize power requirements. Other criteria were introduced until 6 equations in 9 unknowns remained. These were then determined through an optimization of speed.

As systems become larger and design problems more complex, the circuit designer must take full advantage of the digital computer to assist in the design. Programs have been written for the design of computer circuits (Ref. 54); some of these use linear programming techniques (Ref. 55). Circuit design using graphical input-output devices (Ref. 56) seems to have an exciting future.

5.0.1 ARCTURUS Circuit: Design

The circuit design for ARCTURUS was carried out with the background knowledge of a complete range of computer circuits used in two earlier computers, viz; - the digital differential analyser ADA (Ref. 1) and the general purpose digital computer SNOCOM (Ref. 4). All three computers make use of the same type of basic logic circuitry viz: - positive logic diode AND-OR gates and PNP transistor INVERT circuits. However the circuits in ARCTURUS utilise better semiconductor devices and circuit modifications have been made to make these circuits far superior to those in the earlier machine.
There was much overlapping of logical design and circuit design in the ARCTURUS project. An appreciation of the circuit requirements of a tentative logical design enabled diode and transistor types to be selected and a range of logical circuits to be constructed. An appreciation of the circuit capabilities by the logical designer enabled final improvements to the design to be made by making full utilisation of the circuits.

The design criteria of maximising speed, fan-in, fan-out, noise rejection, component tolerances and supply tolerances, and of minimising power requirements, heat dissipation, temperature sensitivity, component rejection and constructional costs were all considered in varying degrees to obtain a range of logical circuits which could be produced in quantity using the facilities of a small research laboratory.

The development of these circuits was carried out by Mr. K.R. Rosolen. The author is deeply indebted to Mr. Rosolen not only for developing these circuits but for the role he played in all stages of the ARCTURUS project. Examples of these circuits and a brief description of their characteristics follow.
5.1 CIRCUIT TYPES AND CHARACTERISTICS

5.1.0 Characteristics of Selected Circuits

Examples of circuit configurations are shown in Figs. 5.0, 5.2, 5.4 and 5.6. All circuits operate with +1 volt and -1 volt as the ONE and ZERO signals. Positive logic is assumed unless otherwise stated.

The logic circuit of Fig. 5.0 shows a typical configuration containing an AND-OR-EF-AND-OR-EF-INVERT-EF logic chain. Typical waveforms through such a chain are shown in Fig. 5.1. An evaluation of waveforms like this indicated that the longest logic chain without regeneration should be limited to an AND-OR-EF-AND-OR-EF configuration. This limitation was never exceeded in the design of the computer; moreover regeneration after fewer logical circuits in cascade was considered wise for some important, heavily-loaded circuits.

The output of the INVERT circuit is a 0 volt/ -2 volts signal. This is shifted positively (by 1 volt) by the divider circuit associated with the final emitter follower. The use of diodes in this circuit produces a voltage shift without an attenuation of signal amplitude. The use of dividers in the outputs of INVERT circuits rather than in the outputs of the AND gates (which was the method used in ADA and SNOCOM) has resulted in a significant component reduction.

Resistor values of approximately double those indicated in Fig. 5.0 were in fact used in earlier circuit configurations. The reduction of these resistor values has increased speed at the expense of increased power requirements.

Typical waveforms associated with the storage gate flip-flop of Fig. 5.2 are shown in Fig. 5.3. A storage gate consists of a resistor-capacitor network which operates on the flip-flop input as an integrating circuit. The output of this circuit is sensed (i.e. gated) by the clock, and the clocked signal is applied to the input of the flip-flop.
The use of storage gates enables setting and resetting signals to flip-flops to be removed at clock time. This property is required by designers of single-phase, synchronous logic systems. Storage gates are used extensively in ARCTURUS for accumulating, shifting and counting.

The waveforms C and D of Fig. 5.3 show the delays involved in the clocking and sensing of a storage-gate flip-flop. These waveforms were generated by the REPEATED OPERATIONS LOOP. In this part of the machine, the G flip-flop (which is a storage-gate flip-flop) is clocked by a $T_{OP}$ pulse, and is used to gate a $T_{125}$ pulse which follows 125 ns after $T_{OP}$ (see section 4.2.2). The waveforms show that the G flip-flop has settled well in advance of the $T_{125}$ pulse. Waveforms like these indicate the circuit tolerances which exist in the computer.

The binary counter circuit of Fig. 5.4 essentially consists of a storage-gate flip-flop whose inputs and outputs are cross-coupled. The addition of the GATE input (see Fig. 5.4) and the associated diode-AND gate enables parallel binary counters to be constructed using the basic binary counter circuit (see Section 4.6.3). The RESET, INPUT and CLOCK inputs enable the binary counter to be initially set to either state using a reset-set cycle. Waveforms associated with a ten-stage, parallel, binary counter are shown in Fig. 5.5. As GATE signals are established for all stages of the counter in the bit-period prior to the activating COUNT pulse, all stages switch almost simultaneously. There is no appreciable delay between the switching of the least-significant stage and the most-significant stage (as there is with a serial counter formed by cascading a number of stages), and depending on variations in switching speed, it is even quite possible for the most-significant stage to switch just before the least-significant stage. The waveforms of Fig. 5.5 verify these remarks.

A circuit diagram of a positive trigger monostable multi is shown in Fig. 5.6. Such a circuit can have several trigger inputs each of which can be controlled by a gate signal. As the resistor-capacitor network used for this purpose represents an integrating circuit.
when viewed from the gate terminal, the gate signals must be established well before the front edge of the trigger input pulse. When these conditions are satisfied, the monostable multis with their gated input signals are extremely useful for the insertion or deletion of timing waveforms in chains of timing waveforms. This requirement was met several times in the design of the timing unit for ARCTURUS. Typical waveforms produced by a chain of monostable multis are shown in Fig. 5.7. All waveforms are variable in period with a minimum of about $\frac{1}{2}$ $\mu$s, and a maximum determined by the value of the capacitor which is shown with the typical value of 470 pf in Fig. 5.6.

A second type of monostable multi which triggers off the back edge of the input signal can have a period down to 100 ns, and is used extensively in the timing unit of ARCTURUS. Typical waveforms produced by this type of circuit are shown in Fig. 5.7.
Figure 5.0 - TYPICAL DIODE-TRANSISTOR LOGIC CIRCUIT CONFIGURATION
Figure 5.1 - WAVEFORMS OF DIODE-TRANSISTOR LOGIC CIRCUITS
(10 ns/cm, 1 volt/cm)
Figure 5.2 - CIRCUIT DIAGRAM OF STORAGE-GATE FLIPFLOP
Figure 5.3 - STORAGE-GATE FLIP-FLOP WAVEFORMS
(1 volt/cm)
Figure 5.4 - CIRCUIT DIAGRAM OF BINARY COUNTER
Figure 5.5 - WAVEFORMS OF TEN-STAGE, PARALLEL BINARY COUNTER
Figure 5.6 - CIRCUIT DIAGRAM OF POSITIVE-TRIGGER MONOSTABLE MULTI
(A) Superimposed memory timing waveforms generated by a train of positive-trigger monostable multis (500 ns/cm)

(B) Switching delays of positive-trigger monostable multis (100 ns/cm)

(C) Superimposed timing waveforms generated by a train of negative-trigger monostable multis (100 ns/cm)

(D) A train of four pulses produced by negative-trigger monostable multis and a binary counter (200 ns/cm)

Figure 5.7 - WAVEFORMS OF MONOSTABLE MULTIS (1 volt/cm)
5.1.1 Summary of circuit characteristics

The characteristics of the logical circuits used in ARCTURUS are summarised in Table 5.0. Only those parameters which are representative of over-all circuit performance, and which are of major importance to the logical designer are listed in this table. For example, although the output of an AND gate may feed one OR gate, one flip-flop or one emitter follower, these do not represent the maximum fan-out of the gate, and the use of the basic AND-OR-EF and AND-OR-INVERT-EF configurations in a large part of the machine makes figures for the fan-out of individual AND and OR gates of much less importance than figures for the over-all circuit configuration. It is, however, to be noted that where individual AND or OR gates generate signals which are transmitted through the base of the computer, emitter followers are always used. Under these circumstances, the stray capacitance of the signal wire to ground and to other signal wires is often of more importance than the load introduced by other logical circuits. It is also to be noted that emitter followers are used with all outputs of all monostable and bistable circuits.

Although tests on circuit characteristics such as noise rejection, power supply tolerance, component tolerance, temperature sensitivity and vibration sensitivity were carried out on the logical circuits, no attempt has been made in Table 5.0 to list these characteristics quantitatively.

It can be seen from Table 5.0 that the number of basic circuit types used in ARCTURUS is not large. However these circuits may be arranged into an unlimited number of different configurations. The size of the package used to hold the circuits and the number of pins in the package connectors introduce constraints. As well as satisfying these constraints, circuit configurations for the packages are designed to maximise switching capabilities of the package and to provide the logical designer with larger, functional building blocks. These topics will be treated in the next chapter.
### TABLE 5.0
**Summary of Circuit Characteristics**

<table>
<thead>
<tr>
<th>CIRCUIT TYPE</th>
<th>CHARACTERISTIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Fan-in: 2-8 (typical)</td>
</tr>
<tr>
<td></td>
<td>: 20 (special gate)</td>
</tr>
<tr>
<td></td>
<td>Delay: 2-4 ns</td>
</tr>
<tr>
<td>OR</td>
<td>Fan-in: 2-10</td>
</tr>
<tr>
<td></td>
<td>Delay: 2-4 ns</td>
</tr>
<tr>
<td>INVERT</td>
<td>Delay: 30-40 ns</td>
</tr>
<tr>
<td>EF(Emitter follower)</td>
<td>Fan-out: &gt;15</td>
</tr>
<tr>
<td></td>
<td>Delay: 2-4 ns</td>
</tr>
<tr>
<td>AND-OR-EF</td>
<td>Delay: 5-8 ns</td>
</tr>
<tr>
<td>AND-OR-INVERT-EF</td>
<td>Delay: 35-45 ns</td>
</tr>
<tr>
<td>R-S flip-flop Storage-gate FF</td>
<td>Turnover time: 80 ns (typical)</td>
</tr>
<tr>
<td>Binary counter</td>
<td>Max. repetition rate:</td>
</tr>
<tr>
<td>Shift register</td>
<td>4 MHz (design spec.)</td>
</tr>
<tr>
<td></td>
<td>8 MHz (typical)</td>
</tr>
<tr>
<td>Positive-trigger monostable multi</td>
<td>Min. pulse period: $\frac{1}{2}$ μs</td>
</tr>
<tr>
<td></td>
<td>Pulse rise time: 30 ns</td>
</tr>
<tr>
<td></td>
<td>Mark-space ratio: &gt; 2:1</td>
</tr>
<tr>
<td>Negative-trigger monostable multi</td>
<td>Min. pulse period:</td>
</tr>
<tr>
<td></td>
<td>125 ns (design spec.)</td>
</tr>
<tr>
<td></td>
<td>90 ns (typical)</td>
</tr>
<tr>
<td></td>
<td>Pulse rise time: 12-15 ns</td>
</tr>
<tr>
<td></td>
<td>Mark-space ratio: &gt; 2:1</td>
</tr>
<tr>
<td>Clock driver</td>
<td>Load: 0.0015 μF (typical)</td>
</tr>
<tr>
<td></td>
<td>Pulse rise time 10-20 ns</td>
</tr>
<tr>
<td></td>
<td>Pulse width 125 ns</td>
</tr>
<tr>
<td>Memory driver</td>
<td>Current output 100 ma</td>
</tr>
<tr>
<td></td>
<td>Pulse rise time 80 ns</td>
</tr>
<tr>
<td></td>
<td>Pulse width 1 μs</td>
</tr>
</tbody>
</table>
CHAPTER 6

ARCTURUS - CONSTRUCTIONAL TECHNIQUES & COMMISSIONING

6.0 PACKAGING
6.0.0 Standard packages
6.0.1 The KL package
6.0.2 Package tests

6.1 BASE-WIRING
6.1.0 Specification
6.1.1 Noise minimisation
6.1.2 Base-wiring checks

6.2 ENGINEERING CONSTRUCTION
6.2.0 Machine layout
6.2.1 Consoles
6.2.2 Auxiliaries

6.3 COMMISSIONING PROCEDURE
6.3.0 Sectional tests
6.3.1 Built-in engineering tests
6.3.2 Total system tests
6.3.3 Test programs
6.0 PACKAGING

6.0.0 Standard Packages

The method of package construction was essentially the same as that used in ADA and SNOCOM. The electronic components are pushed into holes in a polythene card which is held in juxtaposition with one or more Cannon plug(s) by a stainless-steel band. Two sizes of packages are used; their dimensions are approximately $1\frac{3}{4}'' \times 4''$ and $3\frac{3}{4}'' \times 4\frac{1}{2}''$; and they contain 15 pins and 55 pins respectively. Photographs of packages are shown in Fig. 6.0 and Fig. 6.1. Examples of component layout diagrams which are required for the construction of the packages are shown in Appendix A.

The majority of packages in ARCTURUS are of the 15-pin type. In the design of these packages, the main objectives were to construct as much of the computer as possible using a small number of different types of packages, and to keep all non-standard packages as simple as possible. Logic diagrams of the standard packages used in ARCTURUS are shown in Appendix B.

6.0.1 The KL Package

The 55-pin, KL package contains one stage each of the K register, the L register, the ADDER and the DISTRIBUTOR-FUNCTION-SELECTOR. These large packages which are innovations found only in ARCTURUS, were used to minimise inter-package wiring (base-wiring). This results in a reduction of signal delays and pick-up of extraneous noise. In the design of fast computers, long inter-package wiring introduces significant delays, and hence the layout of the packages within the main-frame is extremely important. An extension of the principle used in the KL packages, viz. the packaging of corresponding stages of all registers and the adder in the same package might be a worth-while consideration. The use of integrated circuits would make this idea quite feasible, and the resultant package would not be unmanageably large.

A logic diagram of the KL package is shown in Fig. 6.2. Most of the symbols used in this diagram are listed in Section 4.0. As the present discussion is concerned with the principles of packaging rather than details of logical design, the remaining symbols will not be
Fig. 6.0 - PHOTOGRAPHS OF PACKAGES
6.0.2 Package Tests

As part of the commissioning procedure, all packages were subjected to the following tests:

(i) Visual inspection of all soldered joints.
(ii) Visual check of wiring according to the component layout diagram.
(iii) Static tests to check the logical functioning of the package according to the logic diagram.
(iv) Dynamic tests to check the dynamic performance of the package with every input of every gate connected to a pulse generator in turn. Unacceptable circuit delays, oscillations and ringing were detected by this test.

In addition to the above tests, selected packages were given temperature and vibration tests. These packages were subjected to alternate blasts of hot and cold air generated by a hot air blower and an improvised blower system using liquid air. Results of these tests were very favourable, and indicated that temperature sensitivity of the circuits was not a major problem. Vibration tests using a motor operated cam were used to check the performance of plugs and sockets and to locate suspected dry joints. Combined temperature and vibration tests gave some indication of the durability of the complete package. These tests, which often lasted many hours, augmented confidence in the circuits and in their method of construction.
6.1 BASE-WIRING

6.1.0 Specification

Wiring lists were obtained from the logic diagrams of functional units of the computer (such as the timing unit diagram and the input-output unit diagram). This information was transferred to a large 10' x 3' base-wiring diagram, which contained information about every package in the main-frame of the computer. An extract from this diagram which shows information pertaining to a particular package is shown in Fig. 6.3.

To minimise errors in the large base-wiring diagram, a system of cross-referencing was introduced. This diagram contains not only a list of destination-points to which the package outputs are connected, but also a list of source-points from which signals are derived for the package inputs. Discrepancies in the system clearly reveal errors in wiring or in specification; these are checked out thoroughly and eliminated.

The large base-wiring diagram has proved useful in another way. The diagram provides a picture of the layout of the entire main-frame, and is extremely useful for choosing the optimum position of a package. The KL clock driver package is a good example of a package where operation can depend on its position within the main-frame. As the KL packages are arranged in line, and extend over several feet of the main-frame, the positioning of the clock drivers near the centre of these packages would minimise wire lengths for the clock lines. This would minimise signal overshoot, and also minimise the variation of the signals applied to each KL package. The above considerations were found to be extremely important, as the KL clock driver package was in fact moved to a position near the centre of the KL packages to improve the reliability of the computer.

6.1.1 Noise minimisation

Tests on the main-frame of the computer showed that the transmission of a 100 ns pulse (with a rise time of 10-15 ns) from one end of the main-frame to the other produced severe ringing.
To overcome this and other noise problems, it was decided to introduce a ground-plane. Unfortunately, when this decision was made, most of the base-wiring had already been carried out, and hence it was necessary to form the ground-plane by inserting a grid of copper strips (with silver and gold flashing) behind the wires, and by soldering the strips at all cross-over points.

Noise on voltage lines has been minimised by providing an even distribution of filter capacitors through-out the main-frame. These capacitors are connected between all voltage lines and the ground plane.

Where capacitive loading is important, wires are held well clear of the ground-plane and well clear of other wires by threading them through a number of stainless-steel wire-mesh brackets.

The above features for noise minimisation are illustrated in the photographs of Fig. 6.4. Since the incorporation of these features, most signal waveforms have been extremely "clean", and pickup of self-induced or extraneous noise has not been a problem.

6.1.2 Base-wiring Checks

As part of the commissioning procedure, a thorough check of the base-wiring was made. All points in the base were firstly inspected for dry or badly soldered joints. As all wires in the base were colour coded (e.g. blue-white wires for J signals, green-yellow wires for clocks and so on), a joint with two different coloured wires connected to it was immediately suspected of being wired incorrectly, and a thorough check of all wires connected to this joint would be made.

A check on the continuity of wires according to the base-wiring diagram was made using a system of probes and a buzzer. The cross-referenced specification of base wires (mentioned earlier) revealed many mistakes. A test for the discontinuity of every pair of signal lines was not attempted because of the time required for such a test. However all voltage lines and some important signal lines were in fact checked for discontinuity.
### Figure 6.3 - BASE WIRING SPECIFICATION

**Point Specification:**

<table>
<thead>
<tr>
<th>Block No.</th>
<th>Package No.</th>
<th>Socket (IF NECESSARY)</th>
<th>Pin No.</th>
<th>Source Points</th>
<th>Input Points</th>
<th>Destination Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>06-6</td>
<td>05</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

**Source Points:**
- Source Points: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

**Input Points:**
- Input Points: 11, 12, 13, 14

**Destination Points:**
- Destination Points: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

**Output Points:**
- Output Points: 11, 12, 13, 14

**Point Specification:**

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Input/Output</th>
<th>Point Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>o/p</td>
<td>14/4/6</td>
</tr>
<tr>
<td>2</td>
<td>o/p</td>
<td>14/4/6</td>
</tr>
<tr>
<td>3</td>
<td>3V21-8</td>
<td>14/4/6</td>
</tr>
<tr>
<td>4</td>
<td>2V</td>
<td>14/4/6</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>14/4/6</td>
</tr>
<tr>
<td>6</td>
<td>6V</td>
<td>14/4/6</td>
</tr>
<tr>
<td>7</td>
<td>6V</td>
<td>14/4/6</td>
</tr>
<tr>
<td>8</td>
<td>2V</td>
<td>14/4/6</td>
</tr>
<tr>
<td>9</td>
<td>0/1</td>
<td>14/4/6</td>
</tr>
<tr>
<td>10</td>
<td>0/p</td>
<td>14/4/6</td>
</tr>
<tr>
<td>11</td>
<td>R13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>R13</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>R13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>R13</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>R13</td>
<td></td>
</tr>
</tbody>
</table>
Fig. 6.4 - PHOTOGRAPHS OF BASE-WIRING
6.2 ENGINEERING CONSTRUCTION

6.2.0 Machine Layout

Photographs of ARCTURUS appear in Fig. 6.5. The main-frame of the computer containing the memory, arithmetic and control units rests on top of a Steelbilt desk. The main console which is connected to the main-frame by short cables is located centrally on the desk for the convenience of the machine operator. The pedestals of the desk and the section between the pedestals contains power controls, power supplies, cooling fans and metering facilities.

The input-output unit consists of a long, narrow desk and a small cabinet. The reader and printer rest on top of this desk and the pedestal contains the Teletype BRPE punch. The small cabinet contains the input-output console and all the electronic circuitry required for the autonomous operation of the peripheral units. The layout of the input-output unit was suggested by experience with SNOCOM, and has been found to be very convenient for the efficient handling of paper tape.

6.2.1 Consoles

A photograph of the main console is shown in Fig. 6.6. There are two sections of this console. The lower section contains all the pushbuttons and switches which are frequently used by the machine operator. This section is clearly engraved and the console arrangement is kept as simple as possible. The upper section contains monitors for all machine registers and the controls for built-in engineering tests (see Section 6.3.1). A toggle register in this section of the console is also used occasionally by the machine operator.

A photograph of the input-output console is shown in Fig. 6.7. The engraving of this console is illuminated in sections.
Each section contains the monitors and controls for each peripheral unit, and only those sections corresponding to operative peripheral units are illuminated. The data flow between these units is also represented by illuminated engraved lines. This feature was found to be very useful particularly when the peripheral units were operated off-line as an editing set. Under these conditions, such a feature would significantly reduce operators' errors.

6.2.2 Auxiliaries

The power supply required by ARCTURUS is a single-phase, 240 V A.C. supply which is readily available from a wall outlet. The A.C. power controls are housed in the left-hand pedestal of the Steelbilt desk. Experience with SNOCOM suggested that a Variac control of the A.C. power to the D.C. power supplies was wise (if not absolutely necessary) to minimise the effects of switch-on and switch-off transients. A Variac for this purpose was used in ARCTURUS.

The D.C. power supplies were constructed as separate, modular units. Design of power supplies for general-purpose, laboratory use (Ref. 57) were utilised, but modifications to these designs were required by some high-current, high voltage supplies. Potentiometer controls to vary the output voltage of each supply within a limited range are provided, and are used in marginal tests on the computer. Metering facilities to monitor currents and voltages of all supplies are housed in the right-hand pedestal.

Cooling fans for the general cooling of the main-frame and for the cooling of specific components (such as the resistors in the memory drive amplifiers) are mounted at strategic points throughout the computer. Forced air ducts are also used in the main-frame.

The transistor amplifiers for the monitor lamps are mounted in units located as close to the base of the main-frame as possible. This arrangement minimises the capacitive loading of signal lines.
Fig. 6.5 - PHOTOGRAPHS OF ARCTURUS SHOWING MACHINE LAYOUT
Fig. 6.6 - PHOTOGRAPHS OF MAIN CONSOLE

Fig. 6.7 - PHOTOGRAPH OF INPUT-OUTPUT CONSOLE
6.3 COMMISSIONING PROCEDURE

6.3.0 Sectional Tests

After the package and base-wiring tests described in Sections 6.0.2 and 6.1.2 had been carried out, the next stage of the commissioning procedure involved groups of packages plugged into the main frame. Groups of packages representing, for example the main timing unit or a complete binary counter were tested in their correct positions in the main-frame, but the control signals and drive pulse were produced externally.

The tests carried out on the first few groups of packages were initially very simple, but as more and more packages were gradually plugged into the main-frame, tests involved larger sections of the machine and hence became more comprehensive.

When sufficient packages were in the main-frame to enable the built-in engineering tests to become effective, the following stages of the commissioning procedure made full utilisation of these tests. They were, in fact, sufficient for testing the remaining sections of the machine prior to tests on the total-system.

6.3.1 Built-in Engineering Tests

The built-in engineering tests are basically very simple, but are extremely useful. These tests require a small amount of additional logic to inhibit some important machine signals, and to insert, in their place, some test signals in the form of a single pulse or a pulse train. These tests are carried out under the control of toggle-switches and push-buttons located in the top section of the main console.

As the functioning of the computer depends so heavily on the correct functioning of the main timing unit, a built-in test of this unit is provided. By replacing the normal START pulse by a train of pulses from a pulse generator, all timing waveforms may be monitored.
With the timing unit operating satisfactorily, single-shot tests can be carried out to check every machine function. Extensive use is made of a toggle register which enables information to be set into all registers. By setting the instruction register (the R register) to hold different machine instructions in turn, all instructions may be checked out on a single-shot basis. By inhibiting the R clocks (at present carried out by removing the R clock package), all instructions may be obeyed repeatedly to enable all relevant control and timing waveforms to be monitored.

As test switches are used to break not only the main timing unit but also the V and W loops as well, elementary steps involved in the execution of a complex instruction (such as the INPUT-OUTPUT instruction) may be tested on a single shot basis.

The MJI (Memory-to-J Inhibit) switch has been found to be extremely useful. By using this switch, tests on the arithmetic and control sections of the machine can be carried out quite independently of the memory, as the required instruction or operand may be set up artificially in J using the toggle register.

Although the tests described above are basically very simple, they were found to be extremely useful in the early stages of commissioning for the location of machine faults. The additional controls required by these tests also provided a means for storing information, and this resulted in the first operation of ARCTURUS as a stored-program computer.

6.3.2 Total-system Tests

Prior to a series of tests using programs read from tape, tests on the "total-system" were carried out with the machine executing simple programs stored a word at a time using the console switches. The aim of these tests was firstly to determine the optimum settings of important adjustable controls, and secondly to obtain some indication of the sensitivity of the total system to temperature changes and to external noise sources.
Marginal tests involving a variation of all voltages in turn were carried out to determine the range of voltages over which reliable operation was produced. "Optimum" settings of these voltages were then chosen near the middle of these ranges.

Some voltages used only by the memory unit determine the memory drive currents, and hence were very critical. To determine the optimum settings of these voltages, modifications were made to the machine to read from (and write into) all locations of the memory in turn. In this way, superimposed playback waveforms from all locations could be viewed on a C.R.O., and optimum settings not only of the memory voltages, but also of the important STROBE pulse, could be determined. It is anticipated that modifications for this test will be incorporated into the "built-in engineering tests" at a later date.

Temperature sensitivity tests involved blasts of hot and cold air directed at sections of the machine. These tests were useful for locating several marginally faulty circuits.

Noise immunity tests involved turning equipment in the computer room on and off. The machine was only affected when the same A.C. line was used. This suggested that a further improvement in machine reliability could be obtained by the use of an isolating transformer and/or line-filter.

6.3.3 Test Programs

A whole range of very simple test programs each involving only one or two words was written to check the execution of every instruction. As the instruction under test was obeyed repeatedly, all timing and control signals associated with this instruction could be monitored. These programs were found to be very useful.

Memory dumps were found to be useful not only for debugging programs, but also for testing the machine. The more complex output formats which are useful for debugging programs were, of course, not necessary for testing purposes, and the sexadecimal dump was invariably used.
A memory test program tested every location with different test patterns. With the test program stored in the first half of the memory, the test pattern (read from tape) was stored in every location of the second half of the memory; the fetch-and-compare operation was then applied to each location in turn, and a print-up of all faulty locations (if any) was produced. By copying itself into the second half of the memory, the test program can test the first half. The whole procedure was repeated many times using different test patterns.

It is to be noted that, with the memory test program described above, the test pattern is stored in many locations before any fetch-and-compare operation is carried out. This is a more severe test than one which consists of a succession of store-fetch-and-compare operations. This is so because in the program which is described, all ferrite-cores corresponding to the test location are subjected to many half-read pulses before a full-read pulse is applied.

During its first five months of operation, ARCTURUS was found to be very reliable, and extensive test programs have not been found necessary. However, the usefulness of a single, comprehensive test program which tests all machine functions and produces a diagnostic print-up is fully appreciated; and in anticipation of an increased work-load for the machine, which puts more emphasis on the machine's reliability, such a test program is being planned.
CHAPTER 7
ARCTURUS - PROGRAMMING

7.0 THE ASSEMBLY PROGRAM API

7.0.0 Introduction & description
7.0.1 Program structure & coding

7.1 PROGRAMMED-OPERATORS

7.1.0 Programming arrangements using API
7.1.1 Examples - Double-length arithmetic
7.1.2 Examples - Floating-point arithmetic

7.2 SUB-ROUTINE HIERARCHY

7.2.0 Jump-to-subroutine instructions
7.2.1 Subroutine hierarchy instructions

7.3 SOFTWARE DEVELOPMENT

7.3.0 Present situation
7.3.1 Plans for the future
7.0 THE ASSEMBLY PROGRAM API

7.0.0 Introduction & Description

Prior to the development of API, all programs for ARCTURUS were written in binary. This entailed working out, from the description of ARCTURUS instructions (Appendix C), the binary patterns (or sexadecimal characters) corresponding to every instruction in the program.

Assembly Programs simplify program preparation for digital computers. One of the main reasons for their use would be to facilitate the use of the program library. The Assembly Program API is a basic assembly program in that only the features which were considered "essential" for the efficient programming of a small scientific computer with limited storage capacity were incorporated. A description of this program follows.

Instructions in API use the sexadecimal character set 0-9, +, -, N, J, F, L. The format of these instructions is:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>INDIRECT ADDRESS</th>
<th>DECIMAL ADDRESS</th>
<th>TERMINATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>J</td>
<td>AAAA</td>
<td>+ (31 Characters)</td>
</tr>
</tbody>
</table>

The symbols used in the above instruction format are defined below:

**TT**

The symbols TT represent two sexadecimal characters which specify the instruction type. The characters, for machine instructions, range from 00 to 1L; and for programmed operators from 20 to 3L.

**J**

The character J specifies indirect addressing. If indirect addressing is not required, this character is omitted.
The symbols AAAA represent a decimal number which specifies an address or a pseudo-address representing an instruction variant. The number may contain any number of decimal digits from zero to six inclusive, but must be less than or equal to 524287. If no address is specified it is assumed to be zero.

The terminator + specifies a change of directive. Details of the operations initiated by this terminator are as follows:

(i) The block number is incremented by one.

(ii) The block number is printed. Blocks are numbered 20 (sexadecimal) onwards to correspond to the locations in which their starting addresses are stored.

(iii) The 31 five-bit characters on the tape following the character + which constitute a "program-identifier" are read and printed, but not stored.

(iv) The block-start-address is printed. The block-start-addresses are stored as a list starting from location 20 (sexadecimal). These addresses may form part of unconditional-transfer instructions required by programmed-operators.

(v) The block-start address specifies the location in which the first assembled word of the next block is to be stored.

The terminator - specifies that the address which it follows is to be considered relative to the starting address of the block specified by the two sexadecimal characters SS.

The terminator N specifies that the instruction is to be obeyed and not to be stored.

The terminator F specifies that the address which it follows is a fixed address (i.e. relative to address zero).

The terminator L specifies that the address, which it follows is to be considered relative to the starting address of the current block.
After API is loaded, the initial conditions of the program are:

(i) The address of the instruction which stores the assembled-instruction is 32 (decimal).

(ii) The block number is 1L (sexadecimal).

Features of API are:

(i) The operation codes for "programmed operators" form a logical extension of the operation codes used for machine instructions.

(ii) Indirect addressing may be specified independently.

(iii) Operand addresses are specified in decimal.

(iv) Programs may be readily assembled using blocks of instructions with each block consisting of either a library sub-routine or one section of the user's program.

(v) Relative addressing with respect to the beginning of the block being currently assembled or with respect to the beginning of any specified block is provided.

(vi) Closed sub-routines may be assembled and then executed before the rest of the program is assembled under the control of API.
   (This is accomplished by terminating a JUMP TO SUB-Routine instruction with the character N).

(vii) Program identifiers are printed to provide a permanent record of the programs being used.

(viii) The assembled program may be started at any fixed address or at the beginning of any specified block.

Some instructions, such as the REGISTER TRANSFER instruction, use individual bits of the instructions to specify independent control functions. This results in a large number of variants for these instructions. No provision is made in API for the independent specification of control functions. However, the severity of this limitation is significantly reduced by the extensive use of the pre-compiled lists of corresponding instruction variants and pseudo-addresses. Examples of these lists are shown in Appendix E.
7.0.1 Program Structure & Coding

The program structure of API is shown on page 156; the detailed coding is shown in Appendix D.

There is some difference of opinion concerning the usefulness of flow diagrams for the description of computer programs. One world-renowned computer authority (Ref. 58) has stated that their usefulness has been over-rated, while other experienced programmers continue to make extensive use of them.

In an attempt to find a suitable substitute for the flow diagram, the Author has made use of what he calls a "program structure". This consists of a list of program statements, some of which are associated with a label. Program statements are to be considered sequentially (from top to bottom), unless a break in sequence is called for by a GO TO or IF-GO TO statement.

The API program contains 57 words. As this program is to be used many times and memory space is at a premium, considerable effort was directed at the production of an optimised coding for this program. By using various coding tricks, the number of words was reduced several times. While a further reduction may still be possible, it can be said with confidence that at least a "near-optimum" coding has been produced.
THE API PROGRAM STRUCTURE

* ENTRY : Read two sexadecimal characters.
  : Assemble operation code and programmed-operator bit in location reserved for the assembled-instruction.

* CLEAR : Clear K and L.

* READ : Read one sexadecimal character.
  : IF decimal GO TO * CONVERT
  : IF +,-,N,J,F,L, GO TO * TERMINATOR +,
  * TERMINATOR -, * TERMINATOR N, * CHARACTER J,
  * TERMINATOR F, * TERMINATOR L respectively.

* CONVERT: Convert decimal address to binary assembling address in L.
  : GO TO * READ.

* TERMINATOR + : Operate carriage return/line feed on printer.
  : Set printer to figure shift.
  : Set printer to lower case.
  : Increment block number.
  : Print block number.
  : Read and print program identifier.
  : Set block-start-address in instruction which stores assembled-instruction.
  : Print block-start-address.
  : GO TO * ENTRY.

* TERMINATOR - : Read two sexadecimal characters specifying a block number.
  : Fetch into K the block-start-address of specified block.

* COLLATE : Remove from K all but address bits.
  : GO TO * TERMINATOR F.

* TERMINATOR N: Add address in L to assembled-instruction.
  : GO TO * ASSEMBLED-INSTRUCTION.

* ASSEMBLED-INSTRUCTION : Obey assembled-instruction.
  : GO TO * ENTRY.

* CHARACTER J : Add indirect address bit to assembled-instruction.
  : GO TO * CLEAR.

* TERMINATOR F: Add address in L and block-start-address in K to assembled-instruction.
  : Store assembled instruction.
  : Increment address of instruction which stores assembled-instruction.
  : GO TO * ENTRY.

* TERMINATOR L: Fetch into K the block-start-address of the current block.
  : GO TO * COLLATE.
7.1 PROGRAMMED-OPERATORS

7.1.0 Programming Arrangements Using API

It will be recalled from previous descriptions, that the programmed-operator feature is a hardware innovation which facilitates the use of sub-routines and which provides effectively a powerful extension of the machine's repertoire of instructions. Programmed-operators carry out these functions, viz:

(i) PLANT LINK - The incremented sequence counter is stored in location 0.
(ii) STORE OPERAND ADDRESS - The operand address is stored in the U register. (N.B. the operand may be indirectly addressed to any number of levels).
(iii) TRANSFER CONTROL TO SUB-ROUTINE - The next instruction is taken from location 32 plus the number represented by the operation code.

The operations involved in the execution of programmed-operators using the facilities of the assembly program API are illustrated by Fig. 7.0. This figure is concerned with an indirectly addressed programmed-operator which requires a double-length operand. The programmed-operator (21 J 70F) is stored in location 500 of the main program. When a programmed-operator is indirectly addressed, the hardware of ARCTURUS has been designed to treat the requirements of indirect addressing before those of the programmed-operator. Indirect addressing may be carried out to any level (as with a normal machine instruction), and the final effective operand address is stored in the U register. In satisfying the programmed operator requirements of the example represented by Fig. 7.0, the address 501 (the incremented sequence counter) is stored in location 0, and control is transferred to location 21 (sexadecimal).

The use of API requires a directory of starting addresses of the program blocks to be stored in the locations starting from 20 (sexadecimal). If a programmed-operator is used as one of these blocks, an UNCONDITIONAL TRANSFER OF CONTROL instruction is used as its entry in the directory. This instruction causes the computer to obey the programmed operator, which, in the example of Fig. 7.0, is a sub-routine for carrying out some operation on a double-length operand.
The first word of this operand may be obtained by planting the operand address from U into a FETCH instruction; the second word may be obtained using an ADD INDEX instruction. The link back to the main program may be carried out by indirectly addressing location 0.
Figure 7.0: THE OPERATIONS INVOLVED IN THE EXECUTION OF PROGRAMMED-OPERATORS
7.1.1 Examples - Double-length Arithmetic

All double-length programmed-operators (as well as all floating-point programmed-operators) make use of a double-length/floating-point accumulator in memory. For example, the double-length-add-subtract programmed-operator, which is described in Appendix F, causes the double-length number specified by the programmed-operator instruction to be added to or subtracted from the contents of the double-length accumulator. This is an example of a programmed-operator with two entry points. This arrangement is used, of course, to take full advantage of the similarity between the programs for the two individual functions.

One precaution which had to be taken with the above program was that of making the overflow logic ineffective for all intermediate arithmetic operations but effective for the final operation as a safeguard against producing incorrect results.

Another precaution was that of making the arrangements for the "directive print-out" of programmed-operators with multiple entry compatible with those required by API.

A full description of the double-length-multiply programmed operator also appears in the Appendix (Appendix G). This and other double-length programmed-operators have been checked on the machine. However, at the time of writing, a better algorithm for double-length division is being considered. The number of words in these programs are shown below:

- Double-length store : 10 words
- Double-length fetch: 8 words
- Double-length add-subtract: 25 words
- Double-length multiply: 46 words
- Double-length divide : 53 words

7.1.2 Examples - Floating-point Arithmetic

The format of a floating-point number in ARCTURUS is shown below:

```
<table>
<thead>
<tr>
<th>Sign bit</th>
<th>Waste</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>30-bit Normalised Mantissa</td>
<td>8-bit Exponent</td>
</tr>
</tbody>
</table>
```
The mantissa is a normalised, 30-bit fraction represented by the two's complement system. The 8-bit "exponent" is a positive integer, C, which defines the factor $2^{C-128}$ by which the mantissa must be multiplied to obtain the true value of the number.

All floating-point programmed-operators assumed that the operands are in the above format, and they convert results into this format before storing them in memory. The waste bit was used to make the format compatible with that of a double-length number in so far as double-length-store and double-length-fetch programmed operators may be used for floating-point numbers.

The normalisation process in the above programs makes good utilisation of the SKIP IF NORMALISED variant of the SKIP instruction. A SHIFT UNTIL NORMALISED variant of the SHIFT instruction (using the U register to count the number of shifts) has been proposed, but has not yet been incorporated in the hardware. This variant would produce a significant improvement in all floating-point programs.

Much programming experience was gained by writing the floating-point programmed-operators. Many coding tricks making use of the subtleties of the machine design were discovered, and many coding problems suggested worthwhile modifications to the hardware. A full description of all these points is beyond the scope of this section. An example of the floating-point programmed-operators is, however, presented in the Appendix (Appendix H). Although these programs have not yet been checked thoroughly on the machine, an advanced stage has been reached in their development. At present, the number of words in these programs are as follows:

Floating-point-add-subtract-multiply-divide: 143 words
Floating-point-input: 99 words
Floating-point-output: 85 words
7.2 SUB-ROUTINE HIERARCHY

7.2.0 Jump-to-Sub-routine Instructions

In early machines like SILLIAC (Ref. 51) the absolute or relative address of a sub-routine-link had to be planted in one of the machine registers by the programmer before the sub-routine was entered.

In machines like the LGP-30 (Ref. 3) and SNOCOM, which were designed in the late 1950's, instructions like the RETURN ADDRESS instruction were introduced to store the link (e.g. sequence counter plus two) in the location specified by the instruction. This made the sub-routine entry instructions independent of their location, but the UNCONDITIONAL TRANSFER OF CONTROL instruction (for entering the sub-routine) which followed immediately after the RETURN ADDRESS instruction was still necessary.

In machines like ARCTURUS and some commercial machines designed in the early 1960's, instructions like the JUMP-TO-SUB-ROUTINE instruction were introduced to carry out both functions of "planting the link" and "entering the sub-routine". With the use of the JUMP-TO-SUB-ROUTINE instruction in ARCTURUS, the sub-routine entry point immediately follows the location in which the link is stored. On first contact, this may not appear to be an ideal arrangement; however, with indirect addressing, there is no disadvantage, and the instruction has proved to be extremely satisfactory.

7.2.1 Sub-routine Hierarchy Instructions

Computer programs frequently use nested sub-routines. If conventional JUMP-TO-SUB-ROUTINE instructions (described above) are used, each sub-routine is only provided with a link back to the sub-routine of the next higher level. It has been suggested (Ref. 59) that "recursive" operations could be readily programmed, if each sub-routine were provided with links to both the next lower and next higher levels of the sub-routine hierarchy. If these facilities are to be provided, it appears that all sub-routine links should be best stored in some list, and a "hierarchy counter" should be provided to specify the level.
In ARCTURUS, the INCREMENT HIERARCHY, DECREMENT HIERARCHY and RETURN HIERARCHY instructions (see Appendix C) use location 1 as the hierarchy counter. This location contains an address which "points" to one sub-routine link in a list of links. As the hierarchy counter may be incremented or decremented many times, not only do these hierarchy instructions enable sub-routines to return to either a lower or higher level, but also the number of levels traversed may be specified.

The execution of the hierarchy instructions require either three or four memory cycles. Although several interesting problems were encountered in the design of the hardware for these instructions, the amount of hardware was not excessive.

Because of the infrequency of their use, these hierarchy instructions may not save much computing time. However, they were introduced, not for this purpose but for the purpose of stimulating thought on special instructions (in particular those required for recursive operations). It is hoped that this will lead to hardware innovations in later machines.
7.3 SOFTWARE DEVELOPMENT

7.3.0 Present Situation

Descriptions of test programs, assembly programs and programmed-operators have been presented in earlier sections. Apart from these, programs have been written by two research students working for higher degrees. These programs are concerned with:

(i) preliminary studies leading to research work on the processing of recordings from radio telescopes (Ref. 60)

and

(ii) the addition of a remote console to ARCTURUS (Ref. 24)

ARCTURUS has also supported several Electrical Engineering Honours projects. These were concerned with:

(i) a digital differential analyser simulator (Ref. 61)

(ii) a floating-point coding system similar to the A9 system (Ref. 51) used with SILLIAC and the KDF9 (Ref. 36)

(iii) the computer requirements of a remote console (Ref. 62)

and (iv) a logic simulator for testing the design of a digital control system (Ref. 63).

7.3.1 Plans for the Future

Short-range plans for the future include:

(i) the further development of programmed-operators,

(ii) the full development of an A9-type coding system,

(iii) the extension of testing and diagnostic facilities,

(iv) the development of software (together with hardware modifications) to enable a remote console to be added to the machine,

and (v) the development of software (together with hardware modifications) to make efficient use of a recently acquired disk file.

Long-range plans centre around the proposed use of ARCTURUS in a multi-level digital control system. The efficient communication between computers and the control strategies to be used at the various levels of the hierarchical structure will pose many software and hardware problems.
The solutions of these problems will be topics of future research projects.

A computer-bureau-type installation was not the main objective of the ARCTURUS project, as a very good computing service was already being provided by commercial machines within the University environment. The main objective was the construction of a computing installation for research purposes. This could take the form of the evaluation of hardware innovations, the stimulation of research work on computer organisations, and the development of both the hardware and software required by complex digital computer control systems. The support of research projects at the levels of both the Honours Bachelor's degree and the Master's degree has already proved that ARCTURUS is a very useful research tool; and its continued use in this role for projects planned for the Ph.D. level will confirm the fact that the ARCTURUS project has been a success.
CHAPTER 8

REVIEW AND APPRAISAL

8.0 PREFACE TO FINAL CHAPTER

8.1 REVIEW OF COMPUTER DEVELOPMENTS
8.1.0 History and introduction
8.1.1 Logical organisation and systems
8.1.2 Logical design and switching theory
8.1.3 Logical elements and circuits
8.1.4 Memories
8.1.5 Input-output systems
8.1.6 Constructional techniques
8.1.7 Software
8.1.8 Time-sharing
8.1.9 Computer control
8.1.10 Miscellaneous applications and concluding remarks

8.2 APPRAISAL OF SNOCOM, NIMBUS & ARCTURUS PROJECTS
8.2.0 Computer development in Australia
8.2.1 SNOCOM
8.2.2 NIMBUS
8.2.3 ARCTURUS
8.2.4 Concluding remarks
8.0 PREFACE TO FINAL CHAPTER

The objective of this final chapter is to present an appraisal of the computer projects described in this thesis. This is carried out by firstly presenting (in Section 8.1) a review of digital computer developments on a world-wide basis, and secondly discussing (in Section 8.2) some of the features of the SNOCOM, NIMBUS and ARCTURUS projects in relation to the computer environment in which these projects were carried out.

There are nearly 200 books and innumerable papers written on various aspects of digital computer technology. Because the technology is so extensive, an outline (in several pages) of the developments in all aspects of the technology can only be a superficial one. However some sections of the outline represent summaries of experience gained in particular aspects of the technology which had to be studied in depth, and the remaining sections have evolved from numerous journals, reprints, books and notes which were accumulated during the course of the projects.

To place the SNOCOM, NIMBUS and ARCTURUS projects in proper perspective, a brief survey of the development of digital computers in Australia is presented in Section 8.2. An appraisal of these projects is then made.
8.1 REVIEW OF COMPUTER DEVELOPMENTS

8.1.0 History and Introduction (Refs. 68-69)

The early computers which played major roles in the evolution of the modern digital computer include the electro-mechanical Harvard Mark I (which brought Babbage's ideas into being), the relay computers built by the Bell Telephone Laboratories and the first electronic computer ENIAC. Early stored-program computers include the EDVAC, the EDSAC and the IAS computer.

Probably the most significant single concept which resulted in the rapid growth of computer technology was that of the "stored-program computer". Since its development by von Neumann and others in the late 1940's, almost all later computers have utilised this concept. However, there have been some important conceptual refinements which include micro-programmable organisations, push-down memories, associative memories and multi-computer systems.

The developments of computer hardware have seen the shift from "first-generation" computers using vacuum-tubes through "second-generation" computers using discrete semiconductor components to "third-generation" computers using integrated circuits.

Since about 1958 computer technology has made extensive use of three major components: the magnetic core, the transistor and the magnetic surface (in the form of drums, tapes and discs). Other components have been used, but at present there are no serious contenders for the roles played by the above three.

The development of computer software has seen the trend towards the increased use of "problem-oriented languages". These have been written for commercial, scientific and other applications, and have resulted in a significant increase in "programmer productivity".

More details of the above aspects of digital computer technology, viz: systems, hardware, software and applications, are presented in following sections. They are treated in approximately this order.

8.1.1 Logical Organisation and Systems (Refs. 70-73)

Since the early 1950's there have been many developments in the logical organisation of digital computers. These have resulted in an increase in the power and speed of computers, a simplification of programming, and an extension of the computer from a computational tool to a
A wide variety of word-lengths, word formats and instruction repertoires are to be found in today's computers. Many contain features which enable numerous different arithmetic and logical operations to be carried out on fixed and floating-point numbers, bits, bytes, data fields and lists. Many of the more sophisticated features are first found only in the large computers, but as these features become established, the trend is towards the incorporation of modest versions of these features in the smaller machines.

Various forms of memory addressing techniques (including indirect and relative addressing) are in use, and much research effort is being directed at content-addressable or associative memories for information storage and retrieval and other applications.

For the specification of instruction sequencing many forms of branching instructions (such as the skip instruction in ARCTURUS) have been used, and features have been introduced for the efficient programming of loops and sub-routines. For real-time applications a priority interrupt system is important.

Microprogramming was proposed as an efficient and flexible means of control for a digital computer. Research effort directed at the production of cheap and fast read-only stores has resulted in this method of control being adopted for some commercial computers (including the lower range of the IBM System/360). The use of a fast-read-slow-write control-memory has resulted in several program-modifiable or stored-logic computers.

As the mismatch between fast central-processing-units and slow peripheral devices increased, the advantages of multiprogramming became apparent. To produce efficient multiprogramming features such as priority interrupts, memory protection, real-time clocks and program relocation were introduced.

As new logical and memory devices are developed and hardware costs are reduced, computer designers will take full advantage of these advances in the technology, and the trend will inevitably be the production of faster, more powerful, cheaper and more reliable computers. The mastery of the batch fabrication of logical and memory elements will result in a radical change in computer organisations. Networks of a large number of interconnected computers (which have already been proposed) may then represent another major break-through in computer technology.
8.1.2 Logical Design and Switching Theory (Refs. 74-77, 48)

There have been a number of clever methods for speeding up the arithmetic operations in digital computers. Examples of these are the "carry-lookahead" and "multiplier-recoding" methods described in an earlier chapter. With the exclusion of the smaller computers, most computers now have floating-point hardware. In general, no macro-instructions such as square-root or sine are provided, as rapidly converging algorithms for these functions are available.

Boolean algebra has been found to be adequate for the specification of combinational circuits. Many Boolean minimisation techniques, such as those for minimising contact springs, diodes, gates, feedback loops, states, hazards, inverters and logical levels, have been applied. However, even the best computerized techniques (which are far more powerful than the well known Quine method) can only handle 20 to 30 variables because of storage and computing time limitations. Hence the classical minimisation techniques are inadequate for many complex design problems in existing large computers.

The mathematical model of the digital computer as a complex sequential switching circuit may be useful for small machines. However the application of the state diagram approach to the design of large systems is limited because the individual steps, such as the state assignment for circuit minimisation, are complex and an exhaustive treatment (even by computers) is prohibitive.

It appears that switching theory is adequate for small logical design problems, but large systems cannot be synthesised by a direct application of this theory. For large systems, the designer must be guided by the principles established by the switching theory, but often experience and intuition play important roles, and a useful approach is the partitioning of the system into a number of interconnected, readily-designable sections.

8.1.3 Logical Elements and Circuits (Refs. 78-81)

Since the invention of the transistor in 1948, and its subsequent introduction into commercial equipment around 1954, semiconductor technology has progressed so rapidly that semiconductor devices are now being used in high-speed switching circuits that require not only transition times as short as one nanosecond but also repetition rates above 100 MHz.
Other logical elements have been used for various reasons; for example magnetic logical elements because of their reliability, magnetic thin films, cryogenic devices and tunnel diodes because of their speed and so on. However the transistor either as a discrete component or as part of an integrated circuit still remains the most important logical element for digital computers.

Integrated electronics technology, which was given substantial support because of the requirements of the aerospace industry, has made a great impact on digital computers.

As well as being fast, small, cheap, low-power-consumption logical elements, integrated circuits can provide the designer with more powerful functional units such as full adder stages and shift registers. As nanowatt devices are feasible, and as many as 1000 logic circuits may go on a single monolithic integrated circuit chip and switch at subnanosecond speeds, integrated microelectronic circuits provide a great promise for the development of more powerful computers.

Other logical elements which have some promise for computers of the future include batch fabricated thin film elements and optical logical elements. The latter may solve some of the problems of interconnecting fast logical elements.

8.1.4 Memories (Refs. 82-85)

The types of memories used in the early computers include the mercury delay-line, the cathode-ray-tube store and the magnetic drum. From these early forms of memories, other types (notably the ferrite core memory) have emerged with greater speeds and capacities, and have played major roles in the development of the modern digital computer.

Most large systems use a hierarchy of storage in which a combination of different types of storage techniques are used to achieve the necessary capacity and speed at a reasonable cost. The present state-of-the-art is represented by the following:

- Magnetic thin-film scratch-pad memories of 2500 to 2X10^5 bit capacity have been constructed with a cycle time of 100 ns to 500 ns at a cost of $0.50 to $2.00 per bit; magnetic core main memories with capacities of 10,000 to 2X10^6 bits have been constructed with 0.7 µs to 4 µs cycle time at a cost of 5¢ to 25¢ per bit; and magnetic disc file auxiliary storage with capacities of 2X10^7 to 2X10^9 bits have been constructed with 15 ms to 150 ms access time at a cost of 0.01¢ to 0.2¢ per bit.
While block transfer of information is usually adopted between auxiliary (backing) stores and the main internal memory, additional hardware can be designed to produce an effective one-level storage system. An example is the "one-level" core-drum system used in ATLAS.

Monolithic integrated circuit arrays (as high-speed control or scratch-pad memories) and magnetic thin film main internal memories have already been used in commercial computers. Further improvements in magnetic core technology and batch fabrication techniques will result in higher performance scratch-pad and main memories of the near future. However it does appear that moving-magnetic-media type electromechanical memories will still be used for auxiliary stores with capacities in excess of $10^9$ bits.

8.1.5 Input-output systems (Refs. 86-88, 56)

Conventional input-output devices for computers include paper tape, punched cards, magnetic tape, typewriters and line printers. More recent or more specialised devices include non-impact printers, visual character display consoles, optical character readers, magnetic-ink character recognition systems and graph plotters.

Graphical devices such as Sutherland's Sketchpad have significantly improved man-machine communication, and have introduced a new generation of extremely promising input-output devices. Graphical devices have been used for automotive styling, for the design of electronic circuits and civil engineering structures and for computer-assisted instruction. It appears that it is in the area of "machine-aided design" that graphical man-computer communication devices hold the greatest promise.

Research work on the computer recognition of speech and handwriting have been carried out, but it does appear that useful operating systems will not be available in the foreseeable future.

8.1.6 Constructional Techniques (Refs. 89-92)

Modular construction for complex systems is of course necessary because of manufacturing, installation and maintenance requirements. Discrete semiconductor components are usually mounted on a printed circuit card, and the cards are interconnected via the printed circuit connectors by the base wiring of the main frame. The base wiring is carried out by hand on a point-to-point basis or by some automatic wire-wrapping machine. To reduce the number of the less-reliable printed circuit connectors, individual printed circuits are sometimes
connected permanently to a mother-board to form larger functional units.

When integrated circuits are mounted on a mother-board, multi-layer printed interconnections are often used. As integrated circuits become faster (with switching times of the order of several nanoseconds), the effects of the interconnections on the operation of the circuits become extremely important. The interconnections must be considered as transmission lines and all lines must be properly terminated to minimise signal reflections. When the layout of the integrated circuits is important, computers can be used to examine all possible layouts to minimise wiring delays and crosstalk.

Trends in constructional techniques point to an increase in the number of circuit functions per chip and possibly the use of a "three dimensional" interconnection technique.

8.1.7 Software (Refs. 93-94)

Automatic programming, which may be defined as all those methods which attempt to shift the burden of formulation and programming of problems for computers onto the machine themselves, has evolved from various assembly programs, interpretive programs and machine-oriented languages to the powerful problem-oriented languages in use today. It has been estimated that by the use of FORTRAN, programmer productivity is increased by a factor of 50 over that resulting from the use of machine language; and a further improvement by a factor of 10 or more is still possible.

Since its inception in 1954, FORTRAN has had extensive use (especially in the United States) as a scientific programming language. However ALGOL is gaining wide-spread acceptance as a common, international programming language for mathematical problems, and is being used extensively as a medium of human communication of algorithms in published form.

Other problem-oriented languages include COBOL for commercial applications, SOL for systems simulation, LISP for list processing, FORMAC for the manipulation of mathematical expressions, PERT for planning and controlling large military and engineering programs, LOTIS for simulating digital systems and DAS for simulating analogue computers. The above examples represent only a small percentage of the program-oriented languages in use today but give some indication of their diversity.
To cater for the diverse requirements of a larger percentage of the users of a scientific computing centre, IBM has introduced PL/1. The potential and scope of this language together with the ability to use subsets of the language will ensure its widespread use for many years to come.

8.1.8 Time-sharing (Refs. 95-97,22)

Aspects of computer utilisation which are associated with time-sharing include multi-access systems, multiprocessing, multiprogramming, real-time processing, interactive (on-line) processing and remote processing. In general the main objective is the sharing of computing resources among a number of users. Systems which contain a large centralized computer or a multiprocessor network which is connected to a large number of remote user terminals may be conveniently described as computer "utilities" which provide adjustable amounts of computing power to individual users whenever it is required.

Not only is the potential of the entire central processor made available to individual users, but also time-sharing provides an "interactive", "personalized" computer usage. Greater efficiency results from a more intimate collaboration between the computer and an individual user, or between the computer and several users working on different aspects of a common problem.

Some of the requirements of time-shared systems include dynamic program relocation, memory protection, priority interrupts, large memories, complex scheduling logic, reliable communications equipment and so on. These represent enormous hardware and software costs to system planners. However these costs are spread over many users, and the many advantages of time-shared systems have created a trend towards centralization. If Grosch's law (which states that a computer's potential is proportional to the square of its cost) is true, the computer cost structure will favour the time-shared large machine in preference to a large number of small, decentralised machines.

8.1.9 Computer Control (Refs. 98-101)

Since computers were first used for industrial control about eight years ago, the world-wide tally of process computer installations has grown to over 1,300. Computer control has had its longest history in the oil, chemical, steel and power industries, but its application to other industrial processes is progressing very rapidly.
Various approaches taken with the use of computers in control systems include data acquisition systems, closed-loop supervisory computer control, direct digital control and direct digital supervision. A hierarchical structure of control computers has been suggested for some complex systems. Whatever the approach taken, the main objective is of course to improve efficiency. As anticipated improvements are seldom greater than five per cent, and the cost of the control computer together with sensing and control equipment represents a significant proportion of overall costs, planning of the overall system must be carried out very carefully.

The computer requirements for real-time operation include priority interrupts and special input-output instructions. However the instruction repertoire of "control computers" is essentially the same as that of any general-purpose computer used for scientific computation. Reliability is of paramount importance in most computer control systems. Redundancy techniques applied even to the extent of providing parallel computing systems have been used to increase reliability. "Back-up" systems in case of failure include the use of manual or analogue controllers. The failure of some early computer control systems (especially in the power supply industry), which were due to unreliable hardware and software, has resulted in computer control being applied with more caution and with a greater emphasis on reliability.

In the computer control field, the relationship between theory and practice is remote. Sophisticated theories and mathematical techniques such as sampled-data theory, dynamic programming and optimal control theory are available, but there seems to be many difficulties in the application of these theories to the analysis, little own the synthesis, of computer control systems. Until there is a stronger bond between theory and practice, computer control will remain an art rather than a science.

8.1.10 Miscellaneous Applications and Concluding Remarks
(Ref. 102)

Computers are extremely versatile as indicated by their use in science where they have been active participants in the analysis of results and in the development of
new theories, in technology where they have become indis­
pensable for the design and control of complex systems, in
organisations where they have provided improved communication
and control resulting in increased efficiency, in education
where they have provided personalised tuition adjusted
to the capabilities of individual students, in information
storage and retrieval where they have improved the efficiency
of libraries and have assisted in research and development,
and in the field of artificial intelligence where they have
provided an understanding of human mental processes. As
the potential of computers increases because of reduced
cost, increased speed, increased reliability, increased
capacity and improved man-machine communication, their
application areas will certainly increase. Many benefits
to humanity have resulted from the use of computers, and
continued progress in computer technology holds great promise
for the future.
8.2 APPRAISAL OF SNOCOM, NIMBUS & ARCTURUS PROJECTS

8.2.0 Computer Development in Australia

Apart from SNOCOM, NIMBUS and ARCTURUS, the only other digital computers constructed in Australia were CSIRAC (Ref. 64), SILLIAC (Ref. 51), ADA (Ref. 1), ATROPOS (Ref. 65) and CIRRUS (Ref. 25-26).

CSIRAC was a serial, general-purpose digital computer using vacuum-tube elements and mercury delay-line storage. It was constructed in the early 1950's by the Radio Physics Laboratory of C.S.I.R.O., and although it suffered from problems of unreliability during its early years, it gave good service within C.S.I.R.O. and at the University of Melbourne before it was retired several years ago.

SILLIAC is a parallel, general-purpose digital computer using vacuum-tube elements and cathode-ray-tube storage. Its design was based on that of the ILLIAC with minor modifications, and was constructed under the supervision of the Basser Computing Department within the School of Physics of the University of Sydney. It was commissioned in 1956 and has given sterling service; its retirement is imminent.

ADA was a serial, digital differential analyser using semiconductor circuits and magnetic-drum storage. It was constructed by the Mathematical Instruments Section of C.S.I.R.O. within the School of Electrical Engineering of the University of Sydney. It was commissioned in 1958, but because of unreliable operation, it was given a premature retirement several years later.

ATROPOS was a parallel, digital computer using semiconductor circuits and ferrite-core storage. It was constructed as an impact-prediction computer by the Weapons Research Establishment. Papers on its design appeared around 1960, but its usefulness and future are not known.

CIRRUS is a parallel, general-purpose digital computer using semiconductor circuits and ferrite-core storage. The computer, which uses a micro-program control unit and has multi-program features, was constructed by the University of Adelaide. It was completed by 1963, and it has had extremely good use as a service computer.
8.2.1 SNOCOM

SNOCOM was the first general-purpose digital computer constructed and installed in Australia which utilised semiconductor components and which has had extensive use. Its construction demonstrated that a useful computer can be built in Australia starting from the description of the logical organisation of a computer which appeared in a single publication. It was also demonstrated that useful features, such as the auto-input, could be designed to fit within the framework of this organisation.

SNOCOM established the suitability of constructional methods which were developed for ADA, and which were later used and extended in ARCTURUS and CIRRUS. It also established the potential (and also some weakness) of circuit designs. These were used as a guide to the development of the faster circuits used in ARCTURUS.

In the course of the SNOCOM project, a SILLIAC program for simulating SNOCOM was written so that SNOCOM programs could be developed even before this machine was completed. Extensions to the simulator program enabled proposed SNOCOM design modifications to be assessed (Ref. 66). The SNOCOM project also stimulated some thought on machine organisation and order codes (Ref. 67).

The success of the SNOCOM project was directly responsible for the continuation of computer research and development at the School of Electrical Engineering of the University of Sydney.

8.2.2 NIMBUS

NIMBUS is the first educational computer of its kind to be built in Australia (and possibly in the world). It represents the first serious attempt to build laboratory equipment for teaching computer fundamentals and digital techniques. Its usefulness in this role has been established during the past four years; and this has encouraged the extension of this educational approach.

It has been established that NIMBUS and similar machines are extremely useful research tools. Applications include the evaluation of logical circuit designs and the bread-board construction of special test equipment.

The success of the NIMBUS project has resulted in the decision to build a similar but more powerful machine (NIMBUS II).
8.2.3 ARCTURUS

With the use of advanced techniques of logical design such as carry-lookahead and multiplier-recoding which produce an accumulate time of 1μsec and a multiply time (for 20 bits) of 5 to 7μsec (depending on the multiplier, the arithmetic unit of ARCTURUS is the fastest of all computer arithmetic units constructed in Australia.

Techniques for the provision of a large number of useful instruction variants (such as those of the register transfer instruction) have been established and a computer with a flexible and reasonably powerful instruction repertoire has been produced.

The ARCTURUS project has demonstrated that relatively advanced concepts such as programmed-operators can be implemented with small amounts of hardware, and it has also demonstrated the usefulness of such concepts.

Aspects of the ARCTURUS design, such as the auto-input and the register configuration, have made some contribution to the solutions of logical organisation problems which affect overall efficiency.

The ARCTURUS project has established methods for designing complex timing units. Configurations of gated monostable-multis (of both positive-trigger and negative-trigger types) and storage-gate flip-flops have been established for timing the micro-steps such as shifting and counting etc., as well as the complex phase sequences which are possible in a digital computer.

The ARCTURUS project has established the usefulness of the completely autonomous input-output unit which can be used for off-line tape editing as well as for checking the computer peripherals.

The ARCTURUS project has made some contribution to the art of constructing systems which use very high-speed logical circuits. Functional packaging, ground planes and special mounting arrangements for important signal lines are examples of the techniques established.

ARCTURUS has stimulated thought on logical organisations. Many suggestions concerning useful instructions (such as the detour, set and repeat instructions) have been made. Some of these suggestions have already been incorporated into the machine and their usefulness has been demonstrated. However, studies of the proposed "micro-mode" and of the existing hierarchy instructions have indicated that these features are not extremely useful in their present form.
ARCTURUS provides a research tool for advanced work on digital control, and plans for the control of a model power system have already been considered. For this application ARCTURUS will be more useful than a commercial machine of comparable cost, as extensions and modifications to the machine to incorporate the interface between the system and the computer may be made more readily.

The success of the ARCTURUS project has demonstrated that a very useful computer can be built within the environment of Australian universities with a component cost of less than $A 10,000. This money was obtained over a period of about 5 years, and the very restricted budget necessitated the salvaging of many diodes and transistors from old equipment. Despite the very difficult conditions, a fast and powerful digital computer incorporating some original and advanced concepts has been constructed.

8.2.4 Concluding Remarks

This thesis has been an account of the development of the three computers SNOCOM, NIMBUS and ARCTURUS. The main objectives have been to describe the original features incorporated in the computers, to present examples of new design techniques and to outline some of the novel engineering details of construction, installation and utilisation.

An outcome of the SNOCOM, NIMBUS and ARCTURUS projects has been that a small but effective research group has been formed. This group has shown that it is capable of taking full advantage of progress in computer technology, and of establishing advanced concepts and techniques. The merit of this thesis lies firstly in the fact that the author has been the principle investigator of this research group, and secondly in the fact that the success of the SNOCOM, NIMBUS and ARCTURUS projects has been a significant contribution to the development of computers in Australia.
REFERENCES

INTRODUCTION


CHAPTER 1


References 1, 2, 3 & 4.

CHAPTER 2


References 5, 6 & 7.

CHAPTER 3


References 1, 4, 7, 8 & 10.
CHAPTER 4


46. INTERNATIONAL BUSINESS MACHINES, IBM 1620 Manual.


Reference 7.

CHAPTER 5


References 1 & 4.
CHAPTER 6


CHAPTER 7


References 3, 24, 36 & 51.

CHAPTER 8


67. CHAPPELL, M., (Snowy Mountains Hydro Electric Authority), Personal communication.
CHAPTER 8 - REVIEW


69. KNIGHT, K.E., Changes in Computer Performance, Datamation, September, 1966, pp 40-54.


Reference 48.


Reference 56


97. TIME-SHARING COMPUTERS (SPECIAL REPORT), Electronics, November, 29, 165, pp 71-89.

Reference 22.

98. PROCESS COMPUTER SCOREBOARD, Control Engineering, September 1966, pp 73-82.

99. DIGITAL COMPUTERS IN INDUSTRY-A CONTROL ENGINEERING SPECIAL REPORT, Control Engineering, September 1966, pp 83-142.

100. COMPUTER EQUIPMENT FOR DIRECT DIGITAL CONTROL, (17 papers), 1965 IEEE International Convention Record.


102. SPECIAL ISSUE ON INFORMATION AND ITS PROCESSING BY COMPUTERS, Scientific American, September 1966.


APPENDIX

CONTENTS

APPENDIX A - Component Layout Diagrams -192-

APPENDIX B - Logic Diagrams of Standard Packages -194-

APPENDIX C - Description of ARCTURUS Instructions -199-

APPENDIX D - Coding of the Assembly Program API -203-

APPENDIX E - Details of ARCTURUS Instruction Variants -205-
- ARCTURUS API & Sexadecimal Operation Codes
- Variants of the Stop Instruction
- Variants of the Skip Instruction
- Variants of the Input-Output Instruction
- Printer Code
- Variants of the Shift Instruction
- Variants of the Register Transfer Instruction

APPENDIX F - Double-length Add-Subtract Programmed Operator -211-

APPENDIX G - Double-length Multiply Programmed Operator -213-

APPENDIX H - Floating-point Input Programmed Operator -217-

APPENDIX J - Paper: The Logical Design of the General Purpose Digital Computer SNOCOM

APPENDIX K - Paper: Laboratory Equipment for Teaching Digital Computer Fundamentals

APPENDIX L - Paper: The Design and Construction of the Digital Computer ARCTURUS
Figure 6.9B - COMPONENT LAYOUT OF STORAGE GATE FLIP-FLOP PACKAGE
Figure 6.10 - LOGIC DIAGRAMS OF STANDARD PACKAGES
(D) INVERT CIRCUIT PACKAGE

(E) RESET-SET FLIPFLOP PACKAGE

(F) R REGISTER PACKAGE

(G) J REGISTER PACKAGE

Figure 6.10 - LOGIC DIAGRAMS OF STANDARD PACKAGES
(H) POSITIVE TRIGGER MONOSTABLE MULTI PACKAGE

(I) NEGATIVE TRIGGER MONOSTABLE MULTI PACKAGE

(X indicates an internal connection)

(J) NOR PACKAGE

Figure 6.10 - LOGIC DIAGRAMS OF STANDARD PACKAGES
APPENDIX B

(K) NOR DECODE PACKAGE

(L) AND DECODE PACKAGE

(M) ALTERNATIVE J REGISTER PACKAGE

Figure 6.10 - LOGIC DIAGRAMS OF STANDARD PACKAGES
Figure 6.10 - LOGIC DIAGRAMS OF STANDARD PACKAGES
APPENDIX C

APPENDIX C - DESCRIPTION OF ARCTURUS INSTRUCTIONS

LIST OF SYMBOLS

J Memory output register.
K Accumulator.
L Multiplier-quotient register.
R Instruction register.
S Sequence counter.
U Operand counter.
V Number of times counter.
DFS Distributor function selector.
D Output of DFS.
AS Address selector.
A Address R_7 - R_{19}.
AU Arithmetic Unit.
R_0 - R_{19} Instruction register outputs.
KL Double length register formed by K_0 - K_{19}, L_1 - L_{19}.
N Number of times an operation is executed (specified by R_{15} - R_{19}).
<K> Memory location with address K_7 - K_{19}.
M = <A> Memory location with address R_7 - R_{19}.
I Number increment = 2^{19}.
() Contents of a register or a memory location before an operation.
()' Contents of a register or a memory location after an operation.
(i) Bit i of a register or a memory location before/after an operation.
(i)_{1} Bits i to j (inclusive) of a register or a memory location before/after an operation.
### APPENDIX C

#### DESCRIPTION OF INSTRUCTIONS

<table>
<thead>
<tr>
<th>CODE</th>
<th>INSTRUCTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>STOP</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00 Unconditional</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>01 Contingent on BP1 ON</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10 Contingent on BP2 ON</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11 Continue</td>
<td></td>
</tr>
<tr>
<td>00001</td>
<td>INPUT-OUTPUT</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>4-bits</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>5-bits</td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td>Reader I</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td>Printer</td>
<td></td>
</tr>
<tr>
<td>R13</td>
<td>Punch</td>
<td></td>
</tr>
<tr>
<td>R14</td>
<td>Number of times</td>
<td></td>
</tr>
<tr>
<td>R15-R19</td>
<td>A cyclic left shift of K 4 or 5 places depending on Rg precedes each operation. Reading overwrites K15 or K16 to K19. If Rg = 0, the character read is the next non-fifth bit character in the buffer or on the tape. The character printed or punched is either K16-K19 or K15-K19. Reading precedes printing or punching.</td>
<td></td>
</tr>
<tr>
<td>00010</td>
<td>REGISTER-TRANSFER</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>Transfer L to J first</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>Transfer K to J first</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Transfer U to J first</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Transfer U to J first</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td>If R12 = 0, R9 and R10 specify the</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>first AU input as follows:</td>
<td></td>
</tr>
<tr>
<td>(00 Zero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>One's complement of J</td>
<td></td>
</tr>
<tr>
<td>R9R10</td>
<td>10 J</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>All ones (-I)</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td>If R12 = 0, R11 specifies that an increment is to be added to the adder output.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Add I to adder output</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>If R9 = 0, R10 = 0, R11 = 0, R12 specifies that the first AU input is the absolute value of J.</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td>Absolute value of J</td>
<td></td>
</tr>
<tr>
<td>R13</td>
<td>and R14 specify the second AU input as follows:</td>
<td></td>
</tr>
<tr>
<td>(00 Zero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>R13R14</td>
<td>10 R</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>All ones (-I)</td>
<td></td>
</tr>
<tr>
<td>R15-R16</td>
<td>specify the distributor output as follows:</td>
<td></td>
</tr>
</tbody>
</table>
## APPENDIX C

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>00011</td>
<td>SHIFT</td>
<td>00 Logical, 01 Cyclic, 10 Arithmetic, 11 Illogical, 0 Separate, 1 Double, 0 Shift L, 1 L left, 0 Shift K, 1 K left, R&lt;sub&gt;15&lt;/sub&gt;-R&lt;sub&gt;19&lt;/sub&gt; Number of times</td>
</tr>
<tr>
<td>00100</td>
<td>TRANSFER, UNCONDITIONAL</td>
<td>(S)' = A</td>
</tr>
<tr>
<td>00101</td>
<td>TRANSFER IF NEGATIVE</td>
<td>(S)' = A if K&lt;sub&gt;0&lt;/sub&gt; = 1</td>
</tr>
<tr>
<td>00110</td>
<td>SKIP</td>
<td>R&lt;sub&gt;9&lt;/sub&gt; Accumulator normalized, 00 - 01 Accumulator &lt; 0, 10 Accumulator = 0, 11 Accumulator &gt; 0, 00 - 01 Sense switch 1 ON, 10 Sense switch 2 ON, 11 Sense switch 3 ON, R&lt;sub&gt;15&lt;/sub&gt;-R&lt;sub&gt;19&lt;/sub&gt; Number of times</td>
</tr>
<tr>
<td>00111</td>
<td>(SPARE)</td>
<td></td>
</tr>
<tr>
<td>01000</td>
<td>ADD INDEX</td>
<td>Add (M) to address of next instruction</td>
</tr>
<tr>
<td>01001</td>
<td>SUBTRACT INDEX</td>
<td>Subtract (M) from address of next instruction</td>
</tr>
<tr>
<td>01010</td>
<td>INDEX SKIP</td>
<td>(M)' = (M) + I, (S)' = (S) + 1 if (M)' = 1</td>
</tr>
<tr>
<td>01011</td>
<td>JUMP TO SUBROUTINE</td>
<td>(M)' = (S), (S)' = A + 1</td>
</tr>
<tr>
<td>01100</td>
<td>COMPARE SKIP</td>
<td>(S)' = (S) + 1 if (M)&lt;sub&gt;i&lt;/sub&gt; = (K)&lt;sub&gt;i&lt;/sub&gt; for all i such that (L)&lt;sub&gt;i&lt;/sub&gt; = 1</td>
</tr>
<tr>
<td>Binary Code</td>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>01101</td>
<td>LOAD K</td>
<td>(K)' = (M)</td>
</tr>
<tr>
<td>01110</td>
<td>LOAD L</td>
<td>(L)' = (M)</td>
</tr>
<tr>
<td>01111</td>
<td>ADD</td>
<td>(K)' = (K) + (M)</td>
</tr>
<tr>
<td>10000</td>
<td>SUBTRACT</td>
<td>(K)' = (K) - (M)</td>
</tr>
<tr>
<td>10001</td>
<td>AND</td>
<td>(K)_i = (K)_i \cdot (M)_i (i = 0 - 19)</td>
</tr>
<tr>
<td>10010</td>
<td>MULTIPLY</td>
<td>(KL)' = (L) \times (M)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L_0 = 0</td>
</tr>
<tr>
<td>10011</td>
<td>DIVIDE</td>
<td>(L)' = (KL) \div (M)</td>
</tr>
<tr>
<td>10100</td>
<td>STORE K</td>
<td>(M)' = (K)</td>
</tr>
<tr>
<td>10101</td>
<td>STORE L</td>
<td>(M)' = (L)</td>
</tr>
<tr>
<td>10110</td>
<td>STORE K ADDRESS</td>
<td>(M)_6-19 = (K)_6-19</td>
</tr>
<tr>
<td>10111</td>
<td>ACCUMULATE</td>
<td>(M)' = (M) + (K)</td>
</tr>
<tr>
<td>11000</td>
<td>(SPARE)</td>
<td></td>
</tr>
<tr>
<td>11001</td>
<td>(SPARE)</td>
<td></td>
</tr>
<tr>
<td>11010</td>
<td>PUSH</td>
<td>(&lt;(M)&gt;)' = (K)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(M)' = (M) + I</td>
</tr>
<tr>
<td>11011</td>
<td>POP</td>
<td>(M)' = (M) - I</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(K)' = (&lt;(M)&gt;)</td>
</tr>
<tr>
<td>11100</td>
<td>DECREMENT HIERARCHY</td>
<td>(&lt;(1)&gt;)' = (S)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1)' = (1) - I</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S)' = A</td>
</tr>
<tr>
<td>11101</td>
<td>INCREMENT HIERARCHY</td>
<td>(&lt;(1)&gt;)' = (S)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1)' = (1) + I</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S)' = A</td>
</tr>
<tr>
<td>11110</td>
<td>RETURN HIERARCHY</td>
<td>R_13 { 0 Decrease</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_14 { 1 Increase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_15-R_19 { Change hierarchy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\langle(1)\rangle)' = (S)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1)' = (1) \pm I (N times)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S)' = (&lt;(1)&gt;)</td>
</tr>
<tr>
<td>11111</td>
<td>EXECUTE</td>
<td>Execute instruction in M</td>
</tr>
</tbody>
</table>
## APPENDIX D
### CODING OF THE ASSEMBLY PROGRAM AP1

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>LABEL</th>
<th>INSTRUCTION</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC.</td>
<td>SEXAD.</td>
<td>#</td>
<td>003N6</td>
</tr>
<tr>
<td>967</td>
<td>3N7</td>
<td>00000</td>
<td>Obey instruction</td>
</tr>
<tr>
<td>968</td>
<td>3N8</td>
<td>08082</td>
<td>Read op. code</td>
</tr>
<tr>
<td>969</td>
<td>3N9</td>
<td>10021</td>
<td>(L)' = (K)</td>
</tr>
<tr>
<td>970</td>
<td>3N+</td>
<td>1810F</td>
<td>Sh.L + 14 logical</td>
</tr>
<tr>
<td>971</td>
<td>3N-</td>
<td>18341</td>
<td>Sh.K + 1 logical</td>
</tr>
<tr>
<td>972</td>
<td>3NN</td>
<td>1804F</td>
<td>Sh.K + 14 logical</td>
</tr>
<tr>
<td>973</td>
<td>3NJ</td>
<td>+03N7</td>
<td>Store bits 0-5</td>
</tr>
<tr>
<td>974</td>
<td>3NF</td>
<td>10003</td>
<td>Clear K and L</td>
</tr>
<tr>
<td>975</td>
<td>3NL</td>
<td>08081</td>
<td>Read 1 sexad, char.</td>
</tr>
<tr>
<td>976</td>
<td>3J0</td>
<td>803L7</td>
<td>Subtract 10</td>
</tr>
<tr>
<td>977</td>
<td>3J1</td>
<td>283LJ</td>
<td>Go to * CONVERT</td>
</tr>
<tr>
<td>978</td>
<td>3J2</td>
<td>203F6</td>
<td>Go to 3F6</td>
</tr>
<tr>
<td>979</td>
<td>3J3</td>
<td>683L+</td>
<td>Fetch CR-FS-LC-LC char.</td>
</tr>
<tr>
<td>980</td>
<td>3J4</td>
<td>08444</td>
<td>Print</td>
</tr>
<tr>
<td>981</td>
<td>3J5</td>
<td>503L9</td>
<td>Increment block number</td>
</tr>
<tr>
<td>982</td>
<td>3J6</td>
<td>683L9</td>
<td>Fetch block number</td>
</tr>
<tr>
<td>983</td>
<td>3J7</td>
<td>1804N</td>
<td>Sh.K + 12 logical</td>
</tr>
<tr>
<td>984</td>
<td>3J8</td>
<td>08042</td>
<td>Print block number</td>
</tr>
<tr>
<td>985</td>
<td>3J9</td>
<td>084JL</td>
<td>Read/print program-identifier</td>
</tr>
<tr>
<td>986</td>
<td>3J+</td>
<td>203FF</td>
<td>Go to 3FF</td>
</tr>
<tr>
<td>987</td>
<td>3J-</td>
<td>08082</td>
<td>Read block number</td>
</tr>
<tr>
<td>988</td>
<td>3JN</td>
<td>+03F0</td>
<td>Store temporary</td>
</tr>
<tr>
<td>989</td>
<td>3JJ</td>
<td>6+3F0</td>
<td>Fetch indirect block no.</td>
</tr>
<tr>
<td>990</td>
<td>3JF</td>
<td>883F1</td>
<td>Remove all but address</td>
</tr>
<tr>
<td>991</td>
<td>3JL</td>
<td>203L3</td>
<td>Go to * TERMINATOR F</td>
</tr>
<tr>
<td>992</td>
<td>3F0</td>
<td>00000</td>
<td>Temporary store</td>
</tr>
<tr>
<td>993</td>
<td>3F1</td>
<td>01LLL</td>
<td>Address mask</td>
</tr>
<tr>
<td>994</td>
<td>3F2</td>
<td>02000</td>
<td>Indirect address bit</td>
</tr>
<tr>
<td>995</td>
<td>3F3</td>
<td>10402</td>
<td>(K)' = (L)</td>
</tr>
<tr>
<td>996</td>
<td>3F4</td>
<td>-3N7</td>
<td>Accumulate address</td>
</tr>
<tr>
<td>997</td>
<td>3F5</td>
<td>203N7</td>
<td>Go to * ASSEMBLED-Instruction</td>
</tr>
<tr>
<td>998</td>
<td>3F6</td>
<td>18043</td>
<td>Sh.K + 3 logical</td>
</tr>
<tr>
<td>999</td>
<td>3F7</td>
<td>+03F0</td>
<td>Store temporary</td>
</tr>
<tr>
<td>1000</td>
<td>3F8</td>
<td>10002</td>
<td>Clear K</td>
</tr>
<tr>
<td>1001</td>
<td>3F9</td>
<td>403F0</td>
<td>Add index</td>
</tr>
<tr>
<td>#</td>
<td>Instruction</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----------------------</td>
<td>---------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>3F- * CHARACTER J</td>
<td>Fetch indirect address bit</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>3FN</td>
<td>Accumulate</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>3FJ</td>
<td>Go to * CLEAR</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>3FF</td>
<td>Fetch indirect block no.</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>3FL</td>
<td>Remove all but address</td>
<td></td>
</tr>
<tr>
<td>1008</td>
<td>3L0</td>
<td>Set block-start-address</td>
<td></td>
</tr>
<tr>
<td>1009</td>
<td>3L1</td>
<td>Print block-start-address</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>3L2</td>
<td>Go to * ENTRY</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>3L3 * TERMINATOR F</td>
<td>(K)' = (K) + (L)</td>
<td></td>
</tr>
<tr>
<td>1012</td>
<td>3L4</td>
<td>Add bits 0-6</td>
<td></td>
</tr>
<tr>
<td>1013</td>
<td>3L5</td>
<td>Store assembled-instruction</td>
<td></td>
</tr>
<tr>
<td>1014</td>
<td>3L6</td>
<td>Increment address</td>
<td></td>
</tr>
<tr>
<td>1015</td>
<td>3L7</td>
<td>Number 10</td>
<td></td>
</tr>
<tr>
<td>1016</td>
<td>3L8</td>
<td>Go to * ENTRY</td>
<td></td>
</tr>
<tr>
<td>1017</td>
<td>3L9</td>
<td>Block number</td>
<td></td>
</tr>
<tr>
<td>1018</td>
<td>3L+</td>
<td>CR-FS-LC-LC char.</td>
<td></td>
</tr>
<tr>
<td>1019</td>
<td>3L- * TERMINATOR L</td>
<td>Fetch indirect block no.</td>
<td></td>
</tr>
<tr>
<td>1020</td>
<td>3LN</td>
<td>Go to * COLLATE</td>
<td></td>
</tr>
<tr>
<td>1021</td>
<td>3LJ * CONVERT</td>
<td>(KL)' = 10(L) + (K)</td>
<td></td>
</tr>
<tr>
<td>1022</td>
<td>3LF</td>
<td>Add 10</td>
<td></td>
</tr>
<tr>
<td>1023</td>
<td>3LL</td>
<td>Go to * READ</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>003N8</td>
<td>Start at 3N8</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX E

APPENDIX E - DETAILS OF ARCTURUS INSTRUCTION VARIANTS

- ARCTURUS API AND SEXADECIMAL OPERATION CODES

<table>
<thead>
<tr>
<th>API</th>
<th>INSTRUCTION</th>
<th>SEXADEC. *</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Stop</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>Input-output</td>
<td>08</td>
</tr>
<tr>
<td>02</td>
<td>Register transfer</td>
<td>10</td>
</tr>
<tr>
<td>03</td>
<td>Shift</td>
<td>18</td>
</tr>
<tr>
<td>04</td>
<td>Transfer unconditional</td>
<td>20</td>
</tr>
<tr>
<td>05</td>
<td>Transfer if negative</td>
<td>28</td>
</tr>
<tr>
<td>06</td>
<td>Skip</td>
<td>30</td>
</tr>
<tr>
<td>07</td>
<td>(Spare)</td>
<td>38</td>
</tr>
<tr>
<td>08</td>
<td>Add index</td>
<td>40</td>
</tr>
<tr>
<td>09</td>
<td>Subtract index</td>
<td>48</td>
</tr>
<tr>
<td>0+</td>
<td>Index skip</td>
<td>50</td>
</tr>
<tr>
<td>0-</td>
<td>Jump to subroutine</td>
<td>58</td>
</tr>
<tr>
<td>0N</td>
<td>Compare skip</td>
<td>60</td>
</tr>
<tr>
<td>0J</td>
<td>Load K</td>
<td>68</td>
</tr>
<tr>
<td>0F</td>
<td>Load L</td>
<td>70</td>
</tr>
<tr>
<td>0L</td>
<td>Add</td>
<td>78</td>
</tr>
<tr>
<td>10</td>
<td>Subtract</td>
<td>80</td>
</tr>
<tr>
<td>11</td>
<td>And</td>
<td>88</td>
</tr>
<tr>
<td>12</td>
<td>Multiply</td>
<td>90</td>
</tr>
<tr>
<td>13</td>
<td>Divide</td>
<td>98</td>
</tr>
<tr>
<td>14</td>
<td>Store K</td>
<td>+0</td>
</tr>
<tr>
<td>15</td>
<td>Store L</td>
<td>+8</td>
</tr>
<tr>
<td>16</td>
<td>Store K address</td>
<td>-0</td>
</tr>
<tr>
<td>17</td>
<td>Accumulate</td>
<td>-8</td>
</tr>
<tr>
<td>18</td>
<td>(Spare)</td>
<td>N0</td>
</tr>
<tr>
<td>19</td>
<td>(Spare)</td>
<td>N8</td>
</tr>
<tr>
<td>1+</td>
<td>Push</td>
<td>J0</td>
</tr>
<tr>
<td>1-</td>
<td>Pop</td>
<td>J8</td>
</tr>
<tr>
<td>1N</td>
<td>Decrement hierarchy</td>
<td>F0</td>
</tr>
<tr>
<td>1J</td>
<td>Increment hierarchy</td>
<td>F8</td>
</tr>
<tr>
<td>1F</td>
<td>Return hierarchy</td>
<td>L0</td>
</tr>
<tr>
<td>1L</td>
<td>Execute</td>
<td>L8</td>
</tr>
</tbody>
</table>

* The sexadecimal operation codes given above assume that bits 5-7 of the instruction are zero.
## APPENDIX E

### VARIANTS OF THE STOP INSTRUCTION

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SEXADEC.</th>
<th>AP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconditional stop</td>
<td>00000</td>
<td>00 F</td>
</tr>
<tr>
<td>Stop contingent on BP1 ON</td>
<td>00001</td>
<td>00 1F</td>
</tr>
<tr>
<td>Stop contingent on BP2 ON</td>
<td>00002</td>
<td>00 2F</td>
</tr>
<tr>
<td>Continue</td>
<td>00003</td>
<td>00 3F</td>
</tr>
</tbody>
</table>

### VARIANTS OF THE SKIP INSTRUCTION

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SEXADEC.</th>
<th>AP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skip once if K is normalised</td>
<td>30401</td>
<td>06 1025F</td>
</tr>
<tr>
<td>Skip once if (K) &lt; 0</td>
<td>30101</td>
<td>06 257F</td>
</tr>
<tr>
<td>Skip once if (K) = 0</td>
<td>30201</td>
<td>06 513F</td>
</tr>
<tr>
<td>Skip once if (K) &gt; 0</td>
<td>30301</td>
<td>06 769F</td>
</tr>
<tr>
<td>Skip once if SS1 is ON</td>
<td>30021</td>
<td>06 33F</td>
</tr>
<tr>
<td>Skip once if SS2 is ON</td>
<td>30041</td>
<td>06 65F</td>
</tr>
<tr>
<td>Skip once if SS3 is ON</td>
<td>30061</td>
<td>06 97F</td>
</tr>
</tbody>
</table>
## APPENDIX E

### VARIANTS OF THE INPUT-OUTPUT INSTRUCTION

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SEXADEC.</th>
<th>AP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read one sexadecimal (4-bit) char.</td>
<td>08081</td>
<td>01 129F</td>
</tr>
<tr>
<td>Read one 5-bit char.</td>
<td>08481</td>
<td>01 1153F</td>
</tr>
<tr>
<td>Print one sexadecimal char.</td>
<td>08041</td>
<td>01 65F</td>
</tr>
<tr>
<td>Print one 5-bit char.</td>
<td>08441</td>
<td>01 1089F</td>
</tr>
<tr>
<td>Punch one sexadecimal char.</td>
<td>08021</td>
<td>01 33F</td>
</tr>
<tr>
<td>Punch one 5-bit char.</td>
<td>08421</td>
<td>01 1057F</td>
</tr>
<tr>
<td>Read and print one sexadecimal char.</td>
<td>080N1</td>
<td>01 193F</td>
</tr>
<tr>
<td>Read and print one 5-bit char.</td>
<td>084N1</td>
<td>01 1217F</td>
</tr>
<tr>
<td>Read and punch one sexadecimal char.</td>
<td>080 N1</td>
<td>01 161F</td>
</tr>
<tr>
<td>Read and Punch one 5-bit char.</td>
<td>084 N1</td>
<td>01 1185F</td>
</tr>
<tr>
<td>Read, print and punch one sexadecimal char.</td>
<td>080F1</td>
<td>01 225F</td>
</tr>
<tr>
<td>Read, print and punch one 5-bit char.</td>
<td>084F1</td>
<td>01 1249F</td>
</tr>
<tr>
<td>Print and punch one sexadecimal char.</td>
<td>08061</td>
<td>01 97F</td>
</tr>
<tr>
<td>Print and Punch one 5-bit char.</td>
<td>08461</td>
<td>01 1121F</td>
</tr>
</tbody>
</table>
## APPENDIX E
### VARIANTS OF THE SHIFT INSTRUCTION

<table>
<thead>
<tr>
<th>L</th>
<th>Logical</th>
<th>I</th>
<th>Illogical</th>
<th>A</th>
<th>Arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Cyclic</td>
<td>S</td>
<td>Separate</td>
<td>D</td>
<td>Double</td>
</tr>
<tr>
<td>O</td>
<td>Zero places</td>
<td>+</td>
<td>Left</td>
<td>-</td>
<td>Right</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VARIANT</th>
<th>SEXAD.</th>
<th>API</th>
<th>INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH L S K+ 0</td>
<td>18040</td>
<td>03 0064</td>
<td>+98368</td>
</tr>
<tr>
<td>SH L S K- 0</td>
<td>18060</td>
<td>03 0096</td>
<td>+98400</td>
</tr>
<tr>
<td>SH L S L+ 0</td>
<td>18100</td>
<td>03 0256</td>
<td>+98560</td>
</tr>
<tr>
<td>SH L S L- 0</td>
<td>18180</td>
<td>03 0334</td>
<td>+98668</td>
</tr>
<tr>
<td>SH L S L+ K+ 0</td>
<td>18140</td>
<td>03 0320</td>
<td>+98624</td>
</tr>
<tr>
<td>SH L S L+ K- 0</td>
<td>18160</td>
<td>03 0352</td>
<td>+98656</td>
</tr>
<tr>
<td>SH L S L- K+ 0</td>
<td>18100</td>
<td>03 0448</td>
<td>+98752</td>
</tr>
<tr>
<td>SH L S L- K- 0</td>
<td>18110</td>
<td>03 0480</td>
<td>+98794</td>
</tr>
<tr>
<td>SH L D K+ 0</td>
<td>18240</td>
<td>03 0576</td>
<td>+98832</td>
</tr>
<tr>
<td>SH L D K- 0</td>
<td>18260</td>
<td>03 0696</td>
<td>+98912</td>
</tr>
<tr>
<td>SH L D L+ 0</td>
<td>18300</td>
<td>03 0768</td>
<td>+99000</td>
</tr>
<tr>
<td>SH L D L- 0</td>
<td>18330</td>
<td>03 0996</td>
<td>+99200</td>
</tr>
<tr>
<td>SH L D L+ K+ 0</td>
<td>18340</td>
<td>03 0932</td>
<td>+99136</td>
</tr>
<tr>
<td>SH L D L+ K- 0</td>
<td>18360</td>
<td>03 0964</td>
<td>+99168</td>
</tr>
<tr>
<td>SH L D L- K+ 0</td>
<td>18350</td>
<td>03 0992</td>
<td>+99264</td>
</tr>
<tr>
<td>SH L D L- K- 0</td>
<td>18370</td>
<td>03 1030</td>
<td>+99336</td>
</tr>
<tr>
<td>SH I S K+ 0</td>
<td>18400</td>
<td>03 1136</td>
<td>+101440</td>
</tr>
<tr>
<td>SH I S K- 0</td>
<td>18460</td>
<td>03 1163</td>
<td>+101472</td>
</tr>
<tr>
<td>SH I S L+ 0</td>
<td>18100</td>
<td>03 1230</td>
<td>+101632</td>
</tr>
<tr>
<td>SH I S L- 0</td>
<td>18180</td>
<td>03 1266</td>
<td>+101760</td>
</tr>
<tr>
<td>SH I S L+ K+ 0</td>
<td>18140</td>
<td>03 1330</td>
<td>+101696</td>
</tr>
<tr>
<td>SH I S L+ K- 0</td>
<td>18160</td>
<td>03 1364</td>
<td>+101728</td>
</tr>
<tr>
<td>SH I S L- K+ 0</td>
<td>18100</td>
<td>03 1350</td>
<td>+103856</td>
</tr>
<tr>
<td>SH I D K+ 0</td>
<td>18140</td>
<td>03 1448</td>
<td>+103952</td>
</tr>
<tr>
<td>SH I D K- 0</td>
<td>18160</td>
<td>03 1480</td>
<td>+103992</td>
</tr>
<tr>
<td>SH I D L+ 0</td>
<td>18180</td>
<td>03 1576</td>
<td>+104144</td>
</tr>
<tr>
<td>SH I D L- 0</td>
<td>18200</td>
<td>03 1600</td>
<td>+104224</td>
</tr>
<tr>
<td>SH I D L+ K+ 0</td>
<td>18140</td>
<td>03 1664</td>
<td>+104256</td>
</tr>
<tr>
<td>SH I D L+ K- 0</td>
<td>18160</td>
<td>03 1700</td>
<td>+104288</td>
</tr>
<tr>
<td>SH I D L- K+ 0</td>
<td>18100</td>
<td>03 1736</td>
<td>+104320</td>
</tr>
<tr>
<td>SH I D L- K- 0</td>
<td>18120</td>
<td>03 1768</td>
<td>+104352</td>
</tr>
<tr>
<td>SH A S K+ 0</td>
<td>18240</td>
<td>03 1832</td>
<td>+104512</td>
</tr>
<tr>
<td>SH A S K- 0</td>
<td>18260</td>
<td>03 1860</td>
<td>+104544</td>
</tr>
<tr>
<td>SH A S L+ 0</td>
<td>18280</td>
<td>03 1928</td>
<td>+104680</td>
</tr>
<tr>
<td>SH A S L- 0</td>
<td>18300</td>
<td>03 2016</td>
<td>+104752</td>
</tr>
<tr>
<td>SH A S L+ K+ 0</td>
<td>18240</td>
<td>03 2064</td>
<td>+104832</td>
</tr>
<tr>
<td>SH A S L+ K- 0</td>
<td>18260</td>
<td>03 2112</td>
<td>+104864</td>
</tr>
<tr>
<td>SH A S L- K+ 0</td>
<td>18280</td>
<td>03 2160</td>
<td>+104900</td>
</tr>
<tr>
<td>SH A S L- K- 0</td>
<td>18300</td>
<td>03 2204</td>
<td>+105024</td>
</tr>
<tr>
<td>SH A D K+ 0</td>
<td>18340</td>
<td>03 2272</td>
<td>+105120</td>
</tr>
<tr>
<td>SH A D K- 0</td>
<td>18360</td>
<td>03 2312</td>
<td>+105152</td>
</tr>
<tr>
<td>SH A D L+ 0</td>
<td>18380</td>
<td>03 2380</td>
<td>+105192</td>
</tr>
<tr>
<td>SH A D L- 0</td>
<td>18400</td>
<td>03 2424</td>
<td>+105224</td>
</tr>
<tr>
<td>SH A D L+ K+ 0</td>
<td>18340</td>
<td>03 2480</td>
<td>+105264</td>
</tr>
<tr>
<td>SH A D L+ K- 0</td>
<td>18360</td>
<td>03 2528</td>
<td>+105296</td>
</tr>
<tr>
<td>SH A D L- K+ 0</td>
<td>18380</td>
<td>03 2576</td>
<td>+105336</td>
</tr>
<tr>
<td>SH A D L- K- 0</td>
<td>18400</td>
<td>03 2624</td>
<td>+105368</td>
</tr>
<tr>
<td>SH C S K+ 0</td>
<td>18440</td>
<td>03 2696</td>
<td>+105496</td>
</tr>
<tr>
<td>SH C S K- 0</td>
<td>18460</td>
<td>03 2736</td>
<td>+105528</td>
</tr>
<tr>
<td>SH C S L+ 0</td>
<td>18500</td>
<td>03 2832</td>
<td>+105664</td>
</tr>
<tr>
<td>SH C S L- 0</td>
<td>18520</td>
<td>03 2872</td>
<td>+105696</td>
</tr>
<tr>
<td>SH C S L+ K+ 0</td>
<td>18540</td>
<td>03 2936</td>
<td>+105720</td>
</tr>
<tr>
<td>SH C S L+ K- 0</td>
<td>18560</td>
<td>03 2976</td>
<td>+105752</td>
</tr>
<tr>
<td>SH C S L- K+ 0</td>
<td>18580</td>
<td>03 3032</td>
<td>+105800</td>
</tr>
<tr>
<td>SH C S L- K- 0</td>
<td>18600</td>
<td>03 3072</td>
<td>+105832</td>
</tr>
<tr>
<td>SH C D K+ 0</td>
<td>18640</td>
<td>03 3144</td>
<td>+105944</td>
</tr>
<tr>
<td>SH C D K- 0</td>
<td>18660</td>
<td>03 3184</td>
<td>+105976</td>
</tr>
<tr>
<td>SH C D L+ 0</td>
<td>18700</td>
<td>03 3240</td>
<td>+106064</td>
</tr>
<tr>
<td>SH C D L- 0</td>
<td>18720</td>
<td>03 3280</td>
<td>+106096</td>
</tr>
<tr>
<td>SH C D L+ K+ 0</td>
<td>18740</td>
<td>03 3344</td>
<td>+106128</td>
</tr>
<tr>
<td>SH C D L+ K- 0</td>
<td>18760</td>
<td>03 3384</td>
<td>+106160</td>
</tr>
<tr>
<td>SH C D L- K+ 0</td>
<td>18780</td>
<td>03 3440</td>
<td>+106200</td>
</tr>
<tr>
<td>SH C D L- K- 0</td>
<td>18800</td>
<td>03 3480</td>
<td>+106232</td>
</tr>
<tr>
<td>SH C D K+ 0</td>
<td>18840</td>
<td>03 3544</td>
<td>+106320</td>
</tr>
<tr>
<td>SH C D K- 0</td>
<td>18860</td>
<td>03 3584</td>
<td>+106352</td>
</tr>
<tr>
<td>SH C D L+ 0</td>
<td>18880</td>
<td>03 3648</td>
<td>+106416</td>
</tr>
<tr>
<td>SH C D L- 0</td>
<td>18900</td>
<td>03 3688</td>
<td>+106448</td>
</tr>
<tr>
<td>SH C D L+ K+ 0</td>
<td>18920</td>
<td>03 3752</td>
<td>+106480</td>
</tr>
<tr>
<td>SH C D L+ K- 0</td>
<td>18940</td>
<td>03 3792</td>
<td>+106512</td>
</tr>
<tr>
<td>SH C D L- K+ 0</td>
<td>18960</td>
<td>03 3840</td>
<td>+106560</td>
</tr>
<tr>
<td>SH C D L- K- 0</td>
<td>18980</td>
<td>03 3880</td>
<td>+106592</td>
</tr>
</tbody>
</table>
APPENDIX E

VARIANTS OF THE REGISTER TRANSFER INSTRUCTION

For L = Add 1
For K = Add 2
For U = Add 4
For KLU = Add 7

LOGICAL OPERATIONS

<table>
<thead>
<tr>
<th>VARIANT</th>
<th>SEXAD</th>
<th>AP1</th>
<th>INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10013</td>
<td>02 0024</td>
<td>+065560</td>
</tr>
<tr>
<td>1</td>
<td>10613</td>
<td>02 1560</td>
<td>+067096</td>
</tr>
<tr>
<td>U</td>
<td>11413</td>
<td>02 5144</td>
<td>+070690</td>
</tr>
<tr>
<td>L</td>
<td>10413</td>
<td>02 1043</td>
<td>+065634</td>
</tr>
<tr>
<td>K</td>
<td>10n13</td>
<td>02 3096</td>
<td>+068632</td>
</tr>
<tr>
<td>NU</td>
<td>10113</td>
<td>02 4652</td>
<td>+070163</td>
</tr>
<tr>
<td>NL</td>
<td>10213</td>
<td>02 0536</td>
<td>+066072</td>
</tr>
<tr>
<td>NK</td>
<td>10+13</td>
<td>02 2594</td>
<td>+068120</td>
</tr>
<tr>
<td>U*K</td>
<td>11430</td>
<td>02 5163</td>
<td>+070704</td>
</tr>
<tr>
<td>L*K</td>
<td>10430</td>
<td>02 1072</td>
<td>+066608</td>
</tr>
<tr>
<td>NU*K</td>
<td>11230</td>
<td>02 4656</td>
<td>+070192</td>
</tr>
<tr>
<td>NL*K</td>
<td>10230</td>
<td>02 0560</td>
<td>+066096</td>
</tr>
<tr>
<td>U*K</td>
<td>11423</td>
<td>02 5160</td>
<td>+070696</td>
</tr>
<tr>
<td>L*K</td>
<td>10423</td>
<td>02 1064</td>
<td>+066600</td>
</tr>
<tr>
<td>NL*K</td>
<td>11223</td>
<td>02 4643</td>
<td>+070184</td>
</tr>
<tr>
<td>U/L</td>
<td>11438</td>
<td>02 5176</td>
<td>+070712</td>
</tr>
<tr>
<td>L/K</td>
<td>10438</td>
<td>02 1080</td>
<td>+066616</td>
</tr>
<tr>
<td>NU/K</td>
<td>11238</td>
<td>02 4664</td>
<td>+070200</td>
</tr>
<tr>
<td>NL/K</td>
<td>10238</td>
<td>02 0563</td>
<td>+066104</td>
</tr>
</tbody>
</table>

ARITHMETIC OPERATIONS

<table>
<thead>
<tr>
<th>VARIANT</th>
<th>SEXAD</th>
<th>AP1</th>
<th>INTEGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10000</td>
<td>02 0000</td>
<td>+065536</td>
</tr>
<tr>
<td>+1</td>
<td>10100</td>
<td>02 0256</td>
<td>+065732</td>
</tr>
<tr>
<td>-1</td>
<td>10600</td>
<td>02 1536</td>
<td>+067072</td>
</tr>
<tr>
<td>-2</td>
<td>10660</td>
<td>02 1632</td>
<td>+067168</td>
</tr>
<tr>
<td>MU</td>
<td>11080</td>
<td>02 4224</td>
<td>+069760</td>
</tr>
<tr>
<td>+U</td>
<td>11400</td>
<td>02 5120</td>
<td>+070656</td>
</tr>
<tr>
<td>-U</td>
<td>11300</td>
<td>02 4364</td>
<td>+070400</td>
</tr>
<tr>
<td>ML</td>
<td>10030</td>
<td>02 0128</td>
<td>+065664</td>
</tr>
<tr>
<td>+L</td>
<td>10400</td>
<td>02 1024</td>
<td>+066560</td>
</tr>
<tr>
<td>-L</td>
<td>10300</td>
<td>02 0763</td>
<td>+066304</td>
</tr>
<tr>
<td>MK</td>
<td>10380</td>
<td>02 2176</td>
<td>+067712</td>
</tr>
<tr>
<td>+K</td>
<td>10n00</td>
<td>02 3072</td>
<td>+068608</td>
</tr>
<tr>
<td>-K</td>
<td>10-00</td>
<td>02 2816</td>
<td>+068352</td>
</tr>
<tr>
<td>MU-1</td>
<td>11010</td>
<td>02 4320</td>
<td>+069356</td>
</tr>
<tr>
<td>+U-1</td>
<td>11460</td>
<td>02 5216</td>
<td>+070752</td>
</tr>
<tr>
<td>-U-1</td>
<td>11360</td>
<td>02 4960</td>
<td>+070496</td>
</tr>
<tr>
<td>ML-1</td>
<td>10019</td>
<td>02 0224</td>
<td>+065760</td>
</tr>
<tr>
<td>+L-1</td>
<td>10460</td>
<td>02 1120</td>
<td>+066656</td>
</tr>
<tr>
<td>-L-1</td>
<td>10360</td>
<td>02 0864</td>
<td>+066400</td>
</tr>
<tr>
<td>MK-1</td>
<td>10810</td>
<td>02 2272</td>
<td>+067808</td>
</tr>
<tr>
<td>+K-1</td>
<td>10n60</td>
<td>02 3168</td>
<td>+068794</td>
</tr>
<tr>
<td>-K-1</td>
<td>10-60</td>
<td>02 2912</td>
<td>+068448</td>
</tr>
<tr>
<td>MU+K</td>
<td>110+0</td>
<td>02 4256</td>
<td>+069792</td>
</tr>
<tr>
<td>+U+K</td>
<td>11420</td>
<td>02 5152</td>
<td>+070683</td>
</tr>
<tr>
<td>-U+K</td>
<td>11320</td>
<td>02 4896</td>
<td>+070432</td>
</tr>
<tr>
<td>ML+K</td>
<td>100+0</td>
<td>02 0160</td>
<td>+065696</td>
</tr>
<tr>
<td>+L+K</td>
<td>10420</td>
<td>02 1056</td>
<td>+066592</td>
</tr>
<tr>
<td>-L+K</td>
<td>10320</td>
<td>02 0800</td>
<td>+066336</td>
</tr>
<tr>
<td>MK+K</td>
<td>108+0</td>
<td>02 2203</td>
<td>+067744</td>
</tr>
<tr>
<td>+K+K</td>
<td>10n20</td>
<td>02 3104</td>
<td>+068640</td>
</tr>
<tr>
<td>U+K-1</td>
<td>11220</td>
<td>02 4640</td>
<td>+070176</td>
</tr>
<tr>
<td>-K-L-1</td>
<td>10220</td>
<td>02 0544</td>
<td>+066080</td>
</tr>
<tr>
<td>-U-2</td>
<td>11260</td>
<td>02 4704</td>
<td>+070240</td>
</tr>
<tr>
<td>-L-2</td>
<td>10260</td>
<td>02 0603</td>
<td>+066144</td>
</tr>
<tr>
<td>-K-2</td>
<td>10+60</td>
<td>02 2656</td>
<td>+068192</td>
</tr>
<tr>
<td>U+1</td>
<td>11500</td>
<td>02 5376</td>
<td>+070912</td>
</tr>
<tr>
<td>L+1</td>
<td>10500</td>
<td>02 1280</td>
<td>+068316</td>
</tr>
<tr>
<td>K+1</td>
<td>10100</td>
<td>02 3323</td>
<td>+068364</td>
</tr>
<tr>
<td>U+K+1</td>
<td>11520</td>
<td>02 5403</td>
<td>+070944</td>
</tr>
<tr>
<td>L+K+1</td>
<td>10520</td>
<td>02 1312</td>
<td>+068348</td>
</tr>
<tr>
<td>K+K+1</td>
<td>10,20</td>
<td>02 3360</td>
<td>+068996</td>
</tr>
<tr>
<td>Decimal Number</td>
<td>FIGURE SHIFT.</td>
<td>LETTER SHIFT.</td>
<td>Binary Code</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>p</td>
<td>00000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>q</td>
<td>00001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>w</td>
<td>00010</td>
</tr>
<tr>
<td>3</td>
<td>/</td>
<td>e</td>
<td>00011</td>
</tr>
<tr>
<td>4</td>
<td>$</td>
<td>r</td>
<td>00100</td>
</tr>
<tr>
<td>5</td>
<td>%</td>
<td>t</td>
<td>00101</td>
</tr>
<tr>
<td>6</td>
<td>&amp;</td>
<td>y</td>
<td>00110</td>
</tr>
<tr>
<td>7</td>
<td>$</td>
<td>u</td>
<td>00111</td>
</tr>
<tr>
<td>8</td>
<td>)</td>
<td>i</td>
<td>01000</td>
</tr>
<tr>
<td>9</td>
<td>+</td>
<td>o</td>
<td>01001</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>k</td>
<td>01010</td>
</tr>
<tr>
<td>11</td>
<td>N</td>
<td>s</td>
<td>01011</td>
</tr>
<tr>
<td>12</td>
<td>J</td>
<td>n</td>
<td>01100</td>
</tr>
<tr>
<td>13</td>
<td>F</td>
<td>j</td>
<td>01101</td>
</tr>
<tr>
<td>14</td>
<td>L</td>
<td>f</td>
<td>01110</td>
</tr>
<tr>
<td>15</td>
<td>Lower Case.</td>
<td>l</td>
<td>01111</td>
</tr>
<tr>
<td>16</td>
<td>Lower Case.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>CR/LF.</td>
<td>d</td>
<td>01000</td>
</tr>
<tr>
<td>18</td>
<td>Letter Shift.</td>
<td>CR/LF.</td>
<td>10010</td>
</tr>
<tr>
<td>19</td>
<td>v</td>
<td>b</td>
<td>10011</td>
</tr>
<tr>
<td>20</td>
<td>a</td>
<td>Letter Shift.</td>
<td>10100</td>
</tr>
<tr>
<td>21</td>
<td>x</td>
<td>v</td>
<td>10101</td>
</tr>
<tr>
<td>22</td>
<td>y</td>
<td>a</td>
<td>10110</td>
</tr>
<tr>
<td>23</td>
<td>Space.</td>
<td>x</td>
<td>10111</td>
</tr>
<tr>
<td>24</td>
<td>Space.</td>
<td>Space.</td>
<td>11000</td>
</tr>
<tr>
<td>25</td>
<td>g</td>
<td>g</td>
<td>11001</td>
</tr>
<tr>
<td>26</td>
<td>m</td>
<td>m</td>
<td>11010</td>
</tr>
<tr>
<td>27</td>
<td>Figure Shift.</td>
<td>Figure Shift.</td>
<td>11011</td>
</tr>
<tr>
<td>28</td>
<td>h</td>
<td>h</td>
<td>11100</td>
</tr>
<tr>
<td>29</td>
<td>c</td>
<td>c</td>
<td>11101</td>
</tr>
<tr>
<td>30</td>
<td>z</td>
<td>z</td>
<td>11110</td>
</tr>
<tr>
<td>31</td>
<td>Upper Case.</td>
<td>Upper Case.</td>
<td>11111</td>
</tr>
</tbody>
</table>
APPENDIX F - DOUBLE-LENGTH-ADD-SUBTRACT PROGRAMMED-OPERATOR

Specification: This program causes the double-length number specified by the programmed-operator instruction to be added to or subtracted from the number in the double-length accumulator (locations 4 and 5). The add-entry is the first word of the routine and the subtract-entry is the fourth. The overflow logic is effective with the last arithmetic instruction; this is stored in the 20th word of the program.

Method: The instructions immediately following the different entry points set into internal temporary storage either a waste order (for addition) or a "negate K" order (for subtraction). This instruction causes the double-length operand to pass unchanged or to be negated before it is added to the double-length accumulator.

The program assumes that the waste bits of both the operand and the number in the double-length accumulator are zero. Hence a carry into the sign position is a true carry digit, and does not represent an overflow condition. To inhibit the stopping of the computer when such a carry digit is produced, a "register-transfer-add" instruction rather than an "add from memory" instruction is used, as the overflow logic is not effective during the execution of the former instruction. The same precaution is taken when the carry digit is added to the first half of the operand. However for the final addition, an "add from memory" instruction is required as the overflow logic should be used as a safeguard against the generation of incorrect results.
# Appendix F

## Double-Length Add-Subtract Programmed-Operator Coding

### Table: Double-Length Add

<table>
<thead>
<tr>
<th>REL. LOC.</th>
<th>ADD ENTRY</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0J 22L</td>
<td>Set waste order in 24L</td>
</tr>
<tr>
<td>1</td>
<td>14 24L</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>04 5L</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REL. LOC.</th>
<th>SUBTRACT ENTRY</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0J 23L</td>
<td>Set K' = -K order in 24L</td>
</tr>
<tr>
<td>4</td>
<td>14 24L</td>
<td>K' = U</td>
</tr>
<tr>
<td>5</td>
<td>02 5122F</td>
<td>Obtain operand address</td>
</tr>
<tr>
<td>6</td>
<td>16 16L</td>
<td>Fetch second half</td>
</tr>
<tr>
<td>7</td>
<td>0F 5F</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>08 16L</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>00 1F</td>
<td>Execute 24L</td>
</tr>
<tr>
<td>10</td>
<td>1L 24L</td>
<td>K' = K + L Add second halves</td>
</tr>
<tr>
<td>11</td>
<td>02 1058F</td>
<td>L = 0</td>
</tr>
<tr>
<td>12</td>
<td>02 1F</td>
<td>Sh.C.D.L + K + 1 Form carry &amp;</td>
</tr>
<tr>
<td>13</td>
<td>03 1857F</td>
<td>Sh.L.S. K - 1 set waste bit = 0</td>
</tr>
<tr>
<td>14</td>
<td>03 97F</td>
<td>Store second half result</td>
</tr>
<tr>
<td>15</td>
<td>14 5F</td>
<td>Add carry to</td>
</tr>
<tr>
<td>16</td>
<td>0J (F)</td>
<td>K' = K + L first half</td>
</tr>
<tr>
<td>17</td>
<td>02 1058F</td>
<td>Execute 24L</td>
</tr>
<tr>
<td>18</td>
<td>1L 24L</td>
<td>Add first half of DL acc.</td>
</tr>
<tr>
<td>19</td>
<td>0L 4F</td>
<td>Store first half result</td>
</tr>
<tr>
<td>20</td>
<td>14 4F</td>
<td>Link</td>
</tr>
<tr>
<td>21</td>
<td>04 JF</td>
<td>Waste order</td>
</tr>
<tr>
<td>22</td>
<td>02 F</td>
<td>K' = -K</td>
</tr>
<tr>
<td>23</td>
<td>02 2818F</td>
<td>Temp. store</td>
</tr>
<tr>
<td>24</td>
<td>00 F</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>00 + DOUBLE LENGTH SUBTRACT</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>2N</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX G

APPENDIX G - DOUBLE-LENGTH-MULTIPLY PROGRAMMED-OPERATOR

Specifications: - This program multiplies the number (the multiplicand) in the double-length accumulator (locations 4 and 5) by the double-length number (the multiplier) specified by the programmed-operator instruction, and places the quadruple-length product in locations 4, 5, 6 and 7. Bit 0 of location 4 is the sign bit of the product, while locations 5, 6 and 7 contain waste bits (set to zero) in the sign position.

Method: - In the following description, a superscript * indicates a double-length number, and subscripts 1 and 2 indicate the more-significant and less-significant halves of the double-length number. Hence:

\[ D^* = \text{double-length multiplicand} \]
\[ D_1 = \text{more-significant half of multiplicand} \]
\[ D_2 = \text{less-significant half of multiplicand} \]
\[ M^* = \text{double-length multiplier} \]
\[ M_1 = \text{more-significant half of multiplier} \]
\[ M_2 = \text{less-significant half of multiplier} \]

From the above definitions we obtain:

\[ D^* = D_1 + 2^{-19} D_2 \]
\[ M^* = M_1 + 2^{-19} M_2 \]

Hence the double-length product \( M^* D^* \) is given by:

\[ M^* D^* = M_1 D_1 + 2^{-19} (M_1 D_2 + M_2 D_1) + 2^{-38} M_2 D_2 \]

All the partial products (\( M_1 D_1 \), \( M_1 D_2 \), \( M_2 D_1 \) and \( M_2 D_2 \)) are double-length numbers and may be represented by a single symbol superscripted with a *. Let:

\[ P^* = M_1 D_1 \]
\[ Q^* = M_1 D_2 \]
\[ R^* = M_2 D_1 \]
\[ S^* = M_2 D_2 \]

Hence the equation for \( M^* D^* \) becomes:

\[ M^* D^* = P^* + 2^{-19} (Q^* + R^*) + 2^{-38} S^* \]
Two major problems in the summation of the partial-products are:

(i) incorrect results produced by overflow

and (ii) arithmetic shifting of double-length numbers required by the factor $2^{-19}$

It is to be noted that $S^*$ is always positive, but $P^*$, $Q^*$ and $R^*$ may be positive or negative.

To overcome the above problems, equivalent expressions for $M^* D^*$ may be derived as follows:

$$M^* D^* = p^* + 2^{-19} (Q^* + T^*) + 2^{-57} S_2$$

where $T^* = R^* + 2^{-19} S_1$

$$M^* D^* = p^* + w^* + 2^{-19} U^* + 2^{-57} S_2$$

where $w^* = 2^{-19} Q_1 + 2^{-19} T_1$

and $U^* = 2^{-19} (Q_2 + T_2)$

$$M^* D^* = v^* + w^* + 2^{-38} U_2 + 2^{-57} S_2$$

where $v^* = p^* + 2^{-19} U_1$

$$M^* D^* = x_1 + 2^{-19} x_2 + 2^{-38} U_2 + 2^{-57} S_2$$

where $x^* = v^* + w^*$

The above expressions give an outline of the steps required to produce four single-length numbers $X_1$, $X_2$, $U_2$ and $S_2$ which represent the quadruple-length product of two double-length numbers. It is to be noted that overflow during intermediate steps of the process cannot occur, and arithmetic shifting of only single-length numbers is required.

The detailed coding of this programmed-operator follows.
# APPENDIX G

## DOUBLE-LENGTH MULTIPLY PROGRAMMED-OPERATOR CODING

### 00+ DOUBLE LENGTH MULTIPLY

<table>
<thead>
<tr>
<th>REL.LOC.</th>
<th>REL. LOC.</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QJ F</td>
<td>Plant Link</td>
</tr>
<tr>
<td>1</td>
<td>16 40L</td>
<td>K' = U Obtain</td>
</tr>
<tr>
<td>2</td>
<td>02 5122F</td>
<td>K' = K * 1 Address</td>
</tr>
<tr>
<td>3</td>
<td>16 20L</td>
<td>Temp. Store D1</td>
</tr>
<tr>
<td>4</td>
<td>02 3330F</td>
<td>K = 0</td>
</tr>
<tr>
<td>5</td>
<td>16 9L</td>
<td>L' = M2</td>
</tr>
<tr>
<td>6</td>
<td>0J 4F</td>
<td>Form S1 = M2 D2</td>
</tr>
<tr>
<td>7</td>
<td>14 45L</td>
<td>Store S2</td>
</tr>
<tr>
<td>8</td>
<td>02 2F</td>
<td>L' = M2</td>
</tr>
<tr>
<td>9</td>
<td>0F (F)</td>
<td>Form T1 = M2 D1 + 2^-19 S1</td>
</tr>
<tr>
<td>10</td>
<td>12 5F</td>
<td>Store T2</td>
</tr>
<tr>
<td>11</td>
<td>15 7F</td>
<td>SH.ADL-K-19</td>
</tr>
<tr>
<td>12</td>
<td>0F J 9L</td>
<td>SH LSL-1 STORE</td>
</tr>
<tr>
<td>13</td>
<td>12 45L</td>
<td>2^-19 T1</td>
</tr>
<tr>
<td>14</td>
<td>15 42L</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>03 3059F</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>03 385F</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>14 43L</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>15 44L</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>02 2F</td>
<td>K = 0</td>
</tr>
<tr>
<td>20</td>
<td>0F (F)</td>
<td>L' = M1</td>
</tr>
<tr>
<td>21</td>
<td>12 5F</td>
<td>Form Q1 = M1 D2</td>
</tr>
<tr>
<td>22</td>
<td>15 41L</td>
<td>Store Q2</td>
</tr>
<tr>
<td>23</td>
<td>03 3059F</td>
<td>SH ADL-K-19 STORE</td>
</tr>
<tr>
<td>24</td>
<td>03 385F</td>
<td>SH LSL-1 2^-19 Q1 1N</td>
</tr>
<tr>
<td>25</td>
<td>14 4F</td>
<td>D.L. ACC.</td>
</tr>
<tr>
<td>26</td>
<td>15 5F</td>
<td>DL ADD: W1 = 2^-19 Q1 + 2^-19 T1</td>
</tr>
<tr>
<td>27</td>
<td>22 43L</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>0F 42L</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>0J 41L</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>02 1057F</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>02 2F</td>
<td>L' = K + L FORM</td>
</tr>
<tr>
<td>32</td>
<td>03 833F</td>
<td>K = 0</td>
</tr>
<tr>
<td>33</td>
<td>03 385F</td>
<td>U* = 2^-19 (Q2 + T2)</td>
</tr>
<tr>
<td>34</td>
<td>15 6F</td>
<td>SH LDL + K + 1</td>
</tr>
<tr>
<td>35</td>
<td>0F J 20L</td>
<td>SH LSL - 1</td>
</tr>
<tr>
<td>36</td>
<td>12 45L</td>
<td>Store U2</td>
</tr>
<tr>
<td>37</td>
<td>14 41L</td>
<td>L' = M1</td>
</tr>
<tr>
<td>38</td>
<td>15 42L</td>
<td>Form V1 = P1 + 2^-19 U1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temp. Store</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>39</td>
<td>22</td>
<td>41L</td>
</tr>
<tr>
<td>40</td>
<td>04</td>
<td>(F)</td>
</tr>
<tr>
<td>41</td>
<td>(00</td>
<td>F)</td>
</tr>
<tr>
<td>42</td>
<td>(00</td>
<td>F)</td>
</tr>
<tr>
<td>43</td>
<td>(00</td>
<td>F)</td>
</tr>
<tr>
<td>44</td>
<td>(00</td>
<td>F)</td>
</tr>
<tr>
<td>45</td>
<td>(00</td>
<td>F)</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>2N</td>
</tr>
</tbody>
</table>
APPENDIX H

APPENDIX H - FLOATING-POINT INPUT PROGRAMMED-OPERATOR

**Specification:**

The input number format which this program assumes is:

<table>
<thead>
<tr>
<th>±</th>
<th>DDDDDDDDDDD</th>
<th>±</th>
<th>DD</th>
</tr>
</thead>
<tbody>
<tr>
<td>MANTISSA</td>
<td>MANTISSA</td>
<td>EXPONENT</td>
<td>EXPONENT</td>
</tr>
<tr>
<td>SIGN</td>
<td>(DECIMAL FRACTION)</td>
<td>SIGN</td>
<td>(DECIMAL INTEGER)</td>
</tr>
</tbody>
</table>

The mantissa sign would normally be either + or -. However any even sexadecimal character is interpreted as +, and any odd sexadecimal character as -. The mantissa may consist of 0 to 9 (inclusive) decimal digits. The program assumes that the digits represent a decimal fraction with the decimal point occurring before the first digit.

The exponent sign must be either + or -. The exponent must consist of two decimal digits. A decimal integer is assumed, and this specifies an exponent of 10.

The program will convert a number in the above format into an equivalent number with the format of a binary floating-point number. This number will then be stored in the double-length/floating point-accumulator (locations 4 and 5) as well as in locations n and n + 1, where n is the address specified by the programmed-operator instruction.

**Method:**

At each stage of the input process, a complete representation of the number is produced by the contents of the double-length accumulator and two integers $C_{10}$ and $C_2$. These integers are exponents of 10 and 2 respectively, and determine numbers by which the double length accumulator is to be multiplied. Initially $C_{10} = 0$ and $C_2 = 38 + 128$. The number 38 is necessary as the decimal to binary conversion process produces a double-length integer of 38 magnitude bits, and the number 128 is required as a scaling in the exponent part of a binary floating-point number. As each digit of the mantissa is read, $C_{10}$ is decremented, and a corresponding double-length integer is produced using double-length programmed-operators.
APPENDIX H

The exponent digits are converted into a binary integer which is used to modify $C_{10}$.

The double-length accumulator is normalised (by shifting left) but a complete representation of the number is maintained by adjusting $C_2$ during the shifting process.

At this stage, the aim is to reduce $C_{10}$ to zero. If $C_{10}$ is positive, this is carried out by multiplying the double-length accumulator by $10 \div 2^4$, adding 4 to $C_2$ and subtracting 1 from $C_{10}$. If $C_{10}$ is negative, the double-length accumulator is multiplied by $2^3 \div 10$, 3 is subtracted from $C_2$ and 1 is added to $C_{10}$.

After the final normalisation process, the mantissa is combined with $C_2$, and the floating-point number is stored in the floating-point accumulator, as well as in the locations specified by the programmed-operator instruction.

The program structure of the floating-point programmed-operator follows, and this in turn is followed by the detailed coding.
**APPENDIX II**

Program structure:

* P.O. ENTRY
  : Obtain link
  : Obtain operand address
  : Read mantissa sign and set MNS
    (Mantissa Negative Switch)
  : Clear DL accumulator
  : Set $C_{10}$ to zero
  : Set $C_2$ to 166

* READ
  : Read 1 sexad
  : If exponent sign go to * EXP. SIGN
  : Convert mantissa to binary integer
  : Decrement $C_{10}$
  : Go to * READ

* EXP. SIGN
  : Set ENS (Exponent Negative Switch)
  : If MNS is negative, negate DL accumulator
  : If mantissa is zero, go to * ZERO MANT.
  : Normalise mantissa adjusting $C_2$
    (Jump to sub-routine * NORM.)
  : Read and convert exponent to binary integer
  : Use exponent to form new $C_{10}$

* TEST $C_{10}$
  : If $C_{10} = 0$, go to * COMBINE
  : If $C_{10} < 0$, go to * $C_{10}$ NEG.
  : Multiply mantissa by $10 \div 2^4$
    Add 4 to $C_2$
    Subtract 1 from $C_{10}$
    Normalise mantissa adjusting $C_2$
    (Jump to sub-routine * NORM.)
    Go to * TEST $C_{10}$

**1**
  : Multiply mantissa by $2^3 \div 10$
  : Subtract 3 from $C_2$
  : Add 1 to $C_{10}$
  : Go to **1

* $C_{10}$ NEG.
  : Multiply mantissa by $2^{3+4}$
  : Subtract 3 from $C_2$
  : Add 1 to $C_{10}$
  : Go to **1

* ZERO MANT.
  : Read 2 sexads
  : Go to * DL STORE

* COMBINE
  : Combine mantissa and $C_2$

* DL STORE
  : DL store
  : Go to main program (LINK)
APPENDIX II

* NORM. : Sub-routine link
: Set C₂ in U
: Fetch DL accumulator in K, L' 
: Shift out waste digit

* TEST : If normalised, go to * END
: Shift K, L left
: Decrement U
: Go to * TEST

* END : Introduce waste digit
: Store KL in DL accumulator
: Store U in C₂
: Sub-routine Link
## APPENDIX H
### FLOATING-POINT INPUT PROGRAMMED-OPERATOR CODING

<table>
<thead>
<tr>
<th>REL. LOC.</th>
<th>P.O. ENTRY</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>* P.O. ENTRY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0J  F</td>
<td>Obtain link</td>
</tr>
<tr>
<td>1</td>
<td>16 68L</td>
<td>(K)' = (U)</td>
</tr>
<tr>
<td>2</td>
<td>02 5122F</td>
<td>K = 0, L = 0 Read</td>
</tr>
<tr>
<td>3</td>
<td>16 67L</td>
<td>Read 1 sexad. mantissa sign</td>
</tr>
<tr>
<td>4</td>
<td>02 3F</td>
<td>Sh.C.S.K - 1 set MNS</td>
</tr>
<tr>
<td>5</td>
<td>01 129F</td>
<td>Clear DL accumulator</td>
</tr>
<tr>
<td>6</td>
<td>03 1121F</td>
<td>C_{10} = 0</td>
</tr>
<tr>
<td>7</td>
<td>14 85L</td>
<td>C_{2} = 166</td>
</tr>
<tr>
<td>8</td>
<td>15 4F</td>
<td>K = 0</td>
</tr>
<tr>
<td>9</td>
<td>15 5F</td>
<td>Read 1 sexad</td>
</tr>
<tr>
<td>10</td>
<td>15 87L</td>
<td>Subtract 10 exponent sign</td>
</tr>
<tr>
<td>11</td>
<td>0J 89L</td>
<td>DL Mult. convert</td>
</tr>
<tr>
<td>12</td>
<td>14 88L</td>
<td>DL Fetch mantissa to</td>
</tr>
<tr>
<td>* READ</td>
<td></td>
<td>DL Add binary integer</td>
</tr>
<tr>
<td>13</td>
<td>02 2F</td>
<td>K = -I Decrement C_{10}</td>
</tr>
<tr>
<td>14</td>
<td>01 129F</td>
<td>Go to * READ</td>
</tr>
<tr>
<td>15</td>
<td>14 92L</td>
<td>Sh. C.S.K-5 Set ENS</td>
</tr>
<tr>
<td>16</td>
<td>10 96L</td>
<td>Test MNS</td>
</tr>
<tr>
<td>17</td>
<td>06 774F</td>
<td>Skip K&gt; 0, 3</td>
</tr>
<tr>
<td>18</td>
<td>24 95L</td>
<td>DL Store Negate</td>
</tr>
<tr>
<td>19</td>
<td>21 6F</td>
<td>DL Sub. DL accumulator</td>
</tr>
<tr>
<td>20</td>
<td>22 91L</td>
<td>DL Sub.</td>
</tr>
<tr>
<td>21</td>
<td>02 1538F</td>
<td>Test zero mantissa</td>
</tr>
<tr>
<td>22</td>
<td>17 87L</td>
<td>K = K/L</td>
</tr>
<tr>
<td>23</td>
<td>04 13L</td>
<td>Skip K = 0, 26 Cond. go to * ZERO MANT.</td>
</tr>
<tr>
<td>* EXP. SIGN</td>
<td></td>
<td>Normalise mantissa</td>
</tr>
<tr>
<td>24</td>
<td>03 1125F</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>14 86L</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>0J 85L</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>06 771F</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>20 6F</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>23 6F</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>23 6F</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>0J 4F</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0F 5F</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>02 1082F</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>06 538F</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>0- 69L</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>36</td>
<td>02</td>
<td>3F</td>
</tr>
<tr>
<td>37</td>
<td>01</td>
<td>129F</td>
</tr>
<tr>
<td>38</td>
<td>03</td>
<td>1012F</td>
</tr>
<tr>
<td>39</td>
<td>01</td>
<td>129F</td>
</tr>
<tr>
<td>40</td>
<td>12</td>
<td>96L</td>
</tr>
<tr>
<td>41</td>
<td>0J</td>
<td>87L</td>
</tr>
<tr>
<td>42</td>
<td>15</td>
<td>87L</td>
</tr>
<tr>
<td>43</td>
<td>08</td>
<td>86L</td>
</tr>
<tr>
<td>44</td>
<td>0L</td>
<td>87L</td>
</tr>
<tr>
<td>45</td>
<td>14</td>
<td>87L</td>
</tr>
</tbody>
</table>

* TEST C_10 *

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>0J</td>
<td>87L</td>
</tr>
<tr>
<td>47</td>
<td>06</td>
<td>527F</td>
</tr>
<tr>
<td>48</td>
<td>05</td>
<td>56L</td>
</tr>
<tr>
<td>49</td>
<td>24</td>
<td>90L</td>
</tr>
<tr>
<td>50</td>
<td>0J</td>
<td>98L</td>
</tr>
<tr>
<td>51</td>
<td>17</td>
<td>88L</td>
</tr>
<tr>
<td>52</td>
<td>02</td>
<td>1538F</td>
</tr>
</tbody>
</table>

** 1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>17</td>
<td>87L</td>
</tr>
<tr>
<td>54</td>
<td>0-</td>
<td>69L</td>
</tr>
<tr>
<td>55</td>
<td>04</td>
<td>46L</td>
</tr>
</tbody>
</table>

* C_10 NEG. *

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>24</td>
<td>93L</td>
</tr>
<tr>
<td>57</td>
<td>0J</td>
<td>97L</td>
</tr>
<tr>
<td>58</td>
<td>17</td>
<td>88L</td>
</tr>
<tr>
<td>59</td>
<td>02</td>
<td>258F</td>
</tr>
<tr>
<td>60</td>
<td>04</td>
<td>53L</td>
</tr>
</tbody>
</table>

* ZERO MANT.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>01</td>
<td>130F</td>
</tr>
<tr>
<td>62</td>
<td>04</td>
<td>67L</td>
</tr>
</tbody>
</table>

* COMBINE *

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>03</td>
<td>3180F</td>
</tr>
<tr>
<td>64</td>
<td>11</td>
<td>5F</td>
</tr>
<tr>
<td>65</td>
<td>0L</td>
<td>88L</td>
</tr>
<tr>
<td>66</td>
<td>14</td>
<td>5F</td>
</tr>
</tbody>
</table>

* DL STORE *

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>20</td>
<td>(F)</td>
</tr>
<tr>
<td>68</td>
<td>04</td>
<td>(F)</td>
</tr>
</tbody>
</table>

* NORM. *

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>69</td>
<td>00</td>
<td>(F)</td>
</tr>
<tr>
<td>70</td>
<td>0J</td>
<td>88L</td>
</tr>
<tr>
<td>71</td>
<td>02</td>
<td>3076F</td>
</tr>
<tr>
<td>72</td>
<td>0J</td>
<td>4F</td>
</tr>
<tr>
<td>73</td>
<td>0F</td>
<td>5F</td>
</tr>
<tr>
<td>74</td>
<td>03</td>
<td>257F</td>
</tr>
</tbody>
</table>

\[ K = 0, L = 0 \]

Read 7

Read 1 sexad convert

Sh. C.D.L-K-20 exponent to

Read 1 sexad binary

KL = 10L + K integer in L

Add index ENS new C_10

Test C_10 = 0

 Skip K = 0, 15 Cond. go to * COMBINE

Cond. go to * C_10 NEG.

DL mult. by 10 \( \div 2^4 \)

Add 4 to C_2

K = -1

Accumulate C_10

Normalise mantissa

Go to * TEST C_10

DL Mult. by 2^{3} \div 10

Subtract 3 from C_2

K = + 1

Go to ** 1

Read 2 sexads

Go to *DL STORE

Sh.I.S.K - 12

Set bits 12 to 19 to zero

Add C_2 Combine mantissa and C_2

DL store

Link to main program

Sub-routine link

Set C_2 in U

U = K

Fetch DL acc. in KL

Sh.L.S.L + 1 Shift out waste bit
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CODE</th>
<th>INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>06 1027F</td>
<td>Skip K = norm, 3 Cond. go to END</td>
</tr>
<tr>
<td>76</td>
<td>03 833F</td>
<td>Sh.L.D.L + K + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shift KL left</td>
</tr>
<tr>
<td>77</td>
<td>02 5220F</td>
<td>Decrement U</td>
</tr>
<tr>
<td>78</td>
<td>04 75L</td>
<td>Go to * TEST</td>
</tr>
<tr>
<td>79</td>
<td>03 385F</td>
<td>Sh.L.S.L-1 Introduce waste bit</td>
</tr>
<tr>
<td>80</td>
<td>14 4F</td>
<td>Store KL in DL acc.</td>
</tr>
<tr>
<td>81</td>
<td>15 5F</td>
<td>K = U Store U in C2</td>
</tr>
<tr>
<td>82</td>
<td>02 5122F</td>
<td>Sub-routine link</td>
</tr>
<tr>
<td>83</td>
<td>16 88L</td>
<td>MNS</td>
</tr>
<tr>
<td>84</td>
<td>04 J 69L</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>(00 F)</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>(00 F)</td>
<td>ENS</td>
</tr>
<tr>
<td>87</td>
<td>(00 F)</td>
<td>C10</td>
</tr>
<tr>
<td>88</td>
<td>(00 F)</td>
<td>C2</td>
</tr>
<tr>
<td>89</td>
<td>00 166F</td>
<td>Initial C2</td>
</tr>
<tr>
<td>90</td>
<td>0+ F</td>
<td>DL 10 ÷2²</td>
</tr>
<tr>
<td>91</td>
<td>00 F</td>
<td>DL mantissa digit</td>
</tr>
<tr>
<td>92</td>
<td>(00 F)</td>
<td>DL 2³ ÷ 10</td>
</tr>
<tr>
<td>93</td>
<td>2N J 1638F</td>
<td>DL integer 10</td>
</tr>
<tr>
<td>94</td>
<td>06 J 4915F</td>
<td>integer -3</td>
</tr>
<tr>
<td>95</td>
<td>00 F</td>
<td>integer 4</td>
</tr>
<tr>
<td>96</td>
<td>00 10F</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>3L J 8189F</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>00 4F</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00 2N</td>
<td></td>
</tr>
</tbody>
</table>
The Logical Design of the General Purpose Digital Computer SNOCOM

By D. G. Wong, B.Sc., M.Eng.Sc.

(Graduate)*

Summary.—SNOCOM is a general-purpose digital computer based on the logical design of the LGP-30. In this paper, a description of the functional and logical design of SNOCOM is presented.

The purpose of this paper has been to describe some of the computer processes which were not covered in any detail in Frankel's paper, and also to present some of the features which have been incorporated into the design of SNOCOM.

1.—INTRODUCTION.

In 1955, the development of the Digital Differential Analyser "A.D.A." (Refs. 1, 2) was commenced by the Mathematical Instruments Section of C.S.I.R.O., under the direction of the Professor of Electrical Engineering of the University of Sydney. This project was supported by the Snowy Mountains Hydro-Electric Authority and it was understood that a second D.D.A. was to be designed for the Snowy Mountains Authority be changed from a digital differential analyser to a general-purpose digital computer.

After ADA was completed and officially opened in March, 1958, the logical design of the computer which was to be known as SNOCOM was commenced. The availability of Frankel's design of the LGP-30 (Ref. 3) and the realization that essentially the same circuit and constructional techniques developed for ADA could be used with the second machine, were the main reasons for basing SNOCOM on the logical design of the LGP-30.

The purpose of this paper has been to illustrate how some of these features have been arranged to fit into the general framework of the LGP-30 design.

LIST OF SYMBOLS.

A

Accumulator playback flip-flop.

A*

Double length accumulator playback flip-flop.

B

Input gate to the second stage of the B/P (break-point) flip-flop.

B*/B32

B/P or instruction number waveform.

C

Counter input.

C*

Counter output waveform.

D

Timing waveforms.

F

Phase flip-flop.

F'

Phase flip-flop (complement of F).

F, G, H

Phase flip-flops.

r

Order number flip-flops.

S

Order flip-flops.

T

Counter output waveform.

W

Weight 5 reader buffer flip-flop.

Z

Printer ready signal.

Z0

Punch ready signal.

Z1

Weight 2 reader buffer flip-flop.

Z2

Sign digit.

Z3

Sign digit.

Zo

Weight 1 reader buffer flip-flop.

Z1

Binary counter output of the sector number generator.

Z2

Output of the Z2 (or external transfer) switch.

I

Add-subtractor output.

Add-subtractor input.

S

Add or subtract signal.

T

Duration of the shift register operation of the F flip-flops.

r

Either R or C depending on the phase.

X

Stroke (‘) character flip-flop.

B

Auto input store playback signal (see Section 7.1).

2. GENERAL DESCRIPTION.

SNOCOM is a fixed-point, binary, serial, stored-program, single-address, general purpose digital computer using transistor circuits and magnetic drum storage. The storage capacity of the computer is 2,048 words, each of 32 binary digits. These are arranged on 64 tracks of the drum with 32 words per track. The Model 512-A Bryant magnetic drum (5 in. dia., 12 in. long) runs at 6,000 r.p.m., giving a clock rate of 102.4 k.c/s, a word time of 312 microseconds and an access time of 5 milliseconds.

The accumulator, instruction register and instruction counter are in the form of recirculating registers, with the spacing between record and playback heads corresponding to one word period. The addition of a second playback head on the accumulator recirculating register enables double length numbers to be stored for use in the multiplication and division processes. The computer logic is synchronised by clock pulses derived from a clock track and a once-per-revolution marker permanently recorded on the drum.

An interfaced sector number system enables simple instructions to be obeyed within nine word periods. Another sixty four word periods are required for the multiplication and division instructions. Hence the addition and multiplication times are 2.8 milliseconds and 22.8 milliseconds, respectively. The bench-mark time (i.e., the time to carry out ten additions and one multiplication) is about 51 milliseconds.

The peripheral units consist of (1) a Ferranti TR5 paper tape reader which operates at 300 characters per second, (2) a Teletype BR22 paper tape punch which operates at 50 characters per second, and (3) a modified IBM electric typewriter which operates at 10 characters per second. Both output units are computer-controlled and all units may operate simultaneously.

Fig. 1.—SNOCOM Number and Instruction Words.

A SNOCOM number word consists of a sign bit, thirty magnitude bits and a spacer bit as shown in Fig. 1. The binary point is assumed to be between bit 0 and bit 1, and the normal two's complement system for the representation of negative numbers is used. Hence, all machine variables (x) lie in the range:

\[-1 < x < 1 - 2^{-30}\]

*This paper, No. 1252, was presented before the Engineering Conference, 1962, in Georgia Tech (Feb. 23rd March, 1962). The author is Lecturer, School of Electrical Engineering, The University of Sydney.
A SNOCOM instruction word consists of four order bits, six track bits and five sector bits as shown also in Fig. 1. The track and sector bits, together with the order bits, constitute the full address of the sixteen basic instructions which SNOCOM is capable of executing. The remaining bits have no significance.

### 3. THE SNOCOM ORDER CODE

The order code for SNOCOM’s sixteen basic instructions is shown in Table I. These instructions are defined in detail in Table VII (Appendix IV) and a brief discussion of these is presented in the following sections.

#### TABLE I.

**The SNOCOM Order Code.** (Simplified.)

<table>
<thead>
<tr>
<th>Code</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Stop</td>
</tr>
<tr>
<td>0100</td>
<td>Input</td>
</tr>
<tr>
<td>0101</td>
<td>Divide</td>
</tr>
<tr>
<td>0110</td>
<td>Subtract</td>
</tr>
<tr>
<td>0111</td>
<td>Multiply frac</td>
</tr>
<tr>
<td>1000</td>
<td>Unconditional transfer</td>
</tr>
<tr>
<td>1001</td>
<td>Conditional transfer</td>
</tr>
<tr>
<td>1010</td>
<td>Store</td>
</tr>
<tr>
<td>1011</td>
<td>Store and clear</td>
</tr>
<tr>
<td>1012</td>
<td>Store and output</td>
</tr>
<tr>
<td>1013</td>
<td>Return address</td>
</tr>
<tr>
<td>1014</td>
<td>Skip</td>
</tr>
<tr>
<td>1015</td>
<td>Output</td>
</tr>
<tr>
<td>1016</td>
<td>Step</td>
</tr>
</tbody>
</table>

#### 3.1. Arithmetic and Logical Instructions

The arithmetic operations of addition, subtraction and division are carried out with an accuracy of 30 binary digits plus sign. For multiplication, the complete product consisting of 60 binary digits plus sign is formed (by the use of a second playback head on the accumulator track), and the most significant 30 bits plus sign and the least significant 30 bits are added to the accumulator, depending on whether a 7 order or a 6 order is being executed.

The augend, minuend, multiplier or dividend is entered into the accumulator by a 1 order, while the address of the addend, subtrahend, multiplicand or divisor is specified in the single-address instruction. The results of the arithmetic operations are left in the accumulator.

As the accumulator is used to hold both the multiplier and the product, with a suitable choice of constants for the multiplicand, 7 and 6 orders can be used as right and left shift orders, respectively. For example, a 7 order multiplication by 00000000 (sexadecimal) is equivalent to multiplication by 2⁻⁷ and this results in the contents of the accumulator being shifted 4 binary places to the right. On the other hand, a 6 order multiplication by 00000000 is equivalent to left multiplication by 2⁻⁶ and this results in the contents of the accumulator being shifted 2 binary places to the left.

The 9 order obtains the bit-by-bit logical product of the contents of the accumulator and the next instruction, as the location specified by the instruction, and retains the result in the accumulator. Used in conjunction with "masking" numbers which contain 1s in certain bit positions and 0s elsewhere, the 9 order makes possible the storage of several pieces of information in the same storage location, as each piece of information may be isolated (or extracted) and used individually.

#### 3.2. Transfer of Control Instructions

Interconnection of routines and branching of a program depending on intermediate results are made possible by the unconditional transfer and conditional transfer instructions. Manual control of a program in the computer is also made possible by the external transfer switch Z₉. Reference to Table VII shows that the order 80-ABCO transfers control to JBC if the accumulator is negative or to ABC if the accumulator is positive when the second counter changes from the state 111 to 000. With the above arrangement, one is added to the weight one position of the sector number pattern each time a thirty-word period occurs. The first counter is pulsed to count backwards on every word period, while a second counter is used to count backwards on every eighth word period. The second counter is inhibited from changing when the first counter changes from 111 to 000 as this already represents a subtraction of seven. The second counter is pulsed early in the word period and is used in conjunction with the external transfer switch. The first counter corresponds to the two most significant digits of the sector number plus an additional (more significant) digit which effectively counts odd and even drum revolutions. This second counter is pulsed to count backwards on every word period except every eighth when the first counter changes from the state 111 to 000. With the above arrangement, one is added to the weight one position of the sector number pattern each time a thirty-word period occurs. When the above arrangement, one is added to the weight one position of the sector number pattern each time a thirty-word period occurs. When the first counter changes from 111 to 000 as this already represents a subtraction of seven.

#### 3.3. Record Instructions

Intermediate records are entered on the magnetic drum using the N order (store) or the J order (store and clear), and addresses of instructions in the store are modified using the 2 order (store address). The 3 order (return address) is used "planted" in the machine logic. The order immediately following the 3 order is usually an unconditional transfer order which transfers control to the first order in the subroutine. The address of the second order after the 3 order is planted into the last order of the subroutine (also an unconditional transfer order) which transfers control back to the main program.

#### 3.4. Input Instruction

Reference to Table VII shows that it is possible to enter a single character (four bits) or a complete word (eight characters) on the execution of an instruction. The eight least significant bits are enclosed within two control characters (stroke and #) which direct the input logic to repeat the input process (to be described) until a full set of characters has been received. A given set of control characters have been entered into the accumulator. It is to be noted that all 5th bits characters (including the control characters / and #) never enter the accumulator.

#### 3.5. Output Instruction

If bit 8 (or Fig. 1) of an 8 order is 0, bits 18-22 are punched as a character on the output tape; if bit 17 is 1, and if bits 18-22 correspond to one of the nineteen decoded characters, the output tape is not actuated. A program may be written to either punch or print the results of a computation depending on the setting of the Z₈ switch. This may be carried out very simply and is achieved by generating between a program order for the constant 00004000 (sexadecimal) which may or may not be added to the output order depending on the Z₈ setting.

#### 3.6. Stop Instruction

The conditional stop orders may be very useful in code-checking a new program. If 0 orders with different track numbers are scattered throughout a program, the programmer may stop the computation just before the suspected faulty section of the program by setting the appropriate break-point switches. The computer may then be switched over to single operation mode, and the error located by monitoring the contents of the three main registers and the states of the main flip-flops after the execution of each instruction.

#### 4. THE TIMING UNIT

Waveforms generated by the timing unit are used (1) to synchronize the operation of the logic circuits, (2) to maintain the correct interpretation of digits in a word by extracting the appropriate digits in the machine logic and (3) to announce the number of the sector which is next to appear under the read-write heads of the main store.

The clock and sectors clocks Tₒ and T₂ of Fig. 2 are derived from a clock track on the drum, and are used in the double flip-flop arrangement of Fig. 5. A once-per-revolution marker synchronizes the segments of digits described in Section 4.1, and the sector number generator described in Section 4.2.

#### 4.1. The Segment and Digit Timing Waveforms

To maintain complete flexibility so that any digit within a word may be used, a word period is divided into seven segments with each segment containing either four, five or six digits. The segment waveforms S₇ — S₁ and the digit waveforms D₈ — D₁ of Fig. 2 are generated by two ring counters and interconnecting logic. Any digit within a word is specified in terms of these waveforms. For example, the waste digit and the sign digit are the first and last digits in the word and may be specified as S₇D₇ and S₁D₁ respectively.

#### 4.2. The Sector Number Generator

In SNOCOM, consecutive addresses do not correspond to consecutive locations on a track of the drum, but to locations which are spaced nine word periods apart. In this way, an instruction which refers to an operand, which is appropriately located between itself and the next instruction, is obeyed in the machine logic rather than the drum logic, and the time of thirty-two-word periods (corresponding to one drum revolution) which would otherwise be required. To make the above arrangement possible, a new program number must be formed by subtracting seven from the number of the preceding sector (modulo 32). This results in the sector number sequence shown in Table II.

Two three-stage binary counters are used in the generation of the sector number sequence. The first counter corresponds to the three least significant digits of the sector number and is pulsed to count backwards every word period. The second counter corresponds to the two most significant digits of the sector number plus an additional (more significant) digit which effectively counts odd and even drum revolutions. This second counter is pulsed to count backwards on every word period except every eighth when the first counter changes from the state 111 to 000. With the above arrangement, one is added to the weight one position of the sector number pattern each time a thirty-word period occurs. When the first counter changes from 111 to 000 as this already represents a subtraction of seven.

#### Table II

<table>
<thead>
<tr>
<th>Sector Number Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 — 027 — 034 — 041 — 048 — 055 — 062 — 069 — 076 — 083 — 090 — 097 — 104 — 111 — 118 — 125</td>
</tr>
</tbody>
</table>

In the LGP-30, the sector number pattern is represented by a sequence of 8 bits, with 000 corresponding to the first sector and 177 (or 11111111) corresponding to the last sector of a program. The sector number is stored in the storage location occupied by the program immediately before the program is loaded for execution. This arrangement is compatible with both the LGP-30 and the SNOCOM, as any sector number in the range 000 — 177 may be loaded.

The machine logic consists of two sets of interlaced circuits which are operated in sequence. Each set consists of a sector number generator, a sector number register, a sector number counter, a sector number decoder and a sector number address register. The second set is inhibited from supplying the sector number address register when the sector number counter has reached the number 177 and the second set is inhibited from supplying the sector number address register when the sector number counter has reached the number 177.
5.—MANUAL CONTROLS FOR STARTING A PROGRAM.

Because the three operations for initial filling described in Section 5.1 were designed to share common timing circuits, it is necessary for the machine operator to select the required operation by pressing the appropriate button, before initiating the operation by pressing the "operate" button. Buttons on the machine console must be pressed 28 times in the correct sequence before the bootstrap routine described in Section 5.2 could be initially filled to input a new program. It was realized very early in the development of the computer that this would cause a great deal of inconvenience. To overcome this, "block record" toggle switches with

<table>
<thead>
<tr>
<th>Sector Number</th>
<th>Binary Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00000</td>
</tr>
<tr>
<td>25</td>
<td>11001</td>
</tr>
<tr>
<td>26</td>
<td>10110</td>
</tr>
<tr>
<td>11</td>
<td>01011</td>
</tr>
<tr>
<td>4</td>
<td>00100</td>
</tr>
<tr>
<td>29</td>
<td>11101</td>
</tr>
<tr>
<td>22</td>
<td>10110</td>
</tr>
<tr>
<td>15</td>
<td>01111</td>
</tr>
<tr>
<td>8</td>
<td>01000</td>
</tr>
<tr>
<td>1</td>
<td>00001</td>
</tr>
<tr>
<td>26</td>
<td>11010</td>
</tr>
<tr>
<td>12</td>
<td>01000</td>
</tr>
<tr>
<td>5</td>
<td>00101</td>
</tr>
<tr>
<td>30</td>
<td>11110</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
</tr>
<tr>
<td>9</td>
<td>01001</td>
</tr>
<tr>
<td>2</td>
<td>00010</td>
</tr>
<tr>
<td>20</td>
<td>11011</td>
</tr>
<tr>
<td>13</td>
<td>01101</td>
</tr>
<tr>
<td>6</td>
<td>00110</td>
</tr>
<tr>
<td>31</td>
<td>11111</td>
</tr>
<tr>
<td>24</td>
<td>11000</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
</tr>
<tr>
<td>10</td>
<td>01010</td>
</tr>
<tr>
<td>3</td>
<td>00011</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
<tr>
<td>14</td>
<td>01110</td>
</tr>
<tr>
<td>7</td>
<td>00111</td>
</tr>
<tr>
<td>0</td>
<td>00000</td>
</tr>
</tbody>
</table>

Fig. 2.—Timing Waveforms.

TABLE II

<table>
<thead>
<tr>
<th>Sector Number</th>
<th>Binary Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00000</td>
</tr>
<tr>
<td>25</td>
<td>11001</td>
</tr>
<tr>
<td>26</td>
<td>10110</td>
</tr>
<tr>
<td>11</td>
<td>01011</td>
</tr>
<tr>
<td>4</td>
<td>00100</td>
</tr>
<tr>
<td>29</td>
<td>11101</td>
</tr>
<tr>
<td>22</td>
<td>10110</td>
</tr>
<tr>
<td>15</td>
<td>01111</td>
</tr>
<tr>
<td>8</td>
<td>01000</td>
</tr>
<tr>
<td>1</td>
<td>00001</td>
</tr>
<tr>
<td>26</td>
<td>11010</td>
</tr>
<tr>
<td>12</td>
<td>01000</td>
</tr>
<tr>
<td>5</td>
<td>00101</td>
</tr>
<tr>
<td>30</td>
<td>11110</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
</tr>
<tr>
<td>9</td>
<td>01001</td>
</tr>
<tr>
<td>2</td>
<td>00010</td>
</tr>
<tr>
<td>20</td>
<td>11011</td>
</tr>
<tr>
<td>13</td>
<td>01101</td>
</tr>
<tr>
<td>6</td>
<td>00110</td>
</tr>
<tr>
<td>31</td>
<td>11111</td>
</tr>
<tr>
<td>24</td>
<td>11000</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
</tr>
<tr>
<td>10</td>
<td>01010</td>
</tr>
<tr>
<td>3</td>
<td>00011</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
<tr>
<td>14</td>
<td>01110</td>
</tr>
<tr>
<td>7</td>
<td>00111</td>
</tr>
<tr>
<td>0</td>
<td>00000</td>
</tr>
</tbody>
</table>

Associated logic were introduced to safeguard the unscheduled mutilation of an input routine which was to be stored permanently at the beginning of the store. Control was to be transferred to this routine by simply pressing the "clear C" button. SNOCOM's auto input described in Sections 5.3 and 7.1 has been extremely effective in simplifying the procedure for starting a program, and it has not been necessary to develop programs using the block record feature.

5.1.—Initial Filling:

The three operations provided for initially filling the magnetic drum store with a bootstrap routine are as follows.—

1. Fill Accumulator (→ A)
2. Transfer A to R (A → R)
3. Fill Accumulator (→ A)
4. Execute R (Ex.R)

Steps 1 and 2 put the "store" order (type N) into the instruction register. The word 00-00000 is entered into the accumulator in step 1, then control is transferred to the instruction register in step 2, for this word is filled into location 004 in the main store.

5.2.—A Bootstrap Routine:

With SNOCOM's order code, the minimum number of instructions required for a bootstrap or short input routine is three. In sexadecimal, these instructions are as follows.—

<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00400000</td>
</tr>
<tr>
<td>001</td>
<td>00200030</td>
</tr>
<tr>
<td>002</td>
<td>00400000</td>
</tr>
<tr>
<td>003</td>
<td>(           )</td>
</tr>
<tr>
<td>004</td>
<td>00-00000</td>
</tr>
</tbody>
</table>

This first two words on the input tape must be:

[00N00040] / [00 +00000]

and the routine must be started at location 000 by pressing the "clear C" button on the control console. The latter word is stored in location 004 on the execution of the former (instruction) word which had previously been stored in location 003 by the three orders of the initial bootstrap routine. On the execution of the latter (instruction) word in location 004, control is transferred to location 000, and the following program on the input tape is then effectively under the control of the following routine.—

<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00400000</td>
</tr>
<tr>
<td>001</td>
<td>00200030</td>
</tr>
<tr>
<td>002</td>
<td>00400000</td>
</tr>
<tr>
<td>003</td>
<td>(           )</td>
</tr>
<tr>
<td>004</td>
<td>00-00000</td>
</tr>
</tbody>
</table>

This routine may be used to fill a program into the store when each word of the program is preceded by a "store" or "store and clear" instruction whose operand address corresponds to the location in which the word is to be stored.

The execution of the bootstrap routine can be terminated if the first order of an order pair is an unconditional transfer of control order.

5.3.—The Auto Input:

After the auto input button has been pressed, the normal store playback (V) from sectors 0, 1 and 2 is inhibited and replaced by a pattern of digits generated by the timing unit. This pattern of digits is equivalent to the following three orders.—

[00400030 (from sector 0)]
[00400030 ("clear C")]
[00400030 (A → R)]

If the first two words on a program tape are:

[00N00040] / [00 +00000]

and the computer is started after pressing the "clear C" button, the word 00-00000 will be stored in 004, and the program will then be effectively under the control of the following bootstrap routine.—

<table>
<thead>
<tr>
<th>Location</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00400030</td>
</tr>
<tr>
<td>001</td>
<td>00200030</td>
</tr>
<tr>
<td>002</td>
<td>00400000</td>
</tr>
<tr>
<td>003</td>
<td>(           )</td>
</tr>
<tr>
<td>004</td>
<td>00-00000</td>
</tr>
</tbody>
</table>

The Institution of Engineers, Australia
The normal store playback from sectors 0, 1 and 2 (track 0) is not inhibited permanently, conditions being returned to normal immediately if the "1" is sensed in the waste position of the accumulator in phase 1.

As can be noted, that, because of a simplification of logical design, the normal store playback is inhibited whenever the sector number generator announces a number whose binary equivalent contains a 0 in the position where the digit 4 position, and this applies not only to track 0 but to all tracks. To eliminate these abnormal conditions as soon as possible, it is recommended that all program tapes should begin with the following eight words—

```
00000000         01000000 #
00000000         00000000 #
00000010         00000000 #
00000000         00000000 #
```

When this done, the bootstrap routine discussed above will be actually written into the store, and conditions will be returned to normal by the "1" in the waste position of the eighth word. This is carried out in conjunction with the sector number generator of the timing unit to terminate the current instruction into the instruction register.

6.—FUNCTIONAL DESIGN.

The execution of each instruction stored on the magnetic drum must be executed to eight phases. During computation, information is held on the drum, in the recirculating registers and in flip-flops. The definition of the operations which are to be executed in each phase, as well as the transfer of information from drum to recirculating register or from recirculating register to flip-flop, constitutes the functional design of the computer. The multiplication and division processes which require the additional information is held on the drum, in the recirculating registers and answer digits of the instruction register which are to follow.

For the record orders (types 2, 3, N and J), the store record number generator is activated and information is written onto the magnetic drum. For the input order (type 4), the accumulator digits are effectively shifted four places to the left and the four digits held in P1 — P6, which had previously (in f3) been set according to the character to be input, are shifted into the four least significant places. The input logic has been designed so that any number of characters (up to 8) can be entered into the accumulator on the execution of a single input order. This is carried out in two control characters (/ and #) on the input tape. After a / character has been detected in f3, f4 and not f6 is made to follow f4 so that the f3 — f4 input process (described in Section 3) is terminated after one word period.

For the print or punch order (type 8), the character to be printed is determined by the track digits of the instruction to be printed. They have already been set into the flip-flops (in f3). The flip-flop P1 is used to determine whether the printer or the punch is activated: if P1 = 0, the 5 bit character P2P3P4P5P6 is punched; if P1 = 1, the character corresponding to P2P3P4P5P6 is printed. For a stop order (type 0), the computer completes f4 and remains in f1 until instructed manually to continue its computation. A detection of overflow or division hang-up will also stop the computer.

7.—LOGICAL DESIGN.

The following brief description refers to the simplified block diagram of Fig. 3. More details appear in Appendices II and III.

The logical design of the LGP-30 as described by Frankel in Reference 3 consists primarily of (1) the specification of the conditions under which each of the main fifteen flip-flops is set and reset, (2) the specification of the digits to be recorded in the recirculating registers in each digit period and (3) the specification of the exact location on the magnetic drum in which specified digits are written by the address digits of the instruction register.

Three flip-flops (F1, G1, H) are used to define the eight machine periods (see Table III). Some improvement in machine specification has been made by changing the conditional register which phase three is terminated in SNOCOM. The additional logic described in Section 7.2 utilizes the multi-purpose flip-flop K which is used (1) for sector search, (2) for peripheral unit lock-out and (3) for augmenting the counter address. The L flip-flop is used to delay the carry or borrow digit in the add-subtractor described in Appendix II. The S flip-flop (L or G) is used to detect end of the order digits during the execution phases. The T flip-flop is also used in the blocked-state logic to be described in Section 7.3. The main function of the six flip-flops P1 — P6 is to select the correct track while recording on or playing-back from the magnetic drum.

A signal-flow and block diagram for the recirculating registers is presented in Fig. 7. The derivation of the Boolean equations representing the digits (A', G' and R') presented to the record amplifiers during the processes described in Appendix II. The equations are essentially the same as those described by Frankel, with minor modifications. No modifications to the multiplication and division processes were made and hence these are not described.

The V' logic representing the digits to be recorded and the f logic represent the duration of recording ensure the correct execution of the four record instructions.

Two significant improvements over the LGP-30 design which have been incorporated in SNOCOM are described in Sections 7.1 and 7.2. The logic in Section 7.3 was not described in any detail in Frankel's paper. In these sections, it is assumed that the reader is familiar with the material in the Appendices I, II and III.

7.1.—The Auto Input:

As a peripheral number generator announces the sector number of the next word, when sectors 0 and 1 are being played back, the sector number generator announces 25, 26 and 27, respectively. If the binary numbers of the sectors held in flip-flops P1 — P6 are shifted into the four least significant places of the state digits P2P3P4P5P6 and P1, the relevant sectors are represented by the following—

<table>
<thead>
<tr>
<th>Sector played back</th>
<th>Sector number announcement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 25</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0 0 0 02 26</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 0 0 0 02 27</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 0 0 0 02 28</td>
</tr>
<tr>
<td>4</td>
<td>0 0 0 0 0 0 02 29</td>
</tr>
</tbody>
</table>

8.—Circuits during the subsequent phase (<£4), and the sector digits of the instruction counter may be overwritten by the address digits of the registers.

The normal procedure for inputting a program using the auto input is as follows:

1. Press the initial set button.
2. Press the clear C button.
3. Press the auto input button.
4. Press the operate button.
It can readily be seen that the variable \( \omega_1 \) alone distinguishes sectors 0, 1 and 2 from sectors 3 and 4.

After the auto input button has been pressed, the \( B \) flip-flop is set to the "1" state and while this flip-flop is in this state, the normal \( V \) playback from sectors 0, 1 and 2 (and other sectors but not sectors 3 and 4) is inhibited. The conditions to be satisfied while \( F \) is inhibited are therefore represented by \( \omega_1 \cdot \beta = 1 \).

The normal \( V \) playback is replaced by a pattern of digits generated from the timing unit. For example, an input order 00000000 is represented by a single digit \( S_D_8 \) and the order 00000000 is represented by the 4 digits:

\[
S_D_8 + S_D_8 + S_D_8 + S_D_8 + S_D_8 = (Q!Q_2Q_3Q_4 \cdot Q_5Q_6Q_7Q_8) + (Q!Q_2Q_3Q_4Q_5Q_6Q_7Q_8) + (Q!Q_2Q_3Q_4Q_5Q_6Q_7Q_8) + (Q!Q_2Q_3Q_4Q_5Q_6Q_7Q_8)
\]

The digits from sectors 0, 1 and 2 have \( S_D_8 \) in common. Also, as the address portion of an input order has no significance, the digits \( S_D_9 \) and \( S_D_3 \) may be added to the input orders without harm. The pattern therefore may be simplified to:

\[
S_D_8 = (Q!Q_2Q_3Q_4 \cdot Q_5Q_6Q_7Q_8) + (Q!Q_2Q_3Q_4Q_5Q_6Q_7Q_8) + (Q!Q_2Q_3Q_4Q_5Q_6Q_7Q_8) + (Q!Q_2Q_3Q_4Q_5Q_6Q_7Q_8)
\]

The auto input logic is introduced into the main machine logic by replacing the normal playback by a new variable \( V \), which is equal to \( F \) under normal conditions but equal to the digits generated from the timing unit under the conditions of auto input (i.e., when \( \omega_1 \cdot \beta = 1 \)). Hence:

\[
V = V_1 \cdot \omega_1 \cdot \beta + \omega_2 \cdot \beta (S_D_8 + S_D_8 + S_D_8 + S_D_8 \cdot V_1 \cdot \omega_1 \cdot \beta + S_D_8 \cdot V_1 \cdot \omega_1 \cdot \beta)
\]

The flip-flop \( B \) is set to "1" when the auto-input button is depressed, and is reset to "0" when a "1" is detected in the waste digit of the accumulator in \( \phi \). It is also reset to "0" when the initial set button is depressed so that the computer may be used without the auto input operation. Hence:

\[
\phi = B \cdot \text{auto input button}
\]

A logic diagram corresponding to the above equations is shown in Fig. 4.

7.2. Termination of Phase 3:

In the logical design of the LGP-30 as described by Frankel's summary of equations, \( \phi \) is terminated by sector number agreement for all orders. However, for five of the sixteen orders (viz., 0, +, and — orders) the term \( \omega_1 \) is inhibited are therefore represented by \( \omega_1 \cdot \beta = 1 \). It is also reset to "0" when the initial set button is depressed so that the computer may be used without the auto input operation. Hence:

\[
\omega_1 \cdot \beta = 1
\]

The pattern therefore may be simplified to:

\[
(\omega_1 \cdot \beta = 1)
\]

The states of the \( \phi \) flip-flops corresponding to 0, +, and — orders are \( Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8 \) respectively. The termination of \( \phi \) after one word period for 0, +, and — orders may therefore be represented by:

\[
K' = (Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8)
\]

For type 4 and 8 orders, termination of \( \phi \) is controlled by "reader ready" \( (R, R_1) \), "printer ready" \( (Z_2) \) and "punch ready" \( (Z_3) \) signals, and provided the computer program does not attempt to operate the peripheral units faster than their maximum operating speeds, SNOCMOC's logic is such that \( \omega_1 \) for these orders will last only one word period. The \( K' \) flip-flop is reset to "0" if \( S_D_8 \) is set to "1" state by sector agreement. The "ready signal" of the unit specified by the \( Q_2 \) and \( F \) is then sensed; if the unit is ready, \( \phi \) is terminated by resetting \( K \) to 1 at \( S_D_8 \); if the unit is not ready, the machine will repeatedly lock-out and sense the ready signal at following \( S_D_8 \) times until the unit is ready to initiate another input or output cycle. The above arrangement is represented by the equations:

\[
K' = (Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8)
\]

7.3. The Blocked State:

In the blocked state, the computer remains in \( \phi \). This is accomplished by making the transfer from \( \phi_1 \) to \( \phi_2 \) contingent on the occurrence of \( Q_2 = 1 \). The flip-flop \( Q_2 \) is always found in the 1 state when \( \phi_1 \) is entered. However, it is reset to 0 later in the first word period of \( \phi_1 \) if the stop computer cycle has been previously initiated by (1) the detection of overflow in an addition or subtraction, (2) the detection of an improper division, (3) the execution of a break point order or (4) the setting of the computer in the single operation mode.

Each of the fifteen main flip-flops in the computer is represented by a pair of flip-flops operating with inter-leaved clocks, as shown in Fig. 5. The \( Q_2 \) flip-flop is set to 1 when \( \phi_1 \) is entered by adding a term \( F \cdot F \) to the \( Q_2 \) equation:

\[
Q_2 = F \cdot F
\]

where \( F \) and \( F \) are the 0 and 1 sides of the early and late \( F \) flip-flops, respectively. The term \( F \cdot F \) produces a 1 output only when the phase changes from \( \phi_1 \) to \( \phi_2 \). It is to be noted that \( Q_2 = 1 \) for multiplication and division orders. Hence on entering \( \phi_1 \) from \( \phi_2 \), \( Q_2 \) will be found in the 1 state.

7.3.1. Overflow:

The normal two's complement representation of negative numbers is used. Hence overflow is detected by the presence or absence of the carry or borrow digit into the sign digit position. The four conditions under which overflow occurs is represented by:

\[
O/F = PH_1 Q_0 Q_4 Q_7 Q_{AVL} + Q_{AVL} + Q_{AVL} + Q_{AVL}
\]
The above O/F signal is used to set a flip-flop which illuminates the O/F monitor light and causes the machine to idle in the next state by resetting Q2 0. An overflow on-off switch is provided to inhibit the stopping of the machine when overflow occurs.

7.3.2. Break Point Stop. The track number of a 0 order is held in the P flip-flops during overflow. Reference to Table VII shows that the computer is to stop unconditionally when all Ps are in the 0 state. Otherwise, stopping is contingent on the setting of a B/P switch and the setting of the corresponding P flip-flop in the 1 state. These conditions are represented by:

\[ B/P = B \oplus F \oplus G \oplus H \oplus Q \oplus P \oplus P_1 \oplus P_2 \oplus P_3 \oplus P_4 \oplus P_5 \]

where \( B = B_1 - B_2 \) are the outputs of the B/P switches.

The above B/P signal is used to set a flip-flop which illuminates the B/P monitor light and causes the machine to idle in the next state by resetting Q2 0.

8. CONCLUSIONS.

The problem of starting programs on simple magnetic drum computers has been overcome in SNOCOM by the incorporation of the "Auto Input" feature.

The "Lock-out" circuitry necessary for the simultaneous operation of the computer and the peripheral units has been found extremely effective. This feature, together with the relatively high operating speeds of the peripheral units, has effectively increased the computational speed of the computer for problems involving large amounts of input data and output results.

Starting from the general framework of the LGP-30 design, it has been found relatively simple to introduce a number of desirable features into SNOCOM. These features include the computer control of both the printer and the punch, the allowable selection of the number of characters entered on the execution of an input order, and the reduction of computational time by minimizing the period of the open-search and lock-out phase (Δ).

SNOCOM had run satisfactorily for several months within the Electrical Engineering School of the University of Sydney before its installation in the main offices of the Snowy Mountains Authority in Cooma North, N.S.W. Since the beginning of 1961, its operation in its new environment has also been found satisfactory.

The design, construction, testing and installation of SNOCOM were carried out by an extremely small group of qualified personnel and technicians, some of whom had teaching commitments as well. The success of the SNOCOM project has shown that substantial contributions can be made by such groups in the computer field, and it is the opinion of the author that computer research and development (even by small groups) at the University should be encouraged.

9. ACKNOWLEDGMENTS.

The construction of SNOCOM was made possible by the generous support given by the Snowy Mountains Hydro-Electric Authority to computer development within the Electrical Engineering School of the University of Sydney.

Many of the transistor circuits and engineering methods which were developed by Dr. M. W. Allen of the Mathematical Instruments Section of C.S.I.R.O., for the construction of ADA were utilized in SNOCOM. Dr. Allen was also directly responsible for the initial stages of development of SNOCOM, and he is to be given much of the credit for its success.

The assistance given by Messrs. W. S. Lamond, L. G. Bellamy, J. E. Todd, R. B. White and K. R. Rosolen during various stages of the project is gratefully acknowledged.

Mons. P. T. Brown and D. Linsten, of the Snowy Mountains Authority, assisted in the final stages of testing and installation. Their assistance and that of other members of staff within the Snowy Mountains Authority, the Basser Computing Department and the Electrical Engineering School of the University of Sydney, who have contributed to the project, are also gratefully acknowledged.

APPENDIX I.

PHASE LOGIC.

A block diagram of the phase logic is shown in Fig. 6. The eight machine phases are defined in terms of the settings of the F, G and H flip-flops in Table III.

**APPENDIX I.**

**TABLE III.**

<table>
<thead>
<tr>
<th>Phase</th>
<th>F, G, H States</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>FGH</td>
</tr>
<tr>
<td>T1</td>
<td>FGH</td>
</tr>
<tr>
<td>T2</td>
<td>FGH</td>
</tr>
<tr>
<td>T3</td>
<td>FGH</td>
</tr>
<tr>
<td>T4</td>
<td>FGH</td>
</tr>
<tr>
<td>T5</td>
<td>FGH</td>
</tr>
<tr>
<td>T6</td>
<td>FGH</td>
</tr>
<tr>
<td>T7</td>
<td>FGH</td>
</tr>
</tbody>
</table>

By virtue of the double flip-flop arrangement (Fig. 5) the outputs of a particular flip-flop (for example F and F) may be used in the setting and resetting logic of the same flip-flop. (A single flip-flop with input storage gates would also have been satisfactory.)

The conditions under which a flip-flop (e.g., F) is set to ('1') and reset to ('0') are illustrated by  and , respectively. The sign digit is the last digit of a word period; hence, the term T2 will be included in all the setting and resetting expressions for F, G and H.

As 42 and 44 consist of only one word period, the change in F at the end of these phases is represented by:

\[ F' = F \oplus H, T + F' = F \oplus G, T' + \]

![Fig. 5.—Flip-flop Pair with Interleaved Clocks.](image-url)

![Fig. 6.—Block Diagram of the Phase and Blocked State Logic.](image-url)
In the above and following equations, \( \alpha \) following an expression indicates that other equations show contributions to that input. At the end of \( \alpha \) and \( \beta \), \( \gamma \) is set to 0, hence:

\[ G = H + T_1 + \]

The \( \kappa \) flip-flop is set in the instruction search process of \( \gamma \) or the operand search and look-out processes of \( \delta \). It is found in the "1" state at the end of the word period if the phase is to be terminated. Hence the transfer from \( \alpha \) to \( \beta \) from and to \( \gamma \) may be represented by:

\[ G = FGH K_1 T_1 + FGH K_1 T_1 + GH K T_1 (Q' + Q) = \]

The \( \kappa \) flip-flop is set to 1 if the beginning of each word period. Hence:

\[ K = \frac{1}{2} \]

Disregard the state of the sector generator digits and the sector digits of \( \gamma \) during \( \alpha \) or the sector digits of \( \delta \) during \( \alpha \) or the second sector digit of \( \delta \) during the second sector digit period after the termination of \( \alpha \). The six digit pattern is required for the termination of \( \gamma \). Hence:

\[ \text{Order Setting} \]

The order of digits is set into the \( \kappa \) flip-flops by connecting these flip-flops to the serial logic are in the sequence from the least significant digit to the most significant. In the table, this sequence is represented by the address digits of \( \gamma \) during these two phases; the digits copied during the last word period of \( \gamma \) are entered into \( \alpha \). Hence:

\[ \text{Counter Address Augmentation Examples.} \]

### TABLE IV.

<table>
<thead>
<tr>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
<th>Example 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C'</td>
<td>-</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The above procedure is represented by the equations:

\[ K = G H C + \]

The double flip-flop arrangement for \( K \) (with interleaved clocks) ensures that the first 0 is complemented.

### Conditional Transfer Orders:

A. order is changed into a + order if the sign digit of \( \gamma \) is 1 or if the sign digit of \( \delta \) is 1 and the \( E \) switch is on. As \( \gamma \) alone distinguishes a - order from an - order, the above procedure is simply represented by:

\[ \gamma = Q_1 Q_2 Q_3 (A + R Z_1) + \]

### Accumulator Inputs:

The accumulator recirculates unchanged during every \( \gamma \), \( \alpha \), \( \beta \), and during \( \delta \) of \( 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \) and 6 orders. Hence:

\[ \gamma = A H Q_1 Q_2 + Q_3 t + F H Q_0 Q_2 + A \]

It is to be noted that the two \( \gamma \) equations (above) describe the \( \gamma \) accumulator input for all 16 orders, although only 15 are mentioned explicitly. For a \( \alpha \) order, the accumulator is cleared during \( \delta \) simply by omitting a term involving \( Q_0 Q_2 \) in the \( \gamma \) equations.

### Addition and Subtraction:

The two inputs to the adder-subtractor are denoted by \( \alpha \) and \( \beta \), and the output by \( \delta \). In the execution of \( \gamma \) and \( \delta \) orders occurring during \( \delta \), the two inputs are the \( \gamma \) (accumulator) digits, \( \alpha \) and \( \beta \) (operand) digits, respectively.

\[ \gamma = i A H +\]

Control signals \( \gamma = 0 \) or 1 are used to indicate whether subtraction or addition (respectively) is to be performed. \( \gamma \) alone distinguishes a \( \gamma \) order from an \( \delta \) order. Hence:

\[ S = H Q_1 +\]

The truth table for the formation of the sum and carry digits from the augend digit (i), the addend digit (j) and the carry digit from the next lower order (L) is shown as Table V (See Ref. 6).

### TABLE V.

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The truth table for the formation of the difference and borrow digits from the minuend digit (i), the subtrahend digit (j) and the borrow digit from the next lower order (L) is shown as Table VI.

### TABLE VI.

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Difference

Borrow

0 | 0 | 1 | 1 | 0 | 0 | 1
---|---|---|---|---|---|---|
0 | 0 | 1 | 1 | 0 | 0 | 1
It can be seen from the tables, that for the eight different combinations of \(i_1\) and \(L\), the sum and difference digits are the same. The following equation for \(b\) does not involve \(S\). The terms in the equation represent combinations of \(i_1\) and \(L\) which make the sum or difference digit equal to 1. Hence:
\[
b = i_1 L + i_1 L + i_1 L + i_1 L
\]

The most straightforward way of completing the design of the full adder-subtractor is by using the method described by Frankel. The carry/borrow flip-flop (L) is initially set to 0, and the conditions for setting and resetting the flip-flop are:

- The waste digit (which would depend on the control signal) delayed by 1 digit period and the output of the delay unit used as the control signal.
- The setting and resetting of the carry/borrow flip-flop may be represented by:

\[
\begin{align*}
E' &= (i_1 S + i_1 S) + \tau_a \\
E &= (i_1 S + i_1 S) + \tau_a
\end{align*}
\]

(N.B.—The waste digit \(\tau_a\) is the first digit in the word.)

**APPENDIX III.**

**Main Store Logic.**

The following descriptions refer to the signal flow and block diagram for the main store record and playback logic (Fig. 8).

**Shift Register Operation of the P Flip-flops:**

The six P flip-flops are connected as a shift register during the track of input order. The digit periods during which this shift register operation takes place are denoted by \(p\).

\[
p = GH + FHQ, Q,
\]

The input to the shift register is obtained from the two conditions \(P_r\) and \(P_i\), i.e., the conditions for setting and resetting the first stage.

The input digit is \(S\) (which is \(C\) during \(A\) and \(R\) during \(Q\), while \(G = 1\)), i.e., during \(A\) and \(Q\), and it is \(A\) during \(Q\) (represented by the condition \(G = 1\)).

\[
P_r = \bar{G} + \bar{F} G + \bar{Q} A + Q,
\]

\[
P_i = G + F G + Q A + \bar{Q},
\]

The \(P_i\) flip-flop follows \(P_i\) one digit later while \(P_r\) follows \(P_r\) and so on. Hence:

\[
P_r = \bar{F} + \bar{P},
\]

\[
P_i = F + \bar{P} + \bar{P},
\]

It is to be noted that the shifting operation takes place by virtue of the fact that each of the main flip-flops is actually represented by a pair of flip-flops operating with interlaced clocks.

**APPENDIX IV.**

**TABLE VII.**

The SNOCOM ORDER CODE.

<table>
<thead>
<tr>
<th>Instruction (Hexadecimal)</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>000A ABC0</td>
<td>&quot;Arithmetic and logical instructions&quot;</td>
</tr>
<tr>
<td>000B ABC0</td>
<td>Branch. Replace the contents of the accumulator by the contents of memory location (ABC) (sexadecimal).</td>
</tr>
<tr>
<td>000C ABC0</td>
<td>AND. Add the contents of location (ABC) to the contents of the accumulator and retain the sum in the accumulator.</td>
</tr>
<tr>
<td>000D ABC0</td>
<td>SUBTRACT. Subtract the contents of location (ABC) from the contents of the accumulator and retain the difference in the accumulator.</td>
</tr>
<tr>
<td>000E ABC0</td>
<td>MULTIPLY FRACTION. Multiply the contents of location (ABC) by the contents of the accumulator and retain the sign and most significant 30 bits of the product in the accumulator.</td>
</tr>
<tr>
<td>000F ABC0</td>
<td>MULTIPLY INTEGER. Multiply the contents of location (ABC) by the contents of the accumulator and retain the least significant 30 bits of the product (bits positions 0-29), making bit 30 of the accumulator equal to zero.</td>
</tr>
<tr>
<td>0010 ABC0</td>
<td>DIVIDE. Divide the contents of the accumulator by the contents of location (ABC) and retain the quotient (rounded to 30 bits plus sign) in the accumulator.</td>
</tr>
<tr>
<td>0011 ABC0</td>
<td>EXTRACT. Obtain the &quot;logical product&quot; of the contents of the accumulator and the contents of location (ABC) and retain the result in the accumulator.</td>
</tr>
<tr>
<td>0012 ABC0</td>
<td>&quot;Transfer of control instructions&quot;</td>
</tr>
<tr>
<td>0013 ABC0</td>
<td>&quot;Branch to control instructions&quot;</td>
</tr>
<tr>
<td>0014 ABC0</td>
<td>CONDITIONAL TRANSFER. Transfer control to (ABC) if the number in the accumulator is positive.</td>
</tr>
<tr>
<td>0015 ABC0</td>
<td>CONDITIONAL TRANSFER DEPENDING ON EXTERNAL TRANSFER SWITCH (Z_a). Transfer control to (ABC) if the number in the accumulator is negative or if the external transfer switch (Z_a) is &quot;on.&quot;</td>
</tr>
</tbody>
</table>

**Record Orders:**

- The time during which recording takes place is denoted \(t_1\) and corresponds to all of \(44\) for \(N\) and \(J\) orders but only the address part of \(44\) for 2 and 3 orders. Hence:

\[
f = FG + (Q, Q) (Q, Q)\]

The digits to be recorded are denoted \(V^*\). For \(N, T, Z\), and \(T\) orders they are the digits presented by the accumulator. For \(S\) orders, the address digit of \(G\) augmented again by 1 are to be recorded. This is carried in exactly the same way as the formation of \(C^*\) as in the following equation:

\[
V^* = V + Q \bar{A} + A + Q \bar{O} + (Q \bar{C} + K \bar{G} + C^*)
\]

It is to be noted that the \(K^*\) equation in Appendix II does not involve \(F\).

Hence this equation will be effective for \(44\) for the formation of \(C^*\) as well as in \(44\) for the formation of \(P\) on 3 orders.
Discussion

Dr. M. W. Woods (Member, Adelaide Division)—Can the author give any statistical information on the reliability of the computer in service, expressed as a percentage of hours out of service against total working hours? Such a figure is, to some extent, a "figure of merit" for the machine.

Mr. M. Kovarik (Senior Research Officer, Engineering Section, C.S.I.R.O.)—In coupling relatively slow output devices, such as the electrical typewriter mentioned in the paper, to computers, it is necessary to make sure that the computer is not trying to output a new message element before the output device is ready to accept it. In some output devices a special signal is available for the "ready" condition; otherwise a delay must be generated between message elements to be output by the computer. In the case of a typewriter, a delay long enough to cope with the carriage return function would slow down the machine operation excessively. How has this problem been solved in the SNOCOM design, and if the delay method has been used, what safety margins in the delay period have been found sufficient for trouble-free operation?

Professor A. R. Billings (Department of Electrical Engineering, The University of Western Australia)—I will disagree the chairman's interpretation of the word "reliable", and overstate the author and his co-workers on achieving so much so soon with so little and so few. Having said that, I would like to take issue with him on his contention that similar work in University departments deserves support. I think that in the last forty and early fifties, what computers were in their infancy, much valuable work could be and was contributed from Universities such as Manchester, Illinois and others. I now seriously doubt whether the construction and design of computers is a proper function for a University.

Commercially the principle is unsound. If computer manufacturers with their vast resources can only succeed in producing computers which are obsolete before they are sold, what chance does a University depart­ment have to be and was contributed from Universities such as Manchester, Illinois and others. I now very seriously doubt whether the construction and design of computers is a proper function for a University.

Excessively. How has this problem been solved in the SNOCOM design, and if this provides great advantages over the LGP 30 design. The second is to ask how the selection from several available data locations on the drum which are optimum for any instruction could best be handled by the pro­grammer.

Dr. B. Z. de Ferranti (Associate Member, Adelaide Division).—Mr. Q. Q. Wong's paper has been most comprehensive and paper on its excellent presentation.

SNOCOM has been the outcome of the excellent work which for quite some time has been associated with the Electrical Engineering Depart­ment at the University of Sydney. However, despite my own association with similar projects in universities I do not wish to enter into discussion as to whether or not I agree with the author and his co-workers on achieving this progress. However, point out there have been some significant examples of great success in the university, such as the University of Manchester.

I would like to raise two particular questions which arise from Mr. Wong's paper. The first would be to ask if he can explain the main reasons for building the timing unit rather than using a track on the drum and if this provides great advantages over the LGP 30 design. The second is to ask how the selection from several available data locations on the drum which are optimum for any instruction could best be handled by the pro­grammer.

To Mr. Kovarik.

To Professor Billings.

To Mr. de Ferranti.

To Dr. Woods.

Mr. P. T. Brown in Reply:

Dr. Woods.—During the period of a little more than a year for which SNOCOM has been in service, the computer has undergone unscheduled maintenance, principally associated with the reader and punch, for approximately 5 per cent of working hours. Routine operations, training mainte­nance personnel and minor design modifications to improve reliability occupied an additional 15 per cent of total working hours.

To Mr. de Ferranti.—An optimum assembly program for SNOCOM has been written and this will automatically place a constant or variable in a location which is optimal with respect to a single selected instruction. However, because variables are only available in one or four locations apart in the command sequence have respectively one and two optimal locations in common.

The Author in Reply:

I would like to thank Dr. Woods, Mr. Kovarik, Prof. Billings and Mr. de Ferranti for the intelligent and very thoughtful discussion, and also Mr. Woods for his contribution in reply to some of the questions.

To Mr. Kovarik.—After the initiation of a print cycle, the printer-ready signal mentioned in Section 7.2 inhibits the initiation of a second print cycle for a period which depends on whether or not the carriage- return-line-feed operation is to be executed during the current print instruction. The delay or lock-out periods corresponding to these two conditions are nominally 2 seconds and 100 milliseconds respectively. These delay periods are generated by incommensurate-inidis whose periods may be continuously adjusted. The nominal phrase quoted above include a safety margin of up to 50 per cent to ensure trouble-free operation.

To Professor Billings.—I agree with Prof. Billings in that there is little case for university computer systems to compete with industry in building large, faster and more reliable computers of the von Neumann type, as resources of industry are much greater and the building of such a machine in itself will not contribute to fundamental research.

There is no doubt that university groups do contribute substantially to computer technology without having to build large-scale machines. There is, however, some case for building relatively inexpensive machines at the university for both teaching and research purposes. The teaching computer should be as simple as possible so that the fundamentals of a von Neumann type machine may be readily demonstrated. Such a machine has been constructed within the Electrical Engineering School of the University of Sydney. The construction of a computer for research may be justified in two ways. Firstly, if the digital computer forms part of a more complex, larger system such as a hybrid analogue-digital system in which adaptive control systems (for example) are being studied, then I feel that the construction of the digital machine is completely justified if for any reason (including available financial) commercial machines are unsatisfactory. Secondly, if a fundamentally new computer organisation using new circuit elements promises to have advantages over existing systems, then I feel that these advantages should be established without any doubts by the con­struction of a complete machine. If a significant "break-through" in computer design can be made, there is less likely to be a market for the machine obsolete when it is completed.

To Mr. de Ferranti.—The main reason for generating the segment and digit waveforms of Fig. 2 in a timing unit instead of recording the relevant waveforms on tracks of the drum was that any digit within a week period could be readily derived from these waveforms. This made possible the generation of the bootstrapping sequence of the engineering model and the generation of one or more peripheral units with the computer proper. A timing unit to generate the segment number pattern had to be initially constructed because a particular semiconductor was not to be presented by the store playback amplifier was utilized by the auto input logic. The method of starting programs proved superior to that used in the LGP-30, and was only possible by using waveforms from the timing unit.

A significant improvement in SNOCOM's reliability was made after each period of unscheduled maintenance, and in 1955, shortly after the completion of its assembly and testing. The additional period was not found in the store playback amplifier was utilized by the auto input logic due to long processing times for this computer. It has been found that there is no significant advantage in using waveforms from the timing unit.
Summary

Digital computer fundamentals may be taught more effectively by the use of especially designed educational digital computers. These are experimental units containing sufficient logic and storage elements for the synthesis of a complete stored-programme computer.

This paper contains a description of the educational digital computer which has been used for the last three years by final-year students of electrical engineering at the University of Sydney. A detailed description of the logical design of a simple computer and a game-playing programme for this computer are also provided. Conclusions concerning the suitability of educational units like the one described for undergraduate courses in digital computers are made in the final section of the paper.

List of Symbols

(N.B. The complement of a Boolean variable is represented by a bar over the symbol.)

T<sub>0</sub> Clock pulse.
T<sub>1</sub> Binary counter drive pulse (see figs. 2 and 3).
B<sub>1</sub> - B<sub>16</sub> Binary counter outputs (see figs. 2 and 3).
F<sub>1</sub>-F<sub>4</sub> Waveform differentiated.
F<sub>4</sub>-F<sub>8</sub> Waveform differentiated.
I.S. Initial set.
Z<sub>1</sub>, Z<sub>2</sub> Signals which stop the timing unit (see fig. 2).
G<sub>0</sub> - G<sub>3</sub> Function toggle outputs.
G<sub>0</sub> - G<sub>3</sub> Sequence counter outputs.
C<sup>0</sup> Sequence counter input.
C<sup>1</sup> Sequence counter input.
C<sup>2</sup> Sequence counter drive input.
P<sub>0</sub> - P<sub>3</sub> Address register outputs.
P<sup>0</sup> Address register input.
P<sub>3</sub> Address register drive input.
V<sub>0</sub> Fixed store output.
W<sub>16</sub> Word 16 (= P<sub>0</sub>P<sub>1</sub>P<sub>2</sub>P<sub>3</sub>).
R<sub>0</sub> - R<sub>4</sub> Instruction register (address) outputs.
R<sup>0</sup> Instruction register (address) input.
R<sub>4</sub> Address register (drive) input.
R<sub>4</sub> - R<sub>7</sub> Instruction register (order) outputs.
R<sup>4</sup> Instruction register (order) input.
R<sub>4</sub> Instruction register (order) drive input.
A<sub>0</sub> - A<sub>3</sub> Accumulator outputs.
A<sup>0</sup> Accumulator input.
A<sub>4</sub> Accumulator drive input.
S<sub>0</sub> - S<sub>7</sub> Store register outputs.
S<sup>0</sup> Store register input.
S<sub>4</sub> Store register drive input.
K<sub>0</sub> Signal which sets the K flip-flop (to 1).
K<sup>0</sup> Signal which resets the K flip-flop (to 0).
K Counter augmentation flip-flop output.
L Carry/borrow flip-flop output.
b Sum/difference digit.
V Store output.

1. Introduction

As a result of the rapidly increasing use of digital equipment in Australia, there is a great demand for personnel trained in the fields of electronic data processing and digital systems engineering. Universities must play a leading role in training these personnel.

Training in digital computer fundamentals can be made more effective by the use of "educational digital computers". These are experimental units which have been specifically designed with the specification of a complete computer in mind. The number, type and arrangement of logical circuits provided in the experimental unit enable the synthesis not only of various combinational and sequential switching circuits, but also of a complete stored-programme computer.

The experimental unit to be described has been used for the last three years by final-year students of electrical engineering of the University of Sydney. Students are given the functional design of a simple computer and are asked to carry out the detailed logical design, to construct their computers and also to test their designs by writing and running a number of simple programmes. A better understanding of digital computers is obtained justifiably for the continuation and expansion of this educational approach.

2. Educational Digital Computers

The first digital-logic training devices were made commercially available in 1960, and now they are manufactured by about twenty companies.

The main use of these devices is the teaching of digital-circuit fundamentals to students doing digital computer courses at places of tertiary education as well as to engineers and technicians in industry who are engaged in the design or maintenance of computer systems. Other uses include the bread-boarding of equipment either for manufacture or for testing the performance of computer peripherals.

Some of the desirable features of a digital-logic training device are as follows: the trainer should contain standard transistorized digital circuits for good simulation of actual design problems; the circuit modules may be either fixed or removable, but the arrangement must be flexible so that many circuits may be synthesized; logic panels must be clearly engraved and extensive monitoring facilities must be provided to facilitate the understanding and testing of circuit configurations; the main control unit must enable the monitoring of memory states after each state-change (i.e. after each clock pulse); and finally the device must be "student-proof" so that any interconnection may be made without damage to the equipment.

Most digital-logic training devices only contain sufficient logic modules for the synthesis of a computer sub-assembly. The synthesis of a complete digital computer would be extremely useful to teach engineers computer

Figure 1.—Laboratory equipment for teaching digital computer fundamentals.

Figure 2.—Control and timing unit.

Figure 3.—Timing waveforms.
design techniques and to help programmers visualize the internal organization of a computer. The interconnection of a number of training devices is possible, but the ideal seems to be an experimental unit containing a combination of the devices. Basic circuits are used to teach digital-logic fundamentals such as the design of full-adders, counters and shift-registers, etc., while the additional sub-assemblies enable the synthesis of a complete computer. The sub-assemblies could be, for example, the techniques used for serial, synchronous systems; others could be used for asynchronous systems and so on. There is no doubt that training in the digital computer field would become more effective by the extensive use of this approach.

Most of the above features have been incorporated in an educational digital computer at the University of Sydney. A photographic of the educational computer is shown in fig. 1 and a brief description follows.

The educational computer consists essentially of a large number of logical and storage elements which may be readily patched together. The elements are arranged in six vertical panels and six sloping panels. Each of the two types of panels is removable after the disconnection of a voltage cable, and a package construction for the electronic circuitry is used. These features are desirable from the point of view of construction, testing and maintenance. The inputs and outputs of the elements are brought to sockets on the sloping panels so that a number of points may be linked without the use of busses. However, twenty busses which run the length of the unit are provided to eliminate the need for long wires.

The six vertical panels (from left to right) are as follows:

(i) Control and timing unit (see figs. 2 and 3).
(ii) Function toggles (providing 0/1 output for toggle up/down and sequence counter (a 4-stage shift register).
(iii) Address register (a 4-stage shift register) and fixed store (see sect. 2.3).
(iv) Instruction register (two 4-stage shift registers).
(v) Accumulator (an 8-stage shift register).
(vi) Store register (an 8-stage shift register).

The six sloping panels contain logical elements such as NOR circuits and flip-flops.

2.1 Control and timing unit

A control and timing unit is provided so that experiments on serial, synchronous, digital systems may be carried out. A logic diagram of this unit is shown in fig. 2, and timing waveforms are shown in fig. 3. The repetition rate may be either 50 kc/s or 5 kc/s, and is determined by selecting the appropriate output from two free running multivibrators. Under normal conditions, interposed clock pulses and are generated by three monostable multivibrators. A five-stage binary counter is driven by .

When the START button is depressed, the timing unit generates one pulse, eight pulses, thirty-two pulses or a train of pulses depending on whether the MODE-SELECT switch is in the BIT, PHASE, ORDER or NORMAL positions, respectively. When this switch is in the MONITOR position, the unit runs continuously and all waveforms may be monitored.

A signal appearing at the or inputs will stop the timing unit. The unit may be started again by depressing the RESET button and then the START button.

2.2 Shift registers

Each shift register is associated with two signals. One signal is the input to the first stage and represents the information which is to be stored in the register. The other is the SHIFT or DRIVE input and specifies the period during which the register is to operate as a shift register. This input usually consists of the clock pulse gated with other timing waveforms. When no drive input exists, the information in the register remains in static form. Each stage is monitored by an indicator light.

The eight-stage registers may be set initially to either state by a number of toggle switches and a SET button. The four or three may be initially set to zero by a CLEAR button.

2.3 Fixed store

The outputs of the address register are decoded into sixteen lines which drive the WORD-busses of the "pin-board" store. Component plugs containing sub-miniature diodes may be inserted into the eight WORD-busses with the eight WORD-busses at appropriate places. The pin-board, together with component plugs, forms a diode-matrix which produces (in parallel) the eight bits of the word specified by the contents of the address register. The diode-matrix outputs are combined with the timing waveforms in parallel-to-serial conversion circuits to produce the serial store output .

2.4 Logic panels

The four existing panels each contain two flip-flops and flip-flops with (transient) storage-gates in both input lines, three invertors and nineteen NOR circuits with either five or three inputs.

With the use of NOR circuits, any number of stages of logic may be cascaded without fear of error through cumulative voltage shifts. The circuits used also contain some safe-guard against inadvertent short-circuiting of two outputs.

3. The Design of a Simple Digital Computer

The sub-assemblies and logic circuits described in the preceding section may be interconnected to form a simple digital computer. Computers with different order codes may be constructed, the only restriction being the amount of hardware which is necessary for their implementation. The following sections are given as an example of a complete design.

3.1 Functional design

The computer to be designed is a fixed-point, binary, serial, single-address digital computer with a storage capacity of sixteen eight-bit words of which fifteen are fixed and one is erasable. The computer is capable of carrying out only eleven basic instructions, although four stages in the lower part of the instruction register make provision for sixteen. The order code is shown in table 1.

<table>
<thead>
<tr>
<th>Code</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>INPUT — Illuminate the Z₁ monitor light and stop the computer so that eight digits may be set into the accumulator.</td>
</tr>
<tr>
<td>0001</td>
<td>OUTPUT — Illuminate the Z₂ monitor light and stop the computer so that the accumulator may be monitored.</td>
</tr>
<tr>
<td>0100</td>
<td>TRANSFER IF NEGATIVE — Transfer control to the address specified by the instruction if the accumulator is negative.</td>
</tr>
<tr>
<td>0101</td>
<td>UNCONDITIONAL TRANSFER — Transfer control to the address specified by the instruction.</td>
</tr>
<tr>
<td>1000</td>
<td>INDIRECT STORE IN STORE — Add the contents of the store register (address 13) to the contents of the store register (address 12) in the accumulator.</td>
</tr>
<tr>
<td>1001</td>
<td>BRING — Bring the word specified by the address of the instruction into the accumulator.</td>
</tr>
<tr>
<td>1100</td>
<td>AND — Add the word specified by the address of the instruction to the contents of the accumulator retaining the result in the accumulator.</td>
</tr>
<tr>
<td>1101</td>
<td>SUBTRACT — Subtract the word specified by the address of the instruction from the contents of the accumulator retaining the result in the accumulator.</td>
</tr>
<tr>
<td>1110</td>
<td>TRANSFER IF ZERO — Transfer control to the address specified by the instruction if the accumulator is zero.</td>
</tr>
<tr>
<td>1111</td>
<td>ADD — Add the contents of the accumulator to the contents of the store register (address 14) retaining the result in the store register.</td>
</tr>
</tbody>
</table>

80 Proceedings I.R.E.E. Australia
February, 1965
The execution of each instruction takes place in four phases designated \( \phi_1, \phi_2, \phi_3, \) and \( \phi_4. \) These may be specified in terms of the timing waveforms \( B_3 \) and \( B_4. \)

The counter specifies the address of the next instruction which is to be obeyed. In \( \phi_1, \) the counter digits are set into the address register so that the correct instruction is presented by the store in the next phase.

In \( \phi_2, \) the instruction to be obeyed is transferred from the store to the instruction register. As the instructions in the store are normally obeyed sequentially, the counter must be augmented by one at some time in the machine cycle after it has directed the logic to set the current instruction in the instruction register. This is carried out in \( \phi_1, \) and the new augmented counter digits will be effective in the next \( \phi_3 \) unless the current instruction is an unconditional transfer or a successful conditional transfer order, in which case the augmented counter digits will be over-written in a later phase.

In \( \phi_4, \) the address digits of the instruction register are set into the address register so that the correct operand is presented by the store in the next phase.

In \( \phi_3 \) (the execution phase), the order specified by the order digits of the instruction register is carried out. As only eleven of the sixteen different combinations of \( R_7, R_6, R_5, \) and \( R_4 \) are used, the term \( R_7 \) is only included when the remaining three order digits are associated with two of the eleven instructions. The \( Z_1 \) and \( Z_2 \) computer stops are activated when \( R_7 R_6 R_5 R_4 = 1, \) and when \( R_7 R_6 R_5 R_4 a = 1, \) respectively. When \( R_7 R_6 R_5 R_4 = 1, \) the logical product of the word presented by the store are entered into the accumulator; for a BRING order the digits \( (V) \) presented by the store are entered into the counter register on the execution of TRANSFER orders. In \( \phi_4, \) the digits must be re-circulated unchanged because they may be required again in \( \phi_5. \) The digits are re-circulated unchanged also in \( \phi_3 \) so that the complete instruction may be monitored after its execution. The above conditions are satisfied by the following equations:

\[
R_{ad} = R_7 R_6 R_5 R_4 T_0 \\
R' = R_7 B_0 V + B_1 R_4
\]

### 3.2.3 Address register

The counter digits and the address digits of the instruction register are entered into the address register during the least significant four digit periods of \( \phi_4 \) and \( \phi_5 \) respectively. Hence:

\[
\Phi_1 = R_7 B_0 T_0 \\
\Phi' = R_7 C_0 + R_1 R_4 \Phi_0
\]

### 3.2.4 Accumulator

The shift register operation of the accumulator is required only in \( \phi_4 \) of STORE, BRING, ADD, SUBTRACT, ACCUMULATE IN STORE and EXTRACT orders. Hence:

\[
A' = R_7 B_0 B_4 R_7 T_0
\]

For a STORE order the accumulator re-circulates unchanged; for a BRING order the digits \( (V) \) presented by the store are entered into the accumulator; for ADD and SUBTRACT the accumulator input is the output \( (b) \) of the serial adder-subtractor. For an EXTRACT order the accumulator input is \( A_0 V. \) Hence:

\[
A'' = R_7 B_0 A_0 + R_1 B_0 V + R_1 B_0 b + R_1 R_4 R_1 A_0 V
\]

### 3.2.5 Adder-subtractor

A flip-flop (designated \( L \)) is used to hold the carry-borrow digit in the addition-subtraction process. The inputs to the adder-subtractor are the digits presented by the accumulator \( (A_0) \) and store \( (V). \) The output \( (b) \) is defined by the following equations:

\[
b = A_0 V + A_0 V L + A_0 V L + A_0 V L
\]

Following the design described in reference 3, the setting and resetting equations for the carry-borrow flip-flop are as follows:

\[
\begin{align*}
L' &= B_0 T_0 + (B_0 A_0 V + R_1 A_0 V) B_0 T_0 \\
L &= (B_0 A_0 V + R_1 A_0 V) B_0 T_0
\end{align*}
\]

The \( B_0 T_0 \) term in the \( L' \) equation resets the \( L \) flip-flop (to 0) before the commencement of the addition-subtraction operation in \( \phi_4. \)

### 3.2.6 Store register

The store register operates as a shift register only when word fifteen is indicated by the address register, and also only during \( \phi_4 \) and \( \phi_5 \) of the machine cycle. During \( \phi_4 \) and also during \( \phi_5 \) of all orders except STORE and ACCUMULATE IN STORE orders, the contents are re-circulated unchanged. During \( \phi_5 \) of a STORE order, the contents of the accumulator are stored in this register. During \( \phi_4 \) of an ACCUMULATE IN STORE order, the output of the adder-subtractor are entered into the accumulator.

\[
C_0 = B_0 T_0 \\
C' = C_0 + B_0 B_4 (K C_0 + K C_0) + B_1 B_0 B_4 C_0 + B_1 B_0 B_4 R_0 + B_1 B_0 B_4 R_0 + B_1 B_0 B_4 R_0 + B_0 B_4 R_0 \times (A_0 R_4 + a b R_4) + B_0 B_4 R_0]
\]

\[
K = B_0 T_0 \\
K' = B_0 T_0 \\
\alpha = A_0 A_0 A_0 A_0 A_0 A_0 A_0 A_0 A_0 A_0 A_0 A_0
\]

February, 1965

into this register. Hence:—

$$S_4 = W_1B_6R_5$$

$$S'' = E_5S_s + B_6R_5S_5 + B_8R_9S_5R_k(A_8R_5 + B_9R_k)$$

3.2.7 Store logic

When word fifteen is specified by the address register, the digit presented by the store (V) corresponds to the output of the store register (S). When word fifteen is not specified, V corresponds to the output of the fixed store (V0). Hence:—

$$V = W_1S_s + W_4V_0$$

3.2.8 Input and output

The computer may be stopped before the commencement of S4 by gating the ΔE5 signal with the appropriate order digits and supplying this signal to the Z1 or Z2 stop computer inputs. Hence:—

$$Z_1 = E_5E_6R_5 + S_4$$

$$Z_2 = E_4E_5R_4. ΔE16$$

3.3 Logical circuit configurations

The realization of the Boolean equations of section 3.2 in terms of NOR circuits and flip-flops is straightforward using the approach discussed in reference 4.

4. Programming the Computer

A number of programmes can be written to illustrate some elementary aspects of programming. These include:

(i) the manner in which transfer instructions break the normal sequencing of instructions,

(ii) the modification of instructions, and

(iii) the use of computer words as both numbers and instructions.

A programme which has provided some amusement during computer laboratory classes is shown in table 2. This is a programme for the computer to play one version of the game NIM. The player and the computer must alternately remove a number of pegs from a line. There are 49 pegs initially on the line and the maximum number which may be removed in one turn is 10. The player naturally remove a number of pegs from a line. There are 49 pegs initially on the line and the maximum number which may be removed in one turn is 10. The player

but refuses to take the last peg as it has been programmed to win.

(3) if the computer will take control and win once the player makes a mistake.

This programme to play NIM is the most complex one written for this simple computer. A severe restriction is the amount of erasable storage. This restriction will soon be removed by the addition of a small magnetic drum.

5. Conclusions

With the ever increasing use of the digital computer as a tool for problem-solving, it is becoming clear that computers should be introduced as early as possible in science and engineering courses.

With the use of problem-oriented programming languages there is no reason why this cannot be carried out at the first-year level. In the United States, first-year elective courses in computing are offered at some of the most renowned universities and the trend is towards the introduction of computing into the secondary schools.

For the efficient training of students in the use of computers, good man-machine communication is essential. The multi-console, time-shared computer system developed by project MAC at the Massachusetts Institute of Technology seems to be the ideal solution, and should be used as a guide for future expansion in computer facilities at Australian universities.

With the increasing use of digital equipment for educational purposes, there is no reason why this cannot be carried out at the first-year level. In the United States, first-year elective courses in computing are offered at some of the most renowned universities and the trend is towards the introduction of computing into the secondary schools.

With the use of problem-oriented programming languages there is no reason why this cannot be carried out at the first-year level. In the United States, first-year elective courses in computing are offered at some of the most renowned universities and the trend is towards the introduction of computing into the secondary schools.

With the use of problem-oriented programming languages there is no reason why this cannot be carried out at the first-year level. In the United States, first-year elective courses in computing are offered at some of the most renowned universities and the trend is towards the introduction of computing into the secondary schools.

With the use of problem-oriented programming languages there is no reason why this cannot be carried out at the first-year level. In the United States, first-year elective courses in computing are offered at some of the most renowned universities and the trend is towards the introduction of computing into the secondary schools.


The Design And Construction Of
The Digital Computer Arcturus

by

D. G. WONG,
School of Electrical Engineering,
The University of Sydney,
Sydney, N.S.W., Australia.

SUMMARY

The ARCTURUS computer, which was developed within the
Electrical Engineering School of the University of Sydney, is a
digital general-purpose computer utilizing packaged
transistor circuits, ferrite-core storage and
paper-tape peripheral units. This paper contains a brief
description of the computer, a survey of the design and
constructional techniques, and a detailed description of the
internal circuits, such as the carry-look-ahead-adder and the
multiplication procedure using multiplier recoding.

I. INTRODUCTION

For the past two decades, the School of Electrical Engineer­ing of the University of Sydney has been engaged in computer
research and development. Computing equipment which has
been constructed include the C.S.I.R.O. mechanical differential
computer(1), the digital differential analyser ADA(2) and the
general-purpose digital computer SNOCOM(3).

Computer teaching at the School has developed to the stage where all under-graduates are introduced to the use of
both digital and analogue computers, and most first-year
students elect to take two courses in which the emphasis is
on computer design. An educational digital computer(4, 5)
called NIMRAC was designed for computer design teaching and
found to be very effective for teaching digital computer
fundamentals.

The digital computer ARCTURUS is being constructed to
satisfy some of the ever-increasing teaching and research
requirements of the School. ARCTURUS is a rather large, a
parallel, binary, general-purpose digital computer using
packaged transistor circuits and ferrite-core storage, and
such it will be useful for teaching purposes. However, the
computer does contain some new features, and the ability to
modify the computer's hardware (perhaps to include an inter­
face) will make the computer a useful tool for research.

The design and construction of the peripheral equipment
for ARCTURUS was commenced about five years ago. Work
on the memory followed, leaving the design and construction of the
central processing unit to be carried out last. At the
time of writing all units have been checked out separately and
an advanced stage has been reached with the commissioning of the
computer.

II. SYSTEM DESIGN

2.1 Approach to the System Design

The approach which was taken for the system design of
ARCTURUS was determined largely by the aim of producing a
computer with the best possible specification by a small
group with very limited resources. The realization that the
construction of the computer would take many years deter­
mined the scale in which the units of the computer were
designed and built.

Former experience with the peripheral units of SNOCOM
and the availability of paper-tape preparation equipment
determined the specification of the peripheral units for
ARCTURUS. These took the form of a paper-tape reader,
with the possibility of adding a second, a paper-tape punch
and a monitor printer. Being electro-mechanical, the peri­
pheral units were expected to be the least reliable units of the
computer. Hence, a built-in check in the form of the
“programmed-operator” feature(6) enables a single
instruction to specify the address of an operand, and to store
the sequence counter (return link) and to transfer control to a
subroutine determined by the operation code of the instruction.
Such an instruction appears like a normal machine language
instruction but in fact could cause quite a complex operation (e.g.,
floating-point divide) to be carried out. A single-length
number consists of a sign bit and 19 magnitude bits; a double­
length number (such as the product formed by the multiplica­
tion of two single-length numbers) consists of a sign bit and
38 magnitude bits as shown in Figure 1. A two-complement
representation of negative numbers is used.

A block diagram of ARCTURUS is shown in Figure 3.
The memory output register (I), the accumulator (K), the
multiplexer-operating register (L) and the instruction register
(R) are all full-length registers, while the sequence counter
(S) and the operand counter (U) are only one-address-length.
The arithmetic unit makes extensive use of a high-speed carry
look-ahead-adder. By appropriate selection of its inputs, the
adder can perform such operations as binary addition and
subtraction, incrementing, etc. which are carried out on operands.
The “distributor-function-selector” selects arithmetic or logical
functions of register outputs and distributes these signals to all
register inputs. The computer becomes an arithmetic unit which is quite
powerful (in terms of arithmetic and logical functions which
modifications to be made readily for research purposes was
considered important. Some of the major decisions such as
the decision to use a carry-look-ahead-adder instead of an
asynchronous adder were left until quite late in the project.
However, at this stage the specification of the machine had
to be temporarily frozen to enable the detailed functional and
logical design to be carried out. After the machine has been
commissioned operating, corrections to the initial specifications, modifica­
tions can then be made.

2.2 General Description

ARCTURUS is a fixed-point binary, parallel, single-address,
general-purpose digital computer using packaged diode-transistor
circuits and containing core storage and paper-tape peripheral
units.

The instructions and number formats, which use a word
length of 20 bits, are shown in Figure 2. An operation code
of this code has 32 distinct instructions. Some of these have a
large number of variants specified by the bits normally
used as an address. A 13 bit address enables 8192 words to be
directly addressed, although at present only 1024 words are
available. When the indirect address bit is a ONE, the
address in the instruction is the address of the operand.
This indirect addressing may be carried to any level.

The “programmed-operator” feature(6) enables a single
instruction to specify the address of an operand, to store
the sequence counter (return link) and to transfer control to a
subroutine determined by the operation code of the instruction.
Such an instruction appears like a normal machine language
instruction but in fact could cause quite a complex operation (e.g.,
floating-point divide) to be carried out. A single-length
number consists of a sign bit and 19 magnitude bits; a double­
length number (such as the product formed by the multiplica­
tion of two single-length numbers) consists of a sign bit and
38 magnitude bits as shown in Figure 1. A two-complement
representation of negative numbers is used.
Fig. 1. Input-Output Unit.

Fig. 2. Instruction and Number Formats.

Fig. 3. Block Diagram of Arcturus.

Fig. 4. Memory Timing Logic.

Fig. 5. Memory Timing Waveforms.

Fig. 6. Photograph of Computer Waveforms. (A, B, C, D: Top to Bottom.)

Fig. 7. The Repeated Operations Timing Loop.

Fig. 8. Parallel, Reversible Binary Counter Logic.
2.3 Instruction Code

The ARCTURUS instruction code is shown in Table 1. More details are presented in the appendix. The "programmed-operator" feature.

The DISTRIBUTOR-FUNCTION-SELECTOR. As these tion, the variant bits determine the main control signals of

More details are presented in the appendix. As these possible), and it makes good utilization of the fast adder.

TO SUBROUTINE instruction plants the return link in the looping and the use of subroutines. For example, the JUMP memory. These operations require only a single memory

memory output register; the contents of the accumulator are added to the operand and the result is written back into memory. These operations require only a single memory cycle.

Some instructions (such as the nesting and hierarchy instruc-
tions) are more than one memory cycle for their execution.

The usefulness of these instructions is still to be established by programming experience. However, the imple-
mentation of quite complex instructions is instructive, and if these prove to be useful, faster implementations (requiring more hardware) or the use of machine organizations may evolve. The utilization of novel ideas will stimu-
late either designer or user into thinking about new machine organizations. At least some of the aims of building a computer at the University will be fulfilled.

The ARCTURUS instruction code is shown in Table 1.

3.3 Design of a Parallel Reversible Binary Counter

The design principle of a parallel, reversible binary counter is illustrated in Figure 8. In this figure the binary counter stages have OUTPUTS, which are designated Bn and Bn = 0 - 1 . . . N, where N is the total number of stages). The states consist of flip-flops whose outputs are gated and cross-coupled to their inputs via transient storage gates. When a GATE signal is a ONE, the next COUNT pulse is allowed to gate the G flip-flop to settle to its new state and to carry out the logic of the true circuit diagram of Figure 8. The counter is initially set to any state by using the RESET line and the INPUT PULSE waveforms are

The approach which must be taken to the functional and logical design of a complex digital system like a computer is firstly to break up the system into a number of distinct sections with well-defined links between sections. The separate design and implementation of each section is one way to make use of the computer's hardware in every distinct time period of the computer's operation. Logical design consists of implementing the functionality using the circuit elements and interconnection arrangements.

The approach which must be taken to the functional and logical design of a complex digital system like a computer is firstly to break up the system into a number of distinct sections with well-defined links between sections. The separate design and implementation of each section is one way to make use of the computer's hardware in every distinct time period of the computer's operation. Logical design consists of implementing the functionality using the circuit elements and interconnection arrangements.

The approach which must be taken to the functional and logical design of a complex digital system like a computer is firstly to break up the system into a number of distinct sections with well-defined links between sections. The separate design and implementation of each section is one way to make use of the computer's hardware in every distinct time period of the computer's operation. Logical design consists of implementing the functionality using the circuit elements and interconnection arrangements.

The approach which must be taken to the functional and logical design of a complex digital system like a computer is firstly to break up the system into a number of distinct sections with well-defined links between sections. The separate design and implementation of each section is one way to make use of the computer's hardware in every distinct time period of the computer's operation. Logical design consists of implementing the functionality using the circuit elements and interconnection arrangements.

The approach which must be taken to the functional and logical design of a complex digital system like a computer is firstly to break up the system into a number of distinct sections with well-defined links between sections. The separate design and implementation of each section is one way to make use of the computer's hardware in every distinct time period of the computer's operation. Logical design consists of implementing the functionality using the circuit elements and interconnection arrangements.

The approach which must be taken to the functional and logical design of a complex digital system like a computer is firstly to break up the system into a number of distinct sections with well-defined links between sections. The separate design and implementation of each section is one way to make use of the computer's hardware in every distinct time period of the computer's operation. Logical design consists of implementing the functionality using the circuit elements and interconnection arrangements.
clock pulses and hence determines the maximum repetition rates of the counter. As the total number of stages is increased, the only limitation is the fan-in of the NOR gates. If the fan-in is \( m \), an extra two NOR delays are added to the gate settling time for every \( m - 1 \) stages added.

3.4 Carry-Lookahead-Adder

3.4.1 The Carry-Lookahead-Adder Principle

In a conventional Ripple-CARRY-ADDER, the carry and sum bits of each full-adder stage are formed in succession as carry bits of less significant stages ripple through to more significant stages.

In a CARRY-LOOKAHEAD-ADDER, the carry bits of a number of adder stages are formed simultaneously. The increase in speed realized by such an adder can be seen by first considering two adjacent stages \( n \) and \( n - 1 \), where stage \( n + 1 \) is less significant. Using the normal definitions of the GENERATE and PROPAGATE functions \( (G_n, P_n) \), the relevant carry equations are:

\[
C_n = G_n + P_nC_{n+1}
\]

The direct circuit equivalents of the above equations (as used in the conventional ripple-carry-adder) would result in the \( C_n \) signal settling after \( C_{n+1} \) has settled as the \( C_n \) signal must pass through an AND-OR circuit configuration to form \( C_n \). Both carry signals may be formed simultaneously by generating \( C_n \) directly in terms of \( C_{n+2} \).

A limitation to the application of the above principle to a large number of adder stages is the fan-in of the circuits used. A reasonable number of stages in a group to which the above principle may be applied is five. GENERATE and PROPAGATE signals may be derived for the group, and the carry-lookahead principle may be applied to a number of groups. This process may be continued by designating five groups as a section and then using carry-lookahead circuits between sections. Carry-lookahead between stages within a group will be called ZERO-LEVEL-LOOKAHEAD, between groups within a section will be called FIRST-LEVEL-LOOKAHEAD, and between sections will be called SECOND-LEVEL-LOOKAHEAD.

3.4.2 Description of Adder with Zero-Level and First-Level-Lookahead

A block diagram of the adder is shown in Figure 7. A description follows.

The adder stages are numbered 0 to 19, with stage 0 being the most significant and stage 19 the least significant.

The adder inputs are \( A_i \) and \( B_i \) (\( i = 0 \ldots 19 \)), and the carry digit into the least-significant stage is designated \( C_0 \). The sum and carry digits are designated \( S_i \) and \( C_i \) respectively, PROPAGATE and GENERATE functions \( (P_i, G_i) \) are formed for each stage.

The adder is divided into 4 groups each of 5 stages. PROPAGATE and GENERATE functions may be formed for the three least significant groups. These first-level functions are designated \( P_i \) and \( G_i \) \((n = 1, 2, 3)\) with \( n = 3 \) corresponding to the least-significant group. The defining equations are:

\[
P_0 = p_0p_3p_6p_9
\]
\[
G_0 = g_0 + p_0g_1 + p_0p_3g_4 + p_0p_6g_7 + p_0p_9g_9
\]
\[
P_1 = p_1p_3p_6p_9 + p_1p_4p_8
\]
\[
G_1 = g_0 + p_0g_1 + p_0p_3g_4 + p_0p_6g_7 + p_0p_9g_9
\]
\[
P_2 = p_2p_3p_6p_9 + p_2p_4p_8 + p_2p_5p_7p_9
\]
\[
G_2 = g_0 + p_0g_1 + p_0p_3g_4 + p_0p_6g_7 + p_0p_9g_9
\]
\[
P_3 = p_3p_4p_5p_7p_9
\]
\[
G_3 = g_0 + p_0g_1 + p_0p_3g_4 + p_0p_6g_7 + p_0p_9g_9
\]

By using FIRST-LEVEL-LOOKAHEAD, i.e. applying the lookahead principle to groups of stages, we may look ahead to \( C_{19} \). \( C_{18} \), and \( C_0 \). The equations are:

\[
C_{12} = G_0 + P_0
\]
\[
C_{13} = G_0 + P_0G_1 + P_0P_1
\]
\[
C_{14} = G_0 + P_0G_1 + P_0P_1G_2 + P_0P_1P_2
\]

With the carry signals \( C_0 \), \( C_9 \), \( C_{12} \), \( C_{18} \), \( C_{20} \), the remaining carry signals may be formed using ZERO-LEVEL-Lookahead. The equations for \( C_i \) are:

\[
C_i = G_i + P_iC_i
\]

Similar equations apply to \( C_{19} \), \( C_{18} \), \( C_{17} \), \( C_{16} \), \( C_{15} \), \( C_{14} \).

The sum digits are formed in terms of carry digits and PROPAGATE functions. The equations are:

\[
S_i = p_iC_{i+1} + p_iC_{i+2} \quad (n = 0 \ldots 19)
\]

3.4.3 Selection of Adder Inputs

The adder is used for carrying out arithmetic operations on instructions (e.g., INDEXING instructions) as well as on numbers held in machine registers. A large number of different operations (such as CLEAR, SUBTRACT, DECREMENT, etc.) are required. Both these requirements must be satisfied by the appropriate selection of adder inputs.

Control signals \( a, s, r \) and \( k \) are used to select \( J, \bar{J} \), or 1 as the first adder input and \( A, \bar{A}, 0 \) or 1 as the second adder input. The
The time required for the multiplication processes would then be 

3.5.2 Speeding Up the Multiplication Process

The multiplication time may be reduced by providing input gates to each stage of the K register and the most significant stage of the L register. The ADD/ADD ZERO and SHIFT operations to be initiated by a single clock pulse. The multiplication time may be further reduced by "by-passing the adder". This occasionally involves providing additional input gates to the registers K and L to enable them to shift without using the adder, thus eliminating the ADD ZERO operation. The multiplication time is reduced because the shift time \( t_g \) may be significantly less than the ADD time \( t_a \). The multiplexer digit control the time interval (either \( t_g \) or \( t_a \)) between successive clock pulses. The multiplication time would then vary between \( t_g \) (corresponding to a multiplication of all ZEROS) and \( t_a \) (corresponding to a multiplier of all ONES). The number of SHIFT operations may be increased (and the number of ADD/ADD ZERO operations reduced) by "multiplier recoding". This produces a further reduction in the multiplications time.

3.5.3 Multiplier Recoding

By convention, the binary number \( 00111100 \) represents \( 2^4 + 2^3 + 2^2 + 2^1 \). A better understanding of the steps involved in a multiplication process (when the number is used as a multiplier) may be obtained by replacing PLUS SIGN (+) above the bit position of the multiplier which requires the ADD SHIFT operation (assuming this is possible using a single clock pulse), and a DOT (.) above the bit position which requires a PASS-SHIFT operation, (discarding the content of the adder to zero-bypassing). The PASS part of the PASS-SHIFT operation indicates that the partial product remains unchanged. The above number when used as a multiplier may then be coded as \( 00111101 \), and the sequence of PLUS SIGNS and DOTS moving from right to left (i.e. from the least to the most significant ends) would indicate the steps involved in the multiplication process. The binary number \( 00111100 \) contains a STRING OF ZEROS and represents \( 2^4 + 2^3 + 2^2 + 2^1 \). This suggests an alternative multiplication procedure, in which the binary representation of the multiplier involves only two ADD-SHIFT or SUBTRACT-SHIFT operations required by the process using the conventional coding of the multiplier.

The decision to ADD-SHIFT, SUBTRACT-SHIFT or PASS-SHIFT is made by specifying a number of multiplier bits starting from the least significant end and progressing to the most significant end. A good decision is one that will PASS-SHIFT the largest number of bits, i.e. it will postpone (as far as possible) any ADD-SHIFT or SUBTRACT-SHIFT operation. When two multiplier digits are sensed at each step, the decision is based on the concepts of a STRING OF ONES, ISOLATED ZEROS and ISOLED ONES.

<table>
<thead>
<tr>
<th>REGISTER</th>
<th>SIGNAL</th>
<th>CLOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>Dk</td>
<td>Rk</td>
</tr>
<tr>
<td>L</td>
<td>Dl</td>
<td>Rl</td>
</tr>
</tbody>
</table>

The time required for the multiplication processes would then be 

The number of SHIFT operations may be increased (and the number of ADD/ADD ZERO operations reduced) by "multiplier recoding". This produces a further reduction in the multiplications time.

3.5.3 Multiplier Recoding

By convention, the binary number \( 00111100 \) represents \( 2^4 + 2^3 + 2^2 + 2^1 \). A better understanding of the steps involved in a multiplication process (when the number is used as a multiplier) may be obtained by replacing PLUS SIGN (+) above the bit position of the multiplier which requires the ADD SHIFT operation (assuming this is possible using a single clock pulse), and a DOT (.) above the bit position which requires a PASS-SHIFT operation, (discarding the content of the adder to zero-bypassing). The PASS part of the PASS-SHIFT operation indicates that the partial product remains unchanged. The above number when used as a multiplier may then be coded as \( 00111101 \), and the sequence of PLUS SIGNS and DOTS moving from right to left (i.e. from the least to the most significant ends) would indicate the steps involved in the multiplication process. The binary number \( 00111100 \) contains a STRING OF ZEROS and represents \( 2^4 + 2^3 + 2^2 + 2^1 \). This suggests an alternative multiplication procedure, in which the binary representation of the multiplier involves only two ADD-SHIFT or SUBTRACT-SHIFT operations required by the process using the conventional coding of the multiplier.

The decision to ADD-SHIFT, SUBTRACT-SHIFT or PASS-SHIFT is made by specifying a number of multiplier bits starting from the least significant end and progressing to the most significant end. A good decision is one that will PASS-SHIFT the largest number of bits, i.e. it will postpone (as far as possible) any ADD-SHIFT or SUBTRACT-SHIFT operation. When two multiplier digits are sensed at each step, the decision is based on the concepts of a STRING OF ONES, ISOLATED ZEROS and ISOLED ONES.

There are two cases as follows. Two or more adjacent ONES constitute a STRING OF ONES, and as noted above, the least significant ONE of a STRING OF ONES is interpreted as 1 while the ZEROS or bits to the left and right of the most significant ONE are interpreted as 0. An ISOLATED ZERO is a zero which is flanked by ONES, which form part of a STRING OF ONES, and is to be interpreted as 0. An ISOLATED ONE is a ONE which is not a member of a STRING OF ONES and which is flanked by ZEROS. An ISOLATED ONE is interpreted as 1.

Based on the above concepts, \( P_1 \) and \( P_1 \) represent the PRESENT OPERATION TO BE PERFORMED is determined by the LAST OPERATION PERFORMED and by the two multiplier digits \( p_1 \) and \( p_2 \) which are sensed in the \( r \)-th step \((r = 1, n)\). Before the first step, the LAST OPERATION PERFORMED is assumed to be an ADD-SHIFT operation, and the following table defines the PRESENT OPERATION TO BE PERFORMED.

<table>
<thead>
<tr>
<th>MULTIPLIER BITS</th>
<th>LAST OPERATION PERFORMED</th>
<th>INTERPRETATION</th>
<th>PRESENT OPERATION TO BE PERFORMED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADD-SHIFT ( P_1 )</td>
<td>CHOOSING A STRING TERMINATING A STRING</td>
<td>SUBTRACT-SHIFT ADD-SHIFT</td>
</tr>
<tr>
<td>1</td>
<td>SUBTRACT-SHIFT ( P_2 )</td>
<td>ISOLATED ZERO</td>
<td>ADD-SHIFT ADD-SHIFT</td>
</tr>
<tr>
<td>1</td>
<td>ADD-SHIFT ( P_3 )</td>
<td>ISOLATED ONE</td>
<td>ADD-SHIFT ADD-SHIFT</td>
</tr>
</tbody>
</table>

Table 4

A further saving of multiplication time may be obtained by considering two multipliers digits in each step, and the incremental saving diminishes as more multiplier digits are considered.

3.5.4 An Implementation of a Multiplication Procedure Using Multiplier Recoding

The repeated operations timing loop is shown in Figure 7 is used to generate timing pulses required
by the multiplication process. The number of products is determined by the F counter. This set is initialized to 20 and is decreased to 0 as the processing continues.

The F flip-flop. The F flip-flop is used to determine whether an addition or a subtraction is to be carried out. The control signals on the input of the flip-flop are the sign digit. Hence an initial arithmetic shift of one place is required.

Noting that as L and F are to change at the same time, transient storage gates are necessary to maintain the setting and resetting signals while the flip-flop is turning. The clock for F is T0P. The G flip-flop. The G flip-flop is used to determine whether the operation to be performed is an ADD/SUBTRACT-SHIFT operation or a PASS-SHIFT operation. The former operation is performed when G is ZERO and the latter when G is ONE.

Starting from the information in table 4, it can be shown that an ADD/SUBTRACT-SHIFT operation is to be performed when L - 19 and G = 0. Hence the setting and resetting signals for G are as follows:—

\[ G' = \text{L} \text{L9F} + \text{L} \text{L9F} \]

As L, G, and F are to change at the same time, transient storage gates are also required for G. The clock for G is T0P.

The timing diagrams of Figure 13 correspond to a multiplier of four stages.

It can be seen that the outputs of the ADD/SUBTRACT-SHIFT operation are to be performed, there is a period of 500 ps between T0P and the next Clock. If K0 or K1 but, when a PASS-SHIFT operation is to be performed, this period is reduced to 250 ps. Hence 500 ps are allowed for the setting of the flip-flops F and G, and the adder and the gates, while only 250 ps is allowed when the adder is by-passed.

The SRO flip-flop. The multiplication process requires 20 pairs of counting adders as stages. As the register L contains only 20 digits, an additional multiplier digit must be obtained. Consideration of the negative multiplier digit and a repetition of the multiplier sign digit. Hence an initial arithmetic shift of one place is required, during which the process originally held in L is not lost but is stored in the cascade flip-flops F and G.

During the first nineteen (of the twenty) steps in the multiplication process, the least significant stage of the last significant adder (K0 or K1) is set into L. By this process, the least significant half of the product is gradually formed in L. During the twentieth step, K 0 is stored in the least significant adder(s) of the subtractor (F or G), and a logical shift of one place is carried out in L. These steps ensure the correct positioning of the double-length product in A and L. (See Figure 2).

The SRO flip-flop is used to generate the digit (L) which is to be set into L, to satisfy the above requirements. As V is determined by 1/xn and the shift pulse may be a T0P pulse, only 125 ns would be allowed for the L storage gates to settle if the signal V = 0. It would be better to sense the condition of the least significant adder (K0 or K1) only when a PASS-SHIFT operation is to be performed, this period is reduced to 250 ps. Hence 500 ps are allowed for the setting of the flip-flops F and G, and the adder and the gates, while only 250 ps is allowed when the adder is by-passed.

The SRO flip-flop is initially set to ZERO by the INITIAL SET pulse. By this process, the least significant half of the product is gradually formed in L. During the twentieth step, K 0 is stored in the least significant adder(s) of the subtractor (F or G), and a logical shift of one place is carried out in L. These steps ensure the correct positioning of the double-length product in A and L. (See Figure 2).

The SRO flip-flop Flip-flop is used to generate the digit (L) which is to be set into L, to satisfy the above requirements. As V is determined by 1/xn and the shift pulse may be a T0P pulse, only 125 ns would be allowed for the L storage gates to settle if the signal V = 0. It would be better to sense the condition of the least significant adder (K0 or K1) only when a PASS-SHIFT operation is to be performed, this period is reduced to 250 ps. Hence 500 ps are allowed for the setting of the flip-flops F and G, and the adder and the gates, while only 250 ps is allowed when the adder is by-passed.

The SRO flip-flop is used to generate the digit (L) which is to be set into L, to satisfy the above requirements. As V is determined by 1/xn and the shift pulse may be a T0P pulse, only 125 ns would be allowed for the L storage gates to settle if the signal V = 0. It would be better to sense the condition of the least significant adder (K0 or K1) only when a PASS-SHIFT operation is to be performed, this period is reduced to 250 ps. Hence 500 ps are allowed for the setting of the flip-flops F and G, and the adder and the gates, while only 250 ps is allowed when the adder is by-passed.

The SRO flip-flop is used to generate the digit (L) which is to be set into L, to satisfy the above requirements. As V is determined by 1/xn and the shift pulse may be a T0P pulse, only 125 ns would be allowed for the L storage gates to settle if the signal V = 0. It would be better to sense the condition of the least significant adder (K0 or K1) only when a PASS-SHIFT operation is to be performed, this period is reduced to 250 ps. Hence 500 ps are allowed for the setting of the flip-flops F and G, and the adder and the gates, while only 250 ps is allowed when the adder is by-passed.

The SRO flip-flop is used to generate the digit (L) which is to be set into L, to satisfy the above requirements. As V is determined by 1/xn and the shift pulse may be a T0P pulse, only 125 ns would be allowed for the L storage gates to settle if the signal V = 0. It would be better to sense the condition of the least significant adder (K0 or K1) only when a PASS-SHIFT operation is to be performed, this period is reduced to 250 ps. Hence 500 ps are allowed for the setting of the flip-flops F and G, and the adder and the gates, while only 250 ps is allowed when the adder is by-passed.

The SRO flip-flop is used to generate the digit (L) which is to be set into L, to satisfy the above requirements. As V is determined by 1/xn and the shift pulse may be a T0P pulse, only 125 ns would be allowed for the L storage gates to settle if the signal V = 0. It would be better to sense the condition of the least significant adder (K0 or K1) only when a PASS-SHIFT operation is to be performed, this period is reduced to 250 ps. Hence 500 ps are allowed for the setting of the flip-flops F and G, and the adder and the gates, while only 250 ps is allowed when the adder is by-passed.

The SRO flip-flop is used to generate the digit (L) which is to be set into L, to satisfy the above requirements. As V is determined by 1/xn and the shift pulse may be a T0P pulse, only 125 ns would be allowed for the L storage gates to settle if the signal V = 0. It would be better to sense the condition of the least significant adder (K0 or K1) only when a PASS-SHIFT operation is to be performed, this period is reduced to 250 ps. Hence 500 ps are allowed for the setting of the flip-flops F and G, and the adder and the gates, while only 250 ps is allowed when the adder is by-passed.
a commercially available computer of comparable cost.

Introductory courses on the use of computers can be taught more effectively using an open-shop system with a small (cheap) computer than with a closed-shop system with a large (expensive) computer which does not have time-sharing, multi-console facilities. Teaching in the field of computer design which may be carried out at a more advanced level (e.g., final-year, honours or post-graduate levels) can also be taught more effectively as the staff concerned will have a more intimate knowledge of the hardware of the computer, and design examples involving diverse practical aspects will be more readily available.

In the context of time-sharing systems, much research effort has been directed at special peripheral devices for improving man-machine communication14, and it is very likely that research along these lines will continue for some time15. A home-made computer could be a very useful research tool here, as modifications and additions to the computer could be made readily to incorporate the device. A flexible computer (i.e., one which can be modified easily) can also be useful for research into the organization of computers. Special instructions built into the computer could suggest different configurations or special hardware for improving its performance. Other applications of a flexible computer would include on-line processing of test data and research into the computer control of complex systems. These applications would involve the interconnection of computer and test set or system via analogue-digital converters.

The third area in which the construction of a computer would prove to be more beneficial than the purchase of a commercial machine would be that of staff training. ARCTURUS (like most computers) is a very complex piece of equipment. The approach taken to its design and construction with very limited resources was good engineering training. Such training is essential if university lecturers are to reveal to students scientific and engineering experience concerning the application of abstract theories so that the students' creative abilities are developed as well as their memories.36

The need for Computer Science courses, involving the theory, design, control and application of information-processing systems, to provide adequately trained computer personnel has been recognized for a number of years. Many of these have been available only to post-graduate students or undergraduates in their final years, but it does appear that the trend is towards courses which form a major part of a first-degree program17. Logical design forms an important part of these courses. A completely theoretical treatment of logical design is not sufficient as it is closely related to circuit design and the properties of components. These related subjects are, of course, treated in an Electrical Engineering course. Hence it does appear that hardware projects (involving design) should be carried out at Schools of Electrical Engineering at Australian Universities. Not only will this produce adequately trained staff, but it is likely that computer teaching equipment will evolve as a by-product and student interest will be stimulated in an important area of Computer Science.

VII. ACKNOWLEDGEMENTS

I would like to thank Professor W. N. Christiansen, Head of the School of Electrical Engineering, University of Sydney for his encouragement and support of the ARCTURUS project.

I am gratefully indebted to my colleagues Messrs. K. R. Rosolen and I. M. Cowell for their assistance with the construction of ARCTURUS. Mr. Rosolen is to be given much of the credit for the circuit design and constructional techniques.

VIII. REFERENCES

### IX. APPENDIX

#### TABLE 5. DESCRIPTION OF INSTRUCTIONS

<table>
<thead>
<tr>
<th>CODE</th>
<th>INSTRUCTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>STOP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rₐ₁Rₐ₄</td>
<td>00 Unconditional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 Contingent on BP1 ON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 Contingent on BP2 ON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 Continue</td>
</tr>
<tr>
<td>00001</td>
<td>INPUT-OUTPUT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rₐ₁</td>
<td>0 4-bits</td>
</tr>
<tr>
<td></td>
<td>Rₐ₅</td>
<td>1 5-bits</td>
</tr>
<tr>
<td></td>
<td>Rₐ₆</td>
<td>Reader 1</td>
</tr>
<tr>
<td></td>
<td>Rₐ₇</td>
<td>Printer</td>
</tr>
<tr>
<td></td>
<td>Rₐ₈</td>
<td>Punch</td>
</tr>
<tr>
<td></td>
<td>Rₐ₉-Rₐ₁₄</td>
<td>Number of times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A cyclic left shift of K 4 or 5 places depending on Rₐ₉ precedes each operation. Reading overwrites Kₜₙ or Kₜ₂ to Kₜₙ. If Rₐ₉ = 0, the character read is the next non-fifth bit character in the buffer or on the tape. The character printed or punched is either Kₜₙ-Rₜₙ₄ or Kₜ₂-Rₜ₂₆. Reading precedes printing or punching.</td>
</tr>
<tr>
<td>00010</td>
<td>REGISTER-TRANSFER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rₐ₁Rₐ₄</td>
<td>00 Transfer L to J first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 Transfer K to J first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 Transfer U to J first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 Transfer U to J first</td>
</tr>
<tr>
<td></td>
<td>Rₐ₉</td>
<td>a if Rₐ₉ = 0</td>
</tr>
<tr>
<td></td>
<td>Rₐ₁₀</td>
<td>s if Rₐ₁₀ = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cᵢ if Rₐ₃ = 0</td>
</tr>
<tr>
<td></td>
<td>Rₐ₁₁</td>
<td>if Rₐ₉ = 0, Rₐ₁₀ = 0, Rₐ₁₁ = 0</td>
</tr>
<tr>
<td></td>
<td>Rₐ₁2</td>
<td>k</td>
</tr>
<tr>
<td></td>
<td>Rₐ₁₃-Rₐ₁₄</td>
<td>00 a₉ = 1 ≠ p = 0 p = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 a₉ = 0 ≠ p = 0 p = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 a₉ = 0 = p = 1 p = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 a₉ = 0 ≠ p = 1 p = 1</td>
</tr>
<tr>
<td></td>
<td>Rₐ₁₄</td>
<td>Clock distributor into U</td>
</tr>
<tr>
<td></td>
<td>Rₐ₁₅</td>
<td>Clock distributor into K</td>
</tr>
<tr>
<td></td>
<td>Rₐ₁₆</td>
<td>Clock distributor into L</td>
</tr>
<tr>
<td>00011</td>
<td>SHIFT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rₐ₁Rₐ₄Rₐ₅</td>
<td>000 Logical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>010 Arithmetic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>001 Cyclic together</td>
</tr>
<tr>
<td></td>
<td>Rₐ₁₋₇</td>
<td>00 Shift L</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 Shift K</td>
</tr>
<tr>
<td></td>
<td>Rₐ₉-Rₐ₁₄</td>
<td>Number of times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SY) = A</td>
</tr>
<tr>
<td>00100</td>
<td>TRANSFER UNCONDITIONAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(SY) = A</td>
<td>if Kₙ = 1</td>
</tr>
<tr>
<td>00101</td>
<td>TRANSFER IF NEGATIVE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(SY) = A</td>
<td>if Kₙ = 1</td>
</tr>
<tr>
<td>00110</td>
<td>SKIP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rₐ₁</td>
<td>Accumulator normalized</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 —</td>
</tr>
<tr>
<td></td>
<td>Rₐ₁Rₐ₄</td>
<td>01 Accumulator &lt; 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 Accumulator = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 Accumulator &gt; 0</td>
</tr>
<tr>
<td></td>
<td>Rₐ₅-Rₐ₈</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 Sense switch 1 ON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 Sense switch 2 ON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 Sense switch 3 ON</td>
</tr>
<tr>
<td></td>
<td>Rₐ₅-Rₐ₈</td>
<td>Number of times</td>
</tr>
</tbody>
</table>

#### TABLE 5 (continued)

<table>
<thead>
<tr>
<th>CODE</th>
<th>INSTRUCTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>00111</td>
<td>ADD</td>
<td>ADD (M) to address of next instruction</td>
</tr>
<tr>
<td>00100</td>
<td>ADD INDEX</td>
<td>ADD (M) to address of next instruction</td>
</tr>
<tr>
<td>00101</td>
<td>ADD INDEX</td>
<td>Subtract (M) from address of next instruction</td>
</tr>
<tr>
<td>01000</td>
<td>INDEX SKIP</td>
<td>(MY) = (M) + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SY) = (S) + 1 if (M)ₚ = 1</td>
</tr>
<tr>
<td>01001</td>
<td>JUMP TO SUBROUTINE</td>
<td>(MY) = (S)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SY) = (S) + 1</td>
</tr>
<tr>
<td>01010</td>
<td>INDEX SKIP</td>
<td>(MY) = (S) + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SY) = (S) + 1 if (M)ₚ = 1 (Kₙ) for all i such that (Kₙi) = 1</td>
</tr>
<tr>
<td>01011</td>
<td>LOAD K</td>
<td>(KY) = (K)</td>
</tr>
<tr>
<td>01100</td>
<td>LOAD L</td>
<td>(LY) = (L)</td>
</tr>
<tr>
<td>00111</td>
<td>ADD</td>
<td>(KY) = (K) + (M)</td>
</tr>
<tr>
<td>00000</td>
<td>ADD INDEX</td>
<td>(KY) = (K) – (M)</td>
</tr>
<tr>
<td>00001</td>
<td>AND</td>
<td>(KY) = (K) – (M)</td>
</tr>
<tr>
<td>00100</td>
<td>MULTIPLY</td>
<td>(KY) = (L) × (M)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lₜₙ = 0</td>
</tr>
<tr>
<td>00111</td>
<td>DIVIDE</td>
<td>(LY) = (L) × (M)</td>
</tr>
<tr>
<td>00100</td>
<td>STORE K</td>
<td>(MY) = (K)</td>
</tr>
<tr>
<td>00101</td>
<td>STORE L</td>
<td>(MY) = (L)</td>
</tr>
<tr>
<td>00110</td>
<td>STORE K ADDRESS</td>
<td>(MYₜₙ = (Mₜₙ + Kₜₙ)</td>
</tr>
<tr>
<td>00111</td>
<td>ACCUMULATE</td>
<td>(MY) = (M) + (K)</td>
</tr>
<tr>
<td>10000</td>
<td>(SPARE)</td>
<td></td>
</tr>
<tr>
<td>10001</td>
<td>(SPARE)</td>
<td></td>
</tr>
<tr>
<td>10110</td>
<td>PUSH</td>
<td>((MY)') = (K)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(MY) = (M) + 1</td>
</tr>
<tr>
<td>10111</td>
<td>POP</td>
<td>(KY) = (K)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(MY) = (M) – 1</td>
</tr>
<tr>
<td>11000</td>
<td>DECREMENT HIERARCHY</td>
<td>(K) = (K) – 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SY) = A</td>
</tr>
<tr>
<td>11010</td>
<td>INCREMENT HIERARCHY</td>
<td>(K) = (K) + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SY) = A</td>
</tr>
<tr>
<td>11100</td>
<td>RETURN HIERARCHY</td>
<td>(Kₙ) = 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 Increase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rₐₙ Change hierarchy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rₐₙ-Rₐ₈ Number of times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(C ) = ( ) + (M times)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SY) = (C )</td>
</tr>
<tr>
<td>11111</td>
<td>EXECUTE</td>
<td>Execute instruction in M</td>
</tr>
</tbody>
</table>