PROGRAMMING MANUAL *FOR THE* ASI •••••• 210 COMPUTER

SYSTEM

MINNEAPOLIS 22, MINNESOTA

PROGRAMMING MANUAL

FOR THE

ASI-210 COMPUTER SYSTEM

SECTION I

SECTION II

August 1962

210-3

VOLUME 3

PROGRAMMING MANUAL

TABLE OF CONTENTS

SECTION I: BASIC PROGRAMMING

TABLE OF CONTENTS (CONT.)

TABLE OF CONTENTS (CONT.)

SECTION II: ADVANCED PROGRAMMING

TABLE OF CONTENTS (CONT.)

LIST OF DRAWINGS

SECTION II

LIST OF TABLES

SECTION I

SECTION II

VOLUME 3

PROGRAMMING MANUAL

SECTION I

BASIC PROGRAMMING

1.1   INTRODUCTION

This section of the programming manual was designed to be used
by both the experienced and the inexperienced programmer. For
the experienced programmer this manual contains such necessary
information as instruction lists, breakdown of the instruction word
and use of input/output instructions.

For the inexperienced programmer this manual contains, in addi-
tion to the instruction lists, etc. an explanation of each instruction,
12 sample programs to illustrate instructions and coding procedures
and the mechanics of coding and debugging a program. If the
principles set forth in this manual are correctly applied, the pro-
gramming of the ASI-210 should become, with a reasonable amount
of practice, a fairly easy task.

The purpose of the first section of the programming manual is to
give the beginning programmer a foundation upon which to build his
knowledge of programming the ASI-210.

The first area that must be taken into account when programming
is considered. is not what programming is, but why programming
is necessary.

All computers have a language. In the modern digital computers,
this language is usually the binary number system. (The binary
number system is a number system that uses only two characters,
one and zero.) Any instructions to the computer or any number
that is to be used by the computer must be in the format of the
binary number system.

It is the programmer's responsibility to present the problem he
wishes the computer to solve in the language of the computer.
Therefore, his foremost duty is not one of solving complex prob-
lems, but of translating the problem from the language that is
understandable to men, to the language that is understandable to
the computer.

It should be noted that while solving complex problems is not the programmer's primary duty, it is necessary that he be able to solve a problem before he can instruct the computer in the method of solving the problem.

1.2 PROGRAMMING FUNDAMENTALS

1.2.1 INTRODUCTION

There are several "building blocks" that are necessary to form a good foundation for the study of programming. These building blocks are (1) flow diagrams, (2) loops and (3) subroutines.

The following paragraphs will explain each of these building blocks and illustrate their use in the writing of a program.

1.2.2 FLOW DIAGRAMS

The first thing a programmer does, after determining what exactly the problem is and how he will solve it, is to write a flow diagram of the solution of the problem. A flow diagram is a pictorial representation of the logical sequence of the solution of a problem. It is made up of several different symbols indicating the various functions that are to be performed by the program. (A list of the symbols and their explanation will be found in paragraph 1.2.3 of this section.)

The preparation of a flow diagram consists of several steps. The first step is to write a "general" flow diagram of the program. The blocks in a general flow diagram will illustrate the basic operations to be performed.

For example, the general flow diagram of a program to compute the hypotenuse of a triangle would look like the following:
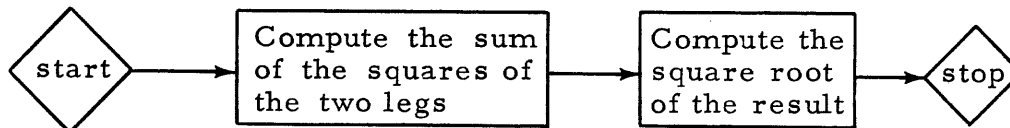
Note the inclusion of the start and stop commands.

Once the general flow diagram is written, the next step is to break each block of the general flow diagram down into

blocks that will require only one or two instructions to the computer. It would probably only require one step to accomplish this for the problem above but for more complex problems, it may take several steps to arrive at the final flow diagram. The number of steps it takes to break the general flow diagram down is not important as long as the logical sequence of the solution is maintained. The final flow diagram for the above problem would look like the following:

```
  ◇           ┌──────────────┐    ┌──────────────┐    ┌──────────────┐
 start ──────▶│ find the value│──▶│ square the   │──▶│ store the result│
  ◇           │ of the first leg│  │ value of the │    │ in memory      │──┐
              └──────────────┘    │ first leg    │    └──────────────┘  │
                                  └──────────────┘                       │
   ┌──────────────┐    ┌──────────────┐    ┌──────────────┐             │
   │ add the squared│◀─│ square the   │◀─│ find the     │◀────────────┘
 ┌─│ values of the │   │ value of the │    │ value of the │
 │ │ two legs     │   │ second leg   │    │ second leg   │
 │ └──────────────┘    └──────────────┘    └──────────────┘
 │
 └─▶ etc.
```

└─▶ etc. until the hypotenuse of the triangle is found.
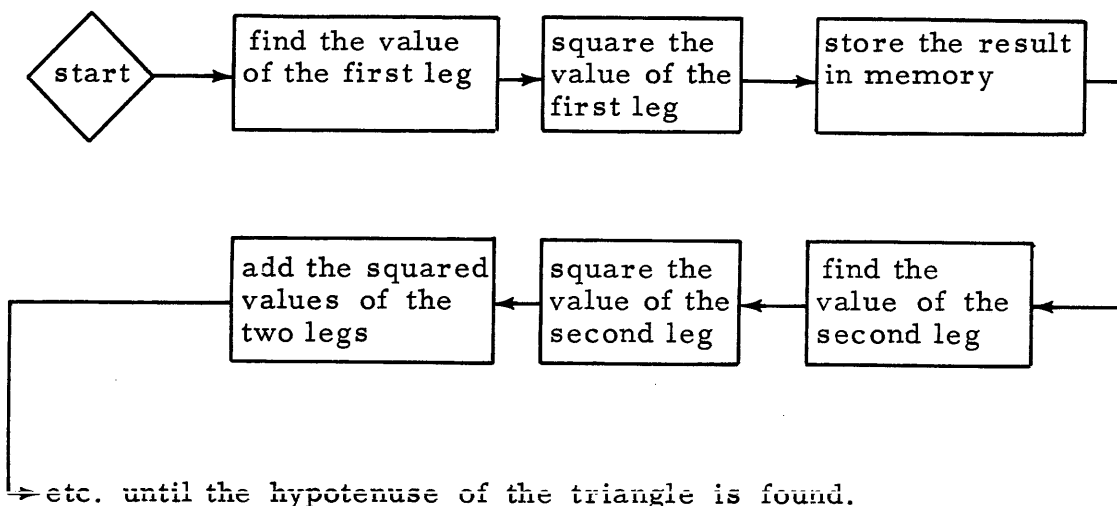
When the flow diagram is in its final form, it is a fairly easy task to translate each block of the flow diagram into instructions to the computer.

1. 2. 3   FLOW DIAGRAM SYMBOLS

1. ┌──────────┐
    │   │
    └──────────┘

statement, explanation or assertions

example   ┌──────────┐
                 │ Add A   │
                 │ and B   │
                 └──────────┘

2.

modified orders

example

Shift A
right X times
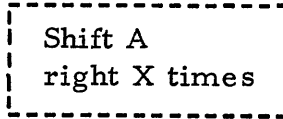
When X is modi-
fied by some
other instructions
in the program

3. (j)

connector, address modifier or special condition

example ———→ (j)
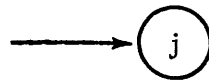
indicates the next
step in the program
will be at point j of
the flow diagram

4.
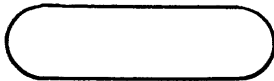
decision junction

example (Does A = zero ?) ———→ Yes

No
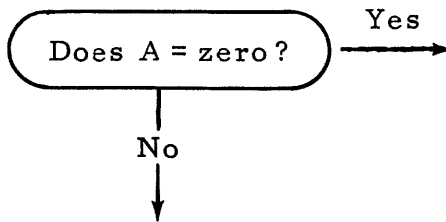
5.

subroutine

example ⟨ square root ⟩
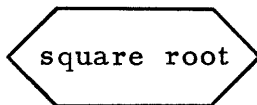
indicates a square
root subroutine

6. ◇
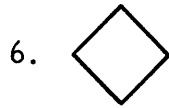
initial or terminal conditions

example

◇ start ◇

7. ⬭

input/output

example

paper tape
input

8. ⟶
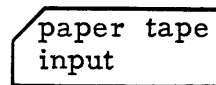
"go to"   "is sent to"

example

◇ start ◇ ⟶ Add A and B
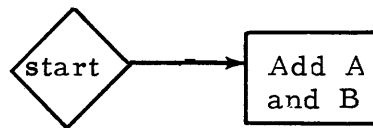
1.2.4   LOOPS

A program loop is a section of a program that is used more than once in succession during the solution of a problem. For example, assume during the course of a program it was necessary to add a series of numbers together and store the result of this addition so that it may be used by the remainder of the program. The flow diagram of this loop would look like the following:

```
                          ┌─────────────┐   ┌──────────────┐   ┌──────────┐
from first                │ load the first│  │ add the second│  │ store the│
part of program ────────▶ │ number into A │─▶│ number and the│─▶│ result   │──┐
                          └─────────────┘   │ contents of A │  └──────────┘  │
                                       ┌───▶└──────────────┘                  │
        ┌─────────────────────────────┘                                      │
        │                                                                    │
        │  ┌──────────────┐   ┌──────────────┐   ╭─────────────────╮         │
        │  │ load the result of│ advance the  │   │  are all the    │         │
        └─▶│ the previous  │◀──│ address of   │◀─ no ─│ numbers added    │◀────────┘
           │ addition      │   │ the second   │   │  together ?     │
           └──────────────┘   │ number       │   ╰─────────────────╯
                              └──────────────┘          │
                                                       yes
                                                        │
                                                        ▼
                                                go on to the
                                                remainder of
                                                the program
```
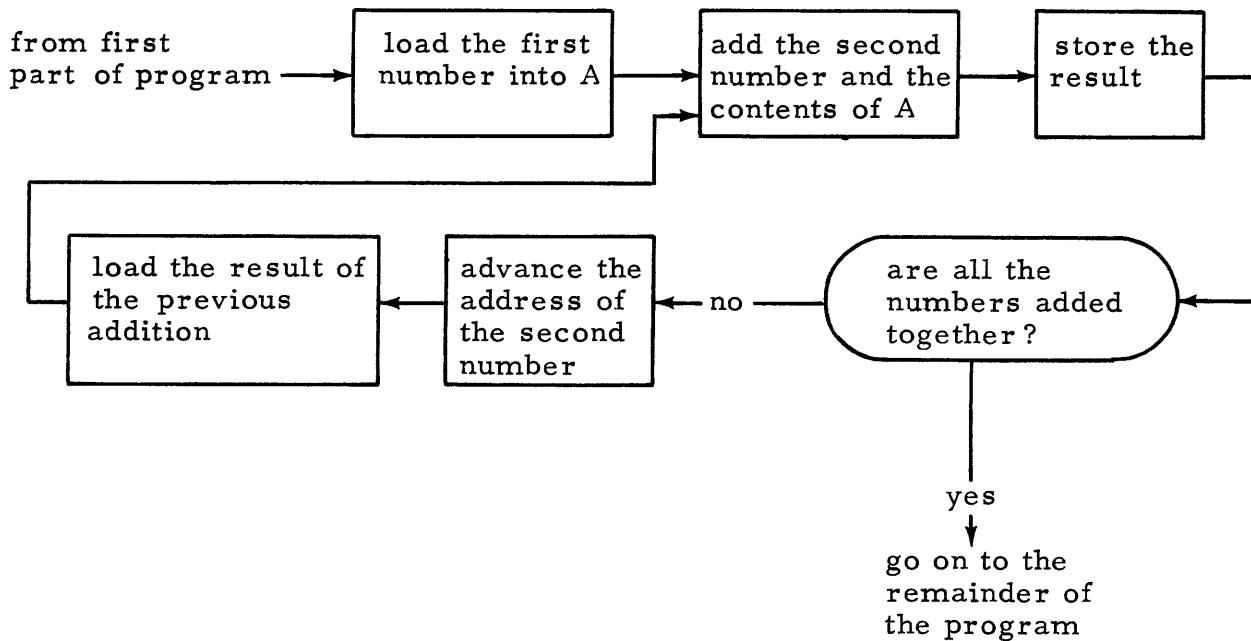
As you have observed, there is always a condition that must be satisfied to end the loop. In this program, it is the condition that all the numbers are added together. This is the primary identifying feature of a loop.

1.2.5 SUBROUTINES

There are many problems that require the same "sub-solution" to be found as a part of the total solution. For example, the sine of an angle. Once these routines are written they may be used by several programs without making it necessary to re-write the same program. These sub-programs are called "subroutines" and are usually kept on a master input tape so that any programmer who has a need for the subroutine may use it. There is also the possibility that the routine will be used more than once by the same program. In both cases, the subroutine is stored in a known place in memory and is referenced by the main program whenever the sub-solution is required.

A subroutine differs from a loop in two ways. First, a subroutine is not used more than once in succession and second, a subroutine is a complete program in itself while a loop is not.

A subroutine is written so that all that is necessary for its operation is the data or information the subroutine is to work with and an address in the main program to go to when the subroutine is completed.

1.3   PROGRAMMING THE ASI-210

1.3.1   DESCRIPTION OF THE ASI-210

1.3.1.1   General

The ASI-210 is a general purpose computer that may be used for engineering and scientific computation, real time system control, data handling and analysis, management support, and satellite operation with other machines. It is a stored program, parallel operation, solid-state machine with a 21-bit word length. The buffered input/output channel, with a transfer rate of 30,000 or more 21-bit words per second, expandable to 2 channels, permits simultaneous computation and data transfer. Data transfer requires no running time (initiation only).

A single bit in the instruction word of the ASI-210 specifies the operand address of all instructions as either a direct or indirect address. The three index words permit successive indirect addressing by indexing at each step.

The megacycle operation enables add times of 10 microseconds, multiply times of 54 microseconds, including memory access time, indexing and input/output channel memory reference. The entire computer requires less than 1350 watts from a standard 110/220 volt, 60 cycle source. No temperature or humidity controls are required in normal operating environments. A paper tape reader and punch are included with the standard computer. Paper tape is read at 600 characters per second and punched at 110 characters per second. There are six sense switches that are available to the programmer for branching or other operations.

1.3.1.2   Principle Registers

Register "A"   Accumulator. The results of all arithmetic or logical operations will be placed in Register "A". (21-bits)

Register "E"   Auxiliary or extension of Register "A". Used when the result of an arithmetic or logical operation is larger than can be contained in Register "A" alone. (21-bits)

Register "S"   Sequence address register. Contains
the address of the next instruction
to be performed. (13-bits)

Register "BM"   Starting address register. Contains
the starting address of memory that
is to be referenced during an input/
output operation. (13-bits)

Register "BL"   Limit address register Contains
the ending address plus 1 of memory
that is to be referenced during an
input/output operation. (13-bits)

1.3.1.3   Memory

The memory unit in the ASI-210 is a magnetic core
array. The 4,096 word memory core stack is made
up of 16 core planes and each plane has a 42 by 128
array. The 8,192 word memory has 32 core planes,
each plane also with a 42 by 128 core matrix. Memory
is organized in a word fashion, rather than a coinci-
dent current fashion. That is, full current flows
through a single word during a read operation with
the usual sense lines available on each bit.

The write system has a partial write current
through one word of memory. An additive digit
current, coincident with the write current, is
used for writing "1's" in the selected bit position
of the word.

The read current is a full current through one word.
The write current is a partial current through one
word plus an additive digit current to write a "1".
Memory is coincident current in this respect; the
organization, however, is basically "word organiza-
tion".

1.3.1.4   Representation of Numerical Values

The only number system the ASI-210 can recognize
is the binary system, therefore all information
within the computer must be in binary format. An
instruction word in the computer is composed of
21 binary bits, ones or zeros. These 21 ones and
zeros are hard to read and difficult to manipulate
by people who are used to a decimal system. To
make the task of those people who are concerned
with any operation of the computer easier, the
octal number system is used outside the machine.

The octal number system uses eight distinct characters, zero through seven. This system was chosen because of the relationship between octal and binary number systems. The relationship is that three binary digits may be represented by one octal digit. For example:

Binary	Octal
000 000	00
000 001	01
000 010	02
000 011	03
000 100	04
000 101	05
000 110	06
000 111	07
001 000	10
001 001	11
001 010	12

The relationship of binary to octal is quite easily seen in the above example and the advantage of using octal over binary should also be easily seen. Therefore, all information and instructions, when used outside the computer, will be in the octal number system.

1.3.1.5   Arithmetic Operations

The adder used in the ASI-210 is a closed loop binary adder using left end-around carry operation in the one's complement addition. The sign bit (21) is a zero (0) for positive numbers and a one (1) for negative numbers. This is useful in most operations using the entire register A. However, in certain instances such as indexing, only a small portion of the adder is used and provisions to prevent end-around carry are made.

Operation is then carried out in the two's complement addition which gives the correct answer without carry. A comparison of the two systems is shown:

Decimal	Binary	1's Comp.	2's Comp.
3	011	+3	+3
2	010	+2	+2
1	001	+1	+1
0	000	+0	+0
7	111	-0*	-1
6	110	-1	-2
5	101	-2	-3
4	100	-3	-4

*Minus zero is meaningless. All carries are forcibly entered in adder so that the number becomes 000 or $^+0$.

It should be noted that there are several operations that will give a result of negative zero. These operations are:

1. Minus zero + minus zero
   $(-0) + (-0) = (-0)$

2. Plus zero - plus zero
   $(+0) - (+0) = (-0)$

3. Minus zero - minus zero
   $(-0) - (-0) = (-0)$

4. Minus zero - plus zero
   $(-0) - (+0) = (-0)$

5. Subtracting a number from itself
   $(+k) - (+k) = (-0)$ and
   $(-k) - (-k) = (-0)$

6. Plus zero X minus zero
   $(+0) \cdot (-0) = (-0)$

7. Any positive number X minus zero
$(+k) \cdot (-0) = (-0)$

8. Any negative number X plus zero
$(-k) \cdot (+0) = (-0)$

9. Plus zero ÷ any negative number
$(+0) \div (-k) = (-0)$

10. Minus zero ÷ any positive number
$(-0) \div (+k) = (-0)$

11. Any negative number ÷ any number
when the remainder is zero
$(-k1) \div (+k2)$ where R is 0 = R of (-0)

or $(-k1) \div (-k2)$ when R is 0 = R of (-0)

Examples of the four basic arithmetic operations follow:

1. Addition

   a. Two positive numbers

Binary			Octal	
1 11	carries		1	carries
000 101			05	
+010 011			+23	
011 000			30	

   b. Two negative numbers (one's complement notation)

Binary			Octal	
111	carries		11	carries
111 010	(-000 101)		53	(-24)
+101 100	(-010 011)		+67	(-10)
100 110	(-011 001)		42	(-35)
→1	(end around carry)		→1	(end around carry)
100 111	(-011 000)		43	(-34)

c. One positive and one negative number

<table>
<tr><td colspan="2">Binary</td><td colspan="2">Octal</td></tr>
<tr><td>-1111 1</td><td>carries</td><td></td><td></td></tr>
<tr><td>110 010</td><td>(-001 101)</td><td>45</td><td>(-32)</td></tr>
<tr><td>+001 111</td><td></td><td>+27</td><td></td></tr>
<tr><td>1</td><td>carry</td><td>74</td><td>(-03)</td></tr>
<tr><td>000 001</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td>(end around carry)</td><td></td><td></td></tr>
<tr><td>000 010</td><td></td><td></td><td></td></tr>
</table>

2. Subtraction

The subtraction process uses the same opera-
tion as the addition except that the subtrahend
is first complemented (changed to the corres-
ponding negative value). For example, to
subtract 23_8 from 37_8 the 23 would first be
changed to 54 (-23 in seven's complement
notation) and then the two numbers will be
added.

3. Multiplication

    a. Two positive numbers :

<table>
<tr><td colspan="2">Binary</td><td>Octal</td></tr>
<tr><td></td><td>001 000</td><td>10</td></tr>
<tr><td>x</td><td>010</td><td>x 2</td></tr>
<tr><td></td><td>000 000</td><td>20</td></tr>
<tr><td>0</td><td>010 00</td><td></td></tr>
<tr><td>0 0</td><td>000 0</td><td></td></tr>
<tr><td>0 0</td><td>010 000</td><td></td></tr>
</table>

    b. Two negative numbers (one's complement
        notation):

        In one's complement notation the significant
        bit is the "zero" bit and not the "one" bit.
        While it is possible to design circuitry
        that will recognize the zero as the signifi-
        cant bit, it is more convenient to check for
        a negative number, and if one is found, to
        change the number to a positive number,
        (by complementation) do the multiplication

with positive numbers and affix the proper sign to the result when the multiplication is complete. This is the procedure that is used by the ASI-210 and therefore, examples of multiplying with negative numbers will not be given.

4. Division

    a. Two positive numbers:

Binary		Octal
0 001 000		10

```
            ┌─────────────────      ┌────
000 011     │ 000 000 011 000    3  │ 30
            │      000 011          │ 3
            └──────000 000 000      └──00
```

    b. Two negative numbers (one's complement notation):

In one's complement notation the significant bit is the "zero" bit and not the "one" bit. While it is possible to design circuitry, that will recognize the zero as the significant bit, it is more convenient to check for a negative number, and if one is found, change the number to a positive number, (by complementation), do the division with positive numbers and affix the proper sign to the result when the division is complete. This is the procedure used by the ASI-210 and therefore, examples of dividing with negative numbers will not be given.

1.3.2 INSTRUCTION WORD

1.3.2.1 Function Code Designator

The function code designator is that portion of the instruction word that tells the computer what operation is to be performed. The five highest order bits of the instruction word comprise the function code and gives the possibility of 32 separate instructions. Each instruction is interpreted and performed separately by the computer.

1.3.2.2 Indirect Address Designator

The indirect address designator specifies whether the operand address is to be used as the address

of the operand (direct) or as the address of the
address of the operand (indirect). The indirect
address designator is bit 16 of the instruction
word and indicates an indirect address if it is
set (1).

1.3.2.3 Index Address

The index address portion of the instruction word
tells the computer whether the operand address of
the instruction is to be modified or not and if the
operand address is to be modified, where the
modifying word will be found. The index address
uses bits 14 and 15 of the instruction word and if
they are both cleared (0's) no modification of the
operand address will take place. If either or both
of the index address bits are set (1's), they indicate
a modification of the operand address is requested
and they also specify the address of the modifying
word.

1.3.2.4 Operand Address

The operand address of the instruction is the lowest
order 13 bits of the instruction word and contains
the address of the operand that is to be used for
this instruction. The operand address may be
modified as described in paragraphs 1.3.2.2 and
1.3.2.3 of this section.

1.3.3 REPERTOIRE OF INSTRUCTIONS

1.3.3.1 Instruction List

Octal Code	Instruction	Time u sec
00	HALT	8
02	JUMP	8
04	RETURN	12
06	JUMP END INTERRUPT	8
10	ADD	10
12	SUBTRACT	10
14	LOAD A	10
16	LOAD E	12
20	ABSOLUTE VALUE	8
22	MINUS	8
24	ZERO	12
26	STORE A	8
30	MULTIPLY	54
32	DIVIDE	56
34	ROUND	14
36	STORE A ADDRESS	10
40	SKIP A HIGH	14
42	SKIP A EQUAL	14
44	JUMP A LESS THAN ZERO	10
46	STORE E	10
50	AUGMENT INDEX	12
52	SKIP INDEX HIGH	10
54	STORE ADDRESS IN INDEX	12
56	LOGICAL OR	12
60	SHIFT	10+2K
62	NORMALIZE A	14+2K
64	NORMALIZE A E	14^+2K
66	LOGICAL AND	12
70	TRAP	8
72	SKIP SENSE SWITCH SET	10
74	EXTERNAL DEVICE	16
76	ASSEMBLY REGISTER	20

1.3.3.2 Symbols and Terms

The following symbols will be used throughout the remainder of this manual:

Symbols	Terms
A	Register A, Accumulator
E	Register E, a second major arithmetic register
S	Register S, contains the address of the next instruction to be performed.
( )	Contents of. For example, (A) signifies the contents of the A register.
$+$	Add
$-$	Subtract
$\cdot$	Multiply
$\div$	Divide
$\longrightarrow$	"is placed in"
$/\ /$	Absolute value. For example, $/(A)/$ signifies the absolute value of the contents of the A register.
$\overline{}$	Complement of. For example, $\overline{(A)}$ signifies the complement of the contents of the A register.
$\oplus$	Logical OR. For example, $\overline{(E)} \oplus (m)$ signifies the logical OR of the complement of the contents of the E register and the operand.
$\odot$	Logical AND. For example, $(E) \odot (m)$ signifies the logical AND of the contents of the E register and the operand.
a	Register A designator in the operand address.
e	Register E designator in the operand address.

Symbols	Terms
s	Shift right designator.
c	Shift circular designator.
g	Gray code to binary shift indicator.
k	Shift count.
I_b	Bits 13 through 1 of the specified index location.
M	The effective operand address of the instruction.
m	The memory location whose address is M.
i	Address of present instruction.
(m)	Operand

Unless otherwise indicated the operand address is subject to indexing and indirect address.

1.3.3.3 Instruction Operation

Code	Instruction	Operation

Code   **Instruction**   **Operation**

00   HALT

HALT
$(m) \longrightarrow S$

Halt and take the next instruction from the operand address.

02   JUMP

$(m) \longrightarrow S$

Take the next instruction from the operand address.

04   RETURN

$(S) + 1 \longrightarrow m$ 13 thru 1

Store the address of the instruction following the next instruction in bits 13 thru 1 of the operand. Bits 21 thru 14 of the operand will be unchanged. By letting the operand be a jump instruction at the end of a subroutine, the program can jump into the subroutine on the next instruction and return to the program at the end of the subroutine.

06   JUMP END INTERRUPT

END INTERRUPT
$M \longrightarrow S$

If the condition which specifies that a priority interrupt routine is in progress is not present, clear the regular interrupt routine condition. In any case clear the priority interrupt routine condition. Take the next instruction from the operand address.

10   ADD

$(A) + (m) \longrightarrow A$

Add the operand to the number that is in register A at the start of this instruction. The resulting sum will be in register A at the end of this instruction. Register E will be unchanged. This instruction will result in an add overflow if the sign of the sum is different from the signs of both commands.

Code	Instruction	Operation

12　　SUBTRACT　　　　　$(A) - (m) \longrightarrow A$

Subtract the operand from the number
that is in register A at the start of this
instruction. The resulting difference
will be in register A at the end of this
instruction. Register E will be unchanged.
This instruction will result in an add
overflow if the sign of the number which
is in A at the start of this instruction,
is different from both the sign of the
operand and the sign of the resulting
difference.

14　　LOAD A　　　　　$(m) \longrightarrow A$

Bring the operand to register A.

16　　LOAD E　　　　　$(m) \longrightarrow E$

Bring the operand to register E.

20　　ABSOLUTE VALUE　　If bit 12· A21:
$$\overline{(A)} \longrightarrow A$$

If bit 11· E21:
$$\overline{(E)} \longrightarrow E$$

*If bit 2: set the flags
specified by bits 9, 8, 7

*If bit 1: clear the flags
specified by bits 9, 8, 7

If bit 12 of this instruction is a one and
if the number that is in register A at
the start of this instruction is negative,
register A will be complemented (made
positive). If bit 11 of this instruction
is a one and if the number that is in
register E at the start of this instruc-
tion is negative, register E will be com-
plemented. If bit 2 of this instruction
is a one, any flags designated by ones
in bits 9, 8, or 7 of this instruction will

* Not in present machines but will be added
in near future.

Code	Instruction	Operation

be set. If bit 1 of this instruction is
a one, any flags designated by bits
9, 8, or 7 of this instruction will be
cleared. Any combination of ones in
bits 12, 11, 9, 8, 7, 2, and 1 may be
used in this instruction with the single
exception that bits 2 and 1 should not
both be ones in the same instruction.

22 NEGATE

If bit 12:
$(\overline{A}) \longrightarrow A$

If bit 11:
$(\overline{E}) \longrightarrow E$

*If bit 2: set the flags
specified by bits 9, 8, 7

*If bit 1: Clear the flags
specified by bits 9, 8, 7

If bits 12 of this instruction is a one,
register A will be complemented
(negated). If bit 11 of this instruc-
tion is a one, register E will be com-
plemented. If bit 2 of this instruction
is a one, any flags designated by ones
in bits 9, 8, or 7 of this instruction
will be set. If bit 1 of this instruction
is a one, any flags designated by ones
in bits 9, 8, or 7 of this instruction will
be cleared. Any combination of ones
in bits 12, 11, 9, 8, 7, 2, and 1 may
be used in this instruction with the
single exception that bits 2 and 1 should
not both be ones in the same instruction.

*Not in present machines but will be
added in the near future.

24 CLEAR

If bit 12:
$0 \longrightarrow A$

If bit 11:
$0 \longrightarrow E$

*If bit 9: $0 \longrightarrow A$
(Same sign as E)

Code	Instruction	Operation

*If bit 8:   0 ⟶ E
(Same sign as A)

*If bit 7: (A) ⟶ E,
(E) ⟶ A

If bit 12 of this instruction is a one,
clear register A. If bit 11 of this
instruction is a one, clear register E.
If bit 9 of this instruction is a one,
register A will be set to either plus
zero or minus zero so that the sign
of register A will be the same as the
sign of register E. If bit 8 of this
instruction is a one, register E will
be set to either plus zero or minus
zero so that the sign of register E
will be the same as the sign of register
A. If bit 7 of this instruction is a one,
the number that is in register A at the
start of this instruction will be in
register E at the end of this instruction,
and the number which was in register E
at the start of this instruction will be in
register A at the end of this instruction.
Bits 12 and 11 may both be ones in the
same clear instruction. If bit 9, 8, or
7 is a one in this instruction, no other
address bit should be a one in the same
instruction.

*Not in present machines but will be
added in the near future.

26	STORE A	(A) ⟶ m

Store the number that is in register A
in the operand address.

30	MULTIPLY	(A) · (m) ⟶ AE

Multiply the number that is in A at the
start of this instruction, by the operand.
At the end of this instruction the most
significant bits of the product will be in
register A, and the least significant bits
of the product will be in register E.

Code	Instruction	Operation
32	DIVIDE	If $\overline{\text{Fault}}$: $(AE) \div (m) \longrightarrow E$ If $\overline{\text{Fault}}$: REMAINDER $\longrightarrow$ A

Divide the double-length number that is in registers A and E at the start of this instruction, by the operand. At the end of this instruction, unless a fault occurs, the quotient will be in register E and the remainder will be in register A. This instruction will result in a fault if the absolute value of the number that is in register A at the start of the instruction, is greater than or equal to the absolute value of the operand. If a fault occurs, registers A and E will be unchanged by this instruction. A fault will result in a fault interrupt if the fault trap is armed and if the program is not already in an interrupt routine.

Code	Instruction	Operation
34	ROUND	If $E21 \cdot \overline{E20}$: $(A)-1 \longrightarrow A$ If $\overline{E21} \cdot E20$: $(A)+1 \longrightarrow A$

If the number that is in register E is negative and if bit E20 is a zero, subtract one from the number that is in register A at the start of this instruction, and leave the result in register A. If the number that is in register E is positive and if bit E20 is a one, add one to the number that is in register A at the start of this instruction, and leave the result in register A. Register E will be unchanged by this instruction. This instruction will result in an add overflow if the sign of register A changes as a result of the addition or subtraction.

Code	Instruction	Operation
36	STORE ADDRESS	(A13 thru A1) $\longrightarrow$ m13 thru m1

Store the information in bits 13 thru 1 of register A in the corresponding bits of the operand address. Register A will be unchanged by this instruction. Bits 21 thru 14 of the operand will be unchanged by this instruction.

40	SKIP A HIGH	If $(A) > (m)$: Skip 1 instruction

If the number which is in register A is greater than the operand, skip the next instruction in sequence. Register A will be unchanged by this instruction.

42	SKIP A EQUAL	If $(A) = (m)$: Skip 1 instruction

If the number which is in register A is equal to the operand, skip the next instruction in sequence. Register A will be unchanged by this instruction.

44	JUMP A LESS THAN ZERO	If $(A) < 0$: M $\longrightarrow$ S

If the number which is in register A is less than zero take the next instruction from the operand address. Register A will be unchanged by this instruction.

46	STORE E	(E) $\longrightarrow$ m

Store the number that is in register E in the operand address.

50	AUGMENT INDEX	$M + (I_b) \longrightarrow I_b$

Add the operand address to the address in bits 13 thru 1 of the index location specified by bits 15 and 14 of this instruction. The sum will be in bits 13

Code	Instruction	Operation

thru 1 of that index location at the end
of this instruction. Bits 21 thru 14 of
that index location will be unchanged
even if an index overflow occurs. Index
overflow will occur if the sum of the
operand address and the address in
bits 13 thru 1 of the specified index
location exceeds 13 bits. The operand
address will not be indexed in this
instruction. If an indirect address is
specified, this address will be indexed
by the index location specified by bits
15 and 14 of the instruction. The index
location which will be augmented will
be the one which is specified by bits
15 and 14 of the word in the final in-
direct address location. Bits 13 thru
1 of the word in this location will not
be indexed and will be used as the
effective operand address of the in-
struction which will be used to augment
the index location.

Code	Instruction	Operation

52   SKIP INDEX HIGH   if $(I_i) > \overline{m}$:
                                                  Skip 1 instruction

If the address which is contained in
bits 13 through 1 of the index location
specified by bits 15 and 14 of this in-
struction exceeds the complement of
the operand address of this instruction
the next instruction in sequence will
be skipped. The operand address will
not be indexed in this instruction. If
an indirect address is specified, this
address will be indexed by the index
location specified by bits 15 and 14
of the instruction. The index location
which will be tested will be the one
which is specified by bits 15 and 14
of the word in the final indirect address
location. Bits 13 through 1 of the word
in this location will not be indexed and
will be used as the effective operand
address of the instruction which will
be used to test the index location.

54   STORE ADDRESS $M \longrightarrow I_b$
         IN INDEX

Store the operand address in bits 13
through 1 of the index location specified
by bits 15 and 14 of the instruction.
The operand address will not be indexed
but it may be indirectly addressed.
The indirect address will not be in-
dexed even if an index address is spec-
ified. Bits 13 through 1 of the word
in the final indirect address location
will be the effective operand address
which will be loaded in the index loca-
tion specified by bits 15 and 14 of the
word in the final indirect address loca-
tion.

Code	Instruction	Operation
56	LOGICAL "OR"	(E) "OR" (m)———►A

Form the logical "Or" of the number that is in register E and the operand and load the result in register A. An example of the bit-for-bit result is as follows:

(E)	1100
(m)	1010
"OR"	1110

The number that is in register E will be unchanged.

60   SHIFT

Instruction Bit	12	11	10	9	8
Shift (A) right	1	0	1	0	0
Shift (A) left	1	0	0	0	0
Shift (A) left circular	1	0	0	1	0
Shift (E) right	0	1	1	0	0
Shift (E) left	0	1	0	0	0
Shift (E) left circular	0	1	0	1	0
Shift (AE) right	1	1	1	0	0
Shift (AE) left	1	1	0	0	0
Shift (AE) left circular	1	1	0	1	0
Convert (E) from gray code to binary leaving the result in register A.	0	0	0	0	1

Right shifts are open-ended. Left shifts may be either open-ended or circular. In open-ended shifts, the bits introduced into the register are identical to the sign bit which remains unchanged. In circular shifts the sign bit is shifted along with the number. The number of shifts is specified by bits 6 through 1

Code	Instruction	Operation

of the instruction in binary code.
The maximum number of shifts
that may be specified is 63 decimal.
For gray to binary conversion the
gray code number should be in register
E at the start of this instruction with
the most significant bit in E20. The
number of bits to be converted should
be specified by bits 6 through 1 of the
instruction. The binary result will
be in the least significant bits of regis-
ter A with zeros in the unused part of
register A.

62   NORMALIZE A $(A) \cdot 2^k \longrightarrow A$
until $A20 \neq A21$
$K + (m) \longrightarrow m$

Shift the number that is in register A
at the start of this instruction left leav-
ing the sign bit (21) unchanged until
A21 is not the same as A20. With
each shift the sign bit (A21) will be
entered in the least significant bit
(A1) of register A. The number of
shifts required will be added to the
operand. If the number in register
A at the start of this instruction is
plus or minus zero 19, shifts will
occur and register A will be unchanged
at the end of the instruction.

64   NORMALIZE AE $(AE) \cdot 2^k \longrightarrow A$
until $A20 \neq A21$
$K + (m) \longrightarrow m$

Shift the double-length number that is
in registers A and E at the start of
this instruction left leaving both sign
bits unchanged until A21 is not the same
as A20. With each shift the sign bit of
the E register (E21) will be entered in

Code	Instruction	Operation

the least significant bit of register
E (E1) and the most significant bit
of register E (E20) will be entered
in the least significant bits of regis-
ter A(A1). The number of shifts re-
quired will be added to the operand.
If the number that is in register A and
E at the start of this instruction is
plus or minus zero, 39 shifts will occur
and registers A and E will be unchanged
at the end of the instruction.

66   LOGICAL "AND"   (E) "AND"

$$(m) \longrightarrow A$$

Form the logical "And" of the number
that is in register E and the operand
and load the result in register A. An
example of the bit-for-bit result is as
follows:

(E)	1100
(m)	1010
"AND"	1000

The number that is in register E will
be unchanged.

Code	Instruction	Operation
70	TRAP	If bit 13 is a 1:

(Add overflow condition) ⟶ A5
(Index overflow condition) ⟶ A7
Clear add and index overflow conditions

If bit 12 is a 1:

(External device trap condition) ⟶ A1
(Busy trap condition) ⟶ A2
(Operator trap condition) ⟶ A3
(Fault trap condition) ⟶ A4
(Add overflow trap condition) ⟶ A5
(Index overflow trap condition) ⟶ A7
(Indicator light #1 condition) ⟶ A8
(Indicator light #2 condition) ⟶ A9

If bit 11 is a 1:

Arm the traps and indicator lights
which are specified by bits 9 thru 1

If bit 10 is a 1:

Disarm the traps and indicator lights
which are specified by bits 9 thru 1

If bit 9 is a 1:

Indicator light #2 is specified

If bit 8 is a 1:

Indicator light #1 is specified

If bit 7 is a 1:

Index overflow trap is specified

If bit 5 is a 1:

Add overflow trap is specified

If bit 4 is a 1:

Fault trap is specified

If bit 3 is a 1:

Operator trap is specified

Code	Instruction	Operation
70 (Continued)		If bit 2 is a 1:

Busy trap is specified

If bit 1 is a 1:

External device trap is specified

If bit 13 of this instruction is a one the add and index overflow conditions will be stored in bits 5 and 7 of register A respectively. All other bits of register A will remain unchanged if bit 13 is a one. If bit 12 is a one all trap and operator light conditions will be stored in bits 9, 8, 7, 5, 4, 3, 2 and 1 of register A as specified above regardless of bits 9 thru 1. All other bits of register A will remain unchanged if bit 12 is a one. Bits 13 and 12 should not both be ones in the same instruction. If bit 11 is a one those traps and indicator lights which are specified by ones in bits 9 thru 1 as indicated above will be armed. If bit 10 is a one those traps and indicator lights which are specified by bits 9 thru 1 as indicated above will be disarmed. Bits 11 and 10 should not both be ones in the same instruction. With the exceptions specified above all combinations of ones and zeros in bits 13 thru 1 are allowable.

| 72 | SKIP SENSE SWITCH SET | *if bit 13 is a one and A21 $\neq$ E21
*Or if bit 12 and E21 are both ones
*Or if bit 11 and indicator light #2 are both ones
*Or if bit 10 and indicator light #1 are both ones
*Or if bit 9 and flag 9 are both ones
*Or if bit 8 and flag 8 are both ones
*Or if bit 7 and flag 7 are both ones |

*Not in present machines but will be added in near future.

Code	Instruction	Operation

72
(Continued)

Or if bit 6 and sense
switch 6 are both ones
Or if bit 5 and sense
switch 5 are both ones
Or if bit 4 and sense
switch 4 are both ones
Or if bit 3 and sense
switch 3 are both ones
Or if bit 2 and sense
switch 2 are both ones
Or if bit 1 and sense
switch 1 are both ones:

Skip 1 instruction

This instruction will result in skipping the
next instruction in sequence if the above
conditions are satisfied. Any combination
of ones and zeroes in 13 thru 1 is allowable
in this instruction.

74   **EXTERNAL**
DEVICE

Interpret the operand as an external device
control word (EDCW). Bits 21 thru 16 of the
EDCW specify the address of an external
device. If the device which is addressed is
busy and if the busy trap is armed a busy
interrupt will result from this instruction.
Bits 13 thru 1 of the EDCW specify what the
addressed external device will be instructed
to do if it is not busy. These bits have a
different significance for each external
device. If the addressed external device is
not busy a start signal will be sent. This
start signal will be recognized only by the
addressed external device and will cause it
to commence executing the instruction speci-
fied by the EDCW. The external device's
own logic will sequence events associated
with this instruction until it is completed.
If bit 14 of the EDCW is a one the addressed
external device will be instructed to interrupt
the central computer program when it has com-
pleted what it is instructed to do. For further
details see Input Output Systems, Volume 3,
Section I, paragraph 1.3.4.

Code	Instruction	Operation
76	ASSEMBLY REGISTER	

Interpret the operand as an assembly
register control word (ARCW). If bit
19 of the ARCW is a zero this instruction
will be directed to channel zero; if it is a
one this instruction will be directed to
channel one. If the specified channel is
busy this instruction will have no effect
except that a busy interrupt will result
if the busy trap is armed. Bits 13 thru
1 of the ARCW represent an ARCW address
which will be indexed if bits 15 and 14 of
the ARCW are not both zero. Indirect
addressing does not apply to the ARCW
address. If bit 17 of the ARCW is a one,
the memory address register (BM) of the
specified channel will be stored in bits 13
thru 1 of the memory location specified by
the ARCW address. If bit 17 is a 0 and bit
18 is a 1, the ARCW address will be trans-
ferred to the memory address register (BM)
of the specified channel. If both bits 17 and 18
are 0 the ARCW address will be transferred
to the limit address register (BL) of the
specified channel. The memory address
register of each channel is the address of
the next memory location with which that
channel may communicate. After an
external device has completed its communi-
cation with the central memory register BM
will contain the address sequentially following
the last memory address with which the
channel communicated. The limit address
register (BL) contains the address following
the last sequential address with which the
channel is allowed to communicate. If
registers BM and BL of the same channel
contain the same address, that channel can
communicate no information.

1.3.4 INPUT/OUTPUT SYSTEMS

1.3.4.1 General Features

All transfer of data to or from the computer is conducted via input/output channels which communicate directly with the magnetic core memory of the ASI-210. The access to the memory is time-shared between the operating program and input/output data transfer; in a typical situation, approximately 15 per cent of the memory time is available for input/ output data transfer. Since the arithmetic and control functions of the operating program do not require access to the memory every computer cycle, they may proceed simultaneously with input/output data transfer with little or no loss in speed.

The standard ASI-210 is provided with one input/output channel. An additional channel may be optionally supplied.

Each piece of on-line peripheral equipment is known as an "External Device" (abbreviated, E. D.). Each external device has an unique address. The ASI-210 can accomodate up to 64 external devices with two channel operation.

The ASI-210 input/output system is provided with program interrupt features so that testing of the condition of the external devices by the running program is not necessary.

1.3.4.2 Input/Output/ Channel

The standard mode of input/output data flow for the ASI-210 is by sequentially transmitted six-bit octal or alphanumeric characters. These characters are assembled (or disassembled) into 21-bit computer words by an input/ output assembly register (B), one of which is employed for each input/output channel. In addition, each input/output channel is provided with two 13-bit address registers. These registers are termed the "Address register" (BM) and the "Limit address register" (BL). (Limit address is the last data address plus one.)

In general, BM and BL define the beginning
and limit locations in the main core memory
into (or from) which a block of data is to be
transferred via the particular input/output
channel. The specific function of BM and BL
depends upon the ED addressed. During the
actual input/output data transfer the various
registers are under the control of the ED.

1.3.4.3 Assembly Register Instruction

The AR instruction has the same format as
the other machine instructions. In this case,
the operand is the "Assembly Register Control
Word" (ARCW). The format of the ARCW is
shown below:

bits 21-20   Store BM   Index
Unused   ┌──────────▶bit 17   Address
      ▲   │   ┌──────────────────────▶bits 15-14
    ⌢⌢   │   │
  B B B B B   B B B B B B B B B B B B B B B B B
    │   └─┐   └─────▶Unused   └──▶AR Operand
Channel   │   bit 16   bits 13-1
bit 19   │
                ▼
              Register
              BM or BL
              bit 18
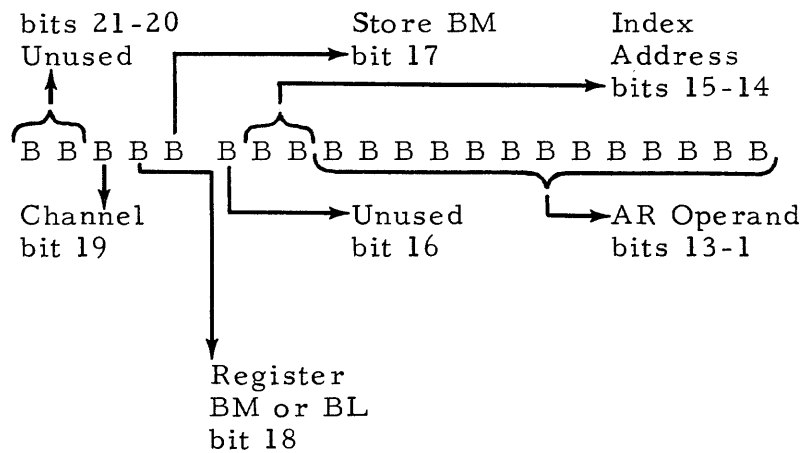
B represents 1 binary digit

Channel (Bit 19)

          If this bit is a one channel #1 is
          specified.

          If this bit is a zero channel #2 is
          specified.

Register BM or BL (Bit 18)

          If this bit is a one, the Effective
          AR Operand will be placed in

BM of the appropriate channel.

If this bit is a zero, the Effective
AR Operand will be placed in BL of
the appropriate channel.

Store BM (Bit 17)

If this bit is a one, bit 18 will be
ignored and the contents of BM of
the appropriate channel will be
stored in the operand address portion
of the memory location specified by
the Effective AR Operand.

If this bit is a zero, this provision
will be ignored.

Index Address

This address specifies the same
index locations employed in the
machine instructions. It is the
address of a memory location,
the operand address portion of
which may be added to the AR
Operand to yield the Effective AR
Operand.

AR Operand

Employed as described above.

1.3.4.4   External Device Instruction

The ED instruction has the same format as
all other machine instructions.

In this case the operand is the "External
Device Control Word" (EDCW).

Format of EDCW:

```
 B B B B B B B B B B B B B B B B B B B B B B
```

ED address
bits 21-16

Number
(used by move
commands)
bits 13-5

Command Code
bits 4-1

Unused
bit 15
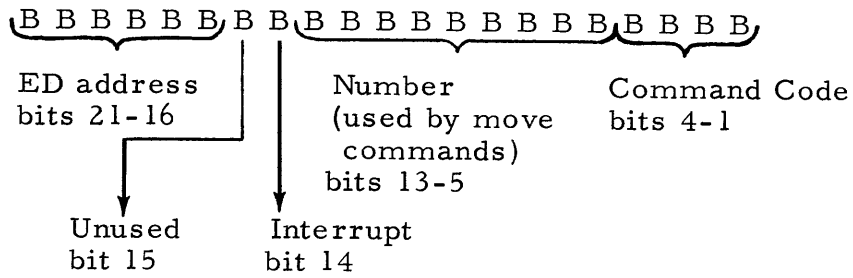
Interrupt
bit 14

B represents one binary digit

ED Address	Address of the particular ED
Interrupt	The "Interrupt" bit 14 commands the ED to interrupt the running program when it has performed the specified operation.
Command Code	These bits specify to the ED what operation is to be performed.

1.3.4.5   Assigned ED Addresses

Device	Address
Typewriter	00
Paper Tape Reader	02
Paper Tape Punch	04
Card Reader	06
Card Punch	10
Line Printer	12
Magnetic Tape 1	14
Magnetic Tape 2	16
Magnetic Tape 3	20

Magnetic Tape 4   22

(Additional Devices will be ass-
igned to locations 00024-00077)

1.3.4.6 Command Codes for ASI External Devices

1.3.4.6.1 Command Codes for the Paper Tape Reader

Command Code	Command
00	Read Packed -
	Input to mem-ory in a packed mode proceeds until (BL) = (BM) or until a stop code is read. (BL) - (BM) words are input with first tape chara-cter on line going to most significant end of memory loc-ation (BM). A six level infor-mation code is used. Lateral odd parity is checked. Any code containing an eight level punch is not read. Unit passes leader and code delete.
01	Read Character -
	Input of all eight tape levels, each character or line going into a single computer word. In memory,

bits 18
through 21 will
be zeros.

Bits 17 through
10 will be filled
from tape levels
8 through 1, res-
pectively.

All other bits
will be zeros.
No parity is check-
ed and input pro-
ceeds until (BL)
(BM). All codes
enter computer,
leader is not pass-
ed.

All even codes same as 00

All odd codes same as 01

If bits 14 of the EDCW was a "one" and
if parity failure does not occur, the
Paper Tape Reader will attempt a normal
External Device interrupt. It will
remain "Busy" until the interrupt is
recognized (occurs). Thus, if interrupt
is specified and the External Device
trap is not armed the Reader remains
"Busy" until the trap is again armed,
allowing the interrupt to occur.

Independently of the EDCW interrupt
bit, a parity failure while reading in
the packed mode only will cause the
reader to attempt an External Device
interrupt to the fail address (00003).
A failure inhibits a normal interrupt
if such an interrupt was specified in
the EDCW. The reader remains "Busy"
until the fail interrupt is recognized.
Thus, if the ED trap is not armed, the
"Reader" remains "Busy" until the
interrupt is allowed. The Assembly
Register is released while the Reader
waits "Busy" for either interrupt rec-
ognitions, thus allowing use of the AR

by other ED's.

1.3.4.6.2 Command Codes for the ASI-A30
Typewriter

Command Code	Command
00	Input -

Input from key-
board is accept-
ed until either
(BL) = (BM)
or until the
"terminate
switch" is dep-
ressed. First
character is
input to most sig-
nificant end of
memory location
BM. If number
of words input is
odd, the last word
contains a maxi-
mum of three char-
acters. Remaining
three bits will be
zeros.

02   Output -

Output to type-
writer occurs
until (BL)=(BM).
First character
is taken from most
significant end of
memory location
(BM). If number
of words output is
odd, the last word
may contain a max-
imum of three char-
acters. Remaining
three bits are ign-
ored.

Any code with bit 2 a "zero" will be
interpreted as 00. Any code with bit
2 a "one" will be interpreted as 02.

If bit 14 of the EDCW is a "one" a
normal interrupt will occur (if armed)
at the conclusion of the command.

If an interrupt is specified, the type-
writer remains "Busy" until interrupt
is recognized. Thus, if bit 14 is a
one in an A30 EDCW, and the External
Device trap is not armed, the A30 is
"Busy" until trap is armed causing
the interrupt to be recognized. The
Assembly Register is released while
the ED waits "Busy" for the interrupt
to occur. Thus, other ED's may use
the AR during this time.

1.3.4.6.3 Command Codes for the ASI Paper Tape
Punch

Command Code	Command
00	Punch Packed -
	The contents of memory from location (BM) to (BL) will be punched on paper tape. Six level information code is used. Level seven is pun- ched, as required, to maintain odd parity. A stop code is generated following the in- formation. If an odd number of words is output, only the most significant eight- een bits of the last word are punched.
01	Punch Character -
	Eight bits of

each of the
memory loc-
ations from
(BM) through
(BL)-1 are
punched in
paper tape.
Bits 17 through
10 are punched
in tape levels
8 through 1,
respectively.
No parity bits
or stop codes
are generated.
Other bits are
ignored.

All even codes same as 00

All odd codes same as 01

If bit 14 of the EDCW was a "one" the
punch will attempt a "normal" interrupt
when the punch is finished. The punch
remains "Busy" until the interrupt
occurs (ED trap armed), but the
Assembly Register is released,
allowing use of the AR by other devices.

1.3.4.6.4 Command codes for the ASI-A11 Mag-
netic Tape Unit

Command Code	Command
00	Test - End of Tape -

If the tape unit
has detected an
End of Tape mark-
er (photo-electric,
reflective spot)
or a Load Point
Marker following
the last command
it received, this
command will
cause the A11 to
effect a "Busy"
interrupt. The
"Busy" interrupt
must be armed if

this interrupt
is desired.
The detection
unit is reset
by all commands
except 00, 01, 02
or 03.

01   Test File Mark-

If the tape unit
has detected a
File Mark on
its last command
(must have been
a "Read" command)
this command will
cause the All to
effect a "Busy"
interrupt. The
"Busy" trap must
be armed if this
interrupt is de-
sired. The detec-
tion unit is reset
by all commands
except 00, 01, 02
or 03.

02   Test Fail -

If the tape unit
has detected a
failure on the last
command this com-
mand will cause the
All to effect a
"Busy" interrupt.
The "Busy" trap
must be armed for
the interrupt to
occur. The fail
condition is reset
by any command
except 00, 01, 02
or 03.

03   Not used.

04   Not used.

05   Write Alpha -

The contents of memory locations from the memory addresses specified by (BM) through (BL)-1 will be written on the magnetic tape with even lateral parity. The information is formatted as a single record with longitudinal parity and a record gap inserted at the end of the information. Code translation occurs (See listing of ASI Magnetic Tape Alphanumeric Codes).

06   Write File Mark -

This command will cause a file mark to be recorded on the tape.

07   Write Binary -

The contents of memory locations from the memory address specified by (BM) through (BL)-1 will be written on the magnetic tape with odd lateral parity. The information is formatted as

a single
record with
longitudinal
parity and a
record gap in-
serted at the
end of the in-
formation. All
combinations of
six bits may be
recorded as a
single character.
No code trans-
lation occurs.

10   Move Reverse
            n Records -

            This command
            will cause the
            tape to move
            reverse n records
            (backspace). n
            may be any number
            from $(0)_{10}$ to
            $(511)_{10}$.

11   Not Used.

12   Rewind -

            This command
            causes the tape
            to move reverse
            at high speed
            until the load
            point (leading
            end of tape
            spot) is detected.

13   Not Used.

14   Move Forward
            n Records -

            This command
            causes the tape
            to be moved for-
            ward n records.
            n may be any
            number from

$(0)_{10}$ to $(511)_{10}.$

15   Read Alpha -

The next complete record on magnetic tape will be transferred to computer memory starting with the memory location specified by the initial (BM) and limited by (BL). If the record requires more memory space than allowed, data transfer will cease when (BL)=(BM). That is, memory location (BL)-1 will be the last to be filled. However, tape motion will continue until the record end is reached. If (BM to (BL)-1 is more than sufficient to hold the record, the entire operation is terminated at the record end. Lateral even parity is checked. Code translation occurs (See listing of ASI Magnetic Tape Alphanumeric Codes).

16   Not used.

17   Read Binary -

The next complete record on magnetic tape

will be trans-
ferred to com-
puter memory
starting with
the location spe-
cified by the
initial (BM) and
limited by (BL).
If the specified
memory area is
not sufficient
to hold the
entire record,
data transfer
ceases at loc-
ation (BL)-1,
but tape motion
continues to the
end of the record.
If the memory area
specified is more
than sufficient,
the operation
terminates at the
record gap. Lat-
eral odd parity is
checked. No code
translation occurs.

If bit 14 of the EDCW was a "one" and
if a fail (fail, E. O. T., F.M.)
condition does not occur, the tape unit
will attempt an ED interrupt at the
conclusion of its operation (interrupt
bit is ignored for commands 00, 01,
02 and 03). The tape unit remains
"BUSY" until the interrupt occurs
(trap armed), but the Assembly Reg-
ister is released.

If a fail condition occurs, the All will
attempt an E. D. interrupt to the fail
address. It will remain "BUSY" until
the interrupt, occurs, but will again
release the Assembly Register. A
fail interrupt is not conditional upon
the interrupt bit 14. A fail condition
inhibits a normal interrupt.

While in use, a "Fail" interrupt from an ASI Model A11 magnetic tape unit may occur as a result of four different conditions:

1. A parity fail occurs during a "read" operation.

2. A parity fail occurs (on echo check) during a "write" operation.

3. A file mark is read during a "read" operation.

4. An end of tape detection occurs during a "read" or "write" operation.

Conditions three and four may be Separated from true fails by interrogating the A11 unit for file mark, end of tape and fail. These interrogations are accomplished by specifying the proper operation code and addressing the tape unit which caused the "Fail" interrupt. If the interrogation results in a "yes" answer, the tape unit will attempt an External Device Busy interrupt. If this interrupt is armed, the proper branch in a fail detection subroutine occurs.

Each time a tape unit command other than 00, 01, 02 or 03 is performed (unit is not busy when commanded) The detection circuits for test conditions will be reset.

1.3.4.7   External Device Control Words

Operation	EDCW
Read Alphanumeric Typewriter	0000000
Write Alphanumeric Typewriter	0000002

Read Binary Paper Tape	0200000
Write Binary Paper Tape	0400000
Read Character Paper Tape	0200001
Write Character Paper Tape	0400001
Read Binary Cards	0600000
Write Binary Cards	1000000
Write Alphanumeric Line Printer	1200000
Read Binary Mag. Tape 1	1400017
Write Binary Mag. Tape 1	1400007
Read Alphanumeric Mag. Tape 1	1400015
Write Alphanumeric Mag. Tape 1	1400005
Rewind Mag. Tape 1	1400012
Space forward, Mag. Tape 1, D records	14ddd14
Space backward, Mag. Tape 1, D records	14ddd10
Write End of File, Mag. Tape 1	1400016
End of Tape Test, Mag. Tape 1	1400000
Test End of File, Mag. Tape 1	1400001
Test Fail, Mag. Tape 1	1400002

1. 3. 4. 8   Data Flow

The transfer of data between ED and the associated
assembly register proceeds under the control of
the ED.   The periodic information rate may be
any value less than 62. 5 KC.   Considerably higher
transfer rates are possible under certain given
conditions.   Input/Output channels may transfer
data simultaneously in two channel operation.
The access to main memory from the in/out
channels is made available alternately.   This is
provided by a data transfer scanner such that all
memory access for in/out purposes is made avail-
able to the channels requiring transfer.   The data
rate and channels active are not restricted except
that the program must not require in/out data trans-
fer which exceeds a peak word rate of 62. 5 KC
considered over both in/out channels.

1. 3. 4. 9   External Device Interrupt

Interrupt provisions have been made to facilitate
the input/output data transfer operations.   In the
case of appropriate ED, specification of the
interrupt bit 14 in the EDCW will result in an ED
interrupt at the conclusion of the specified opera-
tion.   The interrupt requests of the ED are handled
by the interrupt scanner and are processed in the
sequence of ED address numbers.   When an ED
interrupt occurs, the program jump is to the loca-
tion having the same address as the ED.   Provisions
for priority and fail interrupts have been made.
When a priority ED makes an interrupt the interrupt
scanner is returned to the lowest number assigned
to the priority ED interrupt block and scans through
the sequence in order to pick up the highest priority
ED seeking interrupt.   The highest priority ED are
assigned the lowest numbers.   This allows early
access to the computer program for priority ED
even though other ED have previously made interrupt
requests.

1.3.4.10   Busy Interrupt

If a previously specified operation of an ED or in/out
channel is not complete at the time it is given its
next instruction a "busy interrupt" will occur.
This results in a jump to a fixed location (00106)
which may lead to the busy routine.

1.3.4.11   Permanently Assigned Memory Locations

Memory Location	Assignment
00000	Typewriter Interrupt
00001	Typewriter Fault
00002	Paper Tape Reader Interrupt
00003	Paper Tape Reader Fault
00004	Paper Tape Punch Interrupt
00005	Paper Tape Punch Fault
00006	Card Reader Interrupt
00007	Card Reader Fault
00010	Card Punch Interrupt
00011	Card Punch Fault
00012	Line Printer Interrupt
00013	Line Printer Fault
00014	Magnetic Tape #1 Interrupt
00015	Magnetic Tape #1 Fault
00016	Magnetic Tape #2 Interrupt
00017	Magnetic Tape #2 Fault
00020	Magnetic Tape #3 Interrupt
00021	Magnetic Tape #3 Fault
00022	Magnetic Tape #4 Interrupt
00023	Magnetic Tape #4 Fault
00024 to 00077	unassigned
00100	Operator Interrupt
00101	Unassigned
00102	Fault Interrupt
00103	Add Overflow Interrupt
00104	Exponent Overflow Interrupt
00105	Index Overflow Interrupt
00106	Busy Interrupt
00107	Unassigned
00110	Interrupt Fixed Address

Memory Location	Assignment
00111 ⎫	
00112 ⎬ unassigned	
00113 ⎭	
00114	Priority ED Interrupt Fixed Address
00115	Index Address 1
00116	Index Address 2
00117	Index Address 3

1.3.5 CODING PROCEDURES FOR THE ASI-210

1.3.5.1 Writing a Program

There are five definite steps that must be performed when writing a program. They are:

1. Analyze the problem
2. Write a flow diagram of the solution
3. Write the program in machine language from the flow diagram
4. Debug the program
5. Run the corrected program

If the programmer uses these five steps in the order presented he will have the smallest amount of difficulty and will arrive at the solution in the shortest amount of time.

1.3.5.2 Mechanics Of Coding Programs For The ASI-210

The first step in coding a program, as was mentioned above, is to analyze the problem. This analyzing consists primarily of determining:

1. What the problem is
2. What input and input format will be used
3. What output and output format will be used

Once the problem is analyzed and the best procedure for solving it is determined, the programmer will write a flow diagram of the problem. The first flow diagram will be very general and illustrate only

the theme and method that will be used to solve the problem.

After all the basic steps are completed the programmer will break each step down until each block in the flow diagram requires only one or two instructions to the computer.

Once the flow diagram is in this form, it is a fairly simple matter to translate the blocks of the flow diagram into instructions to the computer.

When writing the program in machine language, the address of each instruction is also included. The format of the absolute machine code will be, for example:

Memory Location	Instruction
00120	32 0 02000

When the program is completely written the programmer will have it typed on a Flexowriter, or any other appropriate input device, and a "flex" tape will be prepared. (See Volume 2 for the format and use of "flex" tape.) He will then take the tape and read it into the machine. When the program is in the machine the programmer will run the program. The program may or may not run correctly. If it does run correctly, it is in its final form and may be used to solve the problem anytime it is necessary. If the program does not run correctly it must be "debugged".

1.3.5.3   Debugging a Program

Debugging a program is the process of finding and correcting any and all errors in the program. There are definite steps in debugging a program just as there were definite steps in writing a program. These steps are:

1. Re-analyze the problem to make sure the proper solution was used.
2. Check the flow diagram to insure they agree with the solution and are in a logical sequence.
3. Inspect the machine language program for coding errors - (wrong operations codes, indirect address designators, etc.) Most errors should be found in one of these three steps. If the program still does not run or if the program is so long as to prohibit the first three steps because of the time involved, another method of debugging may be used.

The second method of debugging is done on the console of the computer itself.

1. Check to see if the computer is still running or if it has stopped.
2. If the computer is still running, one of the following two things has _probably_ happened.

    a. A loop has been set up within the program and there is no way to get out of the loop.

    b. An external device is being used and the computer is hung up in a "busy" routine, (a "busy" routine is a routine built into the main program that will cause the computer to wait for an external device until the external device is no longer busy.)

The easiest way to determine what is happening in either of these two cases is to stop the computer and place the machine in "one instruction" mode. By examining the contents of the various registers at the end of each instruction it will be possible to find the error by comparing the actual contents of the registers with what should be contained in the registers.

3.   If the computer is stopped one of two things
     has happened.

     a.   The computer has stopped at the normal
          halt instructions of the program.

     b.   The computer has stopped someplace
          other than at the normal halt instruc-
          tion of the computer.

In either case the easiest way to find the error in
the program is to insert halt instructions at criti-
cal points in the program, usually one for each
block of the general flow diagram, and run the
program in steps. This will cause the program
to be run in a series of short operations and at
the end of each halt instruction the programmer
can determine whether this previous portion of
the program has accomplished what it was de-
signed to accomplish.

BASIC PROGRAMMING

Example 1

Load A, Add, Subtract, Multiply, Store E, Halt

Problem: X = a (b + c - d), all values are integer

           a is in memory location   500

           b is in memory location   1030

           c is in memory location   607

           d is in memory location   501

           x to be stored in memory location   0

Program:

LOC	OPN	ADR	EXECUTION	RESULT
03001	14	01030	Bring b to reg. A	b
03002	10	00607	Add c to (A) $\longrightarrow$ reg. A	b + c
03003	12	00501	Subtract d from (A) $\longrightarrow$ reg. A	b+c-d
03004	30	00500	Multiply a . (A), $\longrightarrow$ reg. AE	a (b+c-d)
03005	46	00000	Store (E) $\longrightarrow$ location 0	x
03006	00	03001	Halt $\longrightarrow$ 3001	

Example 2

Skip A Equal, Skip A High, Jump

Problem: If a < b, transfer program sequence to location 250

           If a > b, transfer program sequence to location 257

           If a = b, transfer program sequence to location 300

           a is in memory location   600

           b is in memory location   610

Program:

LOC	OPN	ADR	EXECUTION
00200	14	00600	Bring a to reg. A
00201	40	00610	Skip the next instruction if a > b
00203	02	00205	a ≤ b, Jump to location 205
00204	02	00257	a > b, Jump to location 257

LOC	OPN	ADR	EXECUTION
00205	42	00610	Skip the next instruction if a = b
00206	02	00257	a < b, Jump to location 250
00207	02	00300	a = b, Jump to location 300

Example 3

Absolute value, Minus, Jump A Less than Zero, Store A

Problem: If $b < 0$, make $b > 0$

   If $c \geq 0$, make $c < 0$

Given: $\pm b$ in location 4766

   $\pm c$ in location 4777

Program:

LOC	OPN	ADR	EXECUTION
04056	14	04766	Bring b to reg. A
04057	44	04061	Jump if (A) < 0
04060	02	04063	b $\geq$ 0, go on
04061	20	04000	b < 0, Make (A) absolute
04062	26	04766	Store (A) in b
04063	14	04777	Bring c to reg. A
04064	44	04100	Jump if (A) < 0, c < 0, go on
04065	22	04000	c $\geq$ 0, Make (A) minus
04066	26	04777	Store (A) in c

Example 4

Trap, Divide, Jump Disable Interrupt, Load E, Zero, Halt

Problem: $x = a/b$

   If $a \geq b$ divide fault will occur; interrupt to location 560

   If $a < b$ store quotient (x) in location 203, remainder

   in location 204, and stop program.

Given: a and b are positive integers

   a is in location 170, b is in location 171

Program:

LOC	OPN	ADR	EXECUTION
00120	14	00131	Set fault interrupt location to accept
00121	26	00102	interrupt in the case where $a \geq b$
00122	70	02010	Arm the fault trap
00123	24	04000	Zero reg. A to sign of dividend
00124	16	00170	Bring dividend (a) to reg. E
00125	32	00171	Divide (AE) by b
00126	46	00203	Store quotient (x) in location 203
00127	26	00204	Store remainder in location 204
00130	00	00120	Halt, go to location 120
00131	06	00560	Jump disable interrupt, transfer to location 560

Example 5

Store Address in Index, Augment Index, Skip if Index High

Problem: Zero memory locations 3300 thru 3456, and halt

Program:

LOC	OPN	ADR	EXECUTION
02200	24	04000	Zero register A
02201	54	20000	Store address zero in index 1
02202	26	23300	Store (A) in memory location (3300 + index 1)
02203	50	20001	Add 1 to index 1
02204	52	37622	Skip if index 1 is higher than 155_8
02206	02	02202	(index 1) $\leq 155_8$, go to location 2202
02207	00	02200	(index 1) $> 155_8$, halt

Example 6

Shift, Logical Or

Problem: Merge three six-bit characters into register A. The characters are right justified in three consecutive memory locations (1031-1033) (00000xx).

Program:

LOC	OPN	ADR	EXECUTION
04006	16	01031	Bring first character to reg. E (00000xx)
04007	60	02406	Circular left shift (E) six-bits (000xx00)
04010	56	01032	Logical Or (E) ⊕ (2nd char.) ⟶ A (000xxxx)
04011	26	01000	Transfer (A) to reg. E
04012	16	01000	
04013	60	02406	Circular Left shift (E) six-bits (0xxxx00)
04014	56	01033	Logical Or (E) ⊕ (3rd char.) ⟶ A (0xxxxxx)

Example 7

Store A Address

Problem: The memory address of a value varies; put the value in Register E.

Given: The variable address of the value is in location 1003

Program:

LOC	OPN	ADR	EXECUTION
00736	14	01003	Bring to register A the variable address
00737	36	00740	Store the address portion of Register A into the address portion of location 740
00740	16	XXXXX	This instruction now reads, load E, with the content of the variable address, value ⟶ E

Example 8

Skip Sense Switch Set

Problem: If Sense Switch 3 is <u>set,</u> go to program sequence starting at location 5070.

If Sense Switch 3 is <u>not set,</u> go to program sequence starting at location 5476.

Program:

LOC	OPN	ADR	EXECUTION
04700	72	00004	Test Sense Switch 3

LOC	OPN	ADR	EXECUTION
04701	02	05476	Not set, go to location 5476
04702	02	05070	Set, go to location 5070

Example 9

Logical And

Problem: Put zeros in X where there are zeros in y.

Given: X (5252525) is in location 2033

        y (2525252) is in location 2036

Program:

LOC	OPN	ADR	EXECUTION	RESULTS
02000	16	02033	Bring X to reg. E	
02001	66	02036	(X ⊙ y), goes to reg. A	
02002	26	02033	Store (A) in X	0000000

Example 10

Normalize

Problem: X.y = P

        Retain 20 most significant bits of product.

Given: X is in location 304 and has a scaling factor of 2^9

        y is in location 305 and has a scaling factor of 2^5

        P is to be stored in location 306

        Scaling factor of X is in location 307

        Scaling factor of y is in location 310

        Scaling factor of P is to be stored in location 311

Program:

LOC	OPN	ADR	EXECUTION	RESULTS
00200	14	00307	Add the scale factors	9
00201	10	00310	of X and y and store	5
00202	26	00311	the sum in loc. 311	14
00203	14	00304	Bring X to reg. A	$X.2^9$
00204	30	00305	Multiply (X.y)	$(X.y).2^{14}$

LOC	OPN	ADR	EXECUTION	RESULTS
00205	64	00311	Normalize (AE), shift count is added to (311)	$P. 2^{(14+K)}$
00206	26	00306	Store (A) in loc. 306	

Example 11

Return

Problem: $ax+b=c$

Solve the preceding equation for c_1, c_2, c_3 using three variables of x (x_1, x_2, x_3).

Given: The x variables are in consecutive locations 540, 541, 542.

Compute each c by use of a subroutine.

The c values will be stored in locations 560, 561, 562.

a is in location 603

b is in location 607

Program:

LOC	OPN	ADR	EXECUTION	RESULTS
00505	14	00540	Bring X_1 to reg. A	X_1
00506	04	01005	Place addr. 510 in location 01005	ax_1+b
00507	02	01003	Jump to subroutine	
00510	26	00560	Store (A) in c_1	c_1
00511	14	00541	Bring x_2 to reg. A	x_2
00512	04	01005	Place addr. 514 in location 01005	ax_2+b
00513	02	01003	Jump to subroutine	
00514	26	00561	Store (A) in c_2	c_2
00515	14	00542	Bring x_3 to reg. A	x_3
00516	04	01005	Place addr. 520 in loc. 01005	ax_3+b
00517	02	01003	Jump to subroutine	
00520	26	00562	Store (A) in c_3	c_3

Subroutine

01003	30	00603	Multiply a.x	ax
01004	10	00607	Add ax+b	ax+b
01005	02	00000	Jump to addr. put here by return instruction	

Example 12

Assembly Register, External Device

Problem: Print on the on-line typewriter the words DOG CAT and stop program.

Given: Memory locations 7003 and 7004 contain the flex codes for DOG CAT.

Program:

LOC	OPN	ADR	EXECUTION
06774	76	07000	Set "BM" or "BL" according to content of Loc. 7000
06775	76	07001	Set "BM" or "BL" according to content of Loc. 7001
06776	74	07002	Select external device and function according to content of Loc. 7002
06777	00	06774	Halt, go to 6774
07000	04	07003	Load "BM" with addr. 7003
07001	00	07005	Load "BL" with addr. 7005
07002	00	00002	Write alphanumeric on typewriter
07003	24	46276	Flex codes for DOG CAT
07004	02	32163	

Table 1 - 1

INSTRUCTIONS

Octal Code	Instruction	Time u sec
00	HALT	8
02	JUMP	8
04	RETURN	12
06	JUMP DISABLE INTERRUPT	8
10	ADD	10
12	SUBTRACT	10
14	LOAD A	10
16	LOAD E	12
20	ABSOLUTE VALUE	8
22	MINUS	8
24	ZERO	12
26	STORE A	8
30	MULTIPLY	54
32	DIVIDE	56
34	ROUND	14
36	STORE A ADDRESS	10
40	SKIP A HIGH	14
42	SKIP A EQUAL	14
44	JUMP A LESS THAN ZERO	10
46	STORE E	10
50	AUGMENT INDEX	12
52	SKIP INDEX HIGH	10
54	STORE ADDRESS IN INDEX	12
56	LOGICAL OR	12
60	SHIFT	10+2K
62	NORMALIZE A	14+2K
64	NORMALIZE A, E	14+2K
66	LOGICAL AND	12
70	TRAP	8
72	SKIP SENSE SWITCH SET	10
74	EXTERNAL DEVICE	16
76	ASSEMBLY REGISTER	20

Table 1 - 2

INSTRUCTIONS ARRANGED BY FUNCTION

Octal Code	Instruction	Time in u sec
Stop Commands		
00	HALT	8
Transfer Commands		
14	LOAD A	10
16	LOAD E	12
26	STORE A	8
36	STORE A ADDRESS	10
46	STORE E	10
54	STORE ADDRESS IN INDEX	12
Arithmetic Commands		
10	ADD	10
12	SUBTRACT	10
20	ABSOLUTE VALUE	8
22	MINUS	8
24	ZERO	12
30	MULTIPLY	54
32	DIVIDE	56
34	ROUND	14
50	AUGMENT INDEX	12
62	NORMALIZE A	14+2K
64	NORMALIZE A, E	14+2K
Logical Commands		
56	LOGICAL OR	12
66	LOGICAL AND	12
Shift Commands		
60	SHIFT	10+2K

Skip Commands

40	SKIP A HIGH	14
42	SKIP A EQUAL	14
52	SKIP INDEX HIGH	10
72	SKIP SENSE SWITCH SET	10

Jump Commands

02	JUMP	8
04	RETURN	12
06	JUMP DISABLE INTERRUPT	8
44	JUMP A LESS THAN ZERO	10

Input/Output Commands

74	EXTERNAL DEVICE	16
76	ASSEMBLY REGISTER	20

Interrupt Recognition Commands

70	TRAP	8

Table 1 - 3

EXTERNAL DEVICE ADDRESSES

Device	Address
Typewriter	00
Paper Tape Reader	02
Paper Tape Punch	04
Card Reader	06
Card Punch	10
Line Printer	12
Magnetic Tape Unit 1	14
Magnetic Tape Unit 2	16
Magnetic Tape Unit 3	20
Magnetic Tape Unit 4	20

Table 1 - 4

FLEXOWRITER CODES

Code	Character UC	LC		Code	Character UC	LC
121	A	a		100	)	0
122	B	b		001	-	+
023	C	c		002	@	2
124	D	d		103	#	3
025	E	e		004	$	4
026	F	f		105	%	5
127	G	g		106	'	6
130	H	h		007	&	7
031	I	i		010	:	8
141	J	j		111	(	9
142	K	k		076	Tab	
043	L	l		032	Shift up	
144	M	m		034	Shift down	
045	N	n		136	Backspace	
046	O	o		156	Carriage return	
147	P	p		200	Leader	
150	Q	q		370	Stop	
051	R	r		377	Delete	
062	S	s		133	•	•
163	T	t		073	,	,
064	U	u		020	o	+
165	V	v		040	¨	
166	W	w		054	!	*
067	X	x		013	ꟾ	=
054	Y	y		061	?	/
171	Z	z		160	space	space

Note: The third bit in the code is a parity bit.

Table 1 - 5

MAGNETIC TAPE BCD CODES

Code	Character	Code	Character
61	A	31	Z
62	B	12	0
63	C	01	1
64	D	02	2
65	E	03	3
66	F	04	4
67	G	05	5
70	H	06	6
71	I	07	7
41	J	10	8
42	K	11	9
43	L	60	+
44	M	40	–
45	N	20	blank
46	O	73	.
47	P	53	$
50	Q	54	*
51	R	33	,
22	S	34	(
23	T	74	)
24	U	14	'
25	V	13	=
26	W	37	record work
27	X	77	group mark
30	Y	17	tape mark

Table 1 - 6

LINE PRINTER CODES

Code	Character	Code	Character
21	A	71	Z
22	B	00	0
23	C	01	1
24	D	02	2
25	E	03	3
26	F	04	4
27	G	05	5
30	H	06	6
31	I	07	7
41	J	10	8
42	K	11	9
43	L	20	+
44	M	40	-
45	N	54	*
46	O	61	/
47	P	33	•
50	Q	73	,
51	R	74	(
62	S	34	)
63	T	13	=
64	U	53	$
65	V	60	space
66	W	14	'
67	X	55	#
70	Y	35	&

VOLUME 3

PROGRAMMING MANUAL

SECTION II

ADVANCED PROGRAMMING

2.1   INTRODUCTION

It is assumed by the authors that the programmer reading this
section has read and understood Section I of the programming
manual.

Advanced Programming was written as a guide to the programmer
in the study of the ASI-210 Assembly Program, Fortran I and the
Mathematical Subroutines. This section will show the programmer
how to write assembly and Fortran programs and gives a list and
function of the mathematical subroutines available.

This section can be treated as three separate parts, the ASI-210
Assembly Program, Fortran I Compiler and Mathematical Sub-
routines. The first two parts contain several examples and sample
programs to help the programmer understand the principles and
procedures outlined in this manual.

2.2   ASI-210 ASSEMBLY PROGRAM

2.2.1   INTRODUCTION

The preparation and machine checking of programs in
machine language caused the users of computers to consider
ways in which they might use machine capabilities to do some
of the arduous preparation and checking with fewer errors.
The coding of programs in machine language caused con-
siderable labor not only in checking out the program initially
but in making any changes at a later date. Thus the concept
of relative addressing was conceived so that changes could
be made and programs relocated in storage with greater
ease. Other ideas such as mnemonic operation codes and

symbolic notation were adopted and the assembly programs which incorporated these concepts became widely accepted. The first assembly programs enabled the users to write only machine instructions in their programs. Macro instructions and pseudo instructions that caused many machine instructions to be generated were later added so that data and often-used sub-programs for subroutine libraries could also be assembled.

Thus an assembly program enables a program to be prepared in a more comprehensive, intelligible language than the underlying machine language. The advantages in using assembly programs are:

      1.   Faster preparation of programs
      2.   Ease in making program corrections
      3.   Ease in segmenting and combining programs

These advantages enable users of computer installations to utilize their staff and their equipment more efficiently by obtaining more solutions per dollar.

2.2.2 FORMAT OF THE ASI-210 ASSEMBLY PROGRAM

The format of the ASI-210 Assembly Program is made up of three "fields". The fields are "location", "OPN" (operation), and "address". An explanation of each field follows:

NOTE: Figure 2-1 is a copy of the coding form used with the ASI-210 Assembly.

2.2.2.1   "Location". The location field is an optional field and is used to identify the instruction, control word or operand that follows the location symbol. The maximum length of the location symbol is seven characters, one of which must be an alphabetic character. If, for example, the location field contained the symbol "PETE", anytime an instruction referred to "PETE" the information identified by the location "PETE" would be taken as the operand of the instruction, or in the case of a jump, program control would be transferred to location "PETE".

ASI-210 ASSEMBLY CODING FORM

NAME

PROGRAM

PAGE NO.

ROUTINE

DATE

LOCN	OPN	*	P	BASE ADDRESS, INDEX	(Do not Punch)	SEQUENCE
1	8			15 35	73	80

ASI-210 ASSEMBLY CODING FORM

FIG. 2-1

2-3

ASI Form 1323CC

2.2.2.2 "OPN". The operation field contains the mnemonic operation code of the instruction. The length of this code will be either three or four characters. The operation code may be followed by an asterisk to denote indirect addressing or by a "P" to denote interrupt in an external device control word.

Examples:

Operation Code	Explanation
ADD	Normal add instruction
SUB*	Subtract instruction using indirect address
WATYP	External device control word "write alphanumeric on the typewriter with interrupt."

2.2.2.3 "Address". The address field contains the memory address, of the instruction, data or parameters. The contents of this field may be symbolic or absolute and in the case of absolute the value must be expressed in decimal, except in the case when the operation field contains the code "OCT" which indicates the address field contains an octal value. If the address field is blank, the assembler will assume the address field contains zeros.

For most machine or macro instructions, the address field consists of two parts; an operand address "y", and an index address "X".

The operand address "y" may be a decimal constant or a symbol. It is optionally followed by plus or minus a constant and/or a comma.

Examples:

PETE
JOE+3
-1347,
MAX-1,

The index address "X" may be a symbol or a
digit, 0-3. A comma preceding indicates the
beginning of the index address.

Examples:

    , IND 1
    , 2

NOTE: (1) for shift instructions, the shift
               count ($\leq$ 63) is used in place of
               the operand address.

          (2) for trap instructions, external device
               control words and assembly registers
               control words, special symbols are
               used in the address field.

2.2.3 ASSEMBLY CONTROL INSTRUCTIONS

Certain operation codes are used to control the assembly
process. An explanation of each code follows:

Operation Code	Explanation
REM	Informs the assembler that the contents of the address field are remarks and are not part of the assembly program. A maximum of 21 characters may be used with one REM operation code.
ORG	Specifies that the beginning address of the program being assembled is in the address field. The address field may be symbolic or absolute.

Operation Code	Explanation
EQU	Used to equate the symbol in the location field to the value of the address field. The address field may contain a symbol if the symbol was assigned an absolute decimal value prior to the EQU instruction.

Examples:

| PETE | EQU | 1234 |
| JOE | EQU | PETE |

JOE and PETE = the decimal value 1234

Reserves n number of memory locations where n is the numerical value of the contents of the address field. The symbol in the location field is assigned to the first reserved location. The address may contain a symbol if the symbol was assigned an absolute decimal value prior to the RES instruction. |
| END | Signifies the end of the program. |

2.2.4 DATA INSERTION OPERATIONS

Several operation codes are used to introduce data words into the program. For all of these the symbol in the LOCN field (if present) is assigned to the data; or, for double-precision insertion, to the first of the two resulting words.

Operation Code	Explanation
DEC	Used to insert a one-word decimal integer or a two-word floating point value

Operation Code	Explanation

into memory. The address field will contain the integer or the floating point value.

(1) Fixed Integer - One to 7 digits; preceding + or - sign optional; maximum magnitude $2^{20}-1$.

Examples:

+16
-1048575
47

For scaled integers, the value may be followed by E and a + or - sign (or no sign) and a decimal exponent, and may also be followed by B and a + or - sign (or no sign) and a binary scaling. For example, for π scaled $2+^{18}$, the following would be used:

3.14159E-5B +18

(2) Floating Value - One or more digits, but must include decimal point; preceding + or - sign optional; may be followed by E and + or - sign (or E and no sign) and one or two-digit decimal exponent of maximum magnitude 76:

Examples:

-352.
3.14159
2.718E-2
0.16E52

Operation Code	Explanation
OCT	Used to insert a one-word octal value; the value consists of one to 7 octal digits; preceding + or - sign optional; if - sign is used, the seven's complement of the corresponding digit value is inserted.
ALF	Forms two words containing seven six-bit alphanumeric codes as specified by characters 1-7 in the ADR field.
FLX	Forms two words containing seven six-bit Flexowriter codes, as specified by characters 1-7 in the address field.

NOTE: Special codes, such as carriage return, are denoted by two-character combinations, each starting with an equal sign:

=R Carriage return
=U Shift to Upper Case
=L Shift to Lower Case
=B Backspace
=T Tabulate
== Equal sign

The latter is necessary because of the use of = for the special codes.

GRY	Forms one word containing a value in the gray code bit configuration. The ADR field contains a maximum of seven decimal digits.

Operation Code	Explanation
PAR	Forms one parameter word containing a 00 operation code, and an operand address containing the symbolic address of the address field.

Two operation codes are used when writing library subroutines, to specify the subroutine names and entry points. These are as follows:

NAME	One or more of these are used at the beginning of a library routine to assign one or more names. The name is limited to 6 alphanumeric characters, one of which must be a letter, and is contained in the address field.
ENTRY	Used immediately before each entry point, to specify the location. Each name corresponds to a name previously given by a NAME pseudo-operation.

For example, for a SIN/COS routine,

```
        NAME    SINF
        NAME    COSF
        ENTRY   SINF
SIN     JMP     **
        (etc.)
        then
        ENTRY   COSF
COS     JMP     **
        (etc.)

        END
```

2.2.5 OPERATION CODES FOR MACHINE INSTRUCTION OF THE ASI-210

Machine Instruction	Operation Code
Integer Add	ADD
Integer Subtract	SUB
Integer Multiply	MPY
Integer Divide	DVD
Zero A	ZOA
Zero E	ZOE
Zero AE	ZAE
Minus A	MNA
Minus E	MNE
Minus AE	MAE
Absolute Value A	AVA
Absolute Value E	AVE
Absolute Value AE	AAE
Round	RND
Load A	LDA
Load E	LDE
Store A	STA
Store A in Address	SAM
Store E	STE
Right Shift A	RSA
Left Shift A	LSA
Left Circ. Shift A	CLA
Right Shift E	RSE
Left Shift E	LSE
Left Circ. Shift E	CLE
Right Shift AE	RAE
Left Shift AE	LAE
Left Circ. Shift AE	CAE
Convert Gray to Binary	GBN
Normalize A	NMA
Normalize AE	NAE
Logical And	ANA
Logical Or	ORA
Store Address in Index	SAX
Augment Index	AUX
Skip Index High	KXH
Return	RTN
Jump	JMP
Halt	HLT
Skip A High	KAH

Machine Instruction	Operation Code
Skip A Equal	KAQ
Jump if Less than Zero	JLZ
Jump End Interrupt	JDI
Assembly Register	ASR
External Device	EXD

Sense Switch Instructions

KSS   m   Skip if any of the switches designated by the corresponding bits of m are set. ($0 \leq m \geq 63$)

KSSj   Skip if switch j set, where j = 1-6.

Trap Instructions

Special operation codes are used to form the desired trap instructions, as follows:

STP		Save trap word in A.
SAT	t, t, (etc.)	Save and arm designated traps.
SDT	t, t, (etc.)	Save and disarm designated traps.
ATP	t, t, (etc.)	Arm designated traps.
DTP	t, t, (etc.)	Disarm designated traps.
STH		Save trap indicators for X0 and A0

where t = trap symbol, as follows:
ED = External Device
OP = Operator
FT = Fault
AO = Add Overflow
XO = Index Overflow
BY = Busy
L1 = Control Light 1
L2 = Control Light 2

2.2.6 MACRO INSTRUCTIONS

A macro instruction is an item which during assembly causes more than one machine instruction to be assembled. Certain preset macros are used to specify double-precision floating point operations. These are used in a manner similar to the machine instructions. However, the operands for these each consist of two computer words.

Instruction	Operative Code	Operation
Float	FLT	(A) fixed ⟶ AE flt.
Unfloat	UFL	(AE) flt. ⟶ A fixed.
Floating Load	FLD	(m) ⟶ A, (m + 1) ⟶ E
Floating Store	FST	(A) ⟶ m, (E) ⟶ m + 1
Floating Add	FAD	(AE) + (m, m + 1) ⟶ AE
Floating Subtract	FSB	(AE) - (m, m + 1) ⟶ AE
Floating Multiply	FMP	(AE) · (m, m + 1) ⟶ AE
Floating Divide	FDV	(AE) ÷ (m, m + 1) ⟶ AE
Floating Absolute	FAV	|(AE)| ⟶ AE
Floating Minus	FMN	-(AE) ⟶ AE
Floating Skip High	FKH	Skip if (AE) $>$ (m)
Floating Skip Equal	FKQ	Skip if (AE) $=$ (m)

2.2.7 CONTROL WORDS

2.2.7.1 Assembly Register Control Words

Each of the following forms an ARCW for setting or saving an assembly register, as follows:

SBMc	y, x	Store BMc in (m)
LBMc	y, x	Load BMc with address (m)
LBLc	y, x	Load BLc with address (m)

where

c = channel number
y = symbolic base address
x = symbolic index designator

2-12

2.2.7.2 External Device Control Words

Each of the following forms an EDCW for initiating the desired external device operation, as follows:

RATY		Read Alphanumeric Typewriter
WATY		Write Alphanumeric Typewriter
RBPT		Read Binary Paper Tape (packed)
WBPT		Write Binary Paper Tape (packed)
RCPT		Read Character Paper Tape
WCPT		Write Character Paper Tape
RBCD		Read Binary Cards
WBCD		Write Binary Cards
RACD		Read Alphanumeric Cards
WACD		Write Alphanumeric Cards
WALP		Write Alphanumeric Line Printer
RBTi		Read Binary Magnetic Tape i
WBTi		Write Binary Magnetic Tape i
RATi		Read Alphanumeric Magnetic Tape i
WATi		Write Alphanumeric Magnetic Tape i
RWDi		Rewind Magnetic Tape i
SFTi	d	Space Forward, Magnetic Tape i, d records
SBTi	d	Space Backward, Magnetic Tape i, d records
WEFi		Write End of File, Magnetic Tape i
ETTi		End of Tape Test, Magnetic Tape i
TEFi		Test End of File, Magnetic Tape i

where

i = Tape unit number

d = Decimal Value $0 \leq d \leq 511(2^9-1)$

The operation code is followed by a (P), if ED interrupt action is desired after the operation.

2.2.8 STANDARD EXTERNAL DEVICE NUMBERS

Device	ED Address	Associated Fixed Locations		
Typewriter	00	=TYP	00000	Normal Interrupt
Paper Tape Reader	02	=PTR	00002	Normal Interrupt
Paper Tape Punch	04	=PTP	00004	Normal Interrupt
Card Reader	06	=CDR	00006	Normal Interrupt
Card Punch	10	=CDP	00010	Normal Interrupt
Line Printer	12	=LNP	00012	Normal Interrupt
Mag. Tape 1	14	=MT1	00014	Normal Interrupt
Mag. Tape 2	16	=MT2	00016	Normal Interrupt
Mag. Tape 3	20	=MT3	00020	Normal Interrupt
Mag. Tape 4	22	=MT4	00022	Normal Interrupt

(Additional devices are assigned to locations 00024 - 00077)

2.2.9 ASSEMBLER OUTPUT

The assembler produces two principle types of data: "real
words", and "artificial words". Associated with each word
is a 5-digit location, and the word consists of a 2-digit
operation code, a one-digit index designator, and a 5-digit
base address, where the digits are all octal.

2.2.9.1 Real Words

Real Words are words that represent instructions
or constants that are to be loaded into the computer.
These are constructed as follows:

Location:	00000 - 17777 if location is fixed.
	20000 - 37777 if location is re-locatable.
Opn Code:	00 - 77
Index Des:	0 - 3
Base Adr:	00000 - 17777 if fixed.
	20000 - 37777 if relocatable positive.
	60000 - 77777 if relocatable negative.

The significance of relocatable addresses is ex-
plained in the description of the paper tape formats
in Volume 2.

2.2.9.2 Artificial Words

Artificial Words are used by the library processor to link a main program and its subroutines together. There are only eight types of these, used as shown in the following table:

TABLE 2-1

ARTIFICIAL WORDS

LOCATION	WORD	PURPOSE
70000	3-1/2 IBM Characters	70000 - 70001 used to identify a library sub-routine; results from a NAME pseudo-operation.
70001	3-1/2 IBM Characters	
70002	3-1/2 IBM Characters	70002 - 70004 used to specify a library sub-routine entry point; results from an ENTRY psuedo-operation.
70003	3-1/2 IBM Characters	
70004	00 Code, 0 index, relocatable address	
70005	3-1/2 IBM Characters	70005 - 70007 used to specify a call for a library subroutine; results from a RTN and JMP to a non-local sub-routine.
70006	3-1/2 IBM Characters	
70007	00 Code, 0 index, fixed or reloc. address	
70010	00 Code, 0 index, fixed or reloc. address	Used to denote last location in routine, plus one.

2.2.10 LIBRARY PROCESSOR

The purpose of the Library Processor is to process library routines referenced by the source program and list them on the end of the output tape (binary format), with relocatable locations and addresses modified.

The following action takes place while processing a library tape:

Step 1 - Processor reads library tape.

> For each "name" on the library tape the processor searches the Call Table. All the names of referenced library routines have previously been placed in a Call Table by the Assembler.

Step 2 - If the name on the tape is the same as one of the names in the Call Table the Processor is set to process subroutine.

Step 3 - The Processor will find an artificial location (70002, 70003, and 70004) with assigned name of subroutine and associated with it the relocatable entry address of library routine, which is inserted in Call Table.

Step 4 - The next information to be loaded and processed will be the subroutine itself. The relocatable loading locations and addresses on each record will be modified by a constant alpha, which enables the subroutine to be readily attached to the source program. After modifications are made the record is punched onto the output tape.

> If reference to another library routine is made within a library routine, (designated by locations 70005, 70006, and 70007) its name and modified call address is stored in Call Table. This newly referenced routine can now be processed when found on the library tape.

Step 5 - After all library routines have been processed, patches are punched, as part of the output, to insert the correct calling addresses at the locations where

the subroutine was first referenced. The correct
addresses and location of call is found in Call
Table.

2.2.11 SAMPLE PROGRAM OF THE ASSEMBLY ROUTINE

2.2.11.1 Introduction

Each of the following sample programs are pre-
sented in two parts; the source program and the
output of the assembly process. Before the pro-
grams are examined, let us review the three fields
of an assembly program.

1. Location Field

   The location field contains a symbol that
   identifies the instruction or data of the re-
   maining fields.

2. Operation

   The operation field contains the instruction
   code, in mnemonic form, to the assembler or
   contains the instruction code in mnemonic form
   of the program that is to be assembled.

3. Address

   The address field contains the operand address
   portion of the instruction, or in the case of a
   REM instruction, any remarks the programmer
   desires.

2.2.11.2 Sample Program Using Fixed Point Arithmetic

$$\text{Equation ZILCH} = \frac{Y \cdot Z}{X} + W$$

x = 3
y = 39
z = 7
w = 20

SOURCE PROGRAM

LOCN	OPN	ADR
	NAME	ZILCH
	REM	PROGRAM FOR
	REM	FINDING
	REM	ZILCH VALUE
	ORG	80
START	LDA	YVAL
	MPY	ZVAL
	DVD	XVAL
	STE	TEMP
	LDA	TEMP
	ADD	WVAL
	STA	ZILCH
	HLT	START
XVAL	DEC	3
ZVAL	DEC	39
YVAL	OCT	7
WVAL	DEC	20
TEMP	RES	1
ZILCH	RES	1
	END	START

ASSEMBLY OUTPUT OF PROGRAM

LOCN	OPN	ADDRESS	LOCN	OPN	ADDRESS
70000	31 3	11436		NAME	ZILCH
70001	37 0	02020			
				REM	PROGRAM FOR
				REM	FINDING
				REM	ZILCH VALUE
00120				ORG	80
00120	14 0	00132	START	LDA	YVAL
00121	30 0	00131		MPY	ZVAL
00122	32 0	00130		DVD	XVAL
00123	46 0	00134		STE	TEMP
00124	14 0	00134		LDA	TEMP
00125	10 0	00133		ADD	WVAL
00126	26 0	00135		STA	ZILCH
00127	00 0	00120		HLT	START
00130	00 0	00003	XVAL	DEC	3
00131	00 0	00047	ZVAL	DEC	39
00132	00 0	00007	YVAL	OCT	7
00133	00 0	00024	WVAL	DEC	20
00134			TEMP	RES	1
00135			ZILCH	RES	1
				END	START

RANGE
70010 00 0 00136

2.2.11.3 Sample Program Using Floating Point Arithmetic

$$\text{Equation ZILCH} = \frac{Y \cdot Z}{X} + W$$

$x = 2.97335 \times 10^5$
$y = 1.47350 \times 10^{-2}$
$z = 7.13425$
$w = 10.4213$

SOURCE PROGRAM

LOCN	OPN	ADR
	REM	PROGRAM FOR
	REM	FINDING
	REM	ZILCH VALUE
	ORG	
START	FLD	YVAL
	FMP	ZVAL
	FDV	XVAL
	FAD	WVAL
	FST	ZILCH
	HLT	START
	REM	CONSTANTS
XVAL	DEC	2.97335E+5
YVAL	DEC	1.47350E-2
ZVAL	DEC	7.13425
WVAL	DEC	10.4213
	REM	VARIABLE STORAGE
ZILCH	RES	1
	END	START

ASSEMBLY OUTPUT OF PROGRAM

LOCN	OPN	ADDRESS	LOCN	OPN	ADDRESS
				REM	PROGRAM FOR
				REM	FINDING
				REM	ZILCH VALUE
20000				ORG	
20000	14 0	20020	START	FLD	YVAL
20001	16 0	20021			
20002	04 0	00000		FMP	ZVAL
20003	02 0	00001			
20004	00 0	20022			
20005	04 0	00000		FDV	XVAL
20006	02 0	00001			
20007	00 0	20016			
20010	04 0	00000		FAD	WVAL
20011	02 0	00001			
20012	00 0	20024			
20013	26 0	20026		FST	ZILCH
20014	46 0	20027			
20015	00 0	20000		HLT	START
				REM	CONSTANTS
20016	22 0	11356	XVAL	DEC	2.97335E+5
20017	00 0	00423			
20020	36 0	13261	YVAL	DEC	1.47350E-2
20021	03 2	03372			
20022	34 2	02274	ZVAL	DEC	7.13425
20023	15 1	03403			
20024	24 3	05732	WVAL	DEC	10.4213
20025	12 0	10404			
				REM	VARIABLE STORAGE
20026			ZILCH	RES	1
				END	START

SUBROUTINES NOT INCLUDED

LOCN	OPN	ADDRESS			
70005	66 2	04477			FMP.
70006	32 0	02020			
70007	00 0	20002			
70005	66 3	04257			FDV.
70006	32 0	02020			
70007	00 0	20005			
70005	66 3	01647			FAD.
70006	32 0	02020			
70007	00 0	20010			

RANGE

| 70010 | 00 0 | 20027 |

2.2.11.4 Typical Input/Output Program

Problem: Read a paper tape and punch a duplicate
of that tape.

SOURCE PROGRAM

LOCN	OPN	ADR
	REM	TAPE DUPLICATE
	REM	AND/OR VERIFY
	REM	SWITCH 1 OFF =
	REM	DUPL.
	REM	SWITCH 1 ON =
	REM	VERIFY
	REM	CHKSUM APPEARS
	REM	IN A
	ORG	80
START	ATP	BY
	DTP	ED
	LDA	BYJMP
	STA	=BY
	ZOA	
	STA	0
	STA	TRLRFL
READ	ASR	W1
	ASR	W2
	EXD	W3
	ASR	W1
	LDA	CHAR
	KAQ	ASILDR
	JMP	*+2
	JMP	TRLRTS
	KAQ	ASIDEL
	JMP	*+2
	JMP	READ
	LDA	NEG
	STA	TRLRFL
	LDA	0
	ADD	CHAR
	STA	0
SWTST	KSS1	
	JMP	PUNCH
	JMP	READ
TRLRTS	LDA	TRLRFL

LOCN	OPN	ADR
	JLZ	*+2
	JMP	SWTST
	LDA	0
	HLT	START
PUNCH	ASR	W1
	ASR	W2
	EXD	W4
	JMP	READ
BYJMP	JMP	*+1
BUSY	LDA	=IP
	SUB	1
	SAM	*+1
	JDI	**
=1	OCT	1
NEG	OCT	7000000
ASILDR	OCT	200000
W1	LBM1	CHAR
W2	LBL1	CHAR+1
W3	RCPT	
W4	WCPT	
CHAR	RES	1
TRLRFL	RES	1
ASIDEL	OCT	377000
	END	

ASSEMBLY OUTPUT OF PROGRAM

LOCN	OPN	ADR	LOCN	OPN	ADR
				REM	TAPE DUPLICATE
				REM	AND/OR VERIFY
				REM	SWITCH 1 OFF=
				REM	DUPLI.
				REM	SWITCH 1 ON=
				REM	VERIFY
				REM	CHKSUM APPEARS
				REM	IN A
00120				ORG	80
00120	70 0	02002	START	ATP	BY
00121	70 0	01001		DTP	ED
00122	14 0	00163		LDA	BYJMP
00123	26 0	00106		STA	=BY
00124	24 0	04000		ZOA	
00125	26 0	00000		STA	0
00126	26 0	00200		STA	TRLRFL
00127	76 0	00173	READ	ASR	W1
00130	76 0	00174		ASR	W2
00131	74 0	00175		EXD	W3
00132	76 0	00173		ASR	W1
00133	14 0	00177		LDA	CHAR
00134	42 0	00172		KAQ	ASILDR
00135	02 0	00137		JMP	*+2
00136	02 0	00152		JMP	TRLRTS
00137	42 0	00201		KAQ	ASIDEL
00140	02 0	00142		JMP	*+2
00141	02 0	00127		JMP	READ
00142	14 0	00171		LDA	NEG
00143	26 0	00200		STA	TRLRFL
00144	14 0	00000		LDA	0
00145	10 0	00177		ADD	CHAR
00146	26 0	00000		STA	0
00147	72 0	00001	SWTST	KSS1	
00150	02 0	00157		JMP	PUNCH
00151	02 0	00127		JMP	READ
00152	14 0	00200	TRLRTS	LDA	TRLRFL
00153	44 0	00155		JLZ	*+2
00154	02 0	00147		JMP	SWTST
00155	14 0	00000		LDA	0
00156	00 0	00120		HLT	START
00157	76 0	00173	PUNCH	ASR	W1
00160	76 0	00174		ASR	W2
00161	74 0	00176		EXD	W4
00162	02 0	00127		JMP	READ
00163	02 0	00164	BYJMP	JMP	*+1
00164	14 0	00110	BUSY	LDA	=1P
00165	12 0	00170		SUB	=1
00166	36 0	00167		SAM	*+1

ASSEMBLY OUTPUT OF PROGRAM

LOCN	OPN	ADR	LOCN	OPN	ADR
00167	06 0	00000		JD1	**
00170	00 0	00001	=1	OCT	1
00171	70 0	00000	NEG	OCT	7000000
00172	02 0	00000	ASILDR	OCT	200000
00173	04 0	00177	W1	LBM1	CHAR
00174	00 0	00200	W2	LBL1	CHAR+1
00175	02 0	00001	W3	RCPT	
00176	04 0	00001	W4	WCPT	
00177			CHAR	RES	1
00200			TRLRFL	RES	1
00201	03 3	17000	ASIDEL	OCT	377000
				END	

RANGE
70010   00 0   00202

2.2.12   Assembly Error Indications

Error Indication	Explanation
R	Range > 8K
D	Duplicate Synbol Assignment
M	Format of M Field
*	Erroneous Attempt to Indirect Address
N	"Name" Error "Name" should come first
E	"Entry" error
L	Location field error
I	Index Error
F	Illegal Flex character
U	Unidentified Symbol
O	Operation Code Error

2.3 FORTRAN I COMPILER

2.3.1 GENERAL

Source program entries, called "statements", are translated into object programs in ASI-210 Assembly language. Statements are entered in the form of Flexo-writer tape.

On Flexowriter tape, each statement occupies one "line", which is defined as a maximum of 72 printing characters or spaces, terminated with a carriage return. A blank line, produced by pressing the carriage return more than once for page editing purposes, is ignored.

2.3.2 REPRESENTATION OF VALUES

2.3.2.1 Fixed-Point Constants:

1 to 6 digits; preceding plus or minus sign optional; maximum magnitude $2^{20}-1$, except where used as an index in a DO statement.

Examples:
                16
                -104857
                4756

2.3.2.2 Floating-Point Constants:

Any number of decimal digits; must include decimal point; preceding plus or minus sign optional; may be followed by E and option plus or minus sign and a one or two digit decimal exponent; maximum magnitude 10^{78}.

Examples:
                -352.
                3.1459
                2.178E-2
                0.16E52

2.3.2.3 Fixed Variables:

1 to 6 characters; first character must be letter I, J, K, L, M, N; other characters may be letters or digits.

Examples:
                ITEM
                JIG17

relative strengths, from strongest to weakest, are given below:

Function and Coefficient
**   exponentation
/ and *   division and multiplication
+ and -   addition and subtraction

2.3.3.3   <u>Mode Rules</u>:

A fixed-point expression is one containing all fixed-point values or variables.

A floating-point expression is one containing, in general, all floating-point values or variables; however, the permissable exceptions for a floating value are its

(1)   subscripts (always fixed)

(2)   exponent (floating or fixed)

Example: $X(I, J) = Y**2 + Z**0.52$

An algebraic equation can be fixed on the left and floating on the right, or vice-versa.

Example: $I = X + Y$

$R = I + J - KING$

2.3.4   SUBSCRIPTED VARIABLES

Fixed or floating variable names can be defined by a Dimension statement as one or two dimensional arrays.

In an algebraic expression, or on the left side of an equation, one element of the array is denoted by the array name, followed by one, or two fixed-point expressions, separated by commas and enclosed in parentheses. The subscripts can be ANY fixed

algebraic expressions.

Example: X(I+2, J-3) = R(I-J/2, K**2)

2.3.5 FUNCTIONS

In an algebraic expression, a value resulting from the execution of a function is denoted by the function name followed by one or more argument expressions, separated by commas and enclosed in parentheses. At present, only the following functions are "standard":

Format	Definition
SINF (x) or SIN (x)	sine x
COSF (x) or COS (x)	cosine x
EXPF (x) or EXP (x)	e^x
LOGF (x) or LOG (x)	$\log_e x$
SQRTF (x) or SQR (x)	square root of x
ATANF (x) or ATN (x)	arc tangent of x

The arguments and results are floating. The alternate names are supplied for compatibility with IBM 1620 source programs.

Hand-coded subroutines or functions can be added to the assembler library, and can be referred to by means of Fortran. However, this is a non-standard procedure.

2.3.6 STATEMENT TYPES

C in column 1, followed by 2 or more blanks	Comment
DIMENSION v_1, v_2... where v = name (d_1) for one-dim. array or v = name (d_1, d_2) for 2-dim. array	Specifies certain variables as arrays; d_1 and d_2 are fixed unsigned constants specifying the maximum attainable size of the corresponding array. Each array must be specified in a DIMENSION statement before the name is used in an arithmetic expression.

a = b	Algebraic statement; causes variable a to be replaced by the results of expression b.
Example:	
X = A + B + SINF(C)	
GO TO n	Go to (jump to, transfer to) statement n, where n is a statement number
Example:	
GO TO 64	
GO TO $(n_1, n_2, \ldots n_m), i$	Go to statement n_i, depending upon the value of fixed variable i.
Example:	
GO TO (2, 3, 17, 6), NAN	
IF (a) n_1, n_2, n_3	Go to statement n_1, n_2, or n_3, corresponding to algebraic expression $a < 0, = 0, > 0$.
Example:	
IF (x-y)17, 20, 20	
IF (SENSE SWITCH i) n_1, n_2	Go to statement n_1, or n_2, corresponding to sense switch i ON or OFF; i = 1 through 6.
Example:	
IF (SENSE SWITCH 6) 2, 4	
PAUSE n	Stop, optional octal fixed constant n is ignored, continue when START is pressed.
Examples:	
PAUSE	
PAUSE 1707	
STOP n	Stop, optional octal fixed constant n is ignored; do not continue if START pressed.
Examples:	
STOP	
STOP 140	
DO ni = m_1, m_2, m_3	Perform following statements, down to and including statement n, repetitively; for each execution, use successive values of unsigned fixed variable i, starting with m_1, increasing i each time by m_3, and ending with $i \geq m_2$; if

2-31

$m_3 = 1$, m_3 can be omitted
from the statement; m_1, m_2,
and m_3 can be unsigned in-
tegers, or fixed variables.
Statement n must not be of a
type that causes a transfer
(i.e., GO TO, IF,) state-
ment. If the last statement
executed in a loop is a
transfer-type, an additional
CONTINUE statement must
be added, to preserve
proper DO loop indexing.

Example:
DO 17 JIG = 2, 14, 2
A = B + C (JIG)
IF (A - X) 16, 17, 16
16 PRINT, A, X
17 CONTINUE

In this example, values A
and X are to be printed if
(A-X) is not equal to zero.
If (A-X) is equal to zero,
printing is to be eliminated.
The CONTINUE statement
provides an orderly means
of proceeding from the IF
statement to the bottom of
the loop, so that proper JIG
indexing will occur.

Example:
DO 6 I = 1, 15
6 X (I) = 0.

This performs statement 6
repeatedly, with values of I
equal to 1, 2, 3, ... 15.
Thus, the "DO loop" clears
15 consecutive values of the
array X.

CONTINUE

A dummy statement used as
the last statement in the
range of DO. It merely sat-
isfies the rule that the last
statement in the range of DO
must not be one that can
cause transfer of control.

ACCEPT n, list

Example:
ACCEPT, X, Y, Z(I), JIG

Accept one or more lines of
fixed and/or floating values
from the input typewriter,
convert to machine represen-

tation, and store in the variables specified by the list (See LIST SPECIFICATIONS); n is a FORMAT statement number and is optional.

TYPE n, list

Example:
TYPE 14, JIG, X(JIG)

Convert values specified by the list into alphanumeric fixed and/or floating values, and type as one or more lines on the output typewriter; n is optional and is ignored.

ACCEPT TAPE n, list

Accept one or more lines of fixed or floating values from the paper tape reader, convert, and store in the variables specified by the list; n is optional, and is ignored.

PUNCH TAPE n, list

Convert values specified by the list into alphanumeric fixed and/or floating values and punch as one or more lines on the paper tape punch; n is optional and is ignored.

READ n, list

Read one or more Hollerith cards, containing fixed and/or floating values, and store as specified by the list; n is optional, and is ignored.

PUNCH n, list

Convert values specified by the list into alphanumeric fixed and/or floating values and punch onto one or more Hollerith cards; n is optional and is ignored.

PRINT n, list

Convert values specified by the list into alphanumeric fixed and/or floating values, and print as one or more lines

on the on-line printer.

FORMAT	Ignored
END	Terminate program

2.3.7 INPUT AND OUTPUT

Data for object programs is read or written by means of the input/output statements onto the following media:

1. On-line typewriter
2. 8-level Flexowriter tape
3. 80-column cards
4. Line-printer

For input, data consists of one or more fixed and/or floating values, separated from each other by one or more blanks. The format for these are as specified in Section 2.3.2, "REPRE-SENTATION OF VALUES", paragraphs 2.3.2.1 and 2.3.2.2, "FIXED-POINT CONSTANTS", and "FLOATING-POINT CON-STANTS". One or more cards or lines are read, until the entire list, accompanying the input statement, is exhausted.

For output, data consists of one or more fixed and/or floating values, in 16-column (character) fields. Floating values are printed as a + or - sign followed by a decimal point and 9-digit fraction, followed by an E and a + or - sign and a two-digit exponent. The values are right-justified within each field.

2.3.8 LIST SPECIFICATIONS

In any of the input/output statements, the statement is termin-ated with a "list" specifying the items to be read or written. Each item in the list represents one or more values. Items are separated by commas.

Subscripts can be used with any variable name. Each non-subscripted or subscripted name represents a single value.

Example: TYPE, A, B, C, I, F(I+1, J)

No "DO"-type indexing is permitted in the list. If an array name is used without subscripts, only the first element of the array is typed.

2.3.9 COMPATABILITY WITH 704/709/7090 FORTRAN

If it is desired to compile and run on the ASI-210, and to compile and run the same source programs on the IBM 704, 709, or 7090, certain rules must be followed:

a. The source programs should be on cards, in the following format:

Column 1	C if comment card, Blank if other statement type
Columns 2-5	Statement number
Column 6	Blank
Columns 7-72	Statement
Columns 73-80	Ignored

b. Subscripts should be limited to the following fixed expressions:

variable
constant
variable $\pm$
constant $*$ variable
constant $*$ variable $\pm$ constant

c. For the arithmetic functions, use the following names:

SINF	LOGF
COSF	SQRTF
EXPF	ATANF

d. For input and output data, <u>always</u> use a FORMAT statement. This means that, for input, one should restrict data to certain card fields, and use a corresponding FORMAT statement.

A simplified scheme is to divide the information into fixed fields, with one value per field, right-justified. Then the corresponding FORMAT statement will consist of a statement number and the word FORMAT followed by one or more terms separated by commas and all

enclosed in parentheses, with the terms
being of the following three forms:

(1) For fixed values, of form

nIw, where   n=repeat count, if more
             than one value
             I = letter I, for "integer"
             w = field width

(2) For consecutive floating values containing
     an E and an exponent, of form

nEw. d where
             n = repeat count
             E = letter "E"
             w = field width
             d = number of places after
                 decimal point

(3) For consecutive floating values expressed
     without exponent, of form

nFw. d where
             n = repeat count
             F = letter "F"
             w = field width
             d = number of places after
                 decimal point

A slash (/) can be used to denote the end of
the information for one card, and the start of
another. Exhaustion of the FORMAT specifi-
cations causes the specifications to repeat
from the previous left parenthesis.

For example, to read two cards containing in-
formation of form

6.23   -.47027E+05   25   67
80   904

where these are each separated by one blank,
one may use the following:

```
        READ 7, A, B, I, K, M, NAN
        7 FORMAT (F4.2, E12.5, 2I3/I2, I4)
```

2.3.10 SAMPLE PROGRAMS

2.3.10.1 Sample Program

Problem: $D = \left(\sum_{i=1}^{30} (A_i \cdot B_i)^2 \right)^{1/2}$

Read A_i, B_i from tape for each i.

Punch $(A_i \cdot B_i)^2$ on tape; if $(A_i \cdot B_i)^2 = 0$, do not punch.

Punch D on tape.

Program:

```
C   FORM D, WHICH EQUALS THE SQRTF OF A SUM.
C   SUM IS FORMED FROM THE TERM (A(I) * B(L)**2,
C   WHERE I GOES FROM 1 UP TO AND INCLUDING 30.
    DIMENSION A(30), B(30)
    SUM = 0
    DO 9 I = 1, 30
    ACCEPT TAPE, A(I), B(I)
    PRODSQR = (A(I) * B(I) ** 2
    IF (PRODSQR), 6, 9, 6
6   SUM = PRODSQR + SUM
    PUNCH TAPE, PRODSQR
9   CONTINUE
    D = SQRTF (SUM)
    PUNCH TAPE, D
    STOP
    END
```

2.3.10.2 Sample Program

Problem: Find the value of F, V, U, W

Input

X_i i = 1 ... 15

y_i i = 1 ... 15

Compute

(1) $\qquad A = \displaystyle\sum_{i=1}^{15} y_i$

(2) $\qquad B = \displaystyle\sum_{i=1}^{15} x_i\, y_i$

(3) $\qquad Z = \displaystyle\sum_{i=1}^{15} x_i^2\, y_i$

(4) $\qquad D = \displaystyle\sum_{i=1}^{15} x_i^3\, y_i$

(5) $\qquad E = \displaystyle\sum^{15} x_i^4\, y_i$

(6) $\qquad F = \dfrac{B}{A}$

(7) $\qquad G = \dfrac{Z}{A}$

(8) $\qquad R = \dfrac{D}{A}$

$$(9) \qquad T = \frac{E}{A}$$

$$(10) \qquad V = G - F^2$$

$$(11) \qquad U = R - 3 \cdot F \cdot G + 2F^2$$

$$(12) \qquad W = T - 4 \cdot F \cdot R \cdot + 6 \cdot F^2 \cdot G - 3G^3$$

Output

F, V, U, W

Program

```
C          MOMENT CALCULATION
           DIMENSION  X(15),  Y(15)
           DO 6  I=1, 15
6          ACCEPT TAPE, X(I)
           DO 7  I=1, 15
7          ACCEPT TAPE, Y(I)
           A=0.
           DO 10  I=1, 15
10         A=A+Y(I)
           B=0
           Z=0
           D=0
           E=0
           DO 12  I=1, 15
           B= B+X(I)*Y(I)
           Z= Z+X(I)*B
           D=DD+X(I)*Z
12         E= E+X(I)*D
           F= B/A
           G= Z/A
           R= D/A
```

```
        T= E/A
        V= G-F**2
        U= R-3.0*F*G+2.0*F**2
        W= T-4.0*F*R+6.0*F**2*G-3.0*G**3
        PUNCH TAPE, F, V, U, W
        PAUSE
        END
```

2.4 ASI-210 MATHEMATICAL SUBROUTINES

The basic mathematical subroutines described here are callable
either by the Fortran compiler or by hand-coded assembly language.
For all these, an argument at location X results in a function value
in the accumulator. The argument and resulting value are both
floating. For the trigonometric functions, the argument is expressed
in radians.

NAME	CALLING SEQUENCE		DESCRIPTION
SIN or SINF	RTN	SIN	Sine X to A
	JMP	SIN+1	
	PAR	X	
COS or COSF	RTN	COS	cos X to A
	JMP	COS+1	
	PAR	X	
EXP or EXPF	RTN	EXP	e^X to A
	JMP	EXP+1	
	PAR	X	
LOG or LOGF	RTN	LOG	$\log_e x$ to A
	JMP	LOG+1	
	PAR	X	
SQR or SQRTF	RTN	SQR	x to A
	JMP	SQR+1	
	PAR	X	
ATN or ATANF	RTN	ATN	$\tan^{-1} x$ to A
	JMP	ATN + 1	
	PAR	X	

For these, the possible error alarms are as follows:

EXPF	ERR	Result too large to be represented
SQRTF	ERR	Negative argument

2-40

ASI

MINNEAPOLIS   22.   MINNESOTA