SIMPLE/360 COMPUTER SIMULATOR

1.0 INTRODUCTION

SIMPLE is a fast simulator which duplicates the actions of a very simple computer. It is especially designed for student use, with most jobs using well under one second of computer time. To the user, SIMPLE is an imaginary computer which can be accessed by means of a SIMPLE Assembly Language (SAL). For each new job, the SIMPLE Assembler is called in, which will read the SAL program. After the program is read in it will be loaded into the SIMPLE memory automatically and executed until normal completion is reached or an error occurs.

1.1 PREPARING INPUT FOR SIMPLE

All input to SIMPLE must consist of 80 column cards punched on a keypunch.[1] These cards are stacked into a card deck which is read by SIMPLE in sequence. These cards can be grouped into three categories: 1) Control cards which control the translation and execution of a job; 2) Program cards through which the user can direct SIMPLE's actions; 3) Data cards which contain numbers to be read by the user's program during execution.

1.2 SYSTEM INPUT

SIMPLE reads cards off of a high speed card reader, with certain columns being used for each of the three types of cards. Information for control cards may be punched between columns 1 and 72, with the user free to use columns 73 through 80 as he desires. The exact formats of the control cards are given in sections 2.0 through 2.3 below. Information for SIMPLE Assembly Language program cards may be punched between columns 1 and 72, once again with the user free to use columns 73 through 80 as he desires. The format for program cards is given in section 3.0 below. Data cards use all 80 columns and therefore

[1] See Appendix A for a keypunch description

the entire card is scanned for input data. Furthermore, numbers to be read as data are not confined to any particular columns, but instead may be spaced freely across the card, with blanks separating the numbers.

1.3 SYSTEM OUTPUT

All SIMPLE output is directed to a high speed printer. The printer is used for program card listings, program execution output, and error messages.

2.0 CONTROL CARDS AND JOB STRUCTURE

Running a job on SIMPLE requires the use of only three system control cards, which are identified by a dollar sign ($) in column one of the card. Every job must utilize all three of the control cards described below. The dollar sign in column one is always immediately followed by a control keyword. No intervening spaces are allowed. Over each of the following figures of card descriptions, numbers appear stating in which columns the information must start and end.

```
1 2   6 8      15 17
$ BEGIN account # name
```

Figure 1 Format of a $BEGIN control card.

2.1 $BEGIN CARD

The $BEGIN card serves two purposes: 1) It tells SIMPLE that a new job is coming up in the card reader and starts the appropriate initialization processes; 2) It identifies the programmer as a valid user of the system through the account number. The $BEGIN card follows the standard control card format of a dollar sign in column one followed by the keyword BEGIN. An eight character account number issued to the user by the computer center is contained in columns 8 through 15. The user's name then appears, starting in column 17.

```
12 5
$DATA
```

Figure 2. Format of a $DATA card.

2.2 $DATA CARD

The $DATA card serves three purposes: 1) It tells SIMPLE that it has read in all of the program cards; 2) It has SIMPLE load the SAL program into the SIMPLE memory, with execution starting at location zero; 3) It indicates that data cards to be scanned for input numbers may be following. The $DATA card has the standard control card format of a dollar sign in column one followed by the keyword DATA in columns 2 through 5. The rest of the card is ignored and should be left blank.

```
1 2   4
$ E N D
```

Figure 3 Format of a $ E N D card.

2.3 $END CARD

The $END card is used to indicate the end of the user's deck and therefore should always be the last card in the deck. It has a dollar sign in column one followed by the keyword END in columns 2 through 4.

2.4 SUMMARY

All SIMPLE jobs should be set up in the order given below:

1. It contains the account number starting in column 8 and the programmer's name starting in column 17.
2. Next come program cards containing the SAL program.
3. The program is followed by a $DATA card.
4. Data cards, if necessary, come next.
5. A $END card is always the last card in the deck.

A pictorial diagram of an input deck is given in Figure 4.

```
$BEGIN    EE536S00   Doucette
        SAL  Program Cards
$DATA
      Optional Data Cards
$END
```

Figure 4. Input deck form.

3.0 THE SIMPLE ASSEMBLY LANGUAGE

The operations of a computer can be reduced to a step by step execution of a relatively few basic instructions. To understand the way in which a computer works, consider Figure 5 below:

Figure 5 Block Diagram of a Typical Computer.

Figure 5 represents a typical computer in block diagram form. In the case of SIMPLE all of these components are present. The input unit would be the card reader. The output unit would be the high speed printer.

The SIMPLE memory is a device in which numbers are stored for calculations and other purposes. Obviously, a memory containing numbers with no method of accessing them for processing would not be a useful device. Therefore a means must be created for getting at these numbers. First, it must be decided how many digits a number must be and in what number system the memory (and the rest of the computer) will operate. SIMPLE operates in the decimal number system with all numbers being four digits long. Second, a means must be found for accessing a particular number in the memory. One method is shown in Figure 6. The four digit numbers can be considered to be arranged one behind the other in memory. Each number is then assigned a location number or address, with the first number having address zero, the second number address one, and so on up through the last number which will have the highest address. The SIMPLE memory has a capacity of 100 numbers, having addresses from 0 to 99. In addition, all addresses have two digits, with zeroes put on the front of the number if necessary. Therefore, the SIMPLE memory has addresses ranging from 00 to 99.

Figure 6 SIMPLE Memory Arrangement.

The arithmetic unit is where all of the computer's calculations are performed. In addition to the electronic circuitry necessary to do these calculations, places are needed for numbers which are to be acted upon to be placed and results made available. These places are called registers, which may be thought of as being similar to the cash register where the results of intermediate calculations (the figuring of the bill) are displayed. SIMPLE has one register, which is called the accumulator, sometimes abbreviated as AC. It is only through the accumulator that the arithmetic unit can communicate with the rest of the computer.

The control unit is the heart of the computer, for it is the control unit which directs every action that the computer performs. The entire subject of computer programming then reduces to having the control unit issue the proper commands to obtain the desired results. The control unit is designed to direct certain basic actions. The SIMPLE control unit can direct only 11 such actions. Each action has a two digit numeric code associated with it called the operation code or opcode for short. This means that SIMPLE has 11 such codes (one for each basic action) running from 00 to 10 (note that two digits are needed for all opcodes since the opcode 10 is a two digit number). For example, if the control unit receives an opcode of 00, it will stop SIMPLE. The next question that arises is how does the control unit receive opcodes. The answer is that the control unit fetches a four digit number from the memory and uses the first two digits as an opcode. The second two digits are used as a memory address for memory accesses during the action being performed and is called the operand part of the number. Numbers which are fetched by the control unit and used to direct operations through their opcodes are called instructions. The control unit fetches instructions sequentially from the memory unit and keeps track of which

instruction it is supposed to fetch next by means of its instruction counter. For each instruction, the control unit goes through the following steps:

1. Fetch the instruction from memory into the control unit. (Location of the instruction is specified by the location counter.)
   Add one to the location counter.
2. Interpret the instruction and route the required data.
3. Execute the instruction.
4. Store the results and return to step one.

At this point the basic actions of SIMPLE, also known as the SIMPLE instruction repetoire or instruction set will be examined.

3.1 THE SIMPLE INSTRUCTION SET

Each instruction in the SIMPLE instruction set is identified by a two digit opcode. As an aid to the programmer, each instruction also has an abbreviation called a MNEMONIC which describes in a compact way what the instruction does. In the list below, the opcode and mnemonic is given first, followed by a brief description of the action performed:

0 0 STP STOP THE COMPUTER

On encountering this opcode, the SIMPLE control unit would stop fetching instructions from memory and all activity would cease. Intervention by the computer operator would be required to restart the machine. In the SIMPLE simulator, this opcode would terminate execution of the job and would have SIMPLE go on to the next job. The operand field of the instruction is not used.

0 1 FET FETCH TO ACCUMULATOR

This opcode has the control unit fetch the number from memory at the address in the operand field and put it in the accumulator. The number in the memory is not changed, but the old number in the accumulator is erased and replaced by the number from memory.

0 2 ADD ADD NUMBER TO ACCUMULATOR

The control unit fetches the number at the operand address in the memory and has the arithmetic unit add it to the number in the accumulator, putting the result back in the accumulator. The old value of the accumulator is destroyed when it is replaced by the sum. Memory remains unchanged.

0 3 SUB SUBTRACT NUMBER FROM ACCUMULATOR

The control unit fetches the number at the operand address and has the arithmetic unit subtract it from the number in the accumulator, putting the difference back in the accumulator. The old value of the accumulator is destroyed when it is replaced by the difference. Memory is unchanged.

0 4 MUL MULTIPLY ACCUMULATOR BY NUMBER

The control unit fetches the number at the operand address and has the arithmetic unit multiply the number in the accumulator by it, putting the product back in the accumulator. The old value of the accumulator is destroyed when in it is replaced by the product. Memory is unchanged.

0 5 BZE BRANCH IF ACCUMULATOR ZERO

The control unit tests the value of the accumulator. If the accumulator is zero, the instruction counter is set to the address in the operand field. Therefore, the next instruction to be executed will be at this address, and execution will proceed sequentially again starting from this location. This is called <u>branching</u> or a <u>transfer</u> of <u>control</u>. If the accumulator is not zero, no branching takes place and execution continues with the next sequential instruction. The numbers in the accumulator and memory remain unchanged.

0 6 BMI BRANCH IF ACCUMULATOR MINUS

The control unit tests the value of the accumulator. If it is negative (minus), the instruction counter is set to the address in the operand field and execution continues from the new address. If the accumulator is non-negative, no branching takes place and execution continues from the next sequential address. Memory and accumulator values are unchanged.

0 7 TRA TRANSFER CONTROL

The control unit sets the instruction counter to the operand field and execution continues from the new address. Memory and accumulator contents are unchanged.

0 8 GTI GET INPUT

A number is read from a data card and is stored into the memory address specified by the operand. All 80 columns of the data card are scanned left to right, with blanks separating different numbers. The numbers may have a sign in front if desired. If no sign is given, a positive sign is assumed. When all the numbers on a data card have been read the next data card is read and scanned until execution ends normally through the execution of a STP instruction, an error occurs which abnormally stops execution, or the end of input data is encountered. The only valid characters for input data are the arithmetic signs in front of the numbers and the decimal characters zero through nine.*

0 9 PTO PUT OUTPUT

The number in the location specified by the operand is transmitted to the printer for output. In the SIMPLE simulator the message:

LOCATION nn =

---

*The old value of the memory location referenced by the operand is erased and the new value read from the data card is substituted. The rest of the memory and the accumulator remain unchanged.

appears on the output printout, where nn is the operand address whose value is being printed. This is immediately followed by the value itself. The memory and accumulator are unchanged.

1 0 STO STORE ACCUMULATOR

The number in the memory location specified by the operand is erased and is replaced by the value of the accumulator. The rest of the memory and the accumulator are unchanged.

3.2 LOADING PROGRAMS INTO SIMPLE

All programs written for SIMPLE consist of the eleven instructions described above. The next step is to put this program into the SIMPLE memory. This is accomplished by program cards which are read by SIMPLE and stored into its memory. The program cards have a fixed format, which means that certain columns on the card must contain specific pieces of information. Each program card contains one instruction to be loaded into the SIMPLE memory. No assumptions may be made regarding the numbers in the various locations in the SIMPLE memory, also referred to as the contents of these locations. The contents of any location not loaded by a program card is unpredictable when exectuion first starts.

The first piece of information needed is in what location in memory to put the instruction constructed from this program card. This two digit location is specified in columns two and three of the card.

The second piece of information is the first two digits of the number to be stored. This can be specified in one of two ways: 1) As an optional sign followed by the first two digits to be stored (if the sign is omitted, a positive sign is assumed) which could represent a two digit opcode if the instruction counter ever reaches this location; 2) As a three letter mnemonic given in the list above which will be translated into a two digit

opcode with positive sign (which could also be the first two digits of a number to be stored as a constant in the memory unit). If the numeric form is used the optional sign appears in column 5, with the two digits in columns six and seven. If the mnemonic form is used the three letter mnemonic appears in columns five, six, and seven.

The third item to be determined is the last two digits of the number, which would correspond to the operand field if the number were to be executed as an instruction. These two digits are placed in columns nine and ten.

The rest of the card is ignored and may be used to contain comments for use in remembering the purpose of the instruction. The number has now been successfully constructed from the program card and is stored in the memory at the appropriate location.

3.3 ADDITIONAL PROGRAM CARD FEATURES

Certain features have been included in the SIMPLE simulator for easier programming.

The first feature is automatic location numbering. If the location field (columns 2 and 3) on the program card are left blank, the location to be loaded is considered to be one greater than the location loaded from the preceding program card. The first program card, if the location field is blank, will automatically be loaded into location zero.

The second feature of interest is the ability to give symbolic names to program cards as well as specific addresses. For example, if a program is written which transfers control to a specific address which contains a certain instruction and the program is modified so that the instruction is at a new address, then the entire program must be modified to transfer control to the

new address. It would be far easier to give the program card a name and transfer control to this name. In the SIMPLE simulator, program cards may have a single letter as a name. No two cards may have the same name, and it is not necessary to give names to program cards. It is merely a feature for the user's convenience. If a program card is to be named, the single letter name must be put in column one of the card. If the program card is to remain unnamed, column one should be left blank.

The use of automatic location numbering and symbolic names makes up a primitive assembly language. Loosely speaking, an assembly language is defined as a language in which one program card (also known as a statement) will translate directly into one machine code instruction. The assembly language described here is called the SIMPLE Assembly Language (SAL) since it is the assembly language of the SIMPLE machine.

Another feature available is that if the operand field is left blank the operand is assumed to be zero.

Finally, a symnolic name may be given to an address by merely putting the name in column one and the address in columns two and three. It should be noted that the contents of the location specified in this manner is unpredictable.

3.4 SIMPLE ASSEMBLY LANGUAGE SAMPLE PROGRAMS

Given below are three examples which are given to bring out some of the features that SIMPLE has to offer. For each example a brief description of the problem is given, followed by a listing of the deck to be run and the output produced by the SIMPLE simulator for this deck.

Example 1: Read numbers from data cards and print them. Stop if a negative number is read. Use specific locations and addresses

INPUT DECK

```
$BEGIN EE536S00 DOUCETTE
 00 GTI 99 READ THE DATA NUMBER INTO LOCATION 99
 01 FET 99 PUT THE NUMBER INTO THE ACCUMULATOR
 02 BMI 05 IF IT'S NEGATIVE,BRANCH TO STP INSTRUCTION
 03 PTO 99 PRINT THE NUMBER
 04 TRA 00 START PROCESS FOR ANOTHER NUMBER
 05 STP    STOP THE MACHINE IF A NEGATIVE NUMBER IS READ (BMI AT 02)
$DATA
   1          10  15 231            16         -3
$END
```

COMPUTER PRINTOUT

```
$BEGIN EE536S00 DOUCETTE

LABEL  LOCATION  OPERATION  OPERAND  COMMENTS
-----  --------  ---------  -------  --------
             0        GTI       99   READ THE DATA NUMBER INTO LOCATION
             1        FET       99   PUT THE NUMBER INTO THE ACCUMULATO
             2        BMI       05   IF IT'S NEGATIVE,BRANCH TO STP INS
             3        PTO       99   PRINT THE NUMBER
             4        TRA       00   START PROCESS FOR ANOTHER NUMBER
             5        STP            STOP THE MACHINE IF A NEGATIVE NUM

$DATA
LOCATION 99=     1
LOCATION 99=    10
LOCATION 99=    15
LOCATION 99=   231
LOCATION 99=    16

STOP AT LOCATION    5

ASSEMBLY TIME=   0.283   TOTAL TIME=    0.433
$END
```

Example 2: Repeat example 1, only use automatic location numbering where possible and symbolic names for program cards.

INPUT DECK

```
$BEGIN EE536S00 SHOOMAN
X    GTI A
     FET A
     BMI I
     PTO A
     TRA X
I    STP
A99
$DATA                                                    3
 1 5                          2
          1          -10
$END
```

COMPUTER PRINTOUT

```
$BEGIN EE536S00 SHOOMAN

LABEL   LOCATION   OPERATION   OPERAND   COMMENTS
-----   --------   ---------   -------   --------
  X          0         GTI       A
             1         FET       A
             2         BMI       I
             3         PTO       A
             4         TRA       X
  I          5         STP
  A         99

$DATA
LOCATION 99=     1
LOCATION 99=     5
LOCATION 99=     2
LOCATION 99=     3
LOCATION 99=     1

STOP AT LOCATION    5

ASSEMBLY TIME=   0.367   TOTAL TIME=    0.533
$END
```

Example 3: Repeat example 1, using numeric opcodes instead of mnemonics.

```
$BEGIN EE536S00 SCHWARTZ
 00   08 99
 01   01 99
 02   06 05
 03   09 99
 04   07 00
 05   00 00
$DATA
   6    4              1 -2
$END
```

COMPUTER PRINTOUT

```
$BEGIN EE536S00 SCHWARTZ

LABEL   LOCATION   OPERATION   OPERAND   COMMENTS
-----   --------   ---------   -------   --------
                0          08        99
                1          01        99
                2          06        05
                3          09        99
                4          07        00
                5          00        00

$DATA
LOCATION 99=     6
LOCATION 99=     4
LOCATION 99=     1

STOP AT LOCATION    5

ASSEMBLY TIME=   0.300   TOTAL TIME=    0.400
$END
```