Using the Implementation of the SIC/XE Simulator

1. Introduction/Simulator Commands

SIC (Simplified Instructional Computer) is a machine designed to illustrate the most commonly encountered computer hardware features and concepts, while avoiding most of the idiosyncracies that are often found in real machines. There is an enhanced version of the machine (SIC/XE), that includes additional features (more addressing modes, floating point instructions, interrupts, virtual memory and memory protection features). A complete description of SIC and SIC/XE is contained in *System Software: An Introduction to Systems Programming* by Beck (Addison-Wesley).

At this version, a partial implementation of SIC/XE is provided via a modified version of a SIC/XE simulator originated by Beck. It is installed on the Unix system. The simulator includes some (but not all) of the SIC/XE features.

Once logged onto the system under UNIX the SIC simulator is started by entering

```
sicsim
```

The simulator will display

```
 SIC SIMULATOR V1.6
File names are:
DEV00
DEVF1
DEVF2
DEVF3
DEV04
DEV05
DEV06
COMMAND: A(CCEPT file names, R(ENAME 1 or more files
```

Section 2 (I/O Device Conventions) describes the function of each of these files. By entering `RENAME` (or `R`) you will be able to alter any of the above (default) file names to correspond to the file names you have decided to use. A null return on any file name prompt leaves it unchanged. The file name list and prompt is repeated until you enter the command `ACCEPT` (or `A`). At this point the primary command line

```
COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
```

is displayed. You may enter any of the commands described below; each command may be abbreviated by entering only its first letter.

START

Entering `S` causes the simulator to read 128 bytes of data from device `00` (default file name `DEV00`) into memory, starting at address `0000`. This command would normally be used to bootstrap a loader or other program into memory. See the section "I/O Device Conventions" below for further information.

RUN

This command causes the simulator to begin executing SIC machine language instructions from a program in memory. There are two forms of the command:

```
R
R  <address>
```

If an address is specified in the command, the next instruction to be executed is the one starting at that address. (All addresses specified in commands are given in hexadecimal.) If no address is specified, the next instruction to be executed is the one following the last previously-executed instruction, if any, or the instruction beginning at address `0000`. Execution continues until an error occurs, or the number of instructions specified by `HCOUNT` has been executed, or a breakpoint specified by `BKPT` is reached (see below).

ENTER

This command is used to enter values into registers or memory locations. The two possible forms of the command are

```
E  R<reg-id>  xxxxxx
E  <address>  xxxx...
```

In the first case, `R`*<reg-id>* is a register identifier (`RA, RX, RL, RB, RS, RT`). Data to be entered into the register is given in hexadecimal notation, with two hexadecimal digits specifying each byte of data. When entering data into a register, exactly three bytes (6 hex digits) must be given.

In the second case, any number of bytes of data may be entered into memory, starting at the address specified. Each byte of data to be entered is specified with two hexadecimal digits, as above.

DUMP

This command is used to display the contents of registers and memory locations. There are three possible forms of the command:

```
D  R
D  <startaddr>-<endaddr>
D  R,<startaddr>-<endaddr>
```

If `R` is specified, the contents of all registers are displayed in hexadecimal, along with the current value of the condition code. If *startaddr* and *endaddr* are specified, the contents of the indicated range of address are displayed; a maximum of 320 (decimal) bytes can be dumped at one time. Because memory is displayed in rows of 16 bytes each, the actual dump may include some bytes before *startaddr* and some bytes after *endaddr*.

HCOUNT

This command is used to specify the maximum number of SIC instructions to be executed in response to a `RUN` command. This limit allows the user to regain control in the case there is an unending loop in the program being simulated. The command has the form

```
H  <nmbr>
```

where *nmbr* is a value given by 1 to 4 decimal digits. The maximum value that can be specified is `9999`; if no `HCOUNT` command is entered, the default value is `1000`. An `HCOUNT` of `1` allows the program to be "single-stepped".

After *n* instructions have been executed, the simulator displays

```
<nmbr> INSTRUCTIONS EXECUTED
P=xxxxxx
```

where `xxxxxx` is the current program counter value (i.e., the address of the next instruction to be executed). Entering `RUN` as the next command will resume execution at this point (for another *nmbr* instructions).

BKPT

This command is used to set a breakpoint to control instruction execution. The form of the command is

```
B <address>
```

When the next instruction to be executed begins at the specified address, the simulator displays

```
 BREAKPOINT REACHED
P=xxxxxx
```

where `xxxxxx` is the current location counter value (i.e., the breakpoint address). Entering `RUN` as the next command will resume execution at this point.

FILES

This command reactivates the `ACCEPT/RENAME` procedure for file names described earlier.

TRACE

This command initiates a display of the last 10 instructions executed.

QUIT

This command is used to terminate simulation.

2. I/O Device Conventions

Device `00` (default file name `DEV00`) is used only by the `START` command; it contains 128 bytes of bootstrap data, represented with two hexadecimal digits (characters `0-9` and `A-F`) per byte. For ease of creation and editing, this file is represented as a text file with four lines of data; each line contains 64 characters (which represent 32 bytes of data). In other words, this file can be created using any of the text editors under UNIX (e.g., "`ex`", "`vi`" or "`ed`").

A device `00` loader (described further below), stored in the file "`loader`" (which must be used in place of the default file "`DEV00`") is supplied for use by COP 3601 students. This loader will handle a **text** file of hexadecimal digits (such as created under "`ex`", "`vi`" or "`ed`") , converting them to true numeric form and storing them 2 hex digits per byte in SIC memory. The device `F1` file (default file name `DEVF1`) is used to provide the input to this loader.

The simulator supports six simulated SIC devices for use by the program: devices F1, F2, and F3 (default file names DEVF1, DEVF2, and DEVF3), which can be used only for input, and devices 04, 05, and 06 (default file names DEV04, DEV05, and DEV06), which can be used only for output. For any of these files, each byte of data is represented as one character. On input, an end-of-line is read as hexadecimal 0A; and end-of-file is read as hexadecimal 04. On output, writing a hexadecimal 0A causes an end-of-line to be inserted.

Device timing delays are simulated via the TD instruction. Except for the first time the device is addressed, a TD (Test Device) issued to a device will return the "device busy" indication from 1 to 4 times before signalling "device ready". An attempt to execute an RD or WD instruction before the device is ready will cause an error message.

3. Notes

1. The largest main memory address is 2FFF. When the simulator is initialized, all registers are set to FFFFFF and all memory locations are set to FF.
2. The file "sic.log" contains a listing of all terminal input and output for the simulation run. This file may be printed to obtain a hard-copy record of the simulation.
3. When the simulator detects a run-time error (for example, illegal machine instruction, address out of range, or arithmetic overflow) it displays an error message and the current program counter value. This value will be either the address of the instruction that caused the error, or the address of the next instruction following it (depending upon the type of error detected).

4. Limitations

This version of the simulator supports all SIC/XE instructions and features except for the following:

1. Floating-point data type and associated instructions (ADDF, SUBF, MULF, DIVF, COMPF, LDF, STF, FIX, FLOAT, NORM)
2. I/O channels and associated instructions (SIO, HIO, TIO)
3. Interrupts and associated instructions (LPS, STI, STSW, SVC)
4. Register SW and associated features (user/supervisor modes, running/idle states)
5. Virtual memory and memory protection

5. Example Session

Consider the following example program:

```
loc   code
            EXAMPLE    START    100
0100  01000C            LDA      #12         . LOAD 12 INTO REG A
0103  190007            ADD      #7          . ADD 7 TO REG A
0106  0F2003            STA       SAVA       . STORE A IN MEMORY
0109  4F0000            RSUB                 . RETURN
010C          SAVA      RESW     1
                        END      EXAMPLE
```

Assume the code is stored in the UNIX file "DEVF1" as follows:

```
000100      <<< load point
000100      <<< starting address
01000C      <<< first line of code
190007
```

```
0F2003
4F0000      <<< last line of code
!           <<< code delimiter
```

A terminal session to execute this code on the SIC simulator might generate the following "sic.log" file (a facsimile generated by the UNIX command "cat sic.log" with annotations later added inside []; see note 2 in Section 3 (Notes)). The loader file which replaces the default file "DEV00" is described in Section 6 (SIC/XE Loader).

```
$sicsim

SIC SIMULATOR V1.6
File names are:
DEV00
DEVF1
DEVF2
DEVF3
DEV04
DEV05
DEV06
COMMAND: A(CCEPT file names, R(ENAME 1 or more files

r                   [<<< r entered to prompt renaming]
DEV00
loader              [<<< system boot redirected to the file "loader"]
DEVF1
                    [<<< null response (so file remains DEVF1)]
DEVF2
                    [<<< null response (so file remains DEVF2)]
DEVF3
                    [<<< null response (so file remains DEVF3)]
DEV04
                    [<<< null response (so file remains DEV04)]
DEV05
                    [<<< null response (so file remains DEV05)]
DEV06
                    [<<< null response (so file remains DEV06)]

File names are:
loader
DEVF1
DEVF2
DEVF3
DEV04
DEV05
DEV06
COMMAND: A(CCEPT file names, R(ENAME 1 or more files
a                   [<<< "a" entered to operate with these files]

COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
d r,0-2f            [<<< dump of registers and memory 0000 through 002F]
A=FFFFFF   X=FFFFFF   L=FFFFFF   B=FFFFFF
S=FFFFFF   T=FFFFFF   P=000000   CC=LT

0000 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0010 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0020 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
s                   [<<< "s" entered to load the boot file (loader)]

COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
d r,0-7f            [<<< dump to show the boot file is now in memory]
A=FFFFFF   X=FFFFFF   L=FFFFFF   B=FFFFFF
```

```
S=FFFFFF   T=FFFFFF   P=000000   CC=LT

0000 B4104B20 2F572071 4B202957 206C4B20
0010 23572067 6B20624B 201A5720 5C4B2014
0020 5720574B 200E5720 524B2008 57C000B8
0030 103F2FF5 1720454B 200EA403 5720404B
0040 20064720 383E2034 E100F133 2FFAB400
0050 D900F129 00303B20 0F1D0030 29000A3B
0060 20031D00 074F0000 29002133 20062900
0070 04372FD4 B4203E20 00FFFFFF FFFFFFFF

COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
b 109              [<<< break point to stop program prior to executing  RSUB]

COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
r                 [<<< "r" entered to initiate execution (from  PC=000000)]
BREAKPOINT REACHED  [loader has finished and turned execution over to loaded]
P=000109           [program which has executed through location  106]

COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
t                 [<<< "t" entered to show last 10 instructions executed]
Trace of last 10 instructions executed
  PC   Instruction  [assembly code from section 7 and program above]
000050 D900F1       [RD       #X'F1']
000053 290030       [COMP     #48]
000056 3B200F       [JLT       EOFCK        <<< branch taken to 68=59+F]
000068 290021       [COMP     #33]
00006B 332006       [JEQ       EXIT        <<< branch taken to 74=6E+6]
000074 B420         [CLEAR     L]
000076 3E2000       [J        @ADDR        <<< branch taken to address stored at 79]
000100 01000C       [LDA      #12]
000103 190007       [ADD      #7]
000106 0F2003       [STA       SAVA]

COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
d 100-110          [<<< dump of the section of memory holding the loaded code]
                   [result boldfaced]
0100 01000C19 00070F20 034F0000 000013FF
0110 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
r                  [<<< "r" entered to continue execution]
ATTEMPT TO READ DEVF1 PAST END OF FILE
P=000053           [loaded program once done returns control to the loader]
                   [which attempts to load the next one (it's not there!)]

COMMAND: S(TART, R(UN, E(NTER, D(UMP, H(COUNT, B(KPT, F(ILES, T(RACE, Q(UIT?
q                  [<<< "q" entered to end simulation and finalize "sic.log"]
```

6. SIC/XE Loader

The SIC/XE loader is a 128 character SIC/XE program designed to fit in the 128 character boot file space. It is composed of fully relocatable code. The loader is designed to input a load module of hex characters from a text file, converting them to true numeric form, and storing them beginning from a specified load point. At the end of the load, control is passed to a specified starting address.

Load module file format

A text file of hex characters as follows:

1. 1st 6 characters = load point

2. Next 6 characters = address to receive control when load is finished ("000000" to re-run the loader and input the next module - see below)
3. Rest of file = code to be stored in memory beginning from the load point
4. Module delimiter = !

Restrictions

5. The loader input is via device F1 (default file name DEVF1)
6. The loader is stored in the file "loader". The SIC simulator will boot from this file under the START command if the default file name for device 00 (DEV00) is replaced by the file name "loader".
7. The load point must be greater than X'00007F'
8. The loader does no error checking for validity of input

Normal object code layout for use by loader

Each load module normally consists only of actual code. In particular, the storage directives RESW and RESB have no corresponding code and so typically provide a natural point at which to terminate a load module (alternately, garbage could be inserted into the code to reserve the needed amount of storage). If more than one load module is employed for a program, then the loader must be restarted for each load module. This is done by putting the loader start address (000000) as the start address component for each load module except the last one. The last load module has the start address for the full program in its start address component. For example, the code

```
loc   code
            EXAMPLE    START    100
0100  01000C            LDA      #12         . LOAD 12 INTO REG A
0103  190007            ADD      #7          . ADD 7 TO REG A
0106  0F2003            J        STORE       . STORE A IN MEMORY
0109          SAVA      RESW     10
0127  0F2FDF   STORE     STA      SAVA
012A  4F0000            RSUB                 . RETURN
                        END      EXAMPLE
```

could be set up for the loader as

```
000100                          <<< load point for the first module
000000                          <<< start address of loader (gets next module loaded)
01000C                          <<< first line of code for this module
190007
0F2003
!                               <<< end of the first module
000127                          <<< load point for the second module
000100                          <<< start address of the program (no more to load)
0F2FDF                          <<< first line of code for this module
4F0000
!                               <<< end of the second module
```

or alternately as the single module

```
000100                          <<< load point for the first module
000100                          <<< start address of program
01000C                          <<< first line of code for this module
190007
0F2003
FFFFFFFFFFFFFFFF                <<< 30 garbage bytes
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFF
```

```
        0F2FDF
        4F0000
!                              <<< end of module
```

Loader algorithm

```
PROCEDURE SICloader

   INTEGER  loadpt, startaddr, i   { i implemented via registers B + X }
   CHARACTER ascii                 { ascii implemented via register A  }
   HEXDIGIT  hex

   LOOP
      Get(loadpt)                 { 3 calls to "GETPAIR"              }
      i = loadpt
      Get(startaddr)              { 3 more calls to "GETPAIR"         }

      LOOP                       { "GETPAIR" strategy                 }

         Input(ascii)

         IF ascii < '0' THEN
            IF ascii = '!' THEN
               EXIT
            ENDIF
         ELSE
            hex = Hexconvert(ascii)  { convert ASCII representation by  }
            Input(ascii)             { subtracting down to range 0..F   }
            IF ascii < '0' THEN
               IF ascii = '!' THEN
                  EXIT
               ENDIF
            ENDIF
         ENDIF

         MEMORY[i] = Hexpair(hex, Hexconvert(ascii))
         i = i + 1                { store the 8 bit pair formed from }
                                  { "hex" and the one converted from }
                                  { "ascii" into memory location i   }
      REPEAT

      CALL loadpt
   REPEAT

END SICloader
```

7. Loader Source Code

```
1.  loc   code
                . . . . . . . . . . . . . . . . . . . . . . . . . . . .
                .                                                      .
                .            SIC/XE LOADER VERSION 1.2                 .
                .                                                      .
                .        FOR USE BY STUDENTS IN CSE2013-01             .
                . . . . . . . . . . . . . . . . . . . . . . . . . . . .
2.                LOADER    START    0
3.   0000  B410              CLEAR    X
4.   0002  4B202F            JSUB     GETPAIR
5.   0005  572071            STCH     ADDR
6.   0008  4B2029            JSUB     GETPAIR       . GET LOAD POINT
7.   000B  57206C            STCH     ADDR2         .    INTO ADDR
8.   000E  4B2023            JSUB     GETPAIR
9.   0011  572067            STCH     ADDR3
10.  0014  6B2062            LDB      ADDR          . MOVE IT TO BASE REG B
```

```
11. 0017  4B201A            JSUB     GETPAIR
12. 001A  57205C            STCH     ADDR
13. 001D  4B2014            JSUB     GETPAIR      . GET START ADDR
14. 0020  572057            STCH     ADDR2        .    INTO ADDR
15. 0023  4B200E            JSUB     GETPAIR
16. 0026  572052            STCH     ADDR3
17. 0029  4B2008  LOOP      JSUB     GETPAIR      . GET A BYTE OF SOURCE
18.                         BASE     ADDR
19. 002C  57C000            STCH     ADDR,X       . STORE AT B+X WITH 0 DISP
20.                         NOBASE
21. 002F  B810              TIXR     X            . (X)=(X)+1
22. 0031  3F2FF5            J        LOOP
                .
                .        SUBROUTINE TO INPUT THE NEXT 2 CHARACTERS (HEX)
                .        CONVERTING TO NUMERIC FORM IN RIGHTMOST BYTE OF A
                .
23. 0034  172045  GETPAIR  STL      RTADDR       . SAVE RETURN ADDRESS
24. 0037  4B200E            JSUB     READ         . GET 1ST HEX DIGIT,
25. 003A  A403              SHIFTL   A,4          .    SHIFT LEFT TO CORRECT
26. 003C  572040            STCH     HEX          .    POSITION & HOLD IT
27. 003F  4B2006            JSUB     READ         . GET 2ND HEX DIGIT,
28. 0042  472038            OR       ORADDR       .    "OR" IT WITH THE 1ST
29. 0045  3E2034            J       @RTADDR       .    TO FORM THE FULL BYTE
                .
                .        SUBROUTINE TO INPUT A CHARACTER (HEX)
                .        AND CONVERT IT TO NUMERIC FORM
                .
30. 0048  E100F1  READ      TD      #X'F1'        . TEST DEVICE (DEVF1)
31. 004B  332FFA            JEQ      READ         .    LOOPING UNTIL READY
32. 004E  B400              CLEAR    A
33. 0050  D900F1            RD      #X'F1'        . INPUT CHARACTER TO REG A
34. 0053  290030            COMP     #48          . SKIP IF NOT A
35. 0056  3B200F            JLT      EOFCK        .    HEX CHARACTER
36. 0059  1D0030            SUB      #48          . CONVERT FROM
37. 005C  29000A            COMP     #10          .    CHARACTER TO
38. 005F  3B2003            JLT      GOBACK       .    NUMERIC FORM
39. 0062  1D0007            SUB      #7
40. 0065  4F0000  GOBACK    RSUB
41. 0068  290021  EOFCK     COMP     #33          . EXIT ON "!" CHARACTER
42. 006B  332006            JEQ      EXIT
43. 006E  290004            COMP     #4           . EXIT ON EOF
44. 0071  372FD4            JGT      READ
45. 0074  B420    EXIT      CLEAR    L            . SET RETURN TO SYSTEM
46. 0076  3E2000            J       @ADDR
47. 0079  FF      ADDR      RESB     1            . STORAGE 1ST FOR LOAD POINT
48. 007A  FF      ADDR2     RESB     1            .    THEN FOR START ADDRESS
49. 007B  FF      ADDR3     RESB     1
50. 007C  FF      RTADDR    RESB     1            . STORAGE FOR RSUB RETURN;
51. 007D  FF      ORADDR    RESB     1            .    THESE 2 BYTES USED BY
52. 007E  FF                RESB     1            .    "OR"; EFFECT ON BYTE
53. 007F  FF      HEX       RESB     1            .    NAMED "HEX" IS ONLY
54.                         END      LOADER       .    PART USED
```

8. Contents of the file "loader"

```
B4104B202F5720714B202957206C4B20235720676B20624B201A57205C4B2014
5720574B200E5720524B200857C000B8103F2FF51720454B200EA4035720404B
20064720383E2034E100F1332FFAB400D900F12900303B200F1D003029000A3B
20031D00074F000029002133200629000 4372FD4B4203E2000FFFFFFFFFFFFF
```

9. Reading a SIC Dump

A dump instruction such as

```
d r,0-1f
```

causes a dump of the registers and SIC memory including addresses from 0000 through 001F with the result:

```
A=FFFFFF   X=FFFFFF   L=FFFFFF   B=FFFFFF
S=FFFFFF   T=FFFFFF   P=000000   CC=LT

0000 B4104B20 2F572071 4B202957 206C4B20
0010 23572067 6B20624B 201A5720 5C4B2014
```

In this case each of the registers (A, X, L, B, S, T) has value FFFFFF, the program counter (P) has the value 000000 indicating the instruction at memory location 000000 is what will be executed next and the condition code (CC) is currently set to less than (LT).

Taking the dump instruction from the example of Section 5:

```
    d 100-110          [dump of the section of memory holding the loaded code]
                       [result boldfaced]
0100 01000C19 00070F20 034F0000 000013FF
0110 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

simply count 0 through F in pairs along each row to find a desired location

```
0100 01000C19 00070F20 034F0000 000013FF
    ^ ^ ^ ^  ^ ^ ^ ^  ^ ^ ^ ^  ^ ^ ^ ^
    0 1 2 3  4 5 6 7  8 9 A B  C D E F
```

and add the lead location to find the memory address. In particular, you then find that the underlined part begins at 0100+C = 010C. This is precisely the location of "SAVA" where the program stored its result. A dump is a good way of verifying correctness of intermediate stages of a program under construction.

10. Additional Support Utilities

SIC/XE assembler

SIC assembler source files can be assembled to the loader format of Section 6 by using the SIC/XE assembler.

For source code stored in the file "sicprog" the usage is

```
sicasm sicprog
```

The assembler report is then stored in the file "sicprog.lst" and the object code in the file "sicprog.obj". Since file names are limited to a maximum of 14 characters, be cautioned that the source file name can be no more than **10** characters (including ".").

The object program (`sicprog.obj`) can be loaded and executed under the SIC simulator by setting file name `DEVF1` to the object file name (`sicprog.obj`).

SIC integer files

Since SIC uses 24 bit integers, a utility is available to COP 3601 classes which will generate files organized as 24 bit integers. These files can then be processed by SIC programs directly (note that the loader described in Section 6 has to do its own conversion from a text file format to a binary format; if the object program was stored in a binary rather than a text format, a much faster loader could be devised).

The utility is called "`sicdtoh`" and is invoked by executing `sicdtoh`

The user is prompted for the target file name and interactively enters (in decimal or in hex) the integers to store in the file in 24 bit format. The Unix command "`od`" can be used to examine the contents of the file produced by `sicdtoh`; for example,

```
od -x sicint
```

will display the file "`sicint`" in hex form.