1105

Bell Laboratories **Cover Sheet for Technical Memorandum**

*The information contained herein is for the use of employees of Bell Laboratories and is not for publication. (See GEI 13.9-3)*

Title- **An Interactive Control Language for SIM**

Date- **June 28, 1976**

TM- **76-1271-8**

Other Keywords-


| Author | Location | Extension | Charging Case- 39919 |
|---|---|---|---|
| **L. L. Cherry** | **MH 2C-516** | **6067** | Filing Case- **39919-12** |



*ABSTRACT*

An interactive control language to be used in conjunction with the computer simulation language SIM [1] is now available on the UNIX operating system. The control language provides an easy way for the user to run a simulator from the terminal, setting and inspecting memory and registers as the simulation progresses. The simulator may be single-stepped or run in a loop until some condition is met. The control language is useful both for checking out a simulator and debugging a program written to run on the simulated machine. Among the features included in the control language are subroutining, selective execution and both command and binary file input.

The control language will be made available under GCOS on request. This paper is intended to be a reference manual and assumes familiarity with SIM.


| | | | |
|---|---|---|---|
| Pages Text 4 | Other 5 | Total 9 | |
| No. Figures 0 | No. Tables 0 | No. Refs. 1 | |

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

<AHO,ALFRED V
<BAKER,BRENDA S
<BECKER,RICHARD A
BROWN,W STANLEY
+CERMAK,I A
+CHAPPELL,S G
<CHEN,STEPHEN
<CHERRY,LORINDA L
+CHOW,W F
<FLEISCHER,H I
<FRASER,A G
<GOLDSTEIN,A JAY
<GRAHAM,R L
<GUTHERY,S B
<HAMILTON,PATRICIA A
HAMMING,R W
+HANNAY,N B
+HAWKINS,RICHARD B
<JOHNSON,STEPHEN C
<KEESE,W M
+KEPLEY,GARRY D
<KERNIGHAN,BRIAN W
<LUDERER,GOTTFRIED W R
<MARANZANO,J F
<MARKY,GERALDINE A
+MC DONALD,H S
<MC GILL,R
MCILROY,M DOUGLAS
MORGAN,SAMUEL P
<OSSANNA,J F,JR
PINSON,ELLIOT N
+PRIM,R C
<RALEIGH,T M
<RIDDLE,GUY G
<SCHLEGEL,C T
+SCHWARTZ,W C
+SIMONE,C F
TERRY,M E
+THOMAS,LEE C
<THOMSON,MAJA-LISA
+VLACK,DAVID
<WALFORD,ROBERT B
<YAMIN,ELAINE E
43 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

ACKERMAN,A FRANK
AHERN,P L
AHRENS,RAINER B
ALBERTS,BARBARA A
ALCALAY,D
ALEXANDER,E J
ALLEN,JAMES R
ALLISON,C E,JR
ALT,DOROTHY L
AMITAY,N
AMORY,ROBERT W
AMOSS,JOHN J
AMRON,IRVING
AMSTER,S J
ANDERSON,KATHRYN J
ANDERSON,L G
ANTOLICK,DAVID R
ARMSTRONG,D B
ARNOLD,GEORGE W
ARNOLD,S L
ARNOLD,THOMAS F
ARTHURS,E
ARTIS,H P
ATAL,BISHNU S
BACCASH,JEANNE M
BACKUS,C F,SR
BALDWIN,GEORGE L
BASEIL,RICHARD J
BATTAGLIA,FRANCES
BAUER,HELEN A
BAUER,STEPHEN M
BAUGH,C R
BAYER,D L
BEL BRUNO,KATHLEEN A
BERGLAND,G D
BERNSTEIN,DANIELLE R
BERNSTEIN,L
BERRYMAN,R D
BERTH,R P
BEYER,JEAN-DAVID
BIANCHI,M H
BICKFORD,NEIL B
BILOWOS,R M
BIRCHALL,R H
BIREN,IRMA B
BISHOP,VERONICA L
BLAZIER,S D
BLEICHER,E
BLINN,J C
BLUE,JAMES L
BLUM,MARION
BODEN,F J
BOHACEK,P K
BOLSKY,MORRIS I
BONANNI,L E

COVER SHEET ONLY TO

BOURNE,STEPHEN R
BOWEN,F W
BOWERS,JERRY L
BOWRA,JAMES W
BOWYER,L RAY
<BOYCE,W M
BOYLE,W S
BRANDT,RICHARD B
BRANTLEY,JOAN T
BREEN,R S,JR
BREILAND,JOHN R
BRONZO,JOSEPH A
BROWNLOW,D L
BROWN,C W
BROWN,JAMES W
BROWN,W R
BUCHANAN,D N E
BULFER,A F
BURGESS,JOHN T,JR
BURNETTE,W A
<BURNETT,DAVID S
BUTLETT,DARRELL L
BYRNE,EDWARD R
<BZOWY,D
CALLAHAN,R L
CANADAY,RUDD H
CANNON,T B
CARDOZA,WAYNE M
CARRAN,JOHN H
CARR,DAVID C
CASPERS,BARBARA L
CATO,H L
CAVINESS,JOHN D
CHAFFEE,N F
CHAMBERS,B C
<CHAMBERS,J M
CHANG,HERBERT Y
CHEE,T
CHENG,CHENG-WEN
CHEN,E
CHIANG,T C
CHODROW,M M
CHRIST,C W,JR
CLAYTON,D P
CLOUTIER,J
CLYMER,J C
COBEN,ROBERT M
COHEN,HARVEY
COKE,ESTHER U
COLE,LOUIS M
COLE,MARILYN O
COLLINS,J P
COLLUM,D JOHN
COLSON,J S,JR
COMMRADE,GERALDINE
CONNERS,RONALD R
COOK,ROBERT W
COOK,T J
COOPER,ARTHUR E
COPP,DAVID H
COREY,D A

COVER SHEET ONLY TO

CORNELL,R G
COSTANTINO,BASIL B
COSTELLO,PETER E
COSTON,W P
CRAGUN,DONALD W
CRANE,R P
CROWE,MARGARET M
CRUME,L L
CUNNINGHAM,S J,JR
CUTLER,C CHAPIN
DAVIDSON,CHARLES LEWIS
DAVIS,D R
DAVIS,R DREW
DE GRAAF,D A
DE JAGER,D S
DE PAOLA,THELMA T
DEERING,PATRICIA A
+DENENBERG,CHARLOTTE G
DERMOND,FAITH L
DESENDORF,JUDITH L
DEUTSCH,DAVID N
DEVLIN,N V
DI PIETRO,R S
DICKMAN,BERNARD N
DIMMICK,JAMES O
DOLAN,MARIE T
DOLOTTA,T A
DONNELLY,MARGARET M
DONOFRIO,LOUIS,J
DOWDEN,DOUGLAS C
DOWDEN,IRIS S
DOWD,P G
DRAKE,LILLIAN
DRECHSLER,R C
DREIZLER,H K
DRISCOLL,P J
DRUMMOND,R E
D'ANDREA,LOUISE A
DSTEFAN,D J
DWYER,T J
EDELSON,DAVID
EDMUNDS,T W
EHLINGER,JAMES C
EIGEN,D J
EITELBACH,DAVID L
ELDREDGE,BARBARA D
ELLIOTT,RUBY J
ELY,T C
EMERSON,ROBERT S
ERRICHIELLO,PHILIP M
ESSERMAN,ALAN R
ESTES,VIRGINIA DANIELLE
ESTOCK,RICHARD G
ESTVANDER,R A
EVENSON,E K
EVERETT,W W
FABISCH,M P
FASCIANO,V
FEDER,J
FELDMAN,STUART I
FELS,ALLEN M

COVER SHEET ONLY TO

FELTON,WILLIAM A
FERIDUN,K K
FERREK,NANCY L
FETTERMAN,C H
FETTE,C J
FISCHER,HERBERT B
FLANDRENA,R
FLEMING,J W
FLUHR,ZACHARY C
FLYNN,MARY L
FOLEY,G
FONG,K T
FONTENOT,SHARON M
FORTNEY,V J
FOUGHT,B T
FOUNTOUKIDIS,A
FOWLER,C F
FOX,A
FOX,PHYLLIS A
FOY,J C
FRANCIS,SAMUEL H
FRANK,AMALIE J
FRANK,H GREGORY
FRANK,RUDOLPH J
FRANZ,ANN M
FREEBURG,JAMES R
FREEDMAN,MARVIN I
FREEMAN,K G
FREEMAN,R DON
FRENCH,WILLIAM G
FRIEDENSON,ROBERT A
FROST,H BONNELL
FULTON,A W
GALLANT,R J
GALVIN,MICHAEL F
GANNON,T F
GARCIA,R F
GATES,G W
GAY,FRANCIS A
GEERS,T J,JR
GELBER,CHERON L
GEPNER,JAMES R
GERGOWITZ,E B
GEYLING,F T
GIBB,KENNETH R
GIBSON,H T,JR
GILBERT,G W
<GILLETTE,DEAN
GILLON,ALEX C
GILMER,G H
GIMPEL,J F
GINSBERG,N
GITHENS,J A
GLASER,W A
GLASSER,ALAN L
GLUCK,F G
<GNANADESIKAN,R
GODWIN,R E
GOFTZ,FRANK M
GOGUEN,N H
GOLABEK,RUTH T
...
732 TOTAL

+ NAMED BY AUTHOR > CITED AS REFERENCE < REQUESTED BY READER (NAMES WITHOUT PREFIX WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION.....

COMPLETE MEMO TO:
127-SUP

COVER SHEET TO:
12-DIR 13-DIR 127

COHART = COMPUTER HARDWARE RELIABILITY, TESTING AND SIMULATION
COPLMP = COMPUTER MICROPROGRAMMING LANGUAGES AND PROCESSORS
UNPL# = UNIX/PROGRAMMING LANGUAGES

---

CHERRY,L L
MH 2C516

TM-76-1271-8
TOTAL PAGES 10

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.

PLEASE SEND A COMPLETE COPY TO THE ADDRESS SHOWN ON THE OTHER SIDE
NO ENVELOPE WILL BE NEEDED IF YOU SIMPLY STAPLE THIS COVER SHEET TO THE COMPLETE COPY.
IF COPIES ARE NO LONGER AVAILABLE PLEASE FORWARD THIS REQUEST TO THE CORRESPONDENCE FILES.

Bell Laboratories

Subject: **An Interactive Control Language for SIM**
Case- 39919 -- File- 39919-12

date: **June 28, 1976**

from: **L. L. Cherry**

TM: **76-1271-8**

*MEMORANDUM FOR FILE*

## 1. Introduction

An interactive control language to be used with the computer simulator language SIM [1] is now available under the UNIX operating system. The control language is useful both for checking out a simulator and debugging a program written to run on the simulated machine.

SIM is a language and compiler designed to ease the task of writing computer and microprocessor simulators. The SIM source language is used to define registers and memory and to name parts of registers. These SIM variable definitions appear at the beginning of the SIM machine description and may be set or retrieved with the interactive controller. The machine operations or cycles are defined as SIM functions which may be executed by the controller.

To create an interactive simulator the user first writes the machine description in the SIM language. The machine description is then compiled to produce a C program. This C program is the source for the simulator functions. If the C program is compiled and loaded with the control language an interactive simulator results. The same SIM output may be compiled and loaded with a user supplied main program or a SIM supplied main program to produce a noninteractive simulator.

## 2. SIM Register and Array Input

SIM variables defined in the beginning of the program may be loaded with a control language statement of the form:

*variable* = *value* ;

or

*variable* <- *value* ;

where *value* may be a number, a variable or an array element. Numbers preceded by 0 are considered octal. The operators = and <- are equivalent.

Array elements may be loaded with the statement

*variable*[*value*] = *value* ;

or

*variable*[*value*] <- *value* ;

Several array elements may be loaded with a statement of the form:

*variable*[*value*] {$value_1, value_2, \ldots, value_n$};

Here *variable*[*value*] is loaded with $value_1$, *register*[*value*+1] with $value_2$, etc.

The control statements to initialize the MAC-8 simulator described in Appendix 1 could be:

```
sp = 0100;
rp = 0;
pc = 040;
mem[pc]{0305,020,0306,040,0};
```

Here the stack pointer is set to 0100, the first 32 byte section of memory is defined as the register block, the program counter is set to 040 and a small program is loaded into memory starting at 040.

The simulator's memory array may also be loaded from a binary file. This method should be used if the program to be simulated has been assembled. Since assemblers usually put a header before the assembled code, the command

**seek** *offset filename* ;

may be used to skip this header. *offset* is the number of host machine words to be skipped from the beginning of the file. +*offset* causes **seek** to skip *offset* words from the current position of the read pointer in the file. *offset***b** indicates the count is in bytes instead of words. *offset* may be either decimal or octal but must be a positive number. The command to load an array with the contents of the file is

**read** *number register*[*value*] *filename* ;

where *number* words (or bytes) are read into *register* starting at *register*[*value*]. As described above *number* may be decimal or octal and a trailing **b** indicates bytes. **read** loads one host machine word (or byte) per array element.

If the program in the previous example has been assembled into file **prog** which has an 8 word header, the commands to load it into memory would be

```
seek 8 prog ;

read 5b mem[pc] prog ;
```

Note that this file must be read in bytes to get memory loaded properly. Because read loads one word (or byte) per array element and the MAC-8 memory is byte-oriented, reading words would cause every other byte to be loaded rather than every byte.

### 3. SIM Register Output

Registers and array elements can be inspected simply by typing a list of their names, separated by , and ending in ;. The output is in octal and there is one line of output per line of input. The command to print the values of pc, rp and mem[0] is

```
pc, rp, mem[0];
```

the output will be

```
pc = 040  rp = 0  mem[0] = 0305
```

## 4. Functions

The simulator that is created by SIM defines machine cycles as functions. These functions may be executed with a control command of the form

*function*( );

The controller also allows for local functions, i.e., functions containing controller commands. Local functions are defined by

*function*( ) {*statement list*}

and are called in the same manner as SIM functions.

The definition of a local function, run, that calls execute and dump is

```
run( ) { execute( );
         dump( );
       }
```

Local functions may be printed with the *list* command as follows:

**list** $function_1, function_2 \cdots$;

## 5. Loops

The control language provides for both while and do-while loops. The while loop has the form

**while**(*value cond value*) *statement*;

or

**while**(*value cond value*){*statement list*}

where *value* is a variable, number or array element and *cond* is one of the conditional operators <, >, <=, >=, != or ==. The substatement or statement list is executed repeatedly until the condition is (*or* becomes) false. The test is performed before each execution. The comparison is done as though all variables were long integers without sign extension. Care should be taken in testing for negativity.

The do-while loop has the form

**do** {*statement list*} **while** (*value cond value*);

The do-while is essentially the same as the while except that the test is performed after execution.

An additional control aid for loops is the **every** statement which has the form

**every** (*number*) *statement* ,

or

**every** (*number*) {*statement list*}

The substatement or statement list is executed every *number*$^{th}$ time thru the loop.

The commands to run the SIM function execute until the program counter is greater than or equal to 050, executing function dump every 3$^{rd}$ time thru the loop are

```
do {
   execute( );
   every(3) dump( );
} while (pc < 050);
```

**6. Controller Input**

The controller normally takes its input from the standard input. The **input** command allows control commands to come from a file. The command is

**input** *file* ;

Succeeding input is taken from *file*. When an end-of-file is reached the controller switches back to the standard input. The **input** command is useful for doing standard initialization that is desired for each simulation run.

**7. Usage**

On UNIX the command

**sim** *file*.**sim**

produces the source for the simulator on *file*.**d**. The command

**simcc** *file*

compiles *file*.**d** and loads it with the controller, leaving an executable interactive simulator in **a.out**.

To produce a non-interactive simulator using either the SIM supplied main program or a user supplied main program the command becomes

**simcc** – *file*

L. L. Cherry

MH-1271-LLC

Attachments
References
Appendix 1

**Reference**

[1] L. L. Cherry, *SIM, a language for simulating computers*, TM-76-1271-2.

**Appendix 1**

```
%{              /* MAC-8 simulator */
                /* op-codes */
#define   ON  1
#define   OFF 0
#define   MOVE     020
#define   MOVE16   030
#define   ADD      025
#define   ADD16    035
#define   INCRDECR 0
#define   COMPL    1
#define   BRANCH   013
long signext();
%}
%mem[1024]<0:7>          /* memory */
%rp<0:15>                /* register pointer */
%sp<0:15>                /* stack pointer */
%pc<0:15>                /* program counter */
%cr<0:7>                 /* condition register */
%neg = cr<0:0>           /* flags */
%zero = cr<1:1>
%ovfl = cr<2:2>
%carry = cr<3:3>
%ones = cr<5:5>
%odd = cr<6:6>
%flag = cr<7:7>
%ir<0:7>                 /* instruction register */
%op = ir<0:4>
%mode = ir<5:7>
%mmode = ir<6:7>
%op1 = ir<5:5>
%ind = mode<0:0>
%twobytes = op<1:1>
%ds<0:7>                 /* source and destination byte */
%d = ds<0:3>
%s = ds<4:7>
%t16<0:15>               /* temporary addressing register */
%sc<0:15>                /* simulator temporary registers */
%dst<0:15>
%wr<0:16>
%er<0:15>
%tmode<0:2>
%1execute
      ir <- mem[pc];
      ds <- mem[pc+1];
      pc =+ 2;
      decode(op){
MOVE:
MOVE16:
      tmode <- mode;
      getsc(); getdst();
      wr <- sc;
      decode(twobytes){
      OFF: neg <- wr<9:9>;
           odd <- wr<16:16>;
```

```
            0:    zero <- 1;
            !0:   zero <- 0;
            0377:ones <- 1;
            !0377:      ones <- 0;
            }
      ON:  neg <- wr<1:1>;
            zero <- 0;
            (wr<1:16> == 0) => zero <- 1;
      }
      putdst();
ADD:
ADD16:
      tmode <- mode;
      getsc(); getdst();
      wr <- sc + dst;
      setcond();
      putdst();
5:    tmode < mmode;
      decode(op1){
      INCRDECR: getdst();
            decode(s<0:0>){
            OFF: wr <- wr + s;
            ON: wr <- wr +(s|0177760);
            }
            setcond();
            putdst();
      COMPL:  getdst();
            wr <- ~dst;
            setcond();
            putdst();
      }
BRANCH: decode(mmode){
      0:    pc <- mem[pc]::ds;              /* branch immediate */
      1:    pc <- pc + signext(ds);         /* branch relative */
      2:    er <- rp +(s << 1);                  /* branch on register bit */
            sc <- mem[er+1] :: mem[er];
            wr <- 1 << (15 - d);
            decode(sc & wr){
            !0:   pc <- pc + signext(mem[pc]);
            0:    pc =+ 1;
            }
      3:    er <- rp + (s << 1);                 /* branch on memory bit */
            sc <- mem[er+1]::mem[er];
            sc <- mem[sc];
            wr <- 1 << (15 - d);
            decode(sc & wr){
            !0:   pc <- pc + signext(mem[pc]);
            0:    pc =+ 1;
            }
      }
}
%2dump
%0getsc           /* routine to put source value in sc
                  see Appendix 2 for meanings of codes */
      er <- rp + (s << 1);
      decode(s,mode,twobytes){
      [0,14],[0,3],OFF:      sc <- mem[er];
```

```
        [0,14],[0,3],ON:
        [0,14],5:       sc <- mem[er+1]::mem[er];
        [0,14],7:       sc <- mem[er+1]::mem[er];
                        decode(twobytes){
                        OFF: wr = sc + 1;
                        ON:  wr = sc + 2;
                        }
                        mem[er] <- wr<9:16>;
                        mem[er+1] <- wr<1:8>;
        15,[0,3],OFF:   sc <- mem[pc];
                        pc =+ 1;
        15,[0,3],ON:
        15,5:           sc <- mem[pc+1]::mem[pc];
                        pc =+ 2;
        [0,14],4:
        [0,14]),6:      sc <- mem[er+1]::mem[er] + signext(mem[pc]);
                        pc =+ 1;
        15,4:
        15,6:           sc <- sp + signext(mem[pc]);
                        pc =+ 1;
        }
        (ind == ON) => decode(twobytes){
              OFF: sc <- mem[sc];
              ON:  sc <- mem[sc+1]::mem[sc];
              }
%0getdst        /* routine to put destination value in dst
                   leaving destination address in t16
                   see Appendix 2 for meanings of codes */
        t16 <- rp + (d << 1);
        decode(d,tmode,twobytes){
        [0,15],0,OFF:
        [0,15],[5,7],OFF:     dst <- mem[t16];
        [0,15],0,ON:
        [0,15],[5,7],ON:
        [0,14],1:
        [0,15],3:       dst <- mem[t16+1]::mem[t16];
        15,1:           dst <- mem[pc+1]::mem[pc];
                        pc =+ 2;
        [0,14],2:
        [0,14],4:       dst <- mem[t16+1]::mem[t16] + signext(mem[pc]);
                        pc =+ 1;
        15,2:
        15,4:           wr <- sp + signext(mem[pc]);
                        pc =+ 1;
        }
        (tmode >= 2 && tmode <= 4)=>{
              t16 <- dst;
              decode(twobytes){
              OFF: dst <- mem[t16];
              ON:  dst <- mem[t16+1]::mem[t16];
              }
        }
%0putdst        /* put wr in destination address */
        mem[t16] <- wr<9:16>;
        (twobytes == ON)=> mem[t16+1] <- wr<1:8>;
        (tmode == 3) => {
              er <- rp + (d << 1);
```

```
            decode(twobytes){
            OFF: t16 =+ 1;
            ON:  t16 =+ 2;
            }
            mem[er] <- t16<8:15>;
            mem[er+1] <- t16<0:7>;
      }
%0setcond      /* set condition codes from wr */
      ovfl <- 0;
      decode(twobytes){
      OFF: neg <- wr<9:9>;
            ((sc<8:8> == dst<8:8>) && (sc<8:8> != wr<9:9>)) => ovfl <- 1;
            carry <- wr<8:8>;
            decode(wr<9:16>){
            0:          zero <- 1;
            !0:         zero <- 0;
            0377:       ones <- 1;
            !0377:      ones <- 0;
            }
            odd <- wr<16:16>;
      ON:  neg <- wr<1:1>;
            decode(wr<1:16>){
            0:    zero <- 1;
            !0:   zero <- 0;
            }
            ((sc<0:0> == dst<0:0>) && (sc<0:0> != wr<1:1>)) => ovfl <- 1;
            carry <- wr<0:0>;
      }
%%
long signext(byte)
long byte; {
      if(byte & 0200)return(byte|0177400L);
      return(byte);
}
dump(){
      /* print registers */
}
```