Simulation of a General-purpose Micro-programmable Computer

by

TM-25

Elis Lieuson Samuel Chanson
Kevin Douglas Mabo Ito
Bary Pollack John Peck
Cindy Chan

SGMC User Manual

1980 January 21

Department of Computer Science
The University of British Columbia
Vancouver, British Columbia V6T 1W5

Abstract

Documentation for the Simulator, Monitor, and Assembler of a General-purpose Micro-programmable Computer (SGMC).

Acknowledgement.
The original design of SGMC is due to S. T. Chanson, M. R. Ito and B. W. Pollack. Implementation started in the summer of 1977 by Kevin Douglas and completed in August, 1978 by Elis Lieuson under the supervision of S. T. Chanson. This work has been supported by the Computer Science Department of UBC and the YEPU project number 2106.03 of British Columbia. A further revision to the Manual and processor was made by John Peck and Cindy Cha in the summer of 1979.

TABLE OF CONTENTS

A. INTRODUCTION.

SGMC is a package containing:

1. the Simulator which simulates a general-purpose micro-programmable computer (It provides arithmetic and logical operations on 32-bit 2's complement integers. It also supports control storage, main storage, and I/O operations),
2. the Monitor which provides facilities to monitor microprogram execution, and
3. the Assembler which translates symbolic microprograms into object codes.

B. SIMULATOR

The simulated machine consists of:

1. the CENTRAL PROCESSING UNIT (CPU) which executes the microporgram. The CPU in turn consists of:
   a. 64 high-speed general-purpose registers,
   b. the ARITHMETIC-LOGIC UNIT (ALU) which performs operations on the high-speed registers,
   c. the SHIFTER which shifts the results of ALU operations,
   d. the MICROPROGRAM CONTROL UNIT (MCU) which conditionally alters flags, control storage, and microprogram flow, and
   e. the STORAGE CONTROL UNIT (SCU) which handles all MS accesses,

2. the CONTROL STORAGE (CS) where the object codes of the microprogram are stored, and

3. the MAIN STORAGE (MS), which has a slower cycle time than the CS. At any time, the microprogram may reset the word and byte sizes (1-32) and the MS addressing mode (word or byte addressing). SGMC keeps everything in 32-bit 2's complement form. The address 'n' (0<=n) refers to the (n+1)st word if the current addressing mode is by word; the (n+1)st byte if the mode is by byte. In a byte I/O operation, the left-most byte of the word referenced is used if the current addressing mode is by word.

B.1. MS AND I/O OPERATIONS.

MS and I/O device operations require more than 1 micro-instruction cycle to complete. Flags are provided to indicate the status of these operations. These flags enable the CPU to perform other operations in the mean time. The CPU should not refer to or alter any register used by the MS and device operations while they are in progress. YOUR MICROPROGRAM MAY NOT WORK PROPERLY IF YOU DO NOT TAKE THIS INTO ACCOUNT. Flags available are: memory done flag for MS operations; interrupt flag for all the I/O devices; and interrupt enabled flag which allows or disallows the interrupt flag to be set. If a MS or I/O operation is initiated before the last similar operation is completed, the CPU is halted until that operation is done.

The MS or I/O operations are done on the start of the:
1. 5th instruction after the command for the MS operations.
2. 10th instruction after the command for the CHAR_READER.
3. 10th instruction after the command for the CHAR_PRINTER.
4. 10th instruction after the command for the NUM_READER.
5. 10th instruction after the command for the NUM_PRINTER.

B.2. DEVICES.

1. Device number 5 is CHAR_READER which reads a character at a time to the given register.
2. Device number 6 is CHAR_PRINTER which writes a character at a time. A write 'eof' will terminate the program.
3. Device number 7 is NUM_READER which reads in a decimal integer at a time.
4. Device number 8 is NUM_PRINTER which writes a decimal integer at a time.
5. Initially, the interrupt enabled flag is off, all devices' done flags are on and the busy flags are off. The interrupt flag is set whenever a device done flag and the interrupt enabled flag are both on. Warning: if a device done flag is altered by a CNTL field operation, the interrupt flag will no longer work correctly.

B.2. DEVICES.

B.3. MICRO-INSTRUCTIONS

All arithmetic operations are done in 2's Complement integers. The fields in a micro-instruction are executed from left to right. The shifter and destination sizes are determined by the opcode.

i. NOTATION.

<X> = The contents of register X.

"X" = The effective value of X with respect to the format of X. See A and AFMT fields for examples.

X* = The register pair [X,X+1].

X$ = The register triple [X,X+1,X+2].

(X,Y) = The value of the concatenated fields X & Y.

ii. REGISTERS.

FLAGS are the status flags (see FMSK field). The arithmetic flags: L, C1, C2, OV, M, P, & EV are affected by: ADD, SUB, AND, OR, XOR, CUM, & SHIFTER. The MD flag is affected only by the MS operations.

r0-r63 are 32-bit general-purpose circularly linked registers. r0 always contains zero. When r0 is the destination of an instruction, no destination is assumed. r0 to r63 are initialized to 0 to 63 respectively when the simulator is started.

RBASE contains a register number which is the user designated target machine's r0 (see AFMT, BFMT, DFMT fields).

PC is the CS program counter. If the current instruction contains a long immediate register format field, the PC is increased by 2. Otherwise, it is increased by 1.

0. OPCODE.GROUP.

OP ALU operation code. Mnemonics are given in upper-case.
= 0 = NOP: no operation.
= 1 = ADD: put "A" + "B" in the shifter.
= 2 = SUB: put "B" - "A" in the shifter.
= 3 = MPY: multiply "A" by "B" and place the double length result in the shifter.
= 4 = DIV: put "A"* / "B" in the shifter. The result is two single-length integers given as a double length result (remainder, quotient). The OV flag is set if the quotient is too large.
= 5 = AND: put "A" AND (logical bitwise) "B" in the shifter.
= 6 = OR : put "A" OR "B" in the shifter.
= 7 = XOR: put "A" EXCLUSIVE-OR "B" in the shifter.
= 8 = CPY: place "A" in the shifter.
= 9 = CPD: place "A"* in the shifter.
=10 = CUM: collapse under mask. The bits of "A" corresponding to 1's bits in "B" are placed in the lower order bits of the shifter. The higher order bits are cleared.
=11 = OUT: initiate output operation.
=12 = IN : initiate input operation.
=13 = TIO: test I/O operation status.
=14 = SSM: set the system MS definition (i.e., set the byte and word size, and indicate byte or word addressing). The size must range from 1 to 32 bits long.
=15 = HLT: stop the CPU.
Note: if the value of "A" to be used in a DIV or CPD operation is a short immediate value (from -32 to +31), then an "*" must append "A" to indicate that its long immediate equivalent should be used in the operation.

1. MS_INFO.GROUP.
(be careful with the timing!)

MSBY indicates that the MS is addressed BY: 0=byte 1=word.

MSBS is the MS Byte Size. The size must range from 1 to 32 bits long. The actual field value is equal to the size less 1.

MSWS is the MS Word Size. The size must range from 1 to 32 bits long. The actual field value is equal to the size less 1.

2. OPERANDS.GROUP.

A is the First operand definition (i.e, the register field).

AFMT is the Format of A (determines the value of "A").
= 0 = <rA>.
= 1 = ~<rA>.
= 2 = -<rA>.
= 3 = the value in bits 0-31 of the next micro-word.
= 4 = A. The result ranges from -32 to +31.
= 5 = <r(<rA>+<RBASE>)>.
= 6 = ~<r(<rA>+<RBASE>)>.
= 7 = -<r(<rA>+<RBASE>)>.

B is the Second operand definition (i.e, the register field).

BFMT is the Format of B. Interpretation is the same as AFMT except for code 3, where bits 32-63 of the next micro-word are used instead.

3. SHIFT OP.GROUP.

SHFT indicates the number of bits to shift (0-63) (see SIMM).

SIMM indicates to shift by: (0=<rSHFT>, 1= SHFT) amount.

STYP indicates that the shift operation is:
0=no shift, 1=logical, 2=arithmetic, 3=rotational.

SDIR indicates to shift to the: (0=left, 1=right).

SLNK indicates wether to include(if 1) or not(if 0) the link during a shift (does not apply to arithmetic shifts).

SFIL specifies the bit supplied to vacated positions (applies only to logical shifts).

SCOM Shifter output: (0=as is, 1=1's complement the result).

D is the destination register definition.

DFMT is the format of D.
= 0 = rD or rD*.
= 1 = r(<rD>+<RBASE>) or r(<rD>+<RBASE>)*.

4. ARITH FMSK.GROUP.

FMSK is the Flag MaSK. A bit 1 indicates the corresponding flag is selected.

- L = link - same as C1 but can be altered in shift.
- C1 = primary -logical high bit- carry.
- C2 = secondary -BCD (low 4 bits)- carry.
- OV = overflow (arithmetic).
- Z = zero: result is zero.
- M = minus: result is negative.
- P = plus: result is positive.
- EV = even: result is even.
- IR = interrupt requested.
- MD = memory done.
- U0 = user-programmable flag 0.
- U1 = user-programmable flag 1.
- U2 = user-programmable flag 2.

5. I/O OP.GROUP.

(be careful with the timing!)

DEV is the device number.

DCS is the Data, or Command, or Status register. The interpretation depends on the value of CTL. Command example: move disk arm to cylinder 5.

CTL is the ConTrol pulse Lines. Issues a CPU-DEVICE communication related command. At most one line can be on at any time.

- ADR = device ADdRess.
- RD = ReaD strobe (read into register "DCS").
- WR = WRite strobe (write from register "DCS").
- ST = put detailed device STatus in register "DCS".
- CD = execute the CommanD in register "DCS".
- IA = Interrupt Acknowledged.
- SCLR= clear the device's status flags.

6. I/O FMSK.GROUP.

IOF is the I/O flag mask.

- IR = interrupt requested.
- IE = interrupt enabled.
- D = device done (operation completed).
- B = device busy (operation underway).

7. CS_OP.GROUP.

FLOAD is the Flag Load control. (0=nop, 1=load FLAGS to r63).

LMD is the Link register, iMmediate operand, or micro-Data register. The interpretation is determined by CNTL.

COND is the branch/control Condition.
- = 0 = no operation.
- = 1 = unconditional.
- = 2 = true if at least one flag selected (see FMSK or IOF) is equal to the desired value (see PATN).
- = 3 = true if all the flags selected are equal to the desired values.

PATN indicates the flags bit values (Pattern) testing for are
- = 0 = 1's.
- = 1 = the bits corresponding to the lower order 13 bits of the MS_OP.GROUP field (the MS_OP.GROUP is not executed).

CNTL indicates the action to be performed when COND is true.
- = 0 = NOP: no operation.
- = 1 = SRB: set RBASE to LMD.
- = 2 = RCS: read control storage. <rLMD$> := micro-word at <rCSB> + <rCSX>.
- = 3 = WCS: write CS from <rLMD$> to <rCSB> + <rCSX>.
- = 4 = BAL: branch-and-link. No link done if LMD=0. <rLMD> := <PC>; <PC> := <rCSB> + <rCSX>.
- = 5 = RET: subroutine Return. <PC> := <rLMD>.
- = 6 = EXC: execute instruction at <rCSB> + <rCSX>.
- = 7 = EXL: execute single word instruction in <rLMD$>.
- = 8 = JMA: jump absolute to (CSB,CSX).
- = 9 = JMI: jump indexed to <rCSB> + <rCSX>.
- =10 = FF0: set the selected Flags (corresponding to the 1 bits in Fmask) to 0.
- =11 = FF1: set the selected Flags to 1.
- =12 = FFC: Complement the selected Flags.
- =13 = FL0: set the Flags corresponding to the 1 bits in <rLMD> to 0.
- =14 = FL1: set the Flags corresponding to the 1 bits in <rLMD> to 1.
- =15 = FLC: Copy from <rLMD> to the Flags.

Note: EXC & EXL execution is part of the current instruction cycle time.

CSB is the CS address Base register.

CSX is the CS address indeX register.

8. MS OP.GROUP.

MEMOP indicates the Main memory operation.
- = 0 = NOP: no operation.
- = 1 = RDB: read the left-most byte at location <rMA> to the lower order bits of the register specified by MD.
- = 2 = WRB: write the byte in the lower order bits of register MD to the left-most bits at location <rMA>.
- = 3 = RDW: read word. Addresses must be alligned on a word boundary.
- = 4 = WRW: write word. Addresses must be alligned on a word boundary.

MA is the MS Address register.

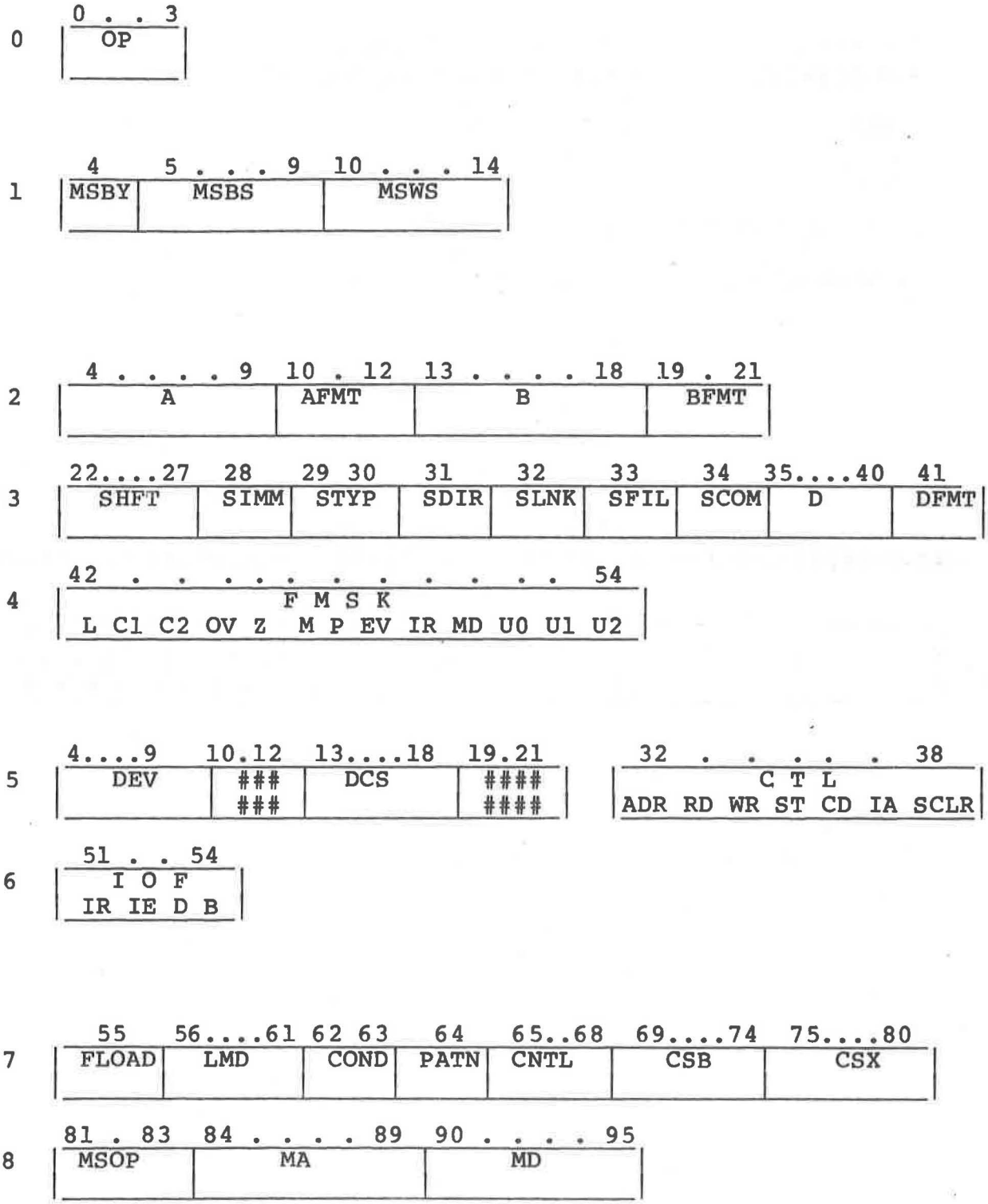
MD is the MS Data register.

B.4. REMARKS ON THE SHIFTER.

1. The number of relevant bits involved in the shift is determined by the length of the result of the current operation, the current MS word size, and whether L is included in the shift or not.
2. The number of bits shifted is "SHFT" mod the number of relevant bits.
3. The maximum number of relevant bits cannot exceed 64 bits. This restriction implies that when the MS word size is 32-bits, and the resultant size of an operation is a double word, then L cannot be involved in the shift.
4. If the SCOM field is set, it will be done only if the shift type is not 'no shift'.

B.4. REMARKS ON THE SHIFTER.

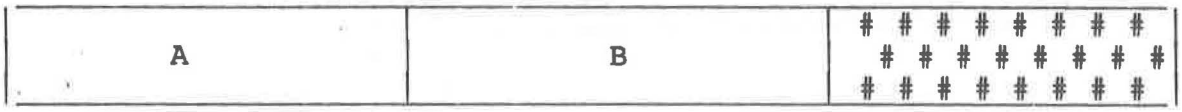
B.5. PICTURE OF INSTRUCTION FIELDS.



B.5. PICTURE OF INSTRUCTION FIELDS.

OPCODE:	CORRESPONDING GROUPS:
HLT	0
SSM	0, 1
TIO,OUT,IN	0, 5, 6, 7,8
(REMAINING)	0, 2,3, 4, 7,8

LONG IMMEDIATE FIELDS:



B.5. PICTURE OF INSTRUCTION FIELDS.

C. MONITOR

C.1. MONITOR COMMANDS.

Commands may be abbreviated by using only the characters underlined. Commands must be separated by ';' or end-of-line. For a numeric parameter (i.e, n, adr, n1, & n2) use a decimal or hex('#'-prefixed) integer, an identifier from the assembler code, or a combination of the above with '+' or '-' operators in between without blanks. Note: n1 <= n2. Addresses are displayed in base 10. Main storage, control storage and registers are displayed in hex. The time printed out is in the form: <number of micro-instruction cycles>.<number of CS instructions done>.

MTS returns to MTS without unloading the simulator.

$... executes a MTS command.

STOP stops the simulator.

QUIT same as STOP.

"EOF" signals error and stops the simulator immediately.

STEP reverses the step switch (initially off). When on, control is returned to the monitor after each micro-instruction execution.

GOTO n starts the CPU at the CS address n.

CONTINUE starts the CPU at the location indicated by the current <PC> (initially 0).

AT n begins an Atpoint definition at the CS address (see Atpoint commands).

RESTORE n
deletes the Atpoint definition previously set at n.

CLEAR deletes all the Atpoint definition currently set.

LOAD CS filename
loads from the MTS file called filename starting from location 0 (see the Assembler for the file format).

LOAD MS filename
: loads from filename to MS (starting from loc 0). The file format is a sequence of integers. The 1st integer, in decimal, specifies the number of 32-bit words to reserve for the MS. The following integers are in hex and are the values to be stored in MS, starting from location 0. Note: the current MS definition (which defaults to 8-bit byte, 32-bit word, byte-addressing) may be set by the SSM instruction. If the wordsize w is set to be < 32, then the left-hand 32 - w bits of MS are not accessed.

SET CS adr = n n n
: places (n n n), each n for each 32-bit part, into CS location adr.

SET MS adr = n
: places n into MS location adr.

SET REGISTER r = n
: places n into the register r.

DISPLAY SYSTEM
: displays FLAGS, PC, RBASE, MS definition, and the internal time.

DISPLAY CS n1 n2
: displays the CS from n1 to n2. Only n1 is shown if n2 is omitted.

DISPLAY MS n1 n2
: displays the MS from n1 to n2. Only n1 is shown if n2 is omitted.

DISPLAY REGISTER n1 n2
: displays the registers from n1 to n2. Only n1 is shown if n2 is omitted.

C.2. ATPOINT DEFINITION.

Atpoints are used to define a set of commands to be executed after the instruction at the given CS address is executed. The effect of an atpoint definition is to store and replace (by a 'STOP') the CS instruction. If a 'STOP' is encountered and the current CS address has an atpoint defintion, the corresponding stored instruction and definition are executed. Note: do not set an atpoint at a location that a EXC references.

Legal commands are:

DISPLAY same as the monitor DISPLAY commands.

STOP stops the system.

QUIT same as STOP above.

END ends the atpoint definition.

BREAK ends the definition, stops the CPU, and returns to the monitor.

D. ASSEMBLER.
The line numbers printed in the compilation are not MTS line numbers but the line position relative to the start of the file.

0. NOTATION:

1. [x y z]
   - ignore it or choose one.

2. {x y z}
   - one must be chosen.

3. <X>
   - the value of the 'variable' <X>.

4. <N>
   - a decimal integer or
   - a hex integer prefixed by the '#' character.

5. [x]
   - zero or more occurrence of x.

6. {{x y z}}
   - example statements to illustrate the syntax.

1. <STATEMENT>:

1. <ID> = { [% + -]<N> } [;]

2. [<UNIQUE_LABEL>:] [<INSTR>] [;]

{{ 1. COUNTER = R7; ZERO=R0
1. M_TWO = -2; MASK = #FFFF
2. AND MASK,%10 TO %10
2. LOOP: CPY ZERO TO COUNTER }}

2. <INSTR>: (fields order similar to their real positions)

1. HLT

2. SSM {BYTE WORD},<BYTESIZE 1-32>,<WORDSIZE>

3. {IN OUT TIO} <N>,%<N>,<CTL> [FLOAD] [<CSOP>] [<MSOP>]

4. [<NOP>] [FLOAD] [<CSOP>] [<MSOP>]

5. <REM> <OPRNDS> [<SHIFT>] [<D>] [FLOAD] [<CSOP>] [<MSOP>]

```
{{  2.  SSM BYTE,8,16
    3.  IN  5,INPUT_CHAR,RD
    3.  TIO 8,,ST IF (D) JMA LOOP
    5.  ADD 1,%2 TO %2
    5.  OR  MASK,PATTERN SRL 4 IF (EV) JMA EVEN
    5.  CPD 0* TO LONG.ZERO  }}
```

3. <REM>:

   1. {ADD SUB MPY DIV AND OR XOR CPY CPD CUM}

4. <OPRNDS>:

   1. [~ -][%]<N>[*] [,[~ -][%]<N>[*]]

   2. If an operand refers to a label or contains an integer too large to fit in the register field, the long immediate format is used.

```
{{  1.  ~MASK,PATTERN
    1.  10,-10
    1.  R5*,R6*
    2.  #FFFF,LABEL1  }}
```

5. <SHIFT>:

   1. <SHIFT-OP> [, ([0 1][,][L])] [%]<N>
      - 0 or 1 specifies the filler bit.
      - L specifies link inclusion.

```
{{  1.  SRL (1,L) R6
    1.  SRRC 0
    1.  SLA TWO  }}
```

6. <SHIFT-OP>:

   1. S{L R}{L A R}[C]
      - {L R} = (Left Right)
      - {L A R} = (Logical Arithmetic Rotational)
      - [C] = (Complement)

D. ASSEMBLER.

7. <D>:

   1. TO %<N>[*]

   {{ 1. TO SUM
      1. TO %6
      1. TO R30* }}

8. <CSOP>:

   1. [{<IF> WITH} (<FLAGS>)] {<NOP-FF> <JMA> <REMAIN>}

   {{ 1. IF (Z) JMA FETCH
      1. IF &(OV, P) JMA ERROR
      1. WITH (M) FFC }}

9. <IF>:

   1. IF [| &]

10. <FLAGS>:

   1. The list of the appropriate flags, separated by commas, that one is interested in. In the case of 'IF', each flag can be preceeded by '~' to indicate testing for 0. In this case, PATN=1 and no MS-OP can appear in the instruction.

11. <NOP-FF>:

   1. {NOP7 FF0 FF1 FFC}

12. <JMA>:

   1. Jma {<LABEL> [+ -]<N>}
      - if [+ -]<N> is used, the address is <PC> [+ -] <N>.

   {{ 1. JMA LOOP
      1. JMA -1 }}

D. ASSEMBLER.

13. <REMAIN>:

   1. <REST-CNTL> %<N>,%<N>,%<N>
      - the registers are for the fields: LMD, CSB, CSX.

   {{ 1. SRB 10,0,0
      1. BAL RET_ADDRS,SUBR_ADDRS,0
      1. RET RET_ADDRS,0,0
      1. JMI 0,BASE_ADDRS,INDEX
      1. EXC 0,SET.FLAG,0 }}

14. <REST-CNTL>:

   1. {SRB RCS WCS BAL RET EXC EXL JMI FL0 FL1 FLC}

15. <MSOP>:

   1. NOP8

   2. {RDB WRB RDW WRW} %<N>,%<N>
      - the registers are for the fields: MA & MD.

   {{ 2. RDW PROG_COUNTER,INSTR_REG
      2. WRB SAVE_AREA,ONE.BYTE }}

16. <NOP-S>:

   1. Since 'NOP' can appear in more than 1 type of operation field, the nearest operation field is taken whenever it is used. To specify the field explicitly, append the field's group number to 'NOP'. Eg. 'NOP3' for the shift operation. Since operation fields defaults to 'NOP', this restriction should not inconvenience anyone.

D. ASSEMBLER.

17. <ID>:

1. A string of alphanumeric, '_', '.', and '$' characters, starting with a non-numeric char. The maximum length is 255 characters long.

2. May appear where <LABEL>, <N>, <KEYWORD> can appear, as long as the <ID-TYPE> is correct.

3. <ID-TYPE> is any one of the following:
Keyword, Register, Integer, or Address.

4. <KEYWORD> is any one of the mnemonic codes (eg. NOP, ADD, CLR, BYTE, IF, JMA, ...)

5. R0-R63 identifiers are initialized to be registers 0 to 63.

6. Scope is the entire program.

7. Assignment operators are '=' for general assignment and ':' for current address assignment. A <KEYWORD> may not be redefined as another <ID-TYPE>. All other identifiers may be assigned any value and type at any time by general assignment. In current address assignment, an identifier (i.e, a statement label) may not be assigned an address value more than once.

8. JMA forward references of labels are resolved at the end of the assembly!!!

18. REMARKS:

1. ';' terminates a statement.

2. '%' indicates that the integer that follows it is a register number.

3. '*' indicates long immediate when used with an integer, and indicates indirection when used with a register number.

{{
```
1.  NUM=R6; CPY NUM TO R1; ADD R1,NUM TO R1
2.  CPY %2 TO %4
3.  TEN=10; CPD TEN*
3.  DIV 20*,R5
3.  ADD R1*,R2* TO R1*
```
}}

D. ASSEMBLER.

19. SPACES AND NEWLINES:

1. Can appear anywhere except inside an <ID>'s string name.

20. COMMENTS:

1. Can appear where a space can.
2. Anything between '/*' and '*/'.
3. Anything from '/'(not followed by '*') to end of line.

21. OBJECT MODULE: (as produced by the assembler)

1. Headed by the control storage size in decimal,
2. followed by the symbol table (where A=address, R=register, and <VALUE> in hex),
   <STRING> {A R} <STRING-VALUE>
3. followed by a zero,
4. followed by the CS word values formated in 3 32-bit word per CS word. Each 32-bit word value is in hex.

22. DEFAULTS: (those values that indicate an empty state)

1. For register fields, register 0 is used.
2. For operation fields, no operation.
3. For switches fields (eg. Link inclusion in shift) the default is 'off'.

23. ERRORS:

1. The error flagged is in the statement that is on or 'just' before the line indicated by the line number.
2. Undefined identifiers are not handled correctly under certain circumstances. Eg. When the name of the destination register in a 'TO' clause is undefined. If an error message does not make sense to you, check and see if an identifier is undefined.

D. ASSEMBLER.

E. HOW TO USE.

To run the assembler:

```
$RUN MCRO:ASM SCARDS=sourcef SPUNCH=objectf SPRINT=listingf T=2
```

- SPUNCH defaults to -LOAD.
- sourcef should not be *source* or *msource*!!!
- SPRINT defaults to *dummy* on terminal, to *msink* on batch.

To run the simulator:

```
$RUN MCRO:SIM SCARDS=commandf
   5=char-reader-f       6=char-printer-f
   7=num-reader-f        8=num-printer-f     T=2
```

- commandf file contains the monitor commands to be executed.
- char-reader-f defaults to MTS GUSER unit (*source*).
- char-printer-f defaults to MTS SERCOM unit (*sink*).
- num-reader-f defaults to MTS GUSER unit.
- num-printer-f defaults to MTS SERCOM unit.
- avoid having any 2 files being the same.
- r0 to r63 are initialized to 0 to 63 respectively.

- a typical commandf file would contain:

```
LOAD MAIN STORE ms-file
LOAD CONTROL STORE cs-file
....other monitor commands (eg.  GOTO 0)...
STOP
```

- results of the on-going printer operations are not seen if the simulator is terminated before the operations are completed.

It is always recommended to put a time constraint on all $RUN commands. Since this system has just been completed, it may not be error-free. The time constraint would avoid possible infinite loops due to the system or your program.

CONTENTS