A macromodular systems simulator (MS2) *

by RICHARD A. DAMMMKOELER
Washington University
St. Louis, Missouri

INTRODUCTION
The macromodular systems simulator is a control program and programming language implemented at Washington University to facilitate the design and subsequent realization of macromodular computer systems. The MS2 control program and language enable an engineer or programmer to describe macromodular systems and run programs on such systems simulated on a conventional digital computer (IBM 360/50). Input to the simulator consists of a standard set of function definitions for the basic macromodules, a description of the organization of the target machine, the program to be executed by the target machine, and data required by that program. Output from the present version of MS2 is a level change map (a continuous trace of the internal control network of the target machine as it executes its program) and the results of computations performed by the target machine. A procedure for preparing wiring tables and diagrams has been developed and will be incorporated into an improved version of the simulator.

The MS2 programming language
The MS2 programming language is, to the user, a symbolic system similar to those designed for assembly language programming with conventional machines. This correspondence was purposely maintained in order to provide a simple transition from programming to systems design. There are seven classes of MS2 commands, including the pseudocommands which provide the capability necessary for external communication between the designer and his simulated macromodular machine.

In the following discussion the Greek symbol $\alpha$ will be used to designate an alphabetic or alphanumeric name assigned by the programmer or designer to a data module. The symbol $\beta$ will designate a control module. Subscripts on either symbol indicate its position on an argument list.

A single data module has 12 binary positions which may be represented by either of two conventions:
1. $\beta_{c1} \beta_{c10} \beta_{c20} \ldots \beta_{c12} \beta_{c20}$, where the subscripts correspond to the associated power of two, or
2. $X_{c1} X_{c2} X_{c3} \ldots X_{c11} X_{c12},$ where the subscripts correspond to the bit ordering from left to right.

Register commands
This class of commands defines operations usually performed on a register stack or between two register stacks.

- **CLEAR,$\alpha$**: The contents of $\alpha$ are set to zero by this command.
- **INDEX,$\alpha$**: This command causes the contents of register $\alpha$ to be increased by 1.
- **COMP,$\alpha$**: This command replaces the contents of register $\alpha$ with its one's complement.
- **SHFTRA,$\alpha$**: This command causes the contents of macromodule $\alpha_1$ to be transmitted to $\alpha_2$.
- **SHFTLZ,$\alpha$**: This implies that the contents of $\alpha_1$ is added to $\alpha_2$ and the result placed in $\alpha_2$. Basic 2's complement arithmetic is used.
- **SHFTRLZ,$\alpha$**: Reverse of SHFTRA.
- **SHFTRZ,$\alpha$**: The contents of $\alpha$ are shifted one bit to the right. A zero is inserted in the high order position and the low order bit is lost.
- **SHFTLA,$\alpha$**: Reverse of SHFTRZ.
- **SHFTRO,$\alpha$**: Same as SHFTRZ but a one

*This research was supported in part by the Advanced Research Projects Agency of the Department of Defense through contract SD-302 and by the Division of Research Facilities and Resources of the National Institutes of Health through grant FR-00218.
is inserted in the high order position.

**SHFTLO, α**: Reverse of SHFTRO.

**SHFTRC, α**: One position right shift with original high order position held constant.

**SHFTLC, α**: Reverse of SHFTRC.

**SHFTRD, α₁, α₂**: One position right shift with high order position of α₁ replaced by the low order position of α₂, an attached data cable. α₂ is unchanged.

**SHFTLD, α₁, α₂**: Reverse of SHFTRD. The low order position of α₁ is replaced by the high order position of α₂, an attached data cable.

**Storage access commands**

Two commands, READ and WRITE provide the facility for accessing the macro modular storage unit. Associated with each 4096 word bank of storage is a single 12-bit address register. The name of this register must appear as the second (α₂) operand of the storage access commands, as it specifies which memory bank is to be used.

**READ, α₁, α₂**: This command causes the contents of the location specified by α₂ to be placed in or on data module α₁.

**WRITE, α₁, α₂**: The WRITE command causes the contents of α₁ to be placed into storage at the location specified by α₂.

**Screwdriver commands**

Certain operations, normally performed manually, can be specified in the MS2 language. Switch settings on parameter units, and formats for the outputs of junction units are established through the use of the following commands. For example, the statement:

**PARAMETER * α, 001000110001**

defines a parameter unit, α, with the setting shown as the second argument. If a parameter unit is to be used as a detector setting, X's may be inserted in the second argument to indicate the 'don't care' condition. An example is shown below:

**DETECTOR * α, XXXX1001XXX1**

Junction units have two 12-bit inputs and one 12-bit output which may be a composite of the inputs and pre-set constants supplied by the junction unit. For example:

JUNCTION * α₁, α₂, X<sub>c</sub><sub>1</sub>, X<sub>c</sub><sub>2</sub>, ... 101 ... Y<sub>c</sub><sub>1</sub>, Y<sub>c</sub><sub>2</sub>, α₃*

defines a junction unit output α₁ with inputs from α₂ and α₃ where the X<sub>c</sub> bits are from α₂ and the Y<sub>c</sub> bits are from α₃. An alternative specification using the power of 2 convention would be

JUNCTION * α₁, α₂, Xβ<sub>c</sub><sub>1</sub>, Xβ<sub>c</sub><sub>2</sub>, ... 101 ... Yβ<sub>c</sub><sub>1</sub>, Yβ<sub>c</sub><sub>2</sub>, α₃*

Junction units may be activated by inserting the name of a previously defined unit as the first argument of the GATE command. For example, if T₁ is the name of a junction unit output, the statement

**GATE, T₁, ACC**

will place the current value of T₁ in or on data module ACC.

**Conditional commands**

Modules with settings or parameters defined previously may be interrogated by the following MS2 statements:

**DETECT, α₁, α₂, β₁, β₂**

where α₁ = name of the module being detected

α₂ = name of previously defined detector

β₁ = control path for TRUE response

β₂ = control path for FALSE response

A decoder may be interrogated by a statement such as

**DECODE, α₁, α₂, β₀, β₁, β₂, β₃ ... β₇**

where α₁ = bits to be decoded written in the form

β<sub>1=1</sub>, β<sub>1=0</sub>, β<sub>2=1</sub>, β<sub>2=0</sub>, or X<sub>c</sub><sub>1</sub>, X<sub>c</sub><sub>2</sub>, X<sub>c</sub><sub>3</sub>

α₂ = name of the data module

(note: this may not be a multiple length module)

β₀ = control path to be taken if the bit pattern (α₁) = 000

β₁ = control path to be taken if the bit pattern (α₁) = 001

... An overflow condition on an adder may be detected by the statement

**ADDOV, α₁, β₁, β₂**
where \( \alpha_i \) is the name of the register stack containing the adder, \( \beta_1 \) is the TRUE (OVERFLOW) return and \( \beta_2 \), the FALSE (NO OVERFLOW) return. Use of the ADDOV command does not reset the overflow indicator.

The contents of two registers may be compared by attaching the data output cable of one register to the data terminal of a detector positioned on a second register. The following MS2 command is used for this operation:

\[
\text{COMPARE, } \alpha_1, \alpha_2, \beta_1, \beta_2
\]

where \( \alpha_1 \) = name of register on which the detector is positioned
\( \alpha_2 \) = name of the register providing detector input
\( \beta_1 \) = TRUE control branch
\( \beta_2 \) = FALSE control branch

Logic function commands
This class of commands specify operations performed in a register stack by the macromodular function unit.

BITSET, \( \alpha_1, \alpha_2 \): The bit pattern of register \( \alpha_2 \) is modified by the pattern of \( \alpha_1 \) using OR logic. Result in \( \alpha_2 \).

BITCLR, \( \alpha_1, \alpha_2 \): If the input bit from \( \alpha_1 \) is zero, the corresponding bit in \( \alpha_2 \) is unchanged. If the input bit from \( \alpha_1 \) is one, the corresponding bit in \( \alpha_2 \) is set to zero.

BITCOMP, \( \alpha_1, \alpha_2 \): If the input bit from \( \alpha_1 \) is zero, the corresponding bit in \( \alpha_2 \) is unchanged. If the input bit from \( \alpha_1 \) is one, the corresponding bit in \( \alpha_2 \) is complemented.

AND, \( \alpha_1, \alpha_2 \): Perform an AND operation between \( \alpha_1 \) and \( \alpha_2 \) bit patterns.

Control commands
Additional commands for the control macromodules are

MERGE, \( \beta_1 \), : which implies the merging of control at point \( \beta_1 \), and

CALL, \( \beta_1, \beta_2 \), : where \( \beta_1 \) is the point at which a reentrant control string begins, and \( \beta_2 \) is the point to which control will be passed following the execution of that control string.

If a reentrant control string contains more than one exit point, (e.g., at detector or other decision module) the decision call macromodule should be used. The simulator command format is

\[
\text{DCALL, } \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6
\]

where as before \( \beta_1 \) specifies the beginning of a control string, and \( \beta_2 \) and \( \beta_3 \) specify the true and false branch points.

A control string may be split into two independent strings by the use of the simulator command

\[
\text{SPLIT, } \beta_1, \beta_2
\]

which specifies that two functions are to be initiated simultaneously; one at a module labeled \( \beta_1 \) and the second at \( \beta_2 \).

Any two control strings can be re-joined using the macromodular rendezvous unit simulated by a command of the following form

\[
\text{WAIT, } \beta_1, \beta_2
\]

where \( \beta_1 \) is the name of a particular rendezvous unit and \( \beta_2 \) is the point to which control will pass after two control strings have been rejoined.

Pseudo-commands
Six pseudo-commands have been implemented in order to facilitate the use of MS2. They are:

\text{INPUT, } \alpha \quad \text{: The INPUT command causes the next octal value, delimited by an apostrophe, to be read from the input stream and placed in data module } \alpha.

\text{OUTPUT, } \alpha \quad \text{: The OUTPUT command prints the contents of data module } \alpha \text{ in octal format.}

\text{RETRN, } \alpha \quad \text{: The RETRN command passes control from the target machine to the MS2 control program. The argument } \alpha \text{ specifies a data module (usually the instruction counter), the contents of which will be printed at the time the return is executed.}

\text{TRUE} \quad \text{: The TRUE command has no arguments as it is used as an argument for the DCALL command when a decision call unit is imbedded in a re-entrant control string called by other target machine strings.}

\text{FALSE} \quad \text{: A command used to identify the FALSE return from a decision}
call unit imbedded in a re-entrant control string.

LOCK, \( \beta_1, \beta_2 \): The LOCK command is a pseudo command which marks the position of a priority selector. The selector module interlocks a control beginning at point \( \beta_1 \) and the second argument \( \beta_2 \) specifies the point to which control passes after the \( \beta_1 \) string has been executed and the interlock released.

Multiple length specification

The user may define and use multiple length data modules by including a slash (/) in argument strings of the Register, Memory Access, Conditional, and Logic Function commands. For example, the MS2 statement:

\[
\text{GATE,A1/A2/A3,BX/BY/BZ}
\]

specifies the gating of a 36-bit data module, with twelve bit sections A1, A2, and A3, to another 36-bit data module, consisting of sections BX, BY, and BZ. Up to 9 twelve bit sections may be coupled with this convention.

The user may ‘assemble’ multiple length detectors, junction units and parameter units defined by

\[
\text{PARAMETER*A1,100011001110}
\]
\[
\text{PARAMETER*A2,100000001110}
\]

to create a 24-bit parameter unit consisting of two 12-bit sections A1 and A2. These sections may be used independently or as a single 24-bit unit as shown in the following example:

\[
\text{TEST* COMPARE, A1,M, NEXT, NOT}
\]
\[
\text{NEXT* COMPARE, A1/A2, ACC1/ACC2, GO, NOGO}
\]

The MS2 control program

Implementation

The simulator control program is implemented in TRAC, a string processing language, selected because of the similarity between the macromodular control network and Mooers’ concept of an active string. But using a combination of TRAC primitives, it is possible to exactly duplicate the functional characteristics of most of the present set of macromodules. The exceptions are the control branch, a macromodule used to initiate parallel control networks, and the interlock module which functions as a priority selector at the intersection of multiple control paths. As parallel operations must be simulated sequentially on a serial machine, the functions of these two modules are handled internally by the simulator control program. However, in order to preserve the validity of the level change map and wiring tables, both control branch and interlock macromodules can be included in the description of the target machine.

Statements describing the internal organization of a computer to be simulated by MS2 are prepared in free-form format using an asterisk (*) as a delimiter for labels of unique control points (control paths) and the character ’ (an apostrophe) to mark the end of each statement. They are read by the control program, analyzed syntactically, and converted to active TRAC strings prior to execution. Unlabelled statements are concatenated with the preceding labelled statement producing a sequential string corresponding to a single control path in the target machine. A complete machine definition consists of one or more such strings, existing as separate within primary core. Linkages between strings are completed at execution time by the control commands described in Section II-F.

TRAC definitions of the basic macromodular functions are also stored internally as active strings as shown in Table I. A complete listing is given in Technical Report No. 18. During execution, an active target machine string initiates requests for the macromodular functions in the sequence indicated by the MS2 machine description. Arguments contained in the target machine string replace the special characters \( \rightarrow_i \) marking the position of the \( i \)th argument, and the resulting string is evaluated by the control program. An example of the ‘evolution’ of an active MS2 string from input through execution is shown in Table II.

The current version of the control program was implemented in BAL, the basic assembly language for the IBM 360 series, and is run under a modified R.A.C.S. (Remote Access Computing System) which provides for conversational interaction between the 360/50 and remote consoles. During MS2 runs, the remote console is used as the operator’s console for the target machine, or alternatively as a reader/printer/punch station.

Table I. Some simple TRAC definitions of macromodular functions

\[
\#(\text{DS,CLEAR}, \#(\text{DS}, \rightarrow_1, 0000))
\]
\[
\#(\text{DS,GATE}, \#(\text{DS}, \rightarrow_2, \#(\text{CL}, \rightarrow_1)))
\]
A Macromodular Systems Simulator (MS2) * 375

for convenience, use the macromodular function definitions as well. Present control program commands are:

**INITIAL */*α** This command calls the initialization routine which edits and analyzes all MS2 statements. $α$ is an arbitrary identification parameter.

**DEFINEBLOCK */*β** This command defines a block $β$ consisting of strings $β_1$ through $β_n$, $n \leq 31$.

**STOREBLOCK */*β** This command stores a block $β$ in disk storage.

**FETCHBLOCK */*β** This command retrieves a block of strings $β$ from the disk and re-establishes the definitions of its constituent strings $β_1$ through $β_n$.

**MAP */*α** This command activates the mechanism for tracing the internal control of a target machine. $α$ is an arbitrary identifier.

**NOMAP */*α** Use of this command deactivates the internal control mapping mechanism.

**β */*α** A command of this form passes control to any previously defined string $β$.

### DISCUSSION

The simulator has been used to evaluate a number of macromodular realizations of specialized computer designs including a Valgol II machine* and the compiler-compiler machine* presented in the following paper. Its principal advantages are related to the ease with which a programmer may describe the organization of his target machine, and the flexibility of implementation which allows additional macromodular functions to be specified as active TRAC strings within the existing MS2 environment. This latter capability has already been of proven value, in that it provided a mechanism for maintaining a correspondence between the evolving macromodular technology and concurrent efforts to develop software for target machines.

---

Table I. Evolution of an active MS2 string

<table>
<thead>
<tr>
<th>Structural form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#(DS,GATE,(#(DS,$→_2$,##(CL,$→_1$))...) )</td>
<td>TRAC definition of the gate macromodule</td>
</tr>
<tr>
<td>START*GATE,P,S'</td>
<td>Input form of MS2 target machine machine description</td>
</tr>
<tr>
<td>#(DS,START,(#(CL, GATE,P,S)) )</td>
<td>Internal form of MS2 statement</td>
</tr>
<tr>
<td>#(CL,GATE,P,S)</td>
<td>Result of initialization of target machine by #(CL,START)</td>
</tr>
<tr>
<td>#(DS,S,##(CL,P))</td>
<td>Result of #(CL, GATE,P,S)</td>
</tr>
</tbody>
</table>

At the beginning of a simulation run approximately 180K bytes of storage are available for the target machine description. The remaining 76K bytes are used by the operating system (~72K) and the control program. The augmented TRAC system, an integral part of the control program, occupies less than twenty-five hundred bytes of core. Storage of target machine data, programs and results is managed by the control program in order to conserve core storage in the 360/50. Such information is stored in octal representation with 8 octal digits occupying 1 word (4 bytes) of 2 μsec core. Packing and unpacking are performed by two new primitives, rm and wm (read memory and write memory), added to the TRAC function set.

**Control program commands**

The MS2 control program commands are defined in terms of the TRAC primitives, and in some cases,
Limited use of MS2 has been made in conjunction with computer design and programming courses offered by the department of Electrical Engineering and the department of Applied Mathematics and Computer Science. In these instances, MS2 descriptions of commercially available machines were developed by the students, and used to obtain solutions for problems assigned in class. Although this ability to verify the results of programs written for a variety of conventional machines has in practice been a valuable supplement to normal laboratory experience, it also demonstrates (indirectly) the design flexibility inherent in the macromodular concept, and the functional completeness of the existing set of processor modules.

REFERENCES

1 C N MOOERS  L P DEUTSCH
   TRAC, a text handling language
   Association for Computing Machinery Proceedings of the
   20th National Conference 229-246 1965

2 R A DAMMKOELER  R A COOK  A R RAWIZZA
   TRAC implementation of macromodular functions
   Technical Memorandum No 18 Computer Research Laboratory
   Washington University St Louis Missouri January 15
   1967

3 A R RAWIZZA
   The valgol II machine
   Masters Thesis Sever Institute of Technology Washington
   University St Louis Missouri 1967

4 W E BALL
   A macromodular meta-II machine
   Proceedings of the Spring Joint Computer Conference 1967