INTEROFFICE MEMORANDUM

M-1091

DATE November 22, 1960

SUBJECT M A C[1] - A Master Subroutine Control System For PDP-1

TO PDP Distribution List

FROM Richard Bennett

I. INTRODUCTION

MAC is a Master Control System for the PDP-1 Computer. Advantages of this sytem include:

1. Reduction of the number of memory registers required to write closed subroutines.
2. Recursive ability of subroutines using this system to call themselves (or other subroutines which may call the original subroutine).
3. Simplification in the writing of subroutines.

MAC requires 174 (octal) locations, including six useful constants, plus 24 (octal) registers of storage. In addition, storage is required for the preservation of subroutine returns and the protected storage of subroutines used recursively. The amount of this latter storage depends on the requirements of the user's subroutines.

MAC was made feasible by the inclusion of the "call" instruction (cal) in the PDP-1. This instruction is equivalent to the sequence of instructions

```
dac 100 (octal)
jsp 101 (octal)
```

(The jsp instruction saves the program location counter in the Accumulator and jumps to the location given in the address part of the instruction.) It should be noted that the address part of the cal instruction is not used by the computer.

MAC is normally located at register 100 (octal) and following. Its main entry point is at 101.

MAC interprets the address part of the cal instruction as the location of the pertinent subroutine. After performing certain preliminary functions, MAC jumps to this location.

Acknowledgment
1. The MAC program and this memordandum was supplied to DEC by Mr. Richard K. Bennett, of Data Processing, Inc., Wellesley, Mass.

The called subroutine, when finished, must return control to the proper point in MAC. The subroutine may also call upon MAC for certain functions before it is finished.

It is possible to call subroutines without employing MAC by using the instructions jsp and jda (Jump and Deposit Accumulator). Working outside the MAC system is normally done only when timing becomes important. Subroutines in and out of MAC are, of course, not compatible.

The characteristics and the rules for the use of MAC are described below. Section II covers the entry and exit of subroutines. The handling of calling-sequence parameters is described in Section III. Section IV discusses recursion and protected storage. The Appendix summarizes the available facilities and gives their execution time.

The facilities described in this memo are those available at the given date. It is anticipated that they may be extended somewhat at the cost of memory registers. Furthermore, in at least one case (dp3 in Section III) the execution time can be cut drastically at the cost of more registers.

This balance between facilities and storage space depends greatly on the application. MAC's design is biased heavily toward the side of storage conservation.

II. ENTRY TO AND EXIT FROM SUBROUTINES

A subroutine in the MAC system is called by executing the instruction

```
cal abc
```

where abc is the symbolic name (location) of the desired subroutine. This instruction effects any entry to MAC, which subsequently jumps to the subroutine abc.

When MAC enters this abc subroutine, the Accumulator (AC) contains the original contents of the AC when the cal abc instruction was executed. In addition, the contents of the AC is available at location 100.

Location 100 has been given the symbolic name mac. (Here mac may be thought of as the Master Accumulator.) It is suggested that mac be used, rather than 100, for it might be necessary to change this value at a future date.

The exit from the subroutine abc must be accomplished via MAC. It is possible to return to L+1, L+2, etc., where L is the location of the cal abc instruction. Furthermore, the AC at the time of the return may contain, at the choice of the coder, either the contents of mac or the AC contents at the time of exit.

To return to L+1 with the AC intact, the subroutine exit is

```
jmp ra1
```

If it is desired to have the AC contain the contents of mac when MAC returns control to L+1, the exit is

```
jmp rm1
```

Similarly, return may be made to L+2 or L+3 with exits

```
jmp ra2
jmp rm2
jmp ra3
jmp rm3
```

The r, of course, stands for return. The a stands for AC (at exit) preserved; the m for mac used to reset AC. The number is the return location, relative to L.

To return to a location beyond L+3, different MAC re-entry points are required. If the AC is to be preserved, the exit is

```
cal ran
```

If mac is to be used to restore the AC on return, the exit is

```
jsp rmn
```

In both cases the relative return location must be stored in the register following the exit. For example,

```
cal ran
6
```

would return control to L+6.

I. CALLING-SEQUENCE PARAMETERS

Quite frequently it is convenient to list subroutine parameters in registers following the one containing the subroutine call. MAC makes this information available to the called subroutine by a process called "displaying."

To conserve computer time, the calling-sequence parameters are not actually moved when they are "displayed." Instead, their locations are stored. The actual parameters are obtained by using the indirect address feature.

At location ipl (standing for "immediate" parameter at L+1), the location L+1 is stored in the address part of the register. The subroutine may use the parameter at L+1 by "deferring" through ipl. For example,

```
lac* ipl
```

would load the AC with the parameter at L+1. (Any other instruction, such as add, dac, etc., can be used instead of lac.)

Quite frequently the calling sequence parameter is the location of the desired value. For example, to multiply x by y, one might write

```
lac x
cal mpy
y
```

The y at L+1 (referenced to the cal) is the location of the variable y.

The value of y in this example is called a "remote" parameter, in contrast to the term "immediate" parameter, which applies when L+1 contains the desired value. MAC provides for remote parameters by storing the location L+1 in the address part of rpl (remote parameter, L+1), where rpl contains a "one" in its defer bit.

To use a remote parameter, a subroutine would defer through rpl. For example,

```
lac* rpl
```

would load the AC with the contents of the register whose location is given in the register at L+1. If the defer bit of the register at L+1 is set (i.e., a "one") the deferring will continue until a register is obtained which has its defer bit clear. Thus, L+1 may contain the location of the location of the value of the parameter, etc.

One calling-sequence parameter (L⌀1) is automatically displayed by MAC. To obtain more parameters, MAC must be re-entered. Three parameters are displayed by the instruction

```
jsp dp3
```

(The letters dp stand for display parameters.) The instruction

```
jsp dpn
```

followed by a number (not greater than 7) will display that number of parameters. For example,

```
jsp dpn
5
```

will display five calling sequence parameters. If more than seven subroutine parameters are required, their locations must be specified by means of pointers.

The displayed parameters are available through the same device described above for the L⌀1 parameter. The number 1 is simply replaced by the appropriate number (up to 7). For example, the L⌀2 parameter is available through ip2 and rp2.

When using displayed parameters in a subroutine, it should be remembered that a call to a lower level subroutine will destroy at least the L⌀1 parameter. Therefore, it may be necessary to redisplay the parameters on returning from a lower level subroutine. This convention has been adopted because automatic redisplay of the parameters would make excessive demands both on computer storage space and on computer time.

IV. RECURSION

There is a growing appreciation of the value of permitting a subroutine to call itself (or to call subroutines which may call the first subroutine). This operation is called recursion.

Subroutines having recursive ability have proved their value in both mathematical as well as logical-type programs. MAC provides the facilities which permit subroutines to be called recursively.

There are two basic requirements in recursion. The first is to protect the return location of a subroutine, and the second is to protect its storage.

These two requirements are separated as a matter of efficiency. If a subroutine's storage was saved before every call to another subroutine, and restored upon its return, then the two requirements could be combined. That is, the return location could be included as part of the subroutine's storage.

However, it is more efficient to adopt the convention that a subroutine must protect the storage of its caller, rather than its own. With this convention a subroutine may call, for example, ten lower-level subroutines (including itself) and yet need only once execute the saving and restoring operation. In contrast, if the subroutine were to protect its own storage, it would, in the above case, have executed the saving and restoring operations ten times.

The protection of a subroutine's return cannot be accomplished under the convention of subroutines protecting their caller's storage. However, since the return location is not needed until the end, the saving and restoring operation need only be executed once.

The first recursion requirement (protection of the return) is automatically handled by MAC. The second requirement (storage protection) is accommodated by MAC very easily.

If a subroutine requires protected storage, the saving operation is executed by MAC using the instruction

```
jsp spl
```

when one register of protected storage is required. (The sp stands for save protected storage.) By substituting 2 or 3 for the 1, two or three registers will be saved.

To save more than three registers, the instruction

```
jsp spn
```

followed by the desired number will save that number of registers.

For example,

```
jsp spn
5
```

will save five registers of protected storage. There will be a fixed maximum number of temporary storage registers per subroutine. The specific value of this number has not yet been decided upon.

The appropriate saving instruction should be executed by the subroutine <u>before</u> it uses the protected storage. To use protected storage, the subroutine simply addresses <u>ps1</u>, ps2, ps3, etc.

The restoring operation is automatically performed by MAC when the subroutine exits. The exits are identical to those of Section II- MAC remembers whether or not protected storage was used and how much.

APPENDIX

SUMMARY OF FACILITIES AND EXECUTION TIMES

A. Available constants

i0	0
i1	1
i3	3
i7	7
m77	77 (octal)
mip	770000 (octal)

B. MAC Operations

Instruction	Time*	Operation
cal x	95	Call subroutine x
jsp sp1	220	Save Protected Storage
jsp sp2	280	
jsp sp3	335	
jsp spn) n)	200+55n	n up to 5
jsp dp3	285**	Display Parameters
jsp dpn) n)	135+65n	n up to 7
jmp ra1	95***	Return to L+1; AC preserved
jmp rm1	85***	Return to L+1; AC from mac
jmp ra2	100***	Return to L+2; AC preserved
jmp rm2	95***	Return to L+2; AC from mac
jmp ra3	100***	Return to L+3; AC preserved
jmp rm3	95***	Return to L+3; AC from mac
cal ran) n)	285	Return to L+n; AC preserved; no limit on n
jsp rmn) n)	110	Return to L+n; AC from mac; no limit on n

C. Protected Storage Restoration

If Protected Storage was saved by a subroutine, additional time is required to restore this storage when the subroutine exits. This time is given by

$$85+55n$$

where n is the number of registers protected.

* Time in microseconds
** Could be reduced to 115 at cost of 13 (octal) registers
*** Additional time required if Protected Storage was saved (see C above)