THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 57, No. 6, July-August 1978
Printed in U. S. A.

UNIX Time-Sharing System:

A Support Environment for MAC-8 Systems

By H. D. ROVEGNO
(Manuscript received January 27, 1978)

An integrated software system based on the UNIX system has been developed for support of the Bell Laboratories 8-bit microprocessor, MAC-8. This paper discusses the UNIX influence on the MAC-8 project, the MAC-8 architecture, the software development and hardware prototyping system, and MAC-8 designer education.*

I. INTRODUCTION

Today's microprocessors perform functions similar to the equipment racks of yesterday. Microprocessor devices are causing a dramatic shift in the economics of computer-controlled systems: product costs and schedules are influenced more by the system architecture and support environment than by the cost or speed of the microprocessor itself. In recognition of this phenomenon, Bell Laboratories has recently introduced a complete set of development support tools based on the UNIX system for its 8-bit microprocessor, MAC-8.[1,2] This paper presents an overview of the MAC-8 architecture and development system.[3]

Development of a microprocessor-based application consists of two activities:

(*i*) Design, construction, and test of the application's hardware.
(*ii*) Design, construction, and test of the application's software.

* UNIX is a trademark of Bell Laboratories.

The development support system described here assists the application designers in both areas. For the hardware designers, a prototyping system that permits emulation as well as stand-alone monitoring of the application's hardware is provided. For the software designers, a high-level language (C), flexible linker-loader, and source-oriented symbolic debugging are supplied. The combination of these tools provides the application designer with a complete and integrated set of tools for system design.

II. WHY UNIX?

At the outset of the MAC-8 development, it was recognized that use of an embedded microprocessor would *increase* the complexity and the scope of applications rather than simply lowering their costs. The tools of the future were going to be programming, documentation, and simulation tools. Considered in this light, a UNIX[4,5] support environment was a natural choice. UNIX possessed many desirable attributes of a "host" environment by providing sophisticated tools for program development and documentation in a cost-effective and highly interactive system. There was already widespread use of the UNIX system not only as a development vehicle in the Business Information Systems Program,[6] but also as part of many "embedded" applications.

III. WHY C?

While the choices of host system and programming language(s) are conceptually independent, there is obvious merit in the consistency of languages and systems. The C language[7] was an obvious choice because it offers high-level programming features, yet also allows enough control of hardware resources to be used in the development of operating systems.

IV. MAC-8

The MAC-8 is a low-cost, single-chip, bus-structured, CMOS microprocessor, whose architecture (Fig. 1) was influenced by the C language. Its major features are:

(*i*) MAC-8 chip, packaged in a 40-pin DIP (dual in-line package), which measures 220×230 mils and uses over 7000 transistors. The chip combines the low power dissipation of CMOS with

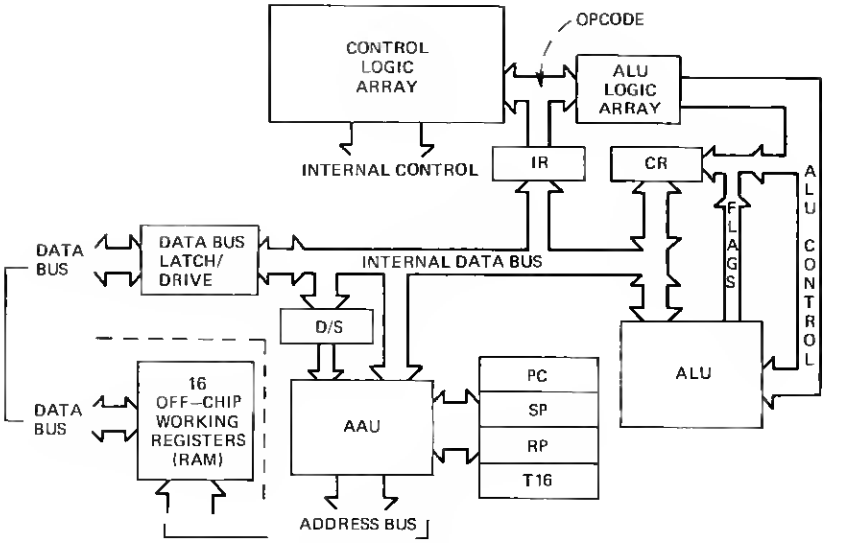

Fig. 1—MAC-8 block diagram.

the gate density of pseudo-NMOS (NMOS with a shared p-channel load transistor).

(*ii*) 16 registers in RAM (random access memory) that are pointed to by the register pointer (a MAC-8 on-chip register). Because of this, the full set of registers can be set aside and a new set "created" by executing only two MAC-8 instructions, which is particularly useful to compiler function-call protocol.

(*iii*) 65K bytes addressable memory space with DMA (direct memory access) capability.

(*iv*) Flexible addressing modes: register, indirect, direct, auto-increment, immediate, and indexed.

(*v*) Communication-oriented CPU (central processing unit) with a wide variety of 8- and 16-bit monadic and dyadic instructions, including arithmetic, logical, and bit-manipulation instructions.

(*vi*) Flexible branch, trap, and interrupt handling.

(*vii*) Processor status brought out to pins, which permits monitoring of CPU activity.

(*viii*) Internal or external clock.

Figure 1 is a block diagram of control. The major blocks are:

(*i*) *Control Logic Array* directs the CPU circuitry through the various states necessary to execute an instruction.

(*ii*) *ALU,* or Arithmetic Logic Unit, performs arithmetic and logical operations.

(*iii*) *ALU Logic Array* controls the operation of the ALU, managing the condition register (CR) flags.

(*iv*) *AAU,* or Address Arithmetic Unit, computes the address in parallel with the ALU operations.

(*v*) *Programmable registers* include the program counter (PC), stack pointer (SP), condition register (CR), and register pointer (RP).

(*vi*) *Internal registers* include instruction register (IR), destination/source register (D/S), and temporary storage register (T16).

V. DEVELOPMENT ENVIRONMENT

The MAC-8, development system (Fig. 2) is an integrated set of software tools, including a C compiler, a structured assembler, a flexible linking loader, a source-oriented simulator, and a source-oriented debugger. All the tools except the debugger reside on

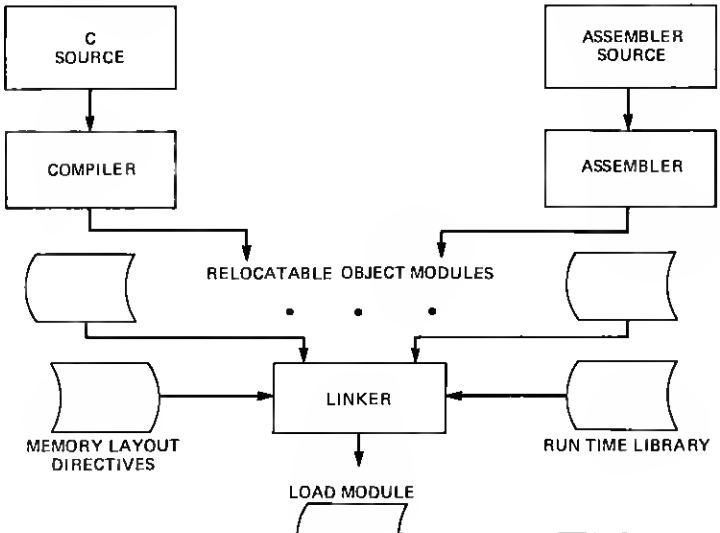

Fig. 2—MAC-8 development system.

UNIX; the debugger resides on a hardware prototyping system called PLAID (Program Logic Aid).

The following sections present a brief discussion of each of those tools.

There is a consistent user interface to all the tools that includes C syntax input language, UNIX file-oriented inter-tool communication, and names analogous to those of the corresponding UNIX tools, e.g., m8cc and m8as.

5.1 MAC-8 C compiler

The MAC-8 C compiler permits the construction of readable and modular programs, due to its structured programming constructs, high-level data aggregates, and a powerful set of operators. C is a good language for microprocessors[8] because it gives control over machine resources by use of primitive data types such as register, character, and pointer variables, and "machine-level" operators such as indirection, "address of," and post/pre- increment/decrement.

5.2 MAC-8 sssembler

The MAC-8 assembler is a conventional assembler in that it permits the use of all hardware instructions; it differs from conventional assemblers in the following ways:

(*i*) The language has C-like syntax as illustrated in Fig. 3. For

```
#define NBYTES 100
char  array[NBYTES];
/*
 * Calculates sum of array elements
 */
sum()
{
        b0 = &array;
        a1 = 0;
        for (a2 = 0; a2 < NBYTES; ++a2) {
                a1 =+ b0;
                ++b0;
        }
}
```

Fig. 3—MAC-8 assembler example.

example, a move instruction looks like a C-like assignment statement. Data layout is accomplished by C-like declarations.

(*ii*) The language has structured programming constructs (e.g., if-else, for, do, while, switch) that permit one to write readable, well-structured code at the assembly level. Each construct usually generates more than one machine instruction.

The reserved words in the language identify the MAC-8 registers and also include many of the reserved words of the C language. The **#define** and **#include**, as well as the other features of the C preprocessor, are supported by the assembler.

5.3 MAC-8 loader

The diverse nature of microprocessor applications with their different types of memories and, often, noncontiguous address spaces requires a flexible loader. Besides performing the normal functions such as relocation and resolution of external references, the MAC-8 loader has the following features:

(*i*) Definition of a unit of relocation (section).

```
lowmem    { f4.o(.text) }

.text     { .=0x100 }
.data     { .=0x5000 }
.bss      { .=0x8000 }

f1.o
f2.o

.bss      {
          . = ( . + 7 ) & 0xfff8
          _RPORG =.
          f3.o(.bss)
          _SPORG = .-1
}

highmem   {
          .=0xa000
          f3.o(.data)
}
```

Fig. 4—Input specification.

(*ii*) Assignment of values to unresolved symbols.
(*iii*) Control of the location counter within sections.

These additional features are specified by an input language that has C-like syntax. For example, the input specification of Fig. 4 will take the relocatable object files (output of the compiler or of the assembler) of Fig. 5a and create the absolute binary output files depicted in Fig. 5b. Fig. 5a consists of four files, the first three containing three sections each, namely .text, .data, and .bss, and the last just .text. _RPORG and _SPORG are unresolved symbols that will

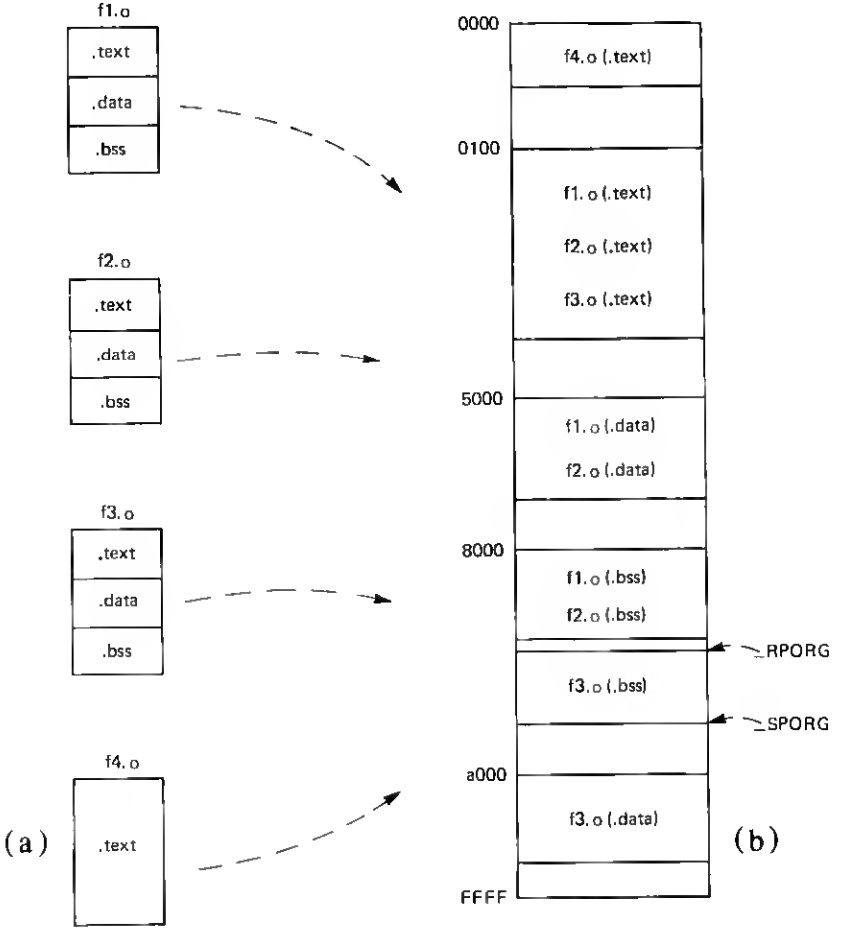

Fig. 5—Loader example.

```
when (23'func1 || glo == 4) {
        if ( flag'func1 ) {
                display i', w;
                userblock;
                continue;
        }

        else display ar'[0] : ar'[n], m'+6;
}
```

Fig. 6—MAC-8 simulator example.

determine initial values of the register pointer and stack pointer, respectively. The expression .= (.+ 7) & 0xfff8 aligns the f3.o (.bss) on a 8-byte boundary.

5.4 MAC-8 simulator

The MAC-8 simulator runs on the UNIX host system and permits debugging of MAC-8 programs without using MAC-8 hardware. The simulator is "source-oriented" and "symbolic," which means that programs can be debugged by referencing variables and function line numbers in terms used on the source listing (compiler or assembler). The symbolic debugging permits the debugging of a C program without worrying about the code generated by the compiler, as illustrated in Fig. 6. The simulator also allows conditional execution of pre-stored procedures of commands and the specification of C expressions containing both user-program and debug symbols, making possible the composition of debugging experiments in an interactive fashion. The C-like input language minimizes the difficulties in changing from one software tool to another. The major features of the MAC-8 simulator are:

(*i*) Loading a program and specifying a memory boundary.
(*ii*) Conditional execution of code and semantic processing when a break point is encountered.
(*iii*) Referencing C identifiers (both local and global) and C source-line numbers on a per-function basis.
(*iv*) Defining a command in terms of other commands to minimize typing.
(*v*) Displaying timing information.
(*vi*) Displaying items in various number bases.
(*vii*) Allocating non-program, user-defined identifiers.

(*viii*) Execution of input commands read from a file on an interactive basis.

(*ix*) Some structured programming constructs, including **if-else** and **while**.

The command illustrated in Fig. 6 will cause a break point when

the program executes line 23 of function **func1** or

the global variable **glo** is equal to 4.

When the break point occurs, if the value of local variable **flag** of function **func1** is non-zero, the values of local variable **i** and global variable **w** are printed, the user-defined block **userblock** is executed, and execution continues; otherwise the contents of local array **ar** for subscripts 0 through *n* and the value of the expression **m'+6** are printed.

5.5 Utilities

Utilities and a library are necessary parts of a support system. The MAC-8 system not only has utilities (like the UNIX system) for determining the size of an object file and the contents of the symbol table, but also a disassembler, a function line-number listing program, and a program to format an object file to permit "down-line" loading into the MAC-8-based application.

5.6 PLAID

A microcomputer-based application or target system typically differs from the host system on which it was developed, particularly in its periphery. Development of a microprocessor application requires hardware/software tools that allow development and debugging in real-time of the target processor and the periphery of its application. The PLAID (Program Logic Aid) system described below is such a tool.

The PLAID hardware includes two MAC-8 systems, each with full memory, and an associated hardware monitor system in a configuration that permits one MAC-8 system (the master) to closely monitor and control the other MAC-8 (the slave). Each MAC-8 has separate I/O, allowing connection to various peripheral devices from the master, and to the application hardware from the slave. The monitor hardware includes various debugging aids, as well as the MAC-cable that allows in-circuit control of any MAC-8 system.

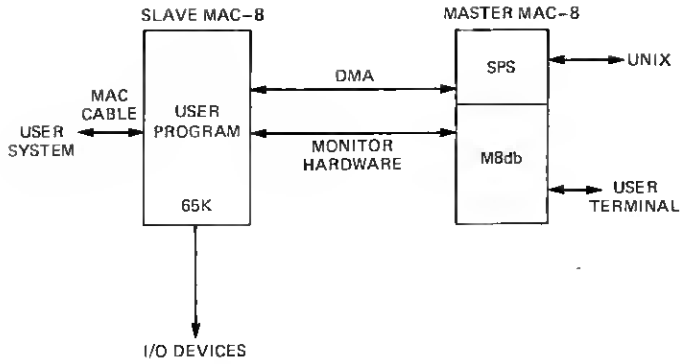

Fig. 7—Level-1 PLAID.

The PLAID software system includes the Satellite Processor System (SPS), which communicates with a host UNIX system, performing operating system functions for the master and monitor hardware, and **m8db**, a source-oriented symbolic debugger whose capability is similar to the MAC-8 simulator with the addition of real-time break points by use of the PLAID monitor system.

In the early stages of development, a fully-instrumented MAC-8 within the PLAID serves as the processor for the target machine, where in later stages, the PLAID monitors and controls the prototype system. Level-1 PLAID is illustrated in Fig. 7. Work is being done on level-2 PLAID, illustrated in Fig. 8. The fundamental difference in hardware between levels 1 and 2 is in the master system, which in level 1 contains a 32K PROM (programmable read-only memory) and a 16K RAM, while level 2 contains a 65K RAM and a dual-drive double-density floppy disk. The SPS executive is replaced in level 2 by a single-user UNIX system; the debugger can be swapped in from the floppy disk, as can other tools.

The satellite processing system of level 1, which is functionally similar to the system described in Ref. 9, resides in the master and controls the flow of program execution. SPS permits communication with a host UNIX system via a dial-up connection, and performs the operating system functions for **m8db**, such as control of the master and monitor hardware. Any UNIX command can be entered at the user terminal (see Fig. 7) and SPS determines whether the command will be processed by PLAID or must be transmitted to UNIX. The SPS interface to **m8db** consists of UNIX-like system calls.

The PLAID-resident symbolic debugger, **m8db**, has a command language which is a superset of the MAC-8 simulator. The additions to the language permit the referencing of the PLAID monitor system

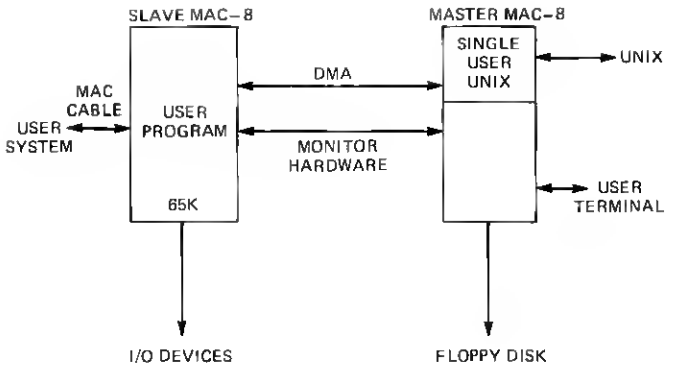

Fig. 8—Level-2 PLAID.

hardware to establish real-time breakpoints. **m8db** has all the features mentioned in Section 5.4, as well as facilities to help debug programs in real time.

The MAC-cable is the channel of communication between the PLAID and the application hardware, and permits control of the MAC-8-based system. During the initial stages in the development of an application, the MAC-cable permits testing out of the slave's MAC-8 and memory, while using the application's I/O. As the development progresses, the MAC-cable permits testing the application's hardware, including its MAC-8 and memory.

The PLAID monitor system keeps track of the actions of the slave or user system enabling the debugging of MAC-8 applications in a real-time environment. The major features of the PLAID monitor system include:

(*i*) *Memory monitor* contains a 65K by 4-bit dynamic RAM that enables trapping ("break-pointing") on a variety of conditions such as:
  (*a*) Memory read, write, or reference.
  (*b*) Arbitrary 8-bit pattern in memory.

(*ii*) *Register monitor* enables "break-pointing" on a read, write, or reference of any of the MAC-8 off-chip registers (see Fig. 1).

(*iii*) *CPU monitor* contains shadow registers that hold the current values of the slave/user MAC-8 on-chip registers (CR, SP, RP, PC), as shown in Fig. 1.

(*iv*) *Event counters* consist of three 16-bit counters and one 32-bit counter that enable "break-pointing" on a variety of events. The events include:

(*a*) Slave/user interrupt acknowledge.
(*b*) Slave CPU clock output.
(*c*) Slave/user memory read, write, or reference.
(*d*) Opcode fetch.
(*e*) Trap.
(*f*) User-supplied backplane signal.

(*v*) *Jump trace* contains a history table of the last 32 program counter discontinuities.

(*vi*) *Instruction trace* consists of a 64-entry trace table of the last 64 cycles executed by the slave.

(*vii*) *Memory access* circuitry permits the selective specification, on a block (256 bytes) basis, of read/write protection. Memory timing characteristics can also be specified on a block basis.

(*viii*) *Clock frequency* for the slave can be selected from a list of predefined frequencies.

VI. DESIGNER EDUCATION

Because of the nature of microprocessor applications, designers must have both hardware and software expertise. Many hardware designers must write programs for the first time, which poses an interesting educational problem. C is a difficult language for nonprogrammers because it is both powerful and concise. This problem can be partially remedied by giving seminars and supplying tutorials on C and on programming style. Offering workshops on the hardware and software aspects of the MAC-8 has also helped.

The MAC-tutor, an "electronic textbook," enables the designer to learn MAC-8 fundamentals. The MAC-tutor is a single-board computer with I/O and can be communicated with by a 28-function keypad or by a terminal. A connection to a UNIX system can also be established for use in loading programs and data into the MAC-tutor memory. The MAC-tutor also includes an executive to control hardware functions, 1K RAM expandable to 2K, sockets for three 1K PROMs, eight 7-segment LED displays, a PROM programming socket, and peripheral interfaces to a terminal and a cassette recorder. The tutor, besides being an educational tool, may be used to develop small MAC-8-based applications.

VII. SUMMARY

The MAC-8 was designed together with an integrated support system. The MAC-8 architecture was influenced by the C language, and

the support tools were influenced by UNIX. The consistent use of C-like language syntax permits easy transition from one tool to another. Software tools that began, in many cases, as spinoffs of existing UNIX tools have evolved to meet the needs of microprocessor applications. The MAC-8 support system continues to evolve to meet the growing needs of microprocessor-based applications.

VIII. ACKNOWLEDGMENTS

The design and development of the MAC-8 and its support system required the cooperative efforts of many people over several years. The author wishes to acknowledge the contributions of all the team members whose work is summarized here.

REFERENCES

1. J. A. Cooper, J. A. Copeland, R. H. Kranbeck, D. C. Stanzione, and L. C. Thomas, "A CMOS Microprocessor for Telecommunications Applications," IEEE Intl. Solid-State Circuits Conf. Digest, *XX* (February 17, 1977), pp. 138-140.
2. H. H. Winfield, "MAC-8: A Microprocessor for Telecommunications," The Western Electric Engineer, special issue (July 1977), pp. 40-48.
3. I. A. Cermak, "An Integrated Approach to Microcomputer Support Tools," Electro Conference Record, session 16/3 (April 1977), pp. 1-3.
4. D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," B.S.T.J., this issue, pp. 1905-1929.
5. D. M. Ritchie, "UNIX Time-Sharing System: A Retrospective," B.S.T.J., this issue, pp. 1947-1969.
6. T. A. Dolotta, R. C. Haight, and J. R. Mashey, "UNIX Time-Sharing System: The Programmer's Workbench," B.S.T.J., this issue, pp. 2177-2200.
7. D. M. Ritchie, S. C. Johnson, M. E. Lesk, and B. W. Kernighan, "UNIX Time-Sharing System: The C Programming Language," B.S.T.J., this issue, pp. 1991-2019.
8. H. D. Rovegno, "Use of the C Language for Microprocessors," Electro Conference Record, session 24/2 (April 1977), pp. 1-3.
9. H. Lycklama and C. Christensen, "UNIX Time-Sharing System: A Minicomputer Satellite Processor System," B.S.T.J., this issue, pp. 2103-2113.