microware®

OS-9 LEVEL TWO

SYSTEM DESIGNER'S GUIDE

OS-9 LEVEL TWO

SYSTEM DESIGNER'S GUIDE

Revision C
Publication date: September 1, 1983

Microware Systems Corporation
5835 Grand Avenue
Des Moines, Iowa 50312 U.S.A
Telephone 515-279-8844
Software Support 515-279-8898
Telex 910-520-2535

INTRODUCTION

The combination of Microware's OS-9 Level Two and the 6809 microprocessor produces an incredibly powerful multiuser/multitasking operating system comparable in performance to most 16-bit micro and minicomputer systems, but at a fraction of the system cost. Level Two systems can be used for general timesharing purposes, as a single user/multitasking system, or for control systems which require large amounts (>64K bytes) of RAM and/or ROM memory.

This purpose of this manual is to provide guidance and design suggestions to designers of 6809 computer systems that will OS-9 Level Two. OS-9 requires specific hardware capabilities, and can utilize optional hardware features that can damatically improve system performance. The goal of this manual is to provide a discussion of these topics and overall guidance to the hardware designer.

Microware does not market individual copies of OS-9 Level Two, rather, the software is available only under OEM license. This means that even if your hardware design is compatible with Level Two, you will have to contact Microware to arrange for software installation and distribution licenses.

In case you're not familiar with Microware's software product line, we would like to point out that this manual applies only to OS-9 Level Two which was designed for multiuser applications or applications requiring in excess of 64K of memory. The Level One version of OS-9 is smaller-scale and does not require memory management hardware, and therefore is limited to applications using 64K memory or less. If you don't need a big system you will find Level One software and hardware more economical.

If you have any questions about Level Two hardware design, be sure to call us. We at Microware are always ready to assist you with hardware or software consultation. It's easier to change designs while they are still on paper, so we encourage you to consult with us before you reach the prototype stage.

HARDWARE DESIGN GOALS

The OS-9 Level Two operating system gives 6809 computers the ability to run multiple, independent tasks utilizing up to 64K memory each up to a system-wide maximum memory size of one or two megabytes of RAM and/or ROM. In order to expand and efficiently utilize this larger memory capacity, OS-9 operates in conjunction with special Memory Management Unit (MMU) hardware. The MMU simultaneously expands the 6809 addressing capability and permits addresses generated by the 6809 chip to be mapped to different actual memory addresses. This function is called Dynamic Address Translation (DAT). Simple bank switching systems cannot be used in place of a true MMU for OS-9.

Even though OS-9 requires the hardware it runs on to have certain minimum MMU/DAT capabilities, the designer still has a great degree of freedom in designing the MMU hardware. The cost/performance tradeoffs typical of microcomputer systems very much applies here. You can design a low-cost minimum parts count MMU, or a more complex MMU that has additional features to increase system performance. This document is intended to offer guidance when making these decisions during the design of an MMU for use with OS-9.

An additional part of the system design process relates to input/output devices and how they should be interfaced to the system for maximum performance.

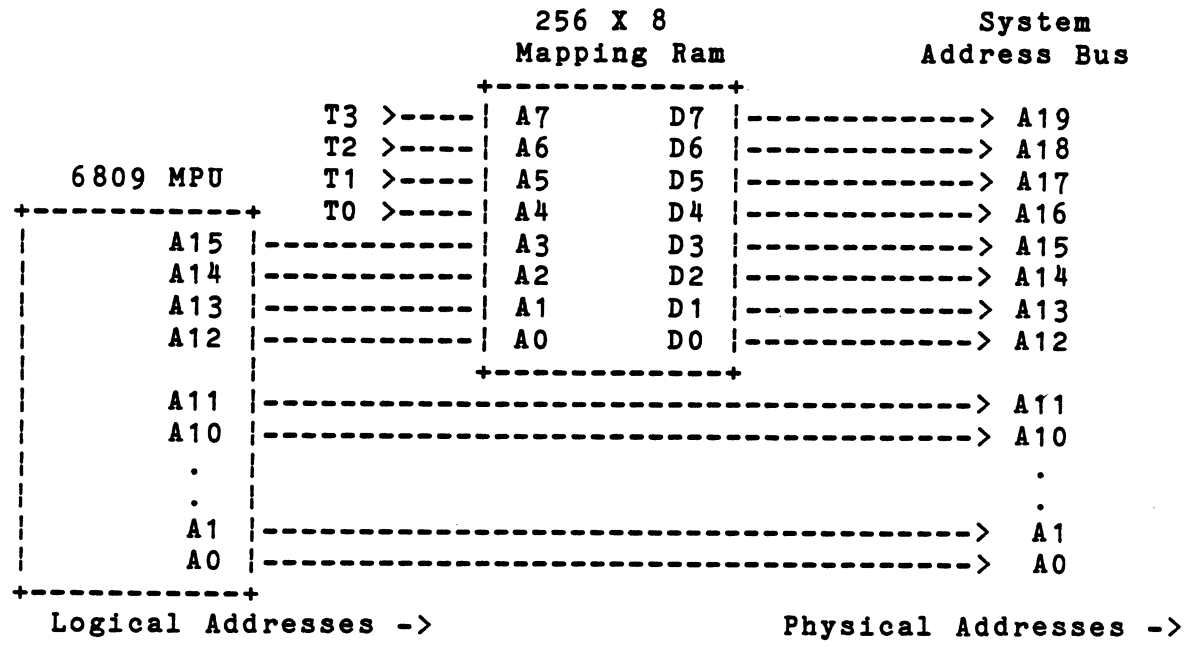
HOW THE MMU WORKS

This section describes the characteristics that any MMU to be used with OS-9 must have.

The basis of an MMU is a high-speed RAM memory (Mapping RAM) which is inserted between the higher address output line of the 6809 and the system address bus. The 16-bit addresses coming out of the 6809 are referred to a "logical addresses" because they are the addresses used by programs. The four or five high-order address bits (A11 or A12 through A15) are fed to the address input lines of the mapping RAM as shown in Figure 1. The data output lines of the mapping RAM are then used as the system-wide address lines.

The number of output bits of the mapping RAM is greater than the number of input logical address bits, which is how the extended address lines (A16 though A19 or A20) are generated. The addresses generated on this bus are referred to as "physical addresses" because they are the addresses decoded to select memory and I/O devices.

Figure 1
BASIC MAPPING RAM LAYOUT



THE TASK REGISTER AND VIRTUAL MAPS

The high-order address lines of the mapping RAM (labelled T0 through T3) in Figure 1) are "task select" lines. These bits are defined by a "Task Select Register" which can be a simple latch or PIA outputs. This effectively divides the mapping RAM into multiple sections (in this example 16 sections). Each section has one data word for each possible combination of high-order MPU address lines. OS-9 uses the various mapping RAM sections to define virtual memory maps for system and user tasks. The software can switch virtual memory maps simply and quickly by changing the value of the task select register. OS-9 requires at least five maps to operate efficiently, but because RAM sizes are even powers of two, eight maps is the practical minimum.

NOTE: THE NUMBER OF SIMULTANEOUS TASKS THE OPERATING SYSTEM CAN RUN IS A SOFTWARE FUNCTION THAT IS NOT LIMITED BY THE NUMBER OF TASK REGISTERS AVAILABLE!

Internally, OS-9 keeps a table for each task called a "process descriptor" which contains infromation such as the task priority, user ID, open files, etc., and also a section called a "DAT image". The DAT image is an exact copy of the contents that must be loaded into the MMU registers as required by the particular task. When OS-9 activates the task, it copies the DAT image into the MMU registers. When the memory map must be changed (for example, to give the task more memory) it is accomplished by updating the DAT image. Use of the DAT image is the reason why the MMU register may be write-only, and also why OS-9 can run more tasks than there are sets of registers in the MMU.

Bacause the DAT image is copied to the MMU at the start of each of the task's time slices, it is important that the MMU registers can be written to directly.

OPERATION OF THE MAPPING RAM

The physical (bus) address generated by the MMU depends on the data word stored at the corresponding mapping RAM address which can be longer and totally different than the MPU (logical) address. Each section of the mapping RAM therefore can map a complete 16-bit (64K) logical address to N out of any M physical address blocks of block size B.

"N" is the number of blocks the logical map is divided into, which depends on the number of MPU address lines routed through the MMU. This is either 4 or 5 lines, giving an "B" of 4K or 2K block sizes, respectively. This number plus the number of bits in the task register defines how many address lines the mapping RAM must have. The block size is the smallest unit of memory that can be allocated by the operating system.

"M" is in effect the maximum physical addressing capability of the MMU which depends on the number of physical address lines generated by the mapping RAM. This is a function of the length of data words in the mapping RAM.

What size mapping RAM should you use? The two main factors are: 1) the desired block size. A 2K block size results in less memory wastage than a 4K block size, but sometimes 4K block hardware has a much lower parts count. 2) What is the desired maximum memory addressing capability? Figure 2 below can assist you in determining the size of the mapping RAM required in your system.

Figure 2
SAMPLE MAPPING RAM CONFIGURATIONS

	Block Size	Number of Task Regs	Maximum System Memory	Mapping RAM Organization
CoCo 3 →	8K	2	524,288	16 x 6
	2K	8	524,288	256 x 8
	2K	8	1,048,576	256 x 9
	2K*	8	2,097,152	256 x 10
	4K	8	524,288	128 x 7
	4K	8	1,048,576	128 x 8
	4K	8	2,097,152	128 x 9
	4K**	16	1,048,576	256 x 8
	4K	16	524,288	256 x 7
SwTPc →	4K	1	1,048,576	16 x 8
CMS 9639 →	4K	128	1,048,576	2048 x 8

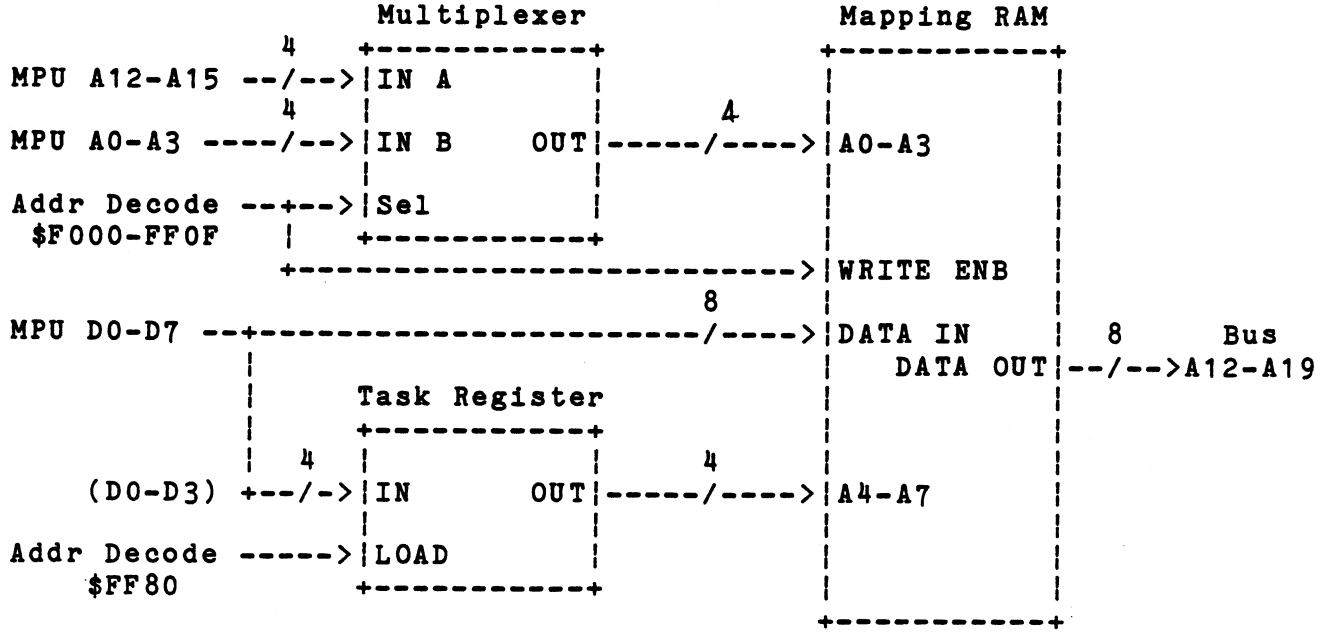
* Typical of two cascaded MC6829 devices.
** Typical of two 93L412/93L422 devices.

HOW THE MAPPING RAM IS ACCESSED

OS-9 must be able to write data to any location within the mapping RAM. This means that the mapping RAM must also respond to memory addresses accessable to the MPU. This is done by a 4 or 5-wide, 2-input multiplexer that switches the address lines from the high-order bits of the MPU (normally) to the low-order bits when writing to the RAM. The multiplexer is controlled by a standard address decoder for the range of addresses assigned to the mapping RAM. Note that this decoder must decode LOGICAL addresses directly from the MPU and not the mapping RAM's output. System complexity can be reduced by not multiplexing the bits generated by the task register because they are already software-definable, and by using the same address decoder used for bootstrap ROM because the mapping RAM and the task select register can be "write-only". RAM devices that have separate data input and output pins can further simplify things if the input pins are always connected to the data bus and the output pins always go to the physical address bus.

Figure 3 below shows the input multiplexer logic for a 256 by 8 type MMU system. Normally, the multiplexer feeds MPU address lines A12-A15 to the low-oder bits of the mapping RAM address inputs. When the MPU writes to the address range $F000-$FF0F (or whatever address is assigned for the MMU) the multiplexer switches the address input to the MPU A0-A3 lines, so each of the 16 addresses in the mapping RAM is decoded sequentially in the address space. The address decoder output also controls the mapping RAM write enable. Note that when writing to the mapping RAM the task register still selects which of the 16 "sets" of 16 registers is addressed.

Figure 3
MAPPING RAM AND ADDRESS MULTIPLEXER DETAIL



ELECTRICAL REQUIREMENTS OF THE MAPPING RAM

First and foremost, the mapping RAM must be FAST because it will introduce a delay in the system address lines. This is especially important if the system is to run at 2 MHz. If the delay is significant you can always use faster RAMs and PROMs but it's hard to get around the minimum address setup time specification of peripheral interface devices such as the 6850, 6821, etc.

Because speed is important, bipolar RAMs are a good choice. Fairchild or Motorola 93L412 (O-C output) or 93L422 (3-S output) have been frequently used as mapping RAMs in OS-9 systems because they are fast (25ns), cheap, and readily available. They are organized as 256 words by 4 bits, so two devices provide a very convenient 256 by 8 configuration.

Motorola MC6829 Memory Management Unit devices use internal MOS "high speed" RAM but even so the 68B29 introduces a 110 ns address delay which may cause problems in 2 Mhz systems. A subsequent section discusses this part in greater detail.

THE SYSTEM BOOTSTRAP ROM

The system must have some ROM to hold the reset vector, code to initialize the MMU and bootstrap the system. Additionally, OS-9 systems use the same ROM for part of the operating system kernel software required for system intialization. The overall size requirement for this ROM should not exceed 4K bytes in total. This ROM is located at addresses $FF000 to $FFFFF and is usually an EPROM device for convenience. It contains a little over 3K bytes of OS-9 code; the rest is reserved for a disk system bootstrap loader.

After system reset the mapping RAM contains random data so the ROM must always be enabled when addresses $F000-$FFFF are generated by the MPU. These addresses are decoded on the MPU side of the MMU, not the bus side. This method is simpler from a hardware standpoint but will limit the maximum address space for each task to 60K bytes.

An alternative method that gives each task the maximum 64K address space requires more complex hardware. The ROM must be selected after reset by disabling the MMU or forcing its output to $FFxxx until a software command is given (such as writing to a special address). In this configuration, the bootstrap ROM is decoded after the MMU using a full 20- or 21-bit address.

OS-9 Level Two can be completely ROMed if desired. In this case, the 4K ROM mentioned above must be provided, plus another 20K (12 for non-disk systems) of fully decoded "normal" ROM which can be located at any convenient address above the end of the RAM area. If user application software is also to be ROMed an appropriate amount of additional ROM capacity should be provided.

TASK SWITCHING

Whenever an interrupt occurs or a user program issues a service request to OS-9 the current virtual map is switched by changing the value of the task select register to zero. Map zero is always the map that contains OS-9 itself, interrupt routines, and I/O device addresses. Because service requests to OS-9 use the SWI2 instruction, hardware can OPTIONALLY be provided to automatically switch to map zero on any interrupt, which can be detected by decoding the interrupt acknowledge state using the 6809 BA and BS lines.

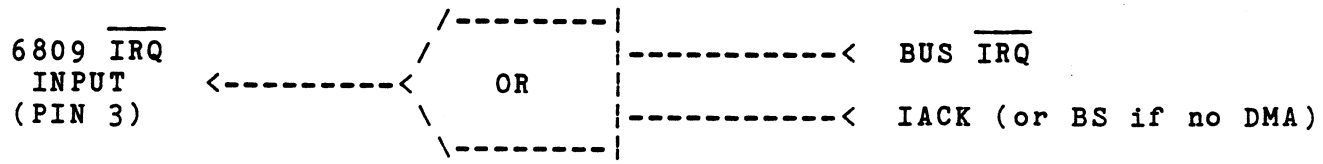
Automatic task switching has the advantage of permitting user programs to use a full 64K virtual address space. This is because non-automatic systems must always map the interrupt vectors and a bit of code to clear the task register into the highest memory block of every user map. This limits maximum memory per task to 62K if a 2K block size is used, or 60K for the 4K block size. Automatic switching also improves interrupt service response time somewhat.

Its disadvantage is complexity, for automatic task switching systems not only have to detect and switch maps on interrupts, but they also have to provide a way to return to a user map. OS-9 returns to user maps using an RTI instruction. A hardware counter triggered by software must count cycles and switch the task select register from zero to the user task register after a certain number of MPU cycles contained in a "fuse register". The 6829 implements this function; consult its data sheets for more information.

One last point about automatic task switching is that IRQs must be externally disabled by hardware from the time the interrupt acknowledge occurs until after the first cycle of the instruction executed after the map switch (which is ORCC #IRQMASK). This is because the SWI2 and SWI3 instructions do not set the IRQ mask and disaster may ensue if another interrupt occurs before OS-9 has a chance to save and reload the stack pointer register, which is pointing to somewhere in the previous map. THIS IS A PECULIARITY OF THE 6809 SO ANY AUTOMATIC TASK SWITCHING SYSTEM INCLUDING THE 6829 HAS THIS PROBLEM AND REQUIRES THE INTERRUPT MASKING HARDWARE.

A simple interrupt lockout circuit is shown in Figure 4 below. Motorola's 6829 application note AN-859 indicates this problem may be solved by inserting an OR gate is series with the 6809 IRQ input with the other input connected to the BS signal. This will only work if the system does not use DMA. If DMA is used, the interrupt acknowledge state (BA=1, BS=1) must be fully decoded as the other OR gate input. If NMI or FIRQ interrupts will be used their respective MPU inputs should have the same circuit.

Figure 4 - IRQ Task Switch Lockout Circuit



THE MOTOROLA MC6829

The MC6829 was designed especially as the Memory Management Unit for the 6809. Additionally, Motorola designers consulted with Microware concerning OS-9 compatibility during development of the 6829. This LSI part does everything a good MMU should do and can replace from ten to thirty discrete bipolar devices.

It major drawback is its speed, or more correctly, lack thereof. The 68B29 can operate with MPU clock rates of 1.0 or perhaps up to 1.5 Mhz, but not at a full 2 Mhz unless faster than usual RAM and PROMs are used and a slow-memory circuit is used to stretch the E clock during read/write cycles to I/O controller addresses.

Designers using the 6829 should carefully note the following:

1) Use exactly two 6829s. One is not enough, three or more are wasteful because they will not be used.

2) Be sure to provide the external IRQ mask circuit discussed previously.

3) Read Motorola data sheets carefully and follow the design information given in them. Motorola application note AN-859 also contains valuable information.

Microware offers an "off-the-shelf" version of OS-9 Level Two preconfigured for use with the 6829.

RAM AND ROM: THE SYSTEM MEMORY MAP

RAM should be in large contiguous blocks starting at physical address 00000. OS-9 performs a memory search during its startup sequence and will find all RAM starting on 2K/4K block boundaries. It automatically adapts to the amount of RAM actually installed in the system without software changes. We recommend a minimum of 64K of RAM; general purpose multiuser systems generally have from 128K to 256K minimum.

There should be at least 4K bytes of ROM at physical addresses $FF000-$FFFFF for the system bootstrap ROM. Some of these addresses can be "stolen" for use by the MMU. Additional ROM locations, if desired, should be at higher addresses, for example, $F0000-$FBFFF. ROMs addresses should start at 2K/4K block boundaries. OS-9 also performs an automatic ROM search at startup.

Figure 5
TYPICAL LEVEL TWO PHYSICAL MEMORY MAP

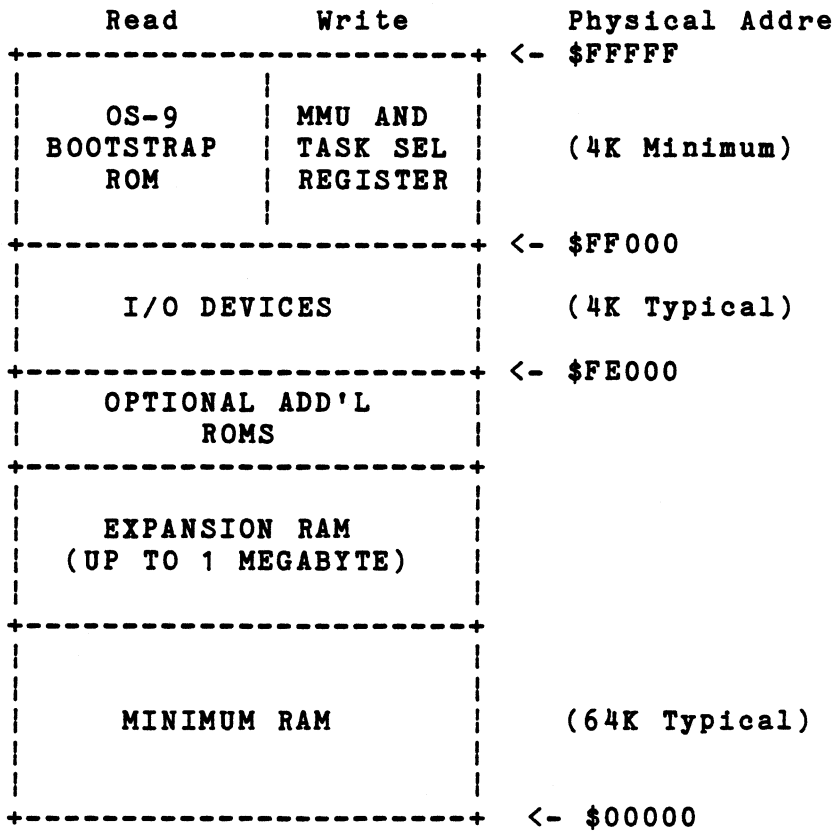
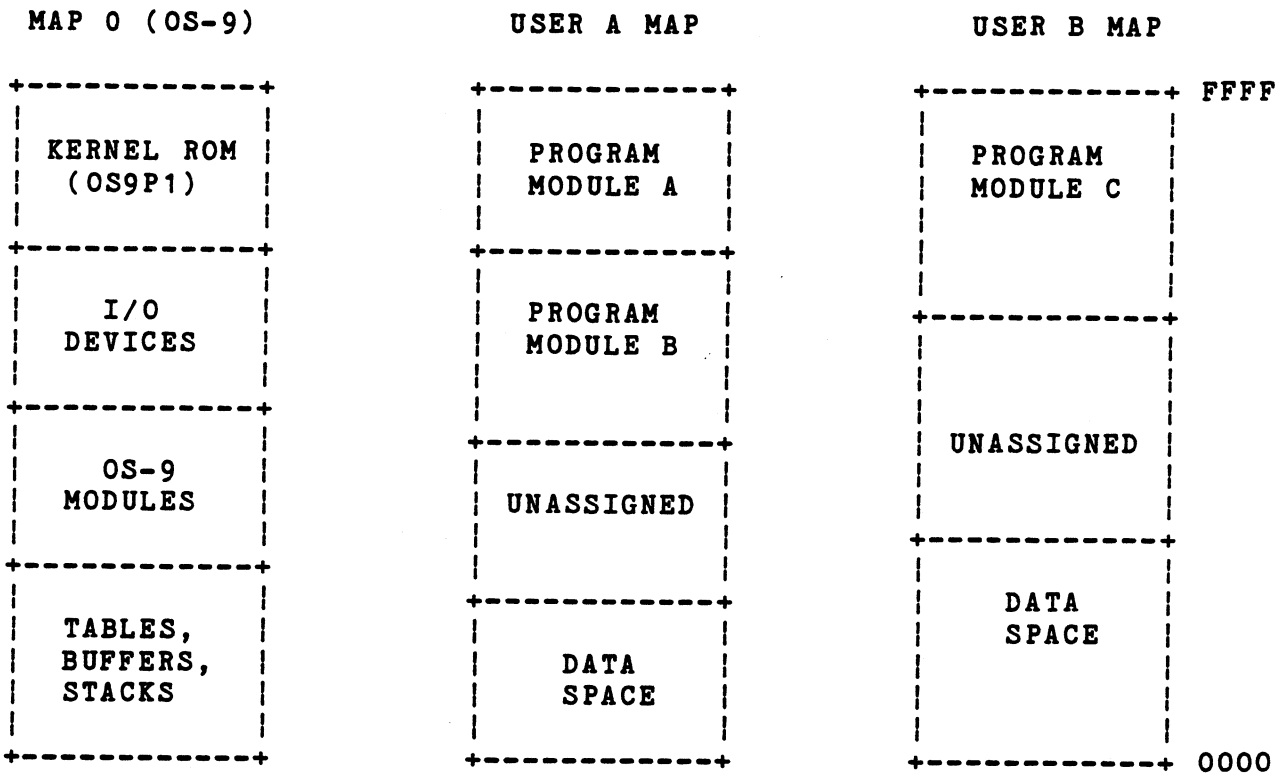


Figure 6
OS-9 LEVEL TWO EXAMPLE LOGICAL MEMORY MAPS
(SYSTEM WITH AUTO TASK SWITCHING)



I/O DEVICES: ADDRESSES AND REQUIREMENTS

No OS-9 Level Two system should be have its performance compromised by I/O devices that can't be interrupt driven. Certain unbuffered high-speed I/O devices (such as disks or data links) may benefit from DMA data transfers capability.

IRQs are the only interrupts actually used by OS-9, so all I/O devices should be tied to the IRQ line. Vectored interrupt systems can improve performance slightly, especially if of the type that gives the software a device number register to read instead of actual vectored jumps. However, the improvement is so slight that Microware only recommends vectored interrupt if the hardware already exists.

DMA is the preferred method for disk I/O transfers. DMA does NOT have to be mapped through the MMU; physical addresses are OK. Also, the DMA does not have to generate a true 20 or 21 bit address. You can use a DMA controller such as the 6844 that generates 16 bit addresses as long as you also include a 4- or 5-bit latch the software can set to define the high-order physical addresses during DMA cycles (only). OS-9 can also handle multiple DMA device contention in software so hardware bus arbitration is not necessary.

Smart disk controllers that are buffered (such as Xebec, DTC, Western Digital, Priam, etc,) can also give satisfactory performance without DMA but will operate faster with DMA.

All I/O devices should have FULLY DECODED PHYSICAL ADDRESSES (e.g., 20 or 21 bit address decoding). These addresses should be assigned in the higher part of the map, for example $FE00 - $FEFF in as few physical blocks as possible.

INSTALLING THE LEVEL TWO SOFTWARE

The installation process for Level Two is quite similar to the Level One installation instructions given in Chapter 9 of the "OS-9 System Programmer's Manual", except that some additional work may be required to adapt the software to the specific Memory Management hardware to be used.

The following items are required for installation:

a) OS-9 Level Two Source Code Files
b) An OS-9 based computer (running either Level One or Two)
c) OS-9 Asssembler and Text Editor
d) PROM programmer and appropriate PROM(s)

In addition, the availability of either a real-time 6809 emulator or a logic analyzer may save debugging time.

Customization of the software is usually concentrated in two areas: the kernel and device drivers. The kernel must be adapted to the physical characteristics of the MMU. You must have device drivers for the terminals and disks plus an appropriate disk bootstrap module (unless you are configuring a ROM-based system). In general, device drivers written for Level One will also work properly on Level Two if they are reassembled using the Level Two symbolic definitions file.

NOTICE: OS-9 LEVEL TWO INSTALLATION REQUIRES USE OF PROPRIETARY SOURCE CODE FURNISHED ONLY UNDER OEM LICENSE BY MICROWARE. SOURCE CODE IS NOT AVAILABLE FOR INDIVIDUAL SYSTEMS OR USERS.

Microware furnishes OS-9 Level Two kernel source code files in two generic versions, one for 4K block size systems without auto task switching, and one for 2K block size systems with auto task switching (6829 version). You should request the source code type from Microware which most closely matches the configuration of the target system. Microware also can provide generic device drivers for many types of I/O devices.

The only parts of OS-9 that are affected by differences in MMU design are the kernel (OS9P1 and OS9P2). The file managers, the shell, and the utilities always use OS-9 system calls to perform MMU-related functions, therefore these modules need not be reassembled and you may use the standard object code files supplied by Microware.

WARNING: THE FILE MANAGERS AND SOME UTILITIES ARE DIFFERENT IN LEVEL TWO THAN LEVEL ONE. BE SURE TO USE ONLY LEVEL TWO MODULES WHEN CONFIGURING THE SYSTEM.

MICROWARE SYSTEMS CORPORATION

DES MOINES