



MOTOROLA
Semiconductor Products Inc.

AN-859
Application Note

MEMORY MANAGEMENT TECHNIQUES USING THE MC6829

Prepared by
Charles Melear
Microprocessor Applications Engineer
and
Ed Rupp
MOS System Design Engineer
Austin, Texas

INTRODUCTION

Eight-bit systems have been the work horse of the computer industry for a long time. This trend will continue because 8-bit systems are easy to build, small-sized, and economical. Most modern microprocessors have a 16-bit address range which limits their memory address range to 64 kilobytes. Many systems need additional memory and utilize inexpensive solid state memory (about 1/50 cent per bit) for mass storage. The access time required for large programs is virtually instantaneous when stored in solid state RAM as compared to the time required in fetching those programs from a disk or tape.

The management of this memory is a major concern. Various techniques are used to allow an 8-bit processor to use these large memories. Basically, bank select and mapping RAM techniques can be used (see Figures 1 and 2). Bank select involves using an addressable latch to select one of several memory banks. That is, the outputs of the latch form part of the chip select circuitry. With the mapping RAM technique, various data can be written to sequential addresses which will correspond to the upper N bits of an expanded address. Usually the mapping RAM will contain a power of 2 words; i.e., 2, 4, 8, 16, 32, etc.

The width of the memory is arbitrary. The required number of address lines from the MPU to address the mapping RAM are applied to it and the remaining address lines go to the system address bus. For instance, the upper three address lines of the MPU could address eight words of mapping RAM. The data output of the mapping RAM then forms the upper address bits. This is basically the technique used in the system presented in this application note.

To understand the system, an explanation of all major components is needed. The building blocks of the system consist of the MC6809 microprocessor unit (MPU), MC6829 memory management unit (MMU), MC6850 asynchronous communications interface adapter (ACIA), MC6840 programmable timer module (PTM), MC6844 direct memory controller (DMAC), and MC6854 advanced data link controller (ADLC). Each of these parts will be discussed here in order to outline their function in the total system under consideration. The function of the system is to act as a satellite processing station to be down loaded from a host computer with programs and data. The local operator can invoke these programs and digest data as the need arises.

SYSTEM CONFIGURATION

The MPU used in the system is an MC6809. With the aid of an MC6829 memory management unit, the memory capability of this system is increased to a maximum of 2 megabytes. One MC6829 MMU can be programmed to address any given 256 kilobytes of the 2 megabytes (the same MMU can be reprogrammed to address a different 256 kilobytes). An MC6850 asynchronous communications interface adapter (ACIA) provides communications to a local terminal. To enable rapid data transfer via a serial link to the main computer, a synchronous data link control (SDLC) protocol network is employed using the MC6854 advanced data link controller (ADLC). Only minimal ROM is needed to operate the system since application programs reside in RAM. The major function of the resident software is to

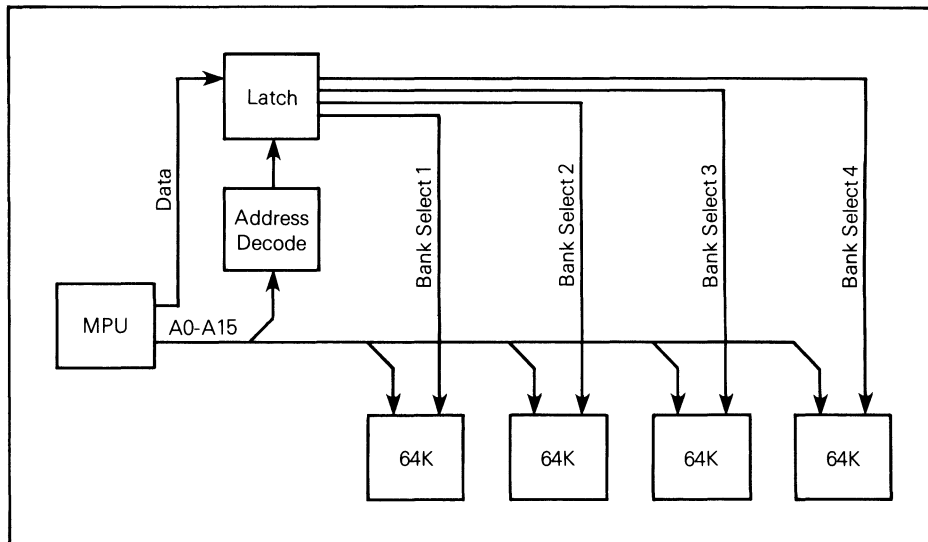


FIGURE 1 — Bank Select Technique

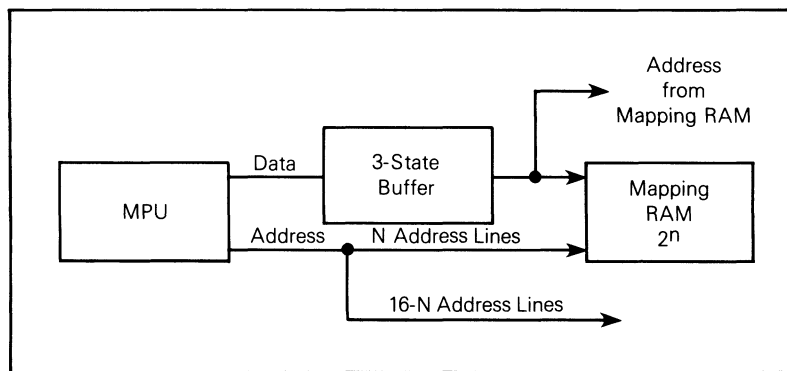


FIGURE 2 — Mapping RAM Technique

allow the local operator to control the system and to maintain control of, and assign memory to, the various tasks as needed.

The system block diagram of Figure 3 illustrates the placement of peripheral devices and buffers. See Figure 4 for a complete schematic diagram. Except for the MMU and DMAC, the address bus is applied to the rest of the system via 74LS244 buffers. The DMAC must appear exactly like the MPU to the entire system; therefore, the MPU and DMAC address buses are connected directly together on the input side of the address bus buffers. The MPU address bus is also connected directly to the MMU input address lines. The DMAC address bus could be applied to the output side of the address buffers; however, extra logic would be required to place the 74LS244 buffers in the high-impedance state whenever the DMAC is active. Since the MC6809 MPU can interface directly with the MC6844 DMAC, it is best to connect their address pins together. This allows the DMAC addresses to be mapped the same as the MPU address.

The chip select signal for the DMAC, as well as all other peripherals, is generated from the entire 21-bit address bus. At first it might appear that the DMAC chip select should be generated only from A0-A15 since this part directly feeds the MPU address bus. This is definitely not the case, as it would allow the DMAC to have a valid chip select signal in every one of the tasks handled by the MC6829 MMU.

In the example of Figure 3, the MC6854 ADLC drives the data bus during a DMA transfer. The read/write line for the ADLC is inverted during DMA and this places the ADLC data bus drivers in conflict with the 74LS245 data bus buffer; i.e., both devices would be driving the data bus. The problem is solved by disabling the 74LS245 data bus buffer during a DMA transfer. This can be accomplished by using $\overline{\text{Tx STB}}$ of the DMAC to act as a disable signal for the 74LS245. During a read of any MMU register, a conflict with the 74LS245 will again occur; therefore, the chip select signal for the MMUs must be used to disable the data bus drivers just as $\overline{\text{Tx STB}}$ was used when the ADLC was in the DMA mode. In general, data bus drivers must always be disabled when a peripheral device is connected to the MPU in parallel with the data bus buffer.

Two problems associated with random garbage accesses to memory must be solved. The first has to do with the timing of the $\overline{\text{R/W}}$ line with respect to the address lines. The second problem is associated with the dead cycles which precede and follow DMA cycles. The mode 1 timing (TSC steal mode) in Figure 5 shows that the internal MPU E clock is stretched during these dead cycle times. The MC6809 automatically places its address bus in the high-impedance state in response to a low bus request ($\overline{\text{DMA/BREQ}}$). The DMAC comes out of or goes into the high-impedance state whenever the MPU does the opposite. During these dead cycles, neither device

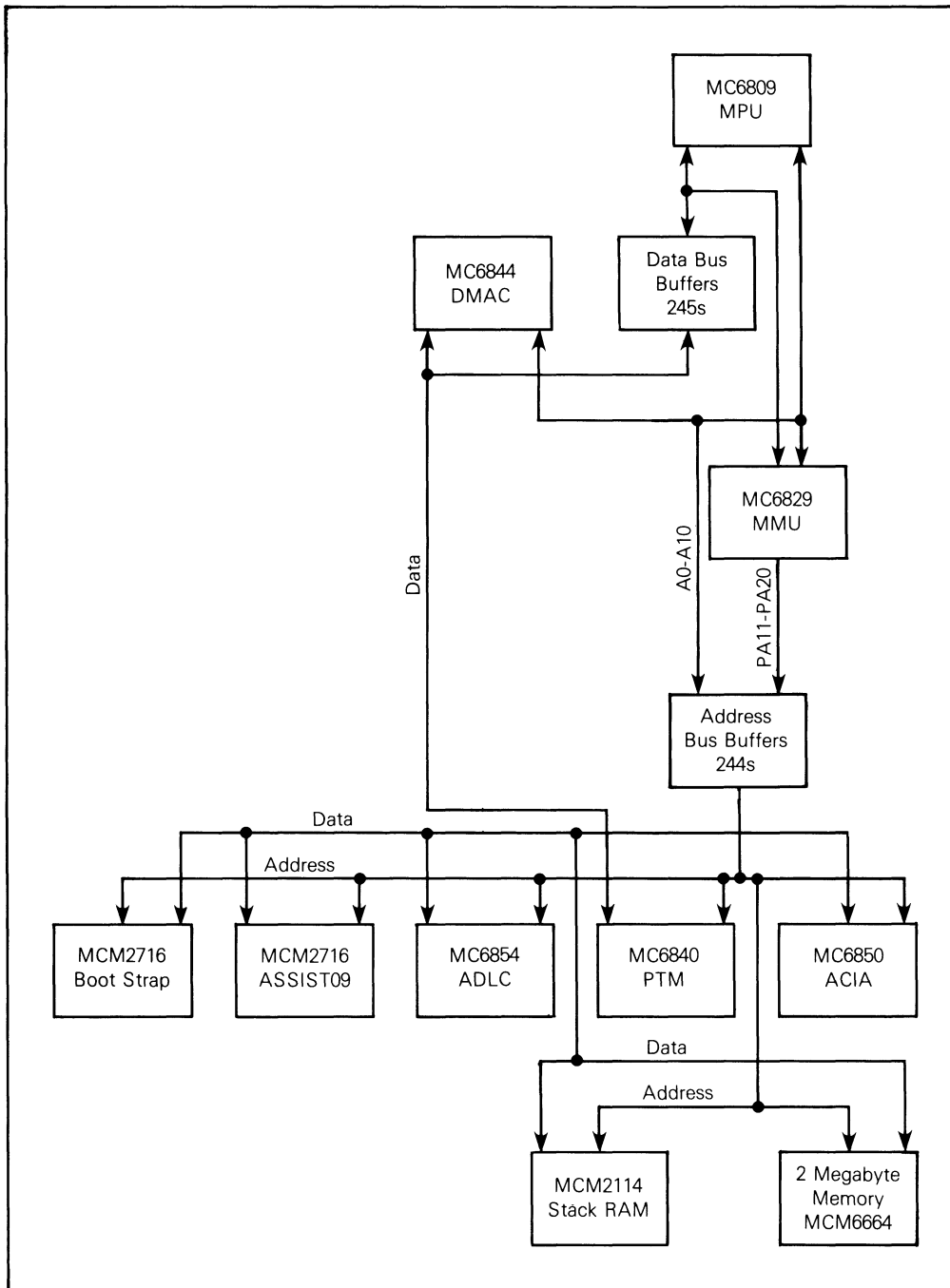


FIGURE 3 — MC6809 MPU System With Memory Management, Block Diagram

has control of the bus; therefore, it is possible for the address bus to float and possibly provide a valid chip select signal to some system device. If this happens, the affected memory location will be written with whatever random information is appearing on the data bus at that time. This problem can be eliminated by generating a DMAVMA signal using the circuit shown in Figure 6.

The output of an exclusive OR gate (SN74LS86) is low while the inputs to it are alike and high when they differ. The MC6809 BA and BS outputs go high during the first dead cycle (generate DGRNT) and return low during the second dead cycle. The DMAVMA signal goes high during the dead cycle as shown in Figure 7 and can be used as a memory

deselect. Both BA and BS are asserted during the first dead cycle; however, the resulting DGRNT output (see Figure 4) is not clocked through the 74LS74 flip-flop until the next negative transition of E. For this cycle the output of the exclusive OR gate is high and provides a memory deselect. Thus, during the time shown as dead in Figure 5, the system cannot be enabled and memories and peripherals are protected during exchange of bus control. Both BA and BS are released in the dead cycle immediately following the DMA transfer; therefore, the inputs to the exclusive OR gate do not match and the output goes high. Thus, while there is no active bus master (dead cycle), the memory cannot be inadvertently accessed.

NOTE: The MC6829 KVA0 through KVA7 lines shown represent connections to eight different parallel connected MMUs. Because of the 1-of-8 decoding action of U7, the KVA (pin 15) input to only one MMU at a time can go low.

Device	Ref. Desig
MC6809	U1
MC6829	U2
74LS244	U3
74LS244	U4
74LS244	U5
74LS245	U6
74LS38A	U7
74LS32	U8
74LS86	U9
74LS74	U10
74LS00	U11
74LS10	U12
74LS04	U13
74LS02	U14
74LS08	U15
74LS133	U16
74LS133	U17
74LS138	U18
74LS04	U19
74LS32	U20
74LS00	U21
74LS21	U22
74LS32	U23
74LS260	U24
74LS21	U25
74LS32	U26
74LS32	U27
74LS138	U28
74LS32	U29
MC6840	U30
MC6850	U31
MC2716	U32
MC2716	U33
MCM2114	U34
MCM2114	U35
74LS133	U36
MC6844	U37
MC6854	U38
MC14411	U39
MC1489	U40
MC1488	U41
MCM2532	U42
74LS32	U43
74LS86	U44
74LS08	U45
74LS153	U46
74LS32	U47
74LS09	U48
74LS04	U49
74LS155	U50

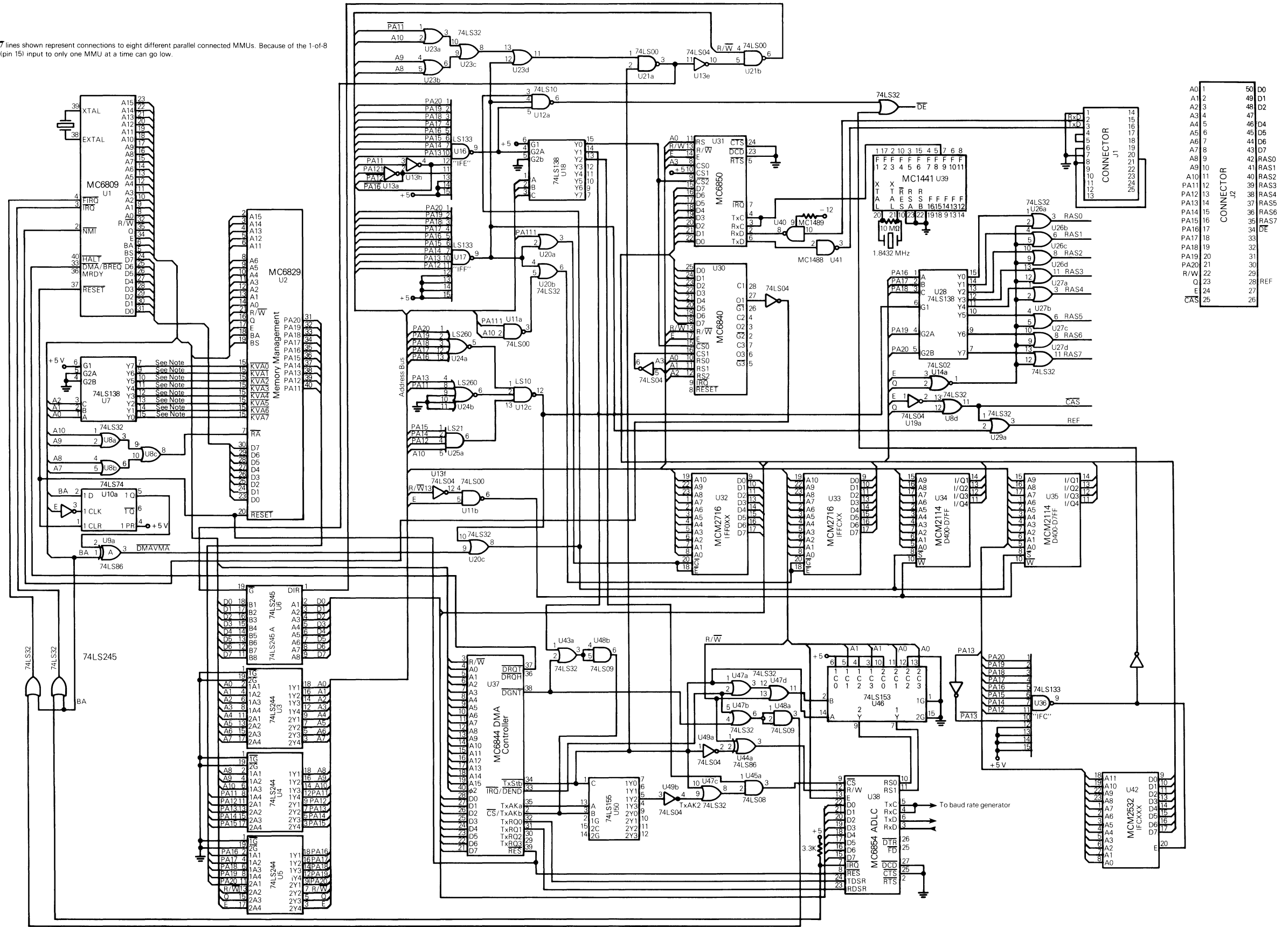


FIGURE 4 - System Schematic Diagram

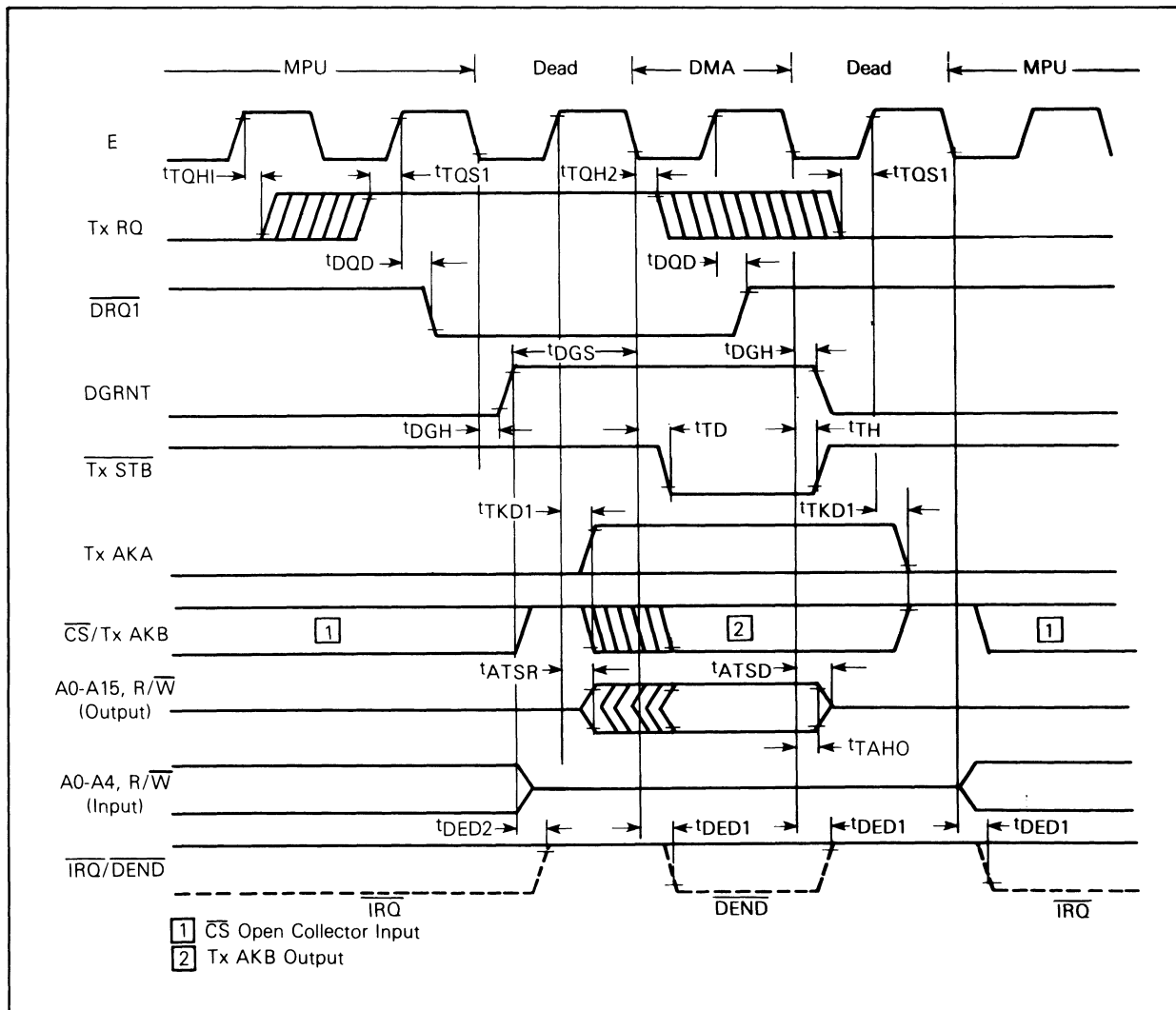


FIGURE 5 — Mode 1 Timing (TSC Steal Mode)

As previously mentioned, the timing of the R/\overline{W} line can cause a problem if a peripheral or memory becomes selected and R/\overline{W} is recognized as a write before all address lines become stable. This problem arises in devices, such as memories, that do not use the enable signal. By gating R/\overline{W} with E or Q of the MC6809 MPU, R/\overline{W} cannot go low until after the address lines have become stable. The circuit in Figure 8 shows a simple circuit to accomplish the proper conditioning of R/\overline{W} . The timing for the read/write delay circuit is shown in Figure 9. This problem does not always occur since it depends upon the individual characteristics of each MPU; however, at some point it may destroy the memory of certain systems in a random fashion if the conditioning circuit shown in Figure 8 is not used.

New interrupts must be delayed if they occur during an interrupt stacking operation for the current interrupt. Refer to the Interrupt Handling paragraph for more interrupt information. As shown in the system schematic diagram (Figure 4), \overline{FIRQ} and \overline{IRQ} are logical ORed with the BA line from the processor. This guarantees that the first instruction of every interrupt routine will be executed. That instruction can mask further interrupts. The \overline{NMI} input to the MPU is not handled in this way and its use in an MMU system for memory management is not advised.

There are two monitor programs used in this system. The first is ASSIST09, a debug program, which makes use of \overline{NMI} when doing single step traces through various programs. Its use is limited to task 0 only and it will not function in any other task because of its extensive use of SWI. The other monitor is CONST which helps program and use the memory management units in this system. The use of CONST is also restricted to task 0; however, this is a function of the software interrupts. This program is called by SWI, thus a task switch is required to enter it. In addition, an initialization program (MMUINIT) allows the MMU to be initialized. The ASSIST09 source code is available in the MC6809-MC6809E Microprocessor Programming Manual, MC6809PM(AD). The source code for CONST and MMUINIT is included at the end of this application note.

Synchronous communications are implemented using the MC6854 advanced data link controller (ADLC). A block diagram of the ADLC is shown in Figure 10. The ADLC is a full duplex communications device which is compatible with IBM SDLC format. The transmit and receive baud rates do not necessarily have to be equal. Modem interface pins are provided although they are not used in this example.

In addition to the processor interface pins, there are two DMA service request outputs associated with the transmit

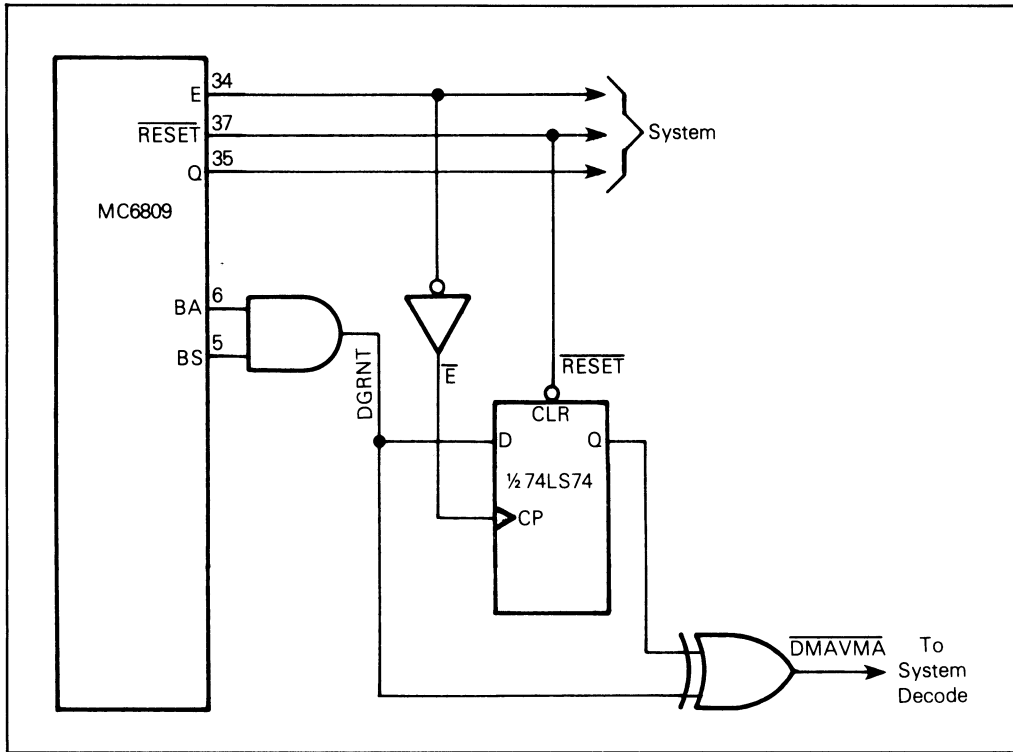


FIGURE 6 — $\overline{\text{DMAVMA}}$ Generation Circuit

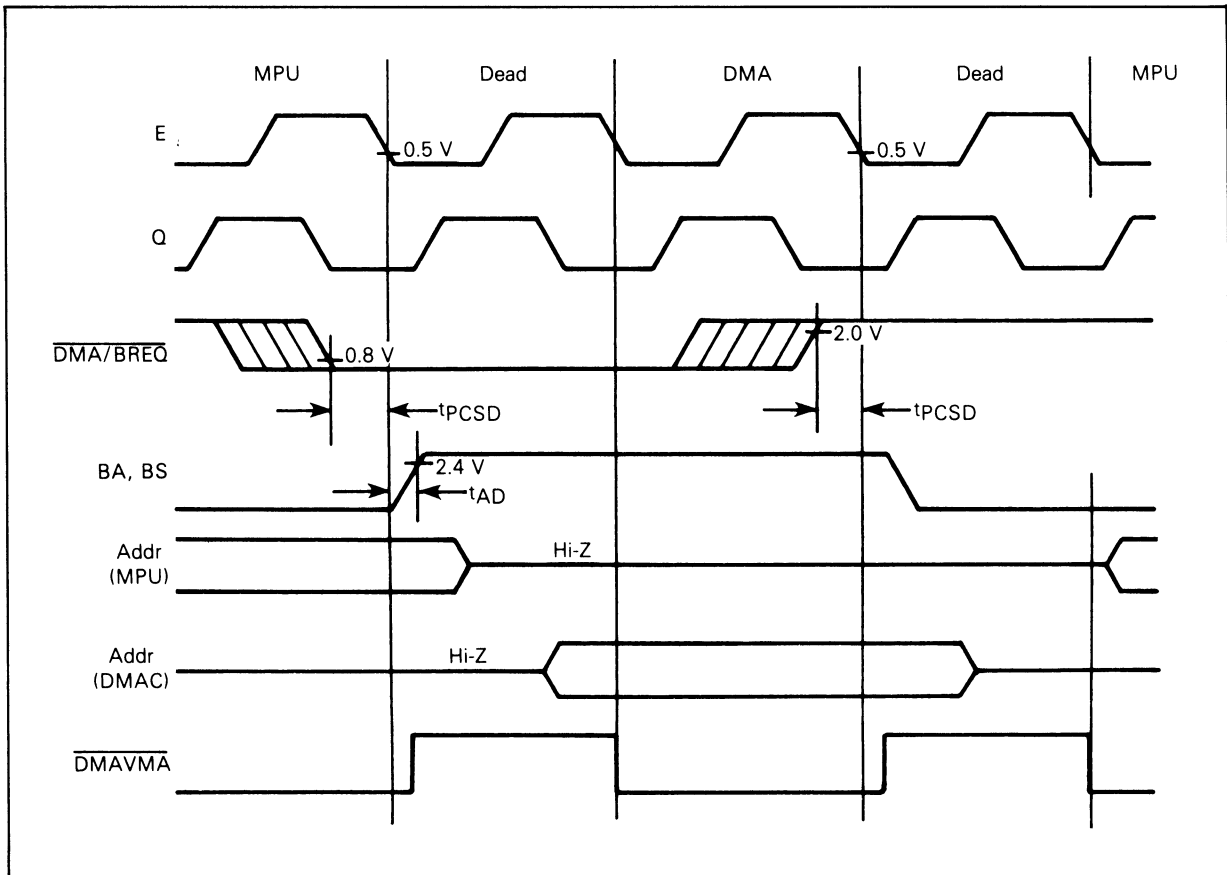


FIGURE 7 — MC6809 Bus Timing During MPU-DMA-MPU Bus Control Transfer

and receive channels, respectively. When enabled, these pins will drive the TxRQ pins of the DMAC. The TDSR (transmit data service request) will be asserted when the transmit data FIFO is empty. The RDSR (receive data service request) is asserted when data is ready to be read from the receive data FIFO.

The transmission format for SDLC is shown in Figure 11. Every message begins with an opening flag of 01111110. The

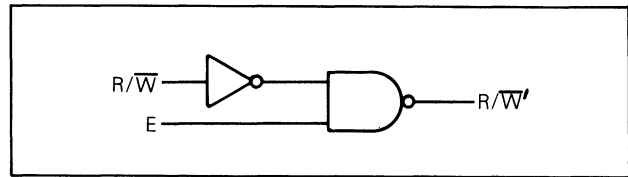


FIGURE 8 — R/W Conditioning Circuit, Gated By E

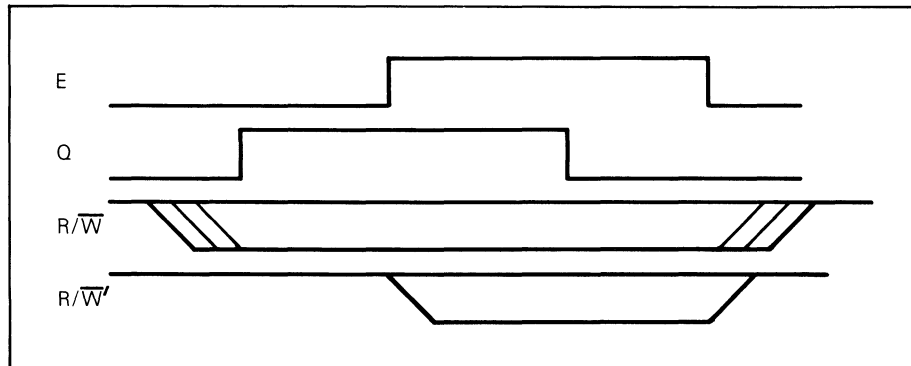


FIGURE 9 — R/W Conditioning Circuit Gated By E, Timing Diagram

next byte is an 8-bit address field and it may be extended in 8-bit increments. The control field follows the address field and is normally eight bits, but it may also be extended in 8-bit increments. The information field can be of any length and contains the data which is to be transmitted. The ADLC automatically calculates and appends a cyclic redundancy check character (CRCC). The polynomial used is $x^{16} + x^{12} + x^5 + 1$. A closing flag (01111110) is appended after the frame check sequence field.

The ADLC contains two read-only status registers, four write-only control registers, a receive data register, a transmit data register, and a transmit last data register. See Table 1 for an internal register structure and Table 2 for a register addressing map. The ADLC functions similar to other serial transmission devices. A receive data available bit and a transmit data register empty bit are monitored. The receive data register is read or the transmit data register is written when the respective bits are asserted. Unlike the MC6850 ACIA, the transmit shift register must always be kept full. The transmit FIFO must be written at a sufficient rate to insure that a transmitter underflow never occurs. Likewise, the receive data register must be read often enough so that a receive overrun does not occur.

The application of SDLC protocol and the MC6854 is considerably more complex than the explanation presented here; however, the basic ideas apply. The main purpose of the ADLC in this application note is to demonstrate a peripheral device using a DMAC in an MC6809-MC6829 system.

The MC6844 direct memory access controller (DMAC) performs the function of transferring data directly between memory and peripheral device controllers (See Figure 12 for block diagram). It accomplishes this transfer by supplying a bus address (instead of being supplied by the MPU) and controlling the data bus such that a memory or peripheral device drives the data bus. Only consecutive-memory-locations-to-peripheral or peripheral-to-consecutive-memory-locations transfers can be handled. The transferred data does not go through the DMA controller. There are four independent

channels, all of which can operate in any of three modes; i.e., TSC steal, halt-steal, and halt-burst. The timing diagrams for these three modes are presented in Figures 13, 14, and 15. To fully understand a typical transfer, a study of the internal DMAC registers, shown in Tables 3 and 4, must be made. For each channel, the starting address for the transfer and the number of bytes to be sent are loaded into the respective address high/low and byte count high/low registers. A control register is associated with each channel to specify the transfer mode, the direction of the data transfer, and if the data is to be accessed in ascending or descending order. The priority control register is used to enable transfer requests (TxRQ 0-3). It also causes the channels to rotate in priority (channel 0, then 1, and 2, . . .) or have a fixed priority (channel 0 is highest and channel 3 is lowest). The interrupt control register is used to enable the $\overline{\text{TRQ/DEND}}$ signal for each individual channel. A data chain enable register allows the address and byte count registers, for either channel 0, 1, or 2, to be loaded with the contents of the channel 3 register when the chained channel byte count register is decremented to \$0000. The data chain register also selects either two or four channels to be active.

A typical sequence for data transfer for the TSC steal mode (see Figure 13) begins with a peripheral device asserting its DMA service request which would be connected to a TxRQ (transfer request) input to the MC6844 DMAC. The high TxRQ input will be recognized within one E clock cycle. The $\overline{\text{DRQ1}}$ output of the DMAC is connected to bus request ($\overline{\text{DMA/BREQ}}$) of an MC6809 and it will be asserted during the cycle in which TxRQ is recognized; see Figure 4. The DMAC activates its address bus during the following cycle while the MPU places its bus in the high-impedance state and issues a bus grant to the DGRNT pin of the DMAC. This cycle is known as a dead cycle. Following the dead cycle is a DMA transfer cycle. During this cycle, the DMAC outputs the address for the data (to be written or read), asserts the proper state on the read/write line, and asserts transfer strobe ($\overline{\text{TxSTB}}$). The transfer strobe acts similar to a chip

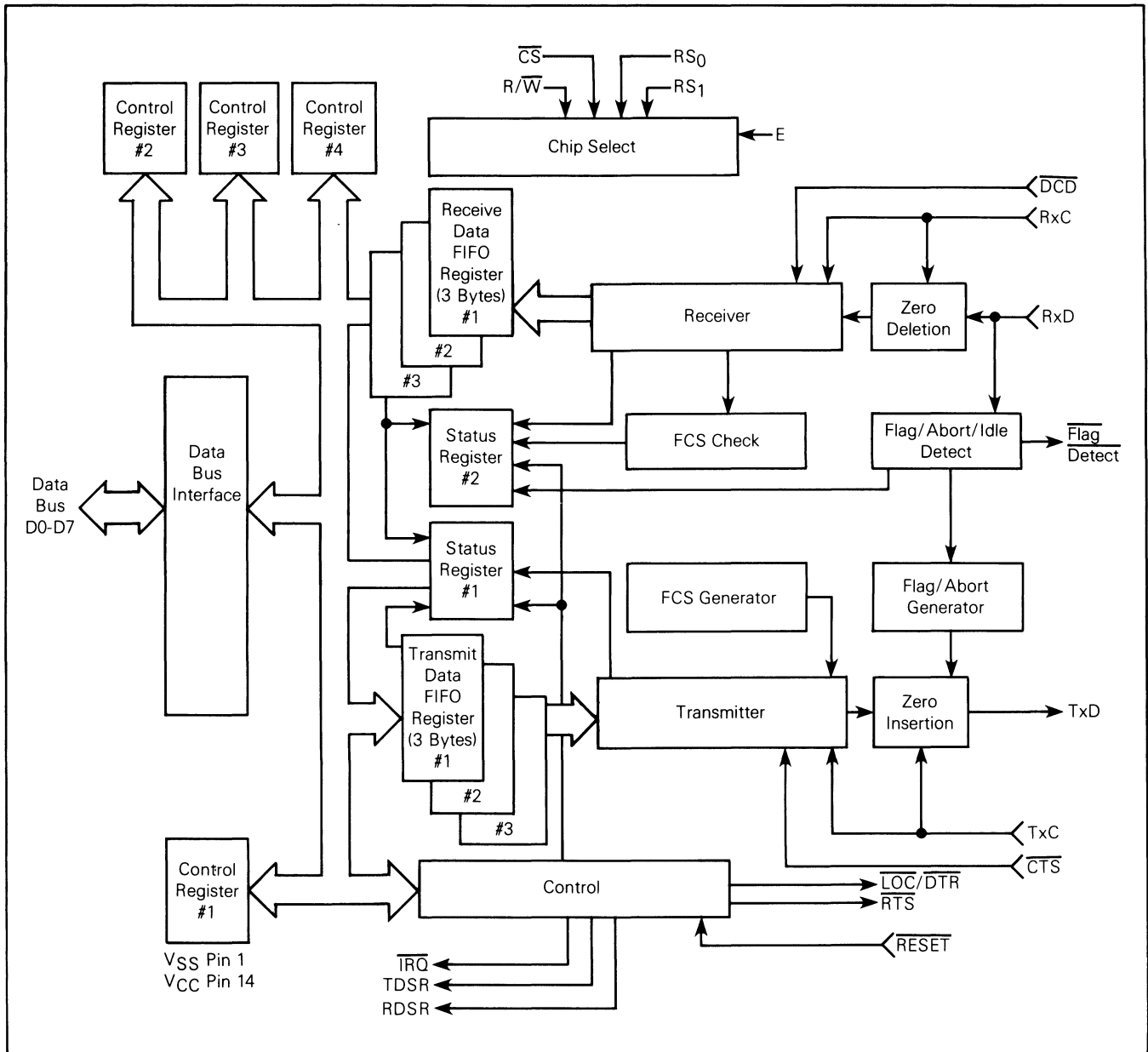


FIGURE 10 — MC6854 ADLC Block Diagram

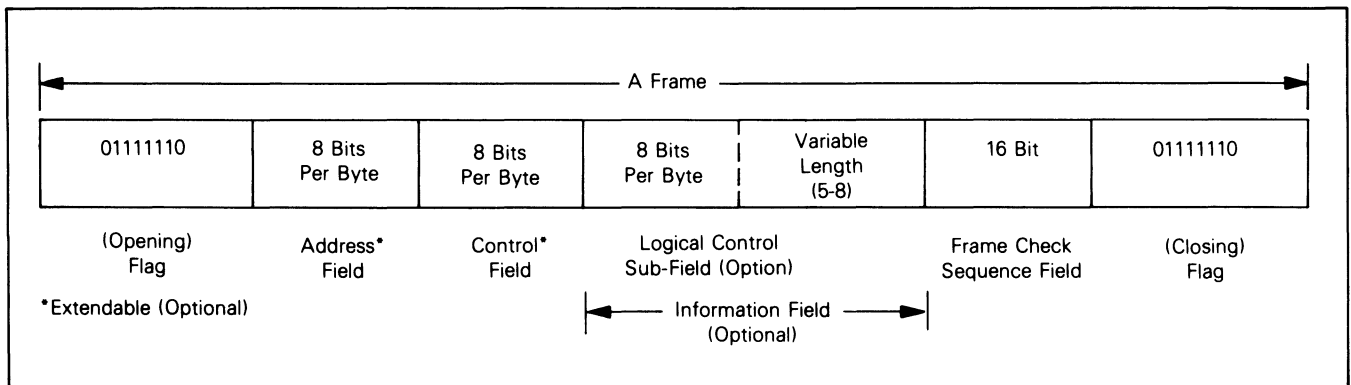


FIGURE 11 — Data Format of a Frame

TABLE 1 — ADLC Internal Register Structure

	Bit #	RS1 RS0 = 00	RS1 RS0 = 01	RS1 RS0 = 10	RS1 RS0 = 11
		Status Register #1	Status Register #2	Receiver Data Register	
Read Only Registers	0	RDA	Address Present	Bit 0	Same as RS1, RS0 = 10
	1	Status #2 Read Request	Frame Valid	Bit 1	
	2	Loop	Inactive Idle Received	Bit 2	
	3	Flag Detected (When Enabled)	Abort Received	Bit 3	
	4	\overline{CTS}	FCS Error	Bit 4	
	5	Tx Underrun	\overline{DCD}	Bit 5	
	6	TDRA/Frame Complete	Rx Overrun	Bit 6	
	7	IRQ Present	RDA (Receiver Data Available)	Bit 7	

	Bit #				Transmitter Data	Transmitter Data	
		Control Register #1	Control Register #2 (C1b0=0)	Control Register #3 (C1b0=1)	(Continue Data)	(Last Data) (C1b0=0)	Control Register #4 (C1b0=1)
Write Only Registers	0	Address Control (AC)	Prioritized Status Enable	Logical Control Field Select	Bit 0	Bit 0	Double Flag/Single Flag Interframe Control
	1	Receiver Interrupt Enable (RIE)	2 Byte/1 Byte Transfer	Extended Control Fixed Select	Bit 1	Bit 1	Word Length Select Transmit #1
	2	Transmitter Interrupt Enable (TIE)	Flag/Mark Idle	Auto, Address Extension Mode	Bit 2	Bit 2	Word Length Select Transmit #2
	3	RDSR Mode (DMA)	Frame Complete/ TDRA Select	01/11 Idle	Bit 3	Bit 3	Word Length Select Receive #1
	4	TDSR Mode (DMA)	Transmit Last Data	Flag Detected Status Enable	Bit 4	Bit 4	Word Length Select Receive #2
	5	Rx Frame Discontinue	CLR Rx Status	Loop/Non-Loop Mode	Bit 5	Bit 5	Transmit Abort
	6	Rx RESET	CLR Tx Status	Go Active on Poll/Test	Bit 6	Bit 6	Abort Extend
	7	Tx RESET	RTS Control	Loop On-Line Control DTR	Bit 7	Bit 7	NRZI/NRZ

TABLE 2 — Register Addressing

Register Selected	R/W	RS1	RS0	Address Control Bit (C1b0)
Write Control Register #1	0	0	0	X
Write Control Register #2	0	0	1	0
Write Control Register #3	0	0	1	1
Write Transmit FIFO (Frame Contine)	0	1	0	X
Write Transmit FIFO (Frame Terminate)	0	1	1	0
Write Control Register #4	0	1	1	1
Read Status Register #1	1	0	0	X
Read Status Register #2	1	0	1	X
Read Receiver FIFO	1	1	X	X

select signal to the peripheral device which is supplying/receiving the data. Following the DMA transfer cycle is a dead cycle which allows the MPU to regain the address bus and the DMAC returns its bus to the high-impedance state.

The halt steal mode is very similar to the TSC steal mode (refer to Figure 14). In this case, $\overline{DRQ2}$, which drives the \overline{HALT} pin of the MPU, is asserted in response to TxRQ. When the current instruction has finished execution, a dead cycle occurs and the processor issues a bus grant (DGRNT). The TxRQ signal must remain valid until after the falling edge of E of the cycle preceding the DMA transfer cycle. A DMA cycle and second dead cycle follow. The $\overline{DRQ2}$ line is released during the DMA cycle.

The halt burst mode is identical to halt steal with one major exception (refer to Figure 15). The $\overline{DRQ2}$ output (driving \overline{HALT} of the MPU) is not released until the byte count has been decremented to \$0000, indicating that all transfers have taken place. As shown in Figure 15, the TxRQ

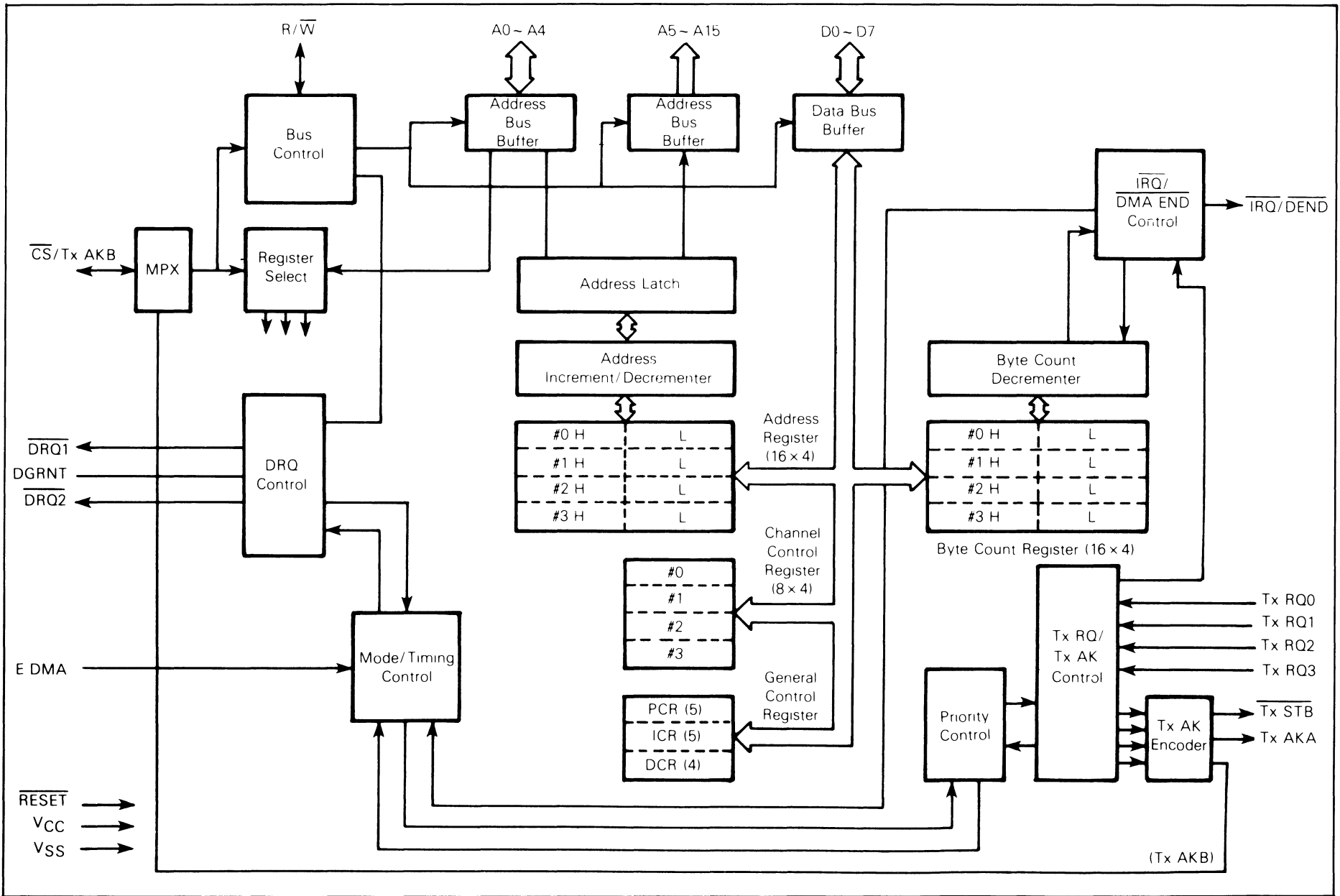


FIGURE 12 — DMAC Block Diagram

TABLE 3 — DMAC Control Registers

Register	Address (Hex)	Register Content							
		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Channel Control	1x*	DMA End Flag (DEND)	Busy/Ready Flag	Not Used	Not Used	Address Up/Down	MCA	MCB	Read/Write (R/W)
Priority Control	14	Rotate Control	Not Used	Not Used	Not Used	Request Enable #3 (RE3)	Request Enable #2 (RE2)	Request Enable #1 (RE1)	Request Enable #0 (RE0)
Interrupt Control	15	DEND IRQ Flag	Not Used	Not Used	Not Used	DEND IRQ Enable #3 (DIE3)	DEND IRQ Enable #2 (DIE2)	DEND IRQ Enable #1 (DIE1)	DEND IRQ Enable #0 (DIE0)
Data Chain	16	Not Used	Not Used	Not Used	Not Used	Two/Four Channel Select (2/4)	Data Chain Channel Select B	Data Chain Channel Select A	Data Chain Enable

*The x represents the binary equivalent of the channel desired.

TABLE 4 — Address and Byte Count Registers

Register	Channel	Address (Hex)
Address High	0	0
Address Low	0	1
Byte Count High	0	2
Byte Count Low	0	3
Address High	1	4
Address Low	1	5
Byte Count High	1	6
Byte Count Low	1	7
Address High	2	8
Address Low	2	9
Byte Count High	2	A
Byte Count Low	2	B
Address High	3	C
Address Low	3	D
Byte Count High	3	E
Byte Count Low	3	F

line is sampled on the following edge of E for each DMA cycle. If TxRQ is high, a transfer takes place; if low, a dummy cycle occurs. The byte count and address registers are not affected by a dummy cycle. Remember, once a halt burst mode has begun, the MPU cannot regain control of the bus until the entire transfer has taken place. This problem can be circumvented by removing DGRNT from the DMAC (thus HALT from the MPU) with external circuitry, then writing the byte count register of the active channel to \$0000.

Asynchronous communications to a local terminal are provided by the MC6850 asynchronous communications interface adapter (ACIA). See Figure 16 for block diagram. This device can be operated in full duplex at speeds up to one megabit per second. Four registers are provided: a transmit data register, a receive data register, a control register, and a status register. The definition of the ACIA register contents is presented in Table 5.

In order to activate the communications link via the ACIA, the control register must be configured. From a power fail/restore or a power-on condition, the ACIA should be placed in a master reset condition. This is done by

writing the lower two bits of the control register (CR0-CR1) to 1s, as shown in Table 6.

A master reset clears the status register except for data carrier detect (DCD) or clear to send (CTS); however, it does not affect any control register bit. In addition to providing master reset, control register bits CR0 to CR1 are also used to select the clock divide ratio for both the transmitter and receiver sections (also shown in Table 6).

Three bits are used as word select bits (CR2, CR3, and CR4) and the configuration of these bits select word length, parity, and number of stop bits. The encoding format is as shown in Table 7.

Two transmitter control bits (CR5 and CR6) provide for the control of the interrupt from a transmit data register empty condition, the request-to-send (RTS) output, and the transmission of a break level. The encoding format and controlled function are as shown in Table 8.

The remaining bit, CR7, is the receive interrupt enable. It allows interrupts caused by the following conditions to be reflected at the IRQ output: receive data register full, receiver overrun, or a positive transition on the data carrier detect (DCD) signal line.

The status register bits provide information on the status of the ACIA; i.e., the conditions present. The bits of the status register are shown in Table 5.

The overall functioning of the MC6850 ACIA can be understood by an examination of the block diagram shown in Figure 16 together with the registers shown in Table 5.

The control register is loaded with the appropriate information to configure the device. When ready to transmit data, the status register is checked to see that the CTS input is low and the transmit data register is empty. Once both of these conditions are met, a data byte can be written to the ACIA transmit data register. Incoming data will then set the receive data register full (RDRF) flag in the status register. Upon polling the status register and finding the RDRF bit set, the data carrier detect (DCD), framing error (FE), receiver overrun (OVRN), and parity error (PE) bits should be examined. If no errors are found, the receive data can be read from the receive data register.

Memory expansion to 2 megabytes is made possible by use of the MC6829 memory management unit (MMU). The MMU allows the system to address 32 blocks of memory in 2 kilobyte increments for a total of 64 kilobytes. These blocks are not required to be in any certain order or to be contiguous. Different sections of memory can be accessed by

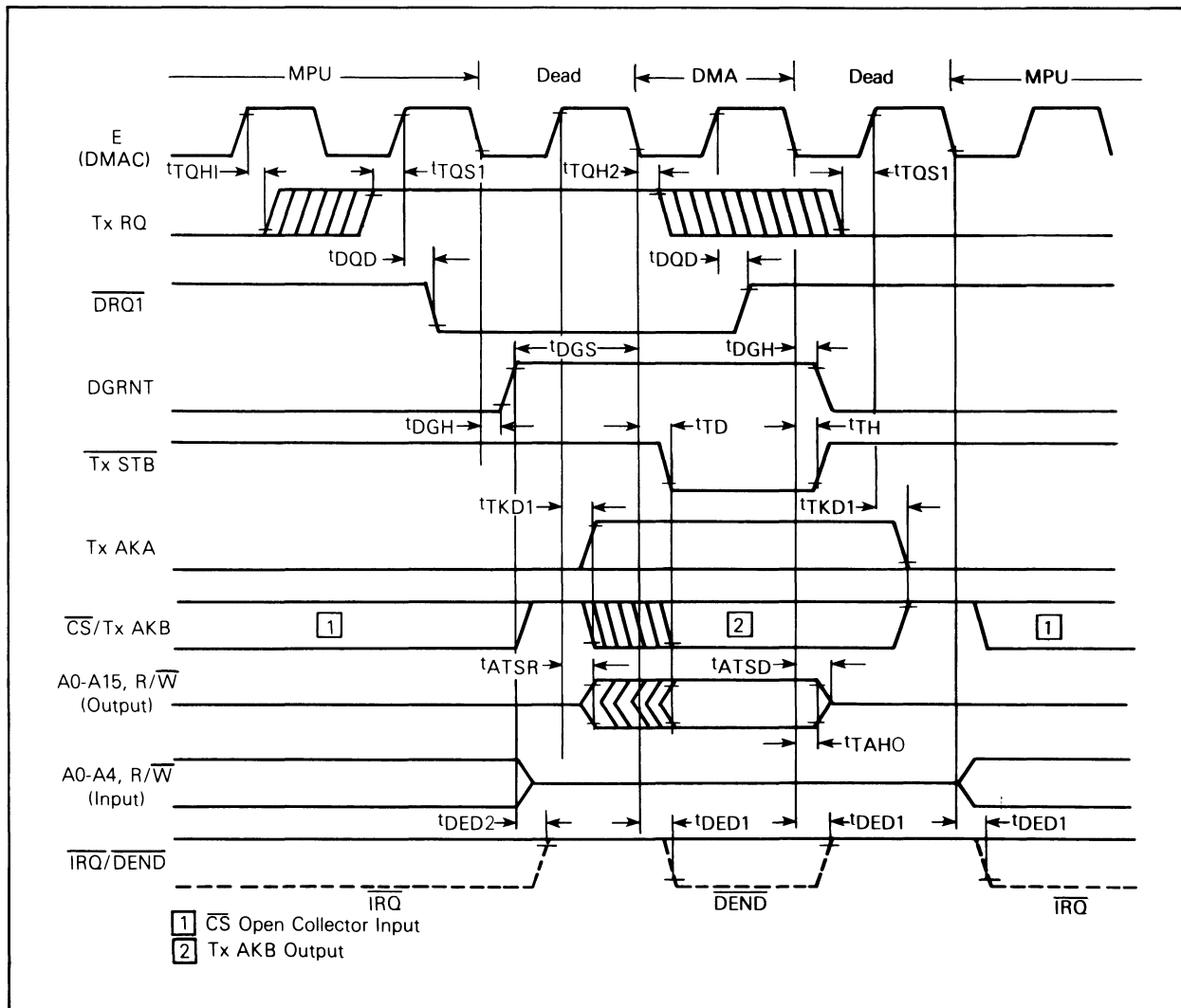


FIGURE 14 — Mode 2 Timing (Halt-Steal Mode)

registers function normally with one exception. When an MMU register is accessed, the physical address bus (PA10-PA20) is driven to all 1s if $S = 1$.

The access key allows the MPU to address the internal registers of the MMU. The upper three bits of this 5-bit register must match the key value register and the S bit must be set before the MMU register can be changed or examined. The lower two bits of the access key select the particular task to be modified.

The operate key register is active only when the S bit is not set and a DMA transfer is not occurring. This assumes that the MMU has been initialized by programming the appropriate registers and initializing the key value register. The upper three bits of the 5-bit operate key register of an MMU must match the key value register to select a particular MMU; otherwise, the physical address bus will be in the high-impedance state. The lower two bits are used to choose the task to be used.

The fuse register is used to determine the exact timing of a task switch. A task switch is always to or from task 0 except for DMA. A number is loaded into the fuse register, and when the fuse register is decremented to zero the MMU automatically switches to the task called for by the operate key. On each successive valid (non-DMA) processor cycle, the

fuse register is decremented by one. Consider the programming example shown in Figure 21.

A jump instruction takes 4 cycles to execute. On the 4th cycle after the jump opcode has been fetched, the fuse register will have been decremented to zero and the program counter will output the destination address of the jump. The next opcode fetch which occurs will be from task N. It is possible to jump to task 0 from task 0. This allows the system to use task 0 like any other task. By jumping to task 0 via the fuse register, the system will not be able to change the internal MMU registers until after an interrupt occurs.

INITIALIZATION OF THE MEMORY MANAGEMENT UNIT

Once the MC6809-MC6829 system is built it must be initialized. However, it is important to understand the state of the system when **RESET** is released. This includes:

1. **An internal MMU reset flag remains set** — This causes all of the physical address lines to be driven high (logical 1, not high-impedance state). This flag does not appear in any register and is cleared by writing to the key value register. It is important to have the physical address lines driven to a known state so that a specific section of memory can always be used for the system

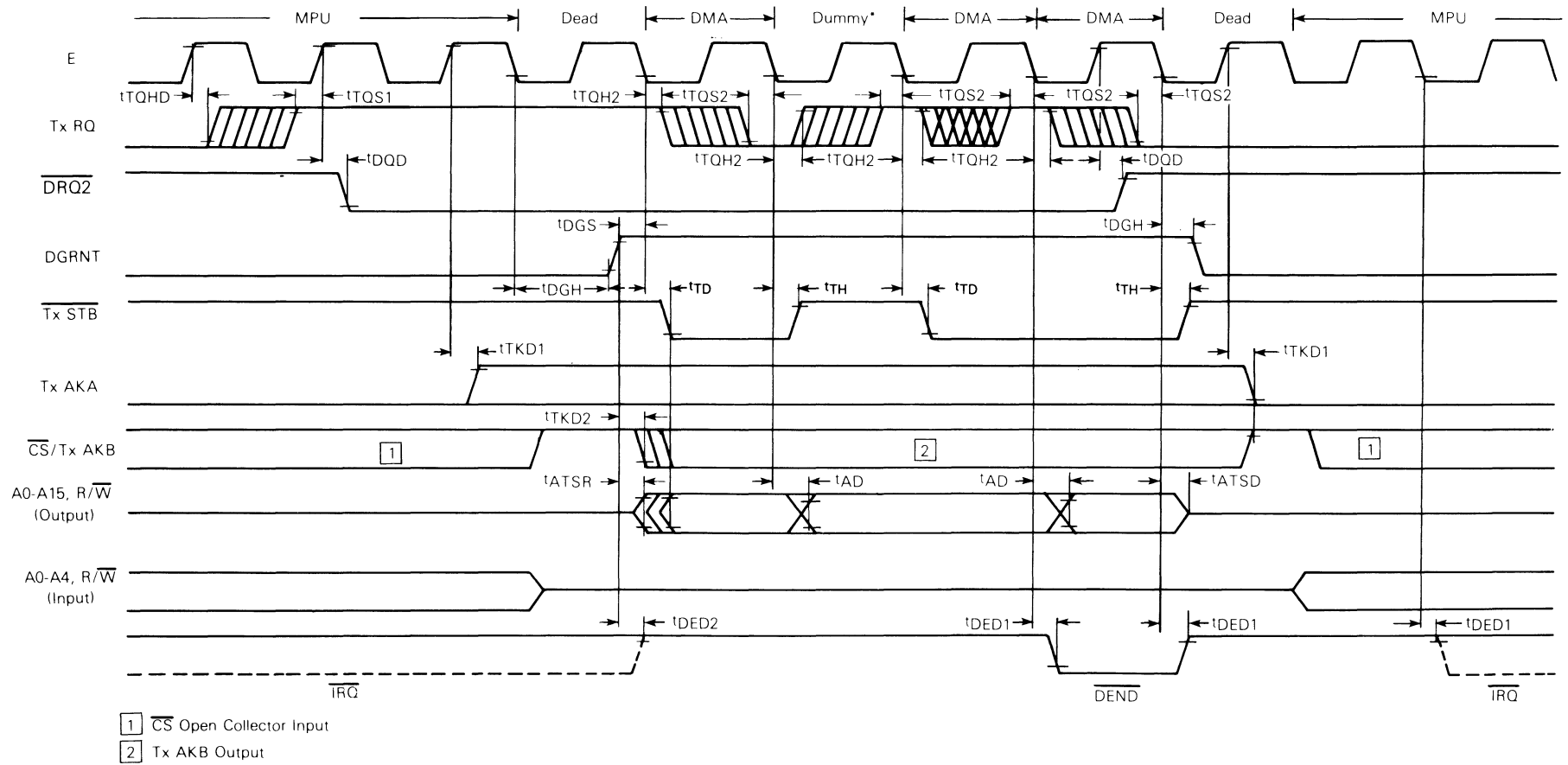


FIGURE 15 — Mode 3 Timing (Halt-Burst Mode)

TABLE 5 — Definition of ACIA Register Contents

Data Bus Line Number	Buffer Address			
	RS • R/W Transmit Data Register	RS • R/W Receive Data Register	RS • R/W Control Register	RS • R/W Status Register
	(Write Only)	(Read Only)	(Write Only)	(Read Only)
0	Data Bit 0*	Data Bit 0	Counter Divide Select 1 (CR0)	Receiver Data Register Full (RDRF)
1	Data Bit 1	Data Bit 1	Counter Divide Select 2 (CR1)	Transmit Data Register Empty (TDRE)
2	Data Bit 2	Data Bit 2	Word Select 1 (CR2)	Data Carrier Detect (DCD)
3	Data Bit 3	Data Bit 3	Word Select 2 (CR3)	Clear to Send (CTS)
4	Data Bit 4	Data Bit 4	Word Select 3 (CR4)	Framing Error (FE)
5	Data Bit 5	Data Bit 5	Transmit Control 1 (CR5)	Receiver Overrun (OVRN)
6	Data Bit 6	Data Bit 6	Transmit Control 2 (CR6)	Parity Error (PE)
7	Data Bit 7***	Data Bit 7**	Receive Interrupt Enable (CR7)	Interrupt Request (IRQ)

*Leading bit = LSB = Bit 0

**Data bit will be zero in 7-bit plus parity modes.

***Data bit is "don't care" in 7-bit plus parity modes.

initialization. In the first two cycles after **RESET** is released, the restart vectors are fetched. At this time, the MMU mapping registers are not used; however, by having PA20-PA11 driven high, the restart vector will always come from \$1FFFE-\$1FFFF.

2. **The fuse register is disabled** — A write to this register will have no effect until the key value register is written and the upper three bits of the operate key register and the key value register are equal.
3. **The system state bit (S bit) is set** — This bit must be set to access any MMU register. It is cleared when the fuse register is written and then decremented to zero which means task 0 is being exited. In order to set the S bit, the BA and BS lines must be 0 and 1, respectively. This occurs when the processor responds to **RESET**, **NMI**, **FIRQ**, **IRQ**, **SWI**, **SWI2**, or **SWI3**. If the system is operating with the S bit clear and the program needs to change an MMU register, the program must execute a software interrupt or cause some external device to input a hardware interrupt.
4. **The operate key register and access key register are both cleared** — Even though contents of these two registers will match the key value register, the internal flag still causes the physical address lines to be high. Not until a byte is written to the key value register will the mapping registers be used for address generation.
5. **The key value register is also cleared after reset** — This register must be written to clear the internal reset bit. This is true even if \$00 is to be written to the key value register.

To initialize a system from reset, the key value register of MMUs 1-7 must be written. Figure 22 is an example of eight MMUs in a single system. For convenience, MMU1 will have \$01 written to its key value register, MMU2 will have \$02, . . . , and MMU7 will have \$07. This causes their

physical address buses to assume a high-impedance condition. The program must be careful not to write the key value register of MMU0 until enough of its mapping registers, in task 0, have been programmed to define the address space. By not writing to the key value register of MMU0, the MPU will stay in the range of \$1FF800-\$1FFFFFF. Remember the physical address lines remain high until the key value register is written in MMU0 clearing its internal reset bit. With the mapping registers of MMU0 task 0 programmed, its (MMU0) key value register may be written.

The MC6809-MC6829 system is now using the mapping registers of MMU0, task 0 to generate physical addresses. The following conditions exist:

1. The mapping registers (32 mapping registers per task, 4 tasks per MMU, 8 MMUs per system) have been programmed to the desired values.
2. The system state bit (S bit) is set in all MMUs.
3. All internal reset bits are clear. This is a mandatory condition to use the mapping registers.
4. Each of the MMU key value registers contains a unique 3-bit value. (Writing the key value register clears the internal reset bit.)
5. The access key and operate key registers are the same for all MMUs.
6. The fuse register in all MMUs are enabled.
7. The system is operating in task 0, MMU0.

Now consider the following program (see Figure 23). The microprocessor stores the A register to the key value register of MMU0. The lower 11 address lines of the microprocessor drives the physical memory directly. As the microprocessor program counter steps from \$F850 through \$F852, the 10 physical address lines are high. This means the 21-bit physical address (A0-A10 of the MPU and PA11-PA20 of the MMU) is \$1FF850 through \$1FF852. As soon as the key value register is written (physical address \$1FF852), the mapping

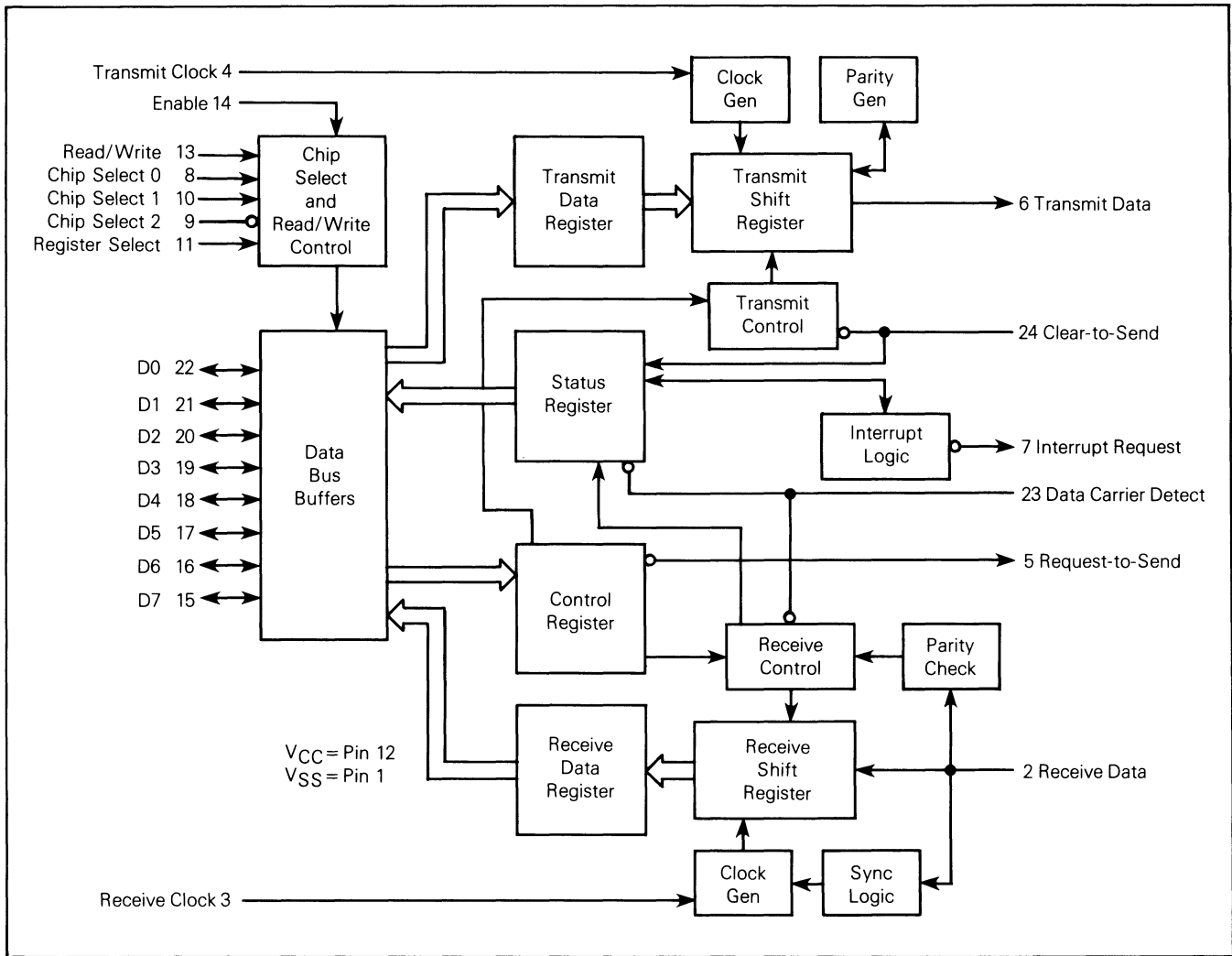


FIGURE 16 – ACIA Block Diagram

TABLE 6 – Counter Divide Select Bits (CR0-CR1)

CR1	CR0	Function
0	0	+ 1
0	1	+ 16
1	0	+ 64
1	1	Master Reset

TABLE 8 – Transmitter Control Bits (CR5 and CR6)

CR6	CR5	Function
0	0	RTS= Low, Transmitting Interrupt Disabled
0	1	RTS= Low, Transmitting Interrupt Enabled
1	0	RTS= High, Transmitting Interrupt Disabled
1	1	RTS= Low, Transmit a Break level on the Transmit Data Output. Transmitting Interrupt Disabled

TABLE 7 – Word Select Bits (CR2, CR3, and CR4)

CR4	CR3	CR2	Function
0	0	0	7 Bits+ Even Parity+ 2 Stop Bits
0	0	1	7 Bits+ Odd Parity+ 2 Stop Bits
0	1	0	7 Bits+ Even Parity+ 1 Stop Bit
0	1	1	7 Bits+ Odd Parity+ 1 Stop Bit
1	0	0	8 Bits+ 2 Stop Bits
1	0	1	8 Bits+ 1 Stop Bit
1	1	0	8 Bits+ Even Parity+ 1 Stop Bit
1	1	1	8 Bits+ Odd Parity+ 1 Stop Bit

TABLE 9 – Condition Codes Bits Set by Interrupts

Interrupt	I Bit	F Bit
NMI	S	S
FIRQ	S	S
IRQ	S	*
SWI	S	S
SWI2	*	*
SWI3	*	*

*Indicates interrupt has no effect on this bit.

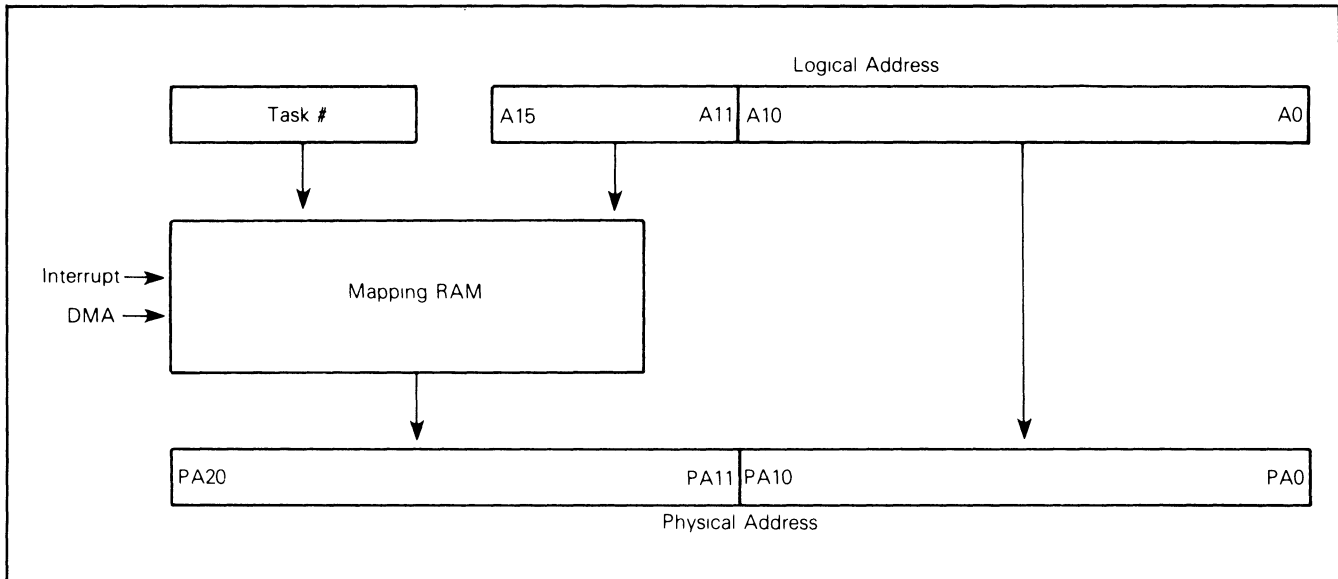


FIGURE 17 — Logic-To-Physical Address Translation Diagram

register becomes active; therefore, when the MPU program counter steps to \$F853, the 21-bit physical address is composed of PA20-PA11 of the mapping register selected by the upper five address lines of the MPU and A10-A0 of the MPU. In this example the upper five address lines of the MPU are all high, selecting the highest order mapping register. The operate key register still contains \$00 since it has never been written and, thus, its contents match the MMU0 key value register. The system is still in task 0 because of the S bit is set and the operate key register is not used. As a matter of convenience and logical program flow, the high order mapping register should contain all 1s so that the next instruction following the write to the key value register (Figure 23) will be fetched from the next physical memory location; i.e., \$1FF853. This is by no means necessary; it is just convenient and will make system software easier to follow.

An examination of the MMU registers show all needed mapping registers are programmed and each key value register contains a unique value. The S bit is set meaning that task 0 of MMU0 is being used. The access key register will contain the key value and task number of the last MMU in which the registers were modified. The operate key will contain the key value and task number that will be used when the program jumps from the operating system. It is absolutely essential to note that as long as the S bit is set, only task 0, of the MMU in which the key value register is \$00 can be used. Only a DMA transfer indicated by BA = 1, BS = 1 will override this condition. Then task 1 of MMU0 is used.

To switch from task 0, the fuse register of all MMUs must be written. This is the only method that can be used to leave task 0 except for DMA cycles. The fuse register is loaded with the number of cycles which must occur after the write to the fuse register until task N is entered. The instruction following the write to the fuse register will generally be a branch, jump, or return from interrupt. The fuse register will be decremented on each successive valid (non-DMA) processor cycle. When the fuse register reaches zero, the system will be operating in task N.

INTERRUPT HANDLING

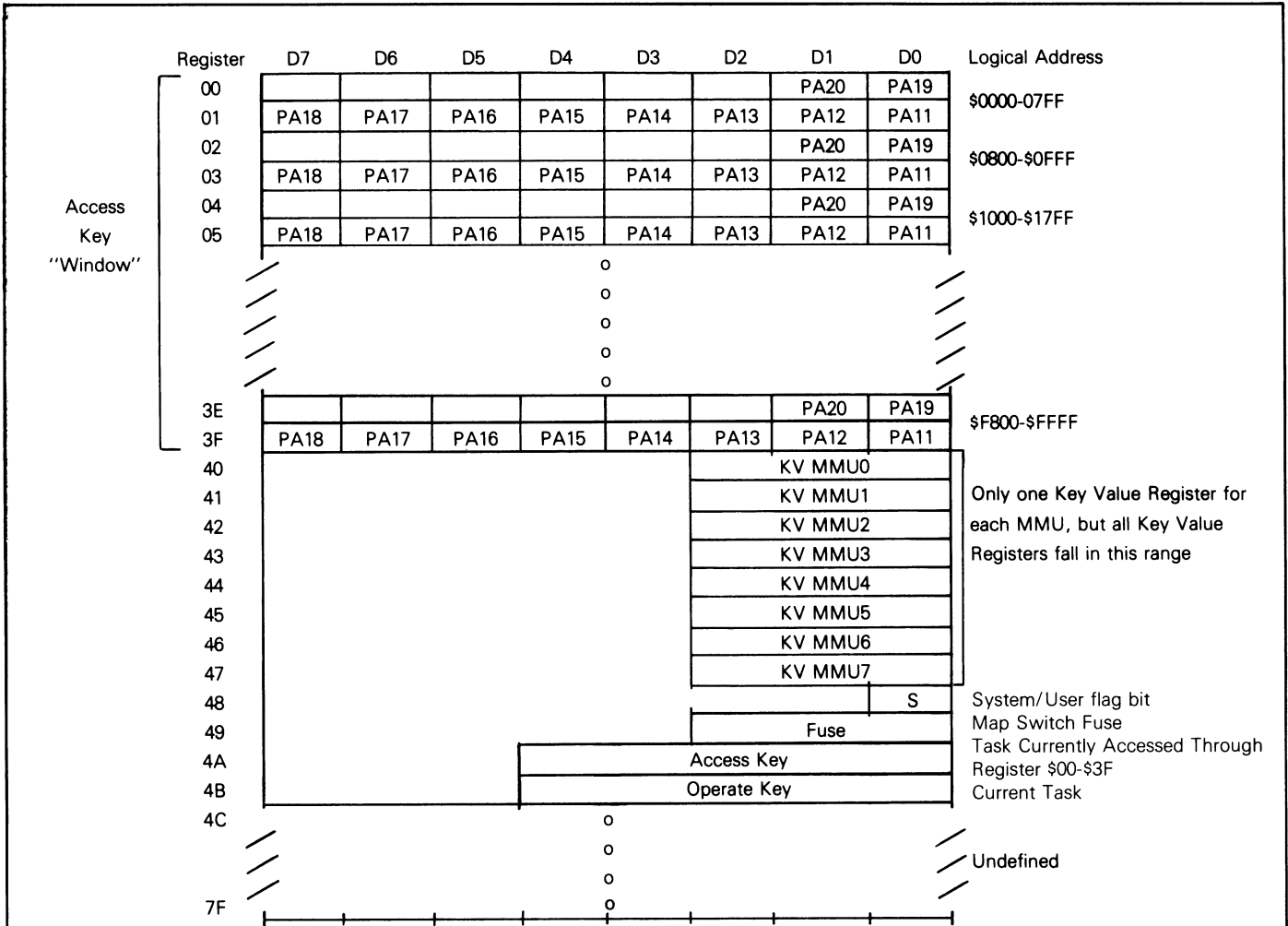
If an interrupt is received by the processor, it will switch automatically to task 0 as soon as the stacking operation is

complete. The stacking operation interrupt timing for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$ is presented in Figure 24 and is shown in Figure 25 for $\overline{\text{FIRQ}}$. The stack pointer is not saved on the stack; therefore, it is the responsibility of the operating system in task 0 to retrieve and store the stack pointer from the task that was just exited. Normally this would be fairly straightforward. Special cases arise when one or more interrupts are recognized during the stacking operation for a current interrupt. For recognition of an additional interrupt to occur, the second interrupt must be valid beginning with cycle 16 of an $\overline{\text{NMI}}$, $\overline{\text{IRQ}}$, SWI, SWI2, or SWI3 stacking operation. Recognition must occur by cycle 7 of an $\overline{\text{FIRQ}}$ stacking operation. For $\overline{\text{NMI}}$ to be recognized, only a transition must be detected. Once the negative transition is recognized, the logic level on the $\overline{\text{NMI}}$ pin is no longer important. However, $\overline{\text{IRQ}}$ and $\overline{\text{FIRQ}}$ must remain at a logic 0 level for the three cycles immediately preceding cycle 16 or cycle 17 of the particular stacking operation or they will not be recognized.

If another stacking operation begins before the stack pointer for the prior task has been stored, data in task 0 may be lost. This means that the program will not be able to find its way back to task N. A method must be found to hold off other interrupts until information can be stored which directs the processor back to task N. An understanding of the interrupts and how they work is needed to develop a simple system of allowing multiple interrupts. The interrupts set the I and F bits according to Table 9.

If $\overline{\text{NMI}}$ occurs first, it will set both interrupt bits of the condition code register, masking further hardware interrupts. The only way for a software interrupt to occur before the stack pointer, from the prior task, could be saved, is for a software interrupt to be the first instruction of the $\overline{\text{NMI}}$ service routine. By causing the $\overline{\text{NMI}}$ service routine to immediately save the task N stack pointer, this problem will never be encountered in returning to task N.

An $\overline{\text{FIRQ}}$ is the next highest priority hardware interrupt. If software interrupts are treated the same as with $\overline{\text{NMI}}$, then SWIs will not be a problem. However, if $\overline{\text{NMI}}$ is recognized before the stack pointer is stored, that data will probably be lost. Since the $\overline{\text{NMI}}$ input is level sensitive, there is nothing that can be done to hold off an $\overline{\text{NMI}}$ once it occurs. This means that there is no way to absolutely guarantee that the



Notes:

1. The contents of bytes \$4C through \$7F are undefined and do not respond to any reads or writes.
2. The Access, Operate and Key Value Registers are cleared on reset. The S-bit is set.
3. Unused bits of defined registers always read zeros.
4. Locations \$40-\$47 are accessible only when $\overline{KVA} = 0$.
5. In multiple MMU configurations, the MMU whose Key Value Register matches the upper three bits of the access key will respond to a processor read of locations \$48-\$4B. Processor writes to these registers will cause the data to be written to all MMUs simultaneously.

FIGURE 18 — MMU Register Model

return information can be saved if \overline{NMI} s are allowed. An \overline{FIRQ} does set both interrupt bits. Thus it will automatically hold off an \overline{IRQ} should it occur.

An \overline{IRQ} only sets the I bit of the condition code register. This does not allow it to automatically mask an \overline{FIRQ} if it occurs. Since both an \overline{IRQ} and \overline{FIRQ} must remain valid for three cycles to be recognized, a hardware property of the MC6809 can be employed to hold off \overline{FIRQ} . If an interrupt signal is ORed with BS before connection to the \overline{FIRQ} input, then \overline{FIRQ} will never be recognized during a stacking operation since \overline{FIRQ} will be removed during cycles 17 and 18 of the stacking operation for \overline{IRQ} . This will allow the \overline{IRQ} service routine to execute at least one instruction. If that instruction sets the F bit in the condition code register, \overline{FIRQ} will successfully be masked. The interrupt routine can enable interrupts when it has saved all necessary information.

An SWI sets both the I and F bits. This keeps \overline{IRQ} and \overline{FIRQ} from preventing the return information from being

lost. An \overline{NMI} cannot be held off; therefore, the system should be designed such that \overline{NMI} s occur only in response to system crashes or the like.

Software interrupts SWI2 and SWI3 do not set either the I or F bits. This means that either \overline{IRQ} or \overline{FIRQ} can prevent the return information from being saved. By ORing \overline{IRQ} and \overline{FIRQ} with BS before applying them to their respective inputs on the MC6809, sufficient time will be allowed to mask off the I and F bits in the condition code register by SWI2 and SWI3. An \overline{NMI} cannot be handled safely as in the case of all other interrupts. Once again, it must be emphasized that there is not a software method to disable \overline{NMI} ; therefore, its use, except for system crash recovery, should be discouraged.

Generally speaking, \overline{NMI} can be successfully handled while operating in task 0. This is because there is no task switch since the program is already in task 0. The stack pointer will remain valid as long as no task switch takes place. If hardware is built to disable \overline{NMI} and if the system is

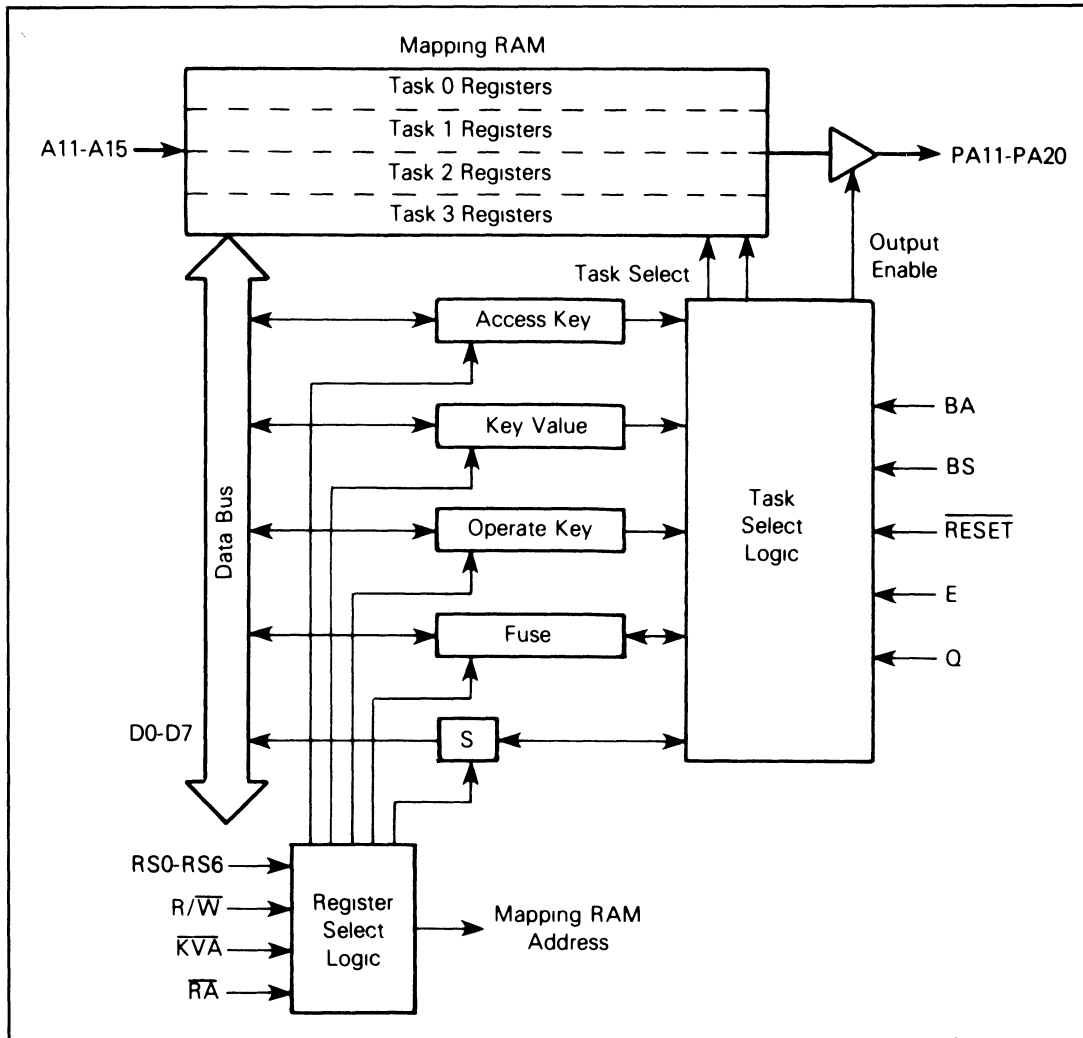


FIGURE 19 — MC6829 MMU Block Diagram

in any task but task 0, then $\overline{\text{NMI}}$ cannot be used to recover from system crashes. Designers must be cautioned again that $\overline{\text{NMI}}$ should not be used for program control in systems using the MC6829 unless all possible avenues for the program to lose the return information have been explored and remedied.

TASK SWITCHING

The operating system must eventually execute an RTI instruction to allow the processor to reload the MPU registers. The map switch must occur after the opcode for the RTI is fetched and before the first register is pulled from the stack. Prior to the RTI, the operating system must reload the saved stack pointer for the task about to run. There must be no interrupts from the time the stack pointer is reloaded until the RTI is executed. An interrupt here would cause the system to treat the task N stack pointer as the task 0 stack pointer. The signal to the MMU that the map should be returned to the user task is noted by a write to the fuse register. When a write to this register is detected, the value written is loaded into the counter and it begins to decrement by one for every non-DMA processor cycle. When the counter underflows (reaches zero), the S bit is cleared and the next processor cycle will be mapped using the task number in the operate key. For most systems, a 1 would be written to

the fuse register immediately before the RTI opcode is executed. This delay allows enough time for the RTI opcode to be fetched (registers are not pulled from the stack until the third cycle of the RTI). Note that DMA operations are still possible within this critical section. The fuse register will count only non-DMA cycles after the write to the fuse register in order to be sure of when to switch the map. Bus dead cycles are also excluded when clocking the fuse register. Thus, the fuse register is inhibited from counting whenever BA is high and for the cycle after the BA high to low transition. The common exit point for all operating system functions is detailed in the programming example shown in Figure 26.

The I and F bits of the condition code register are used to mask interrupts during the RTI (return from interrupt) instruction. When the operating system masks and unmask these bits, no problem is encountered in properly restoring the MPU registers using an RTI instruction. Any time an interrupt occurs, the E bit of the condition code register will be properly set and then the MPU registers will be stacked in the current task. The last register stacked is the condition code register. The condition code register is the first one pulled from the stack upon return. The E and F bits are immediately examined to determine if a return from a fast interrupt is being executed or if the entire status needs to be

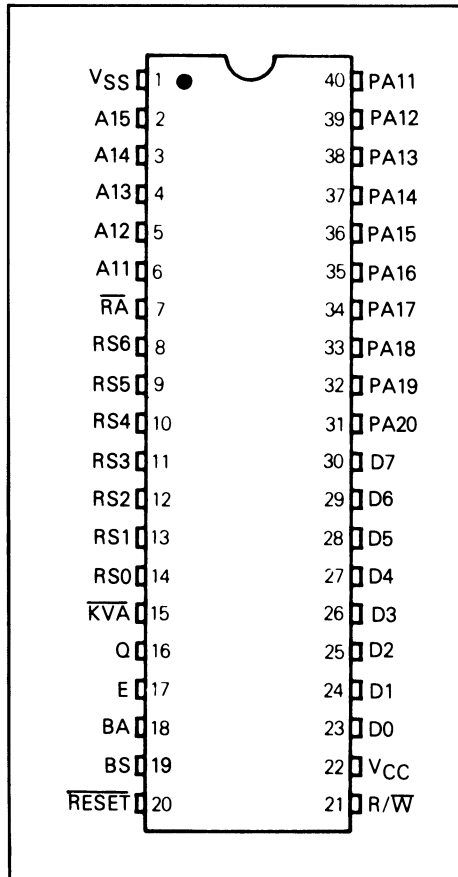


FIGURE 20 — MC6829 MMU Pin Assignment

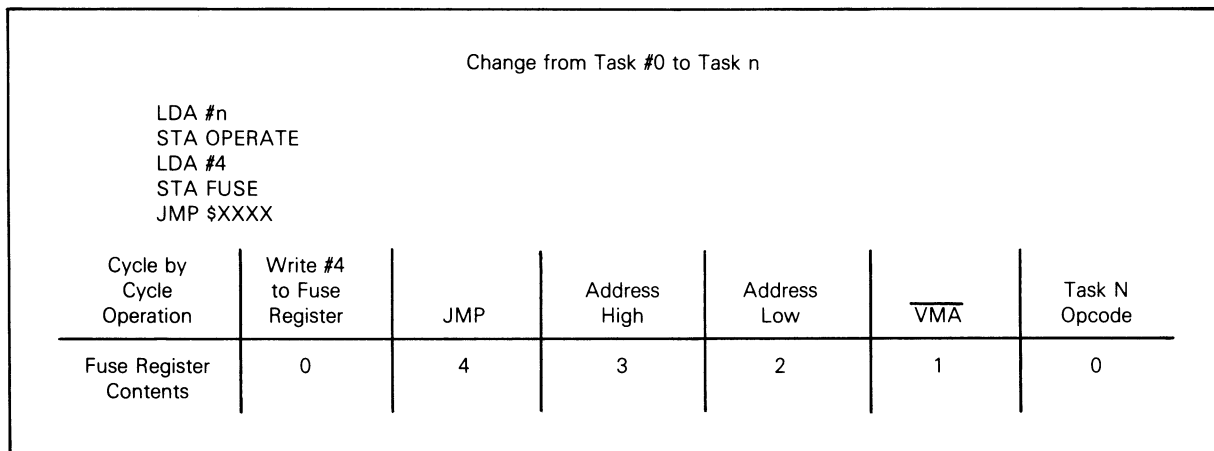


FIGURE 21 — Programming Example to Change from Task 0

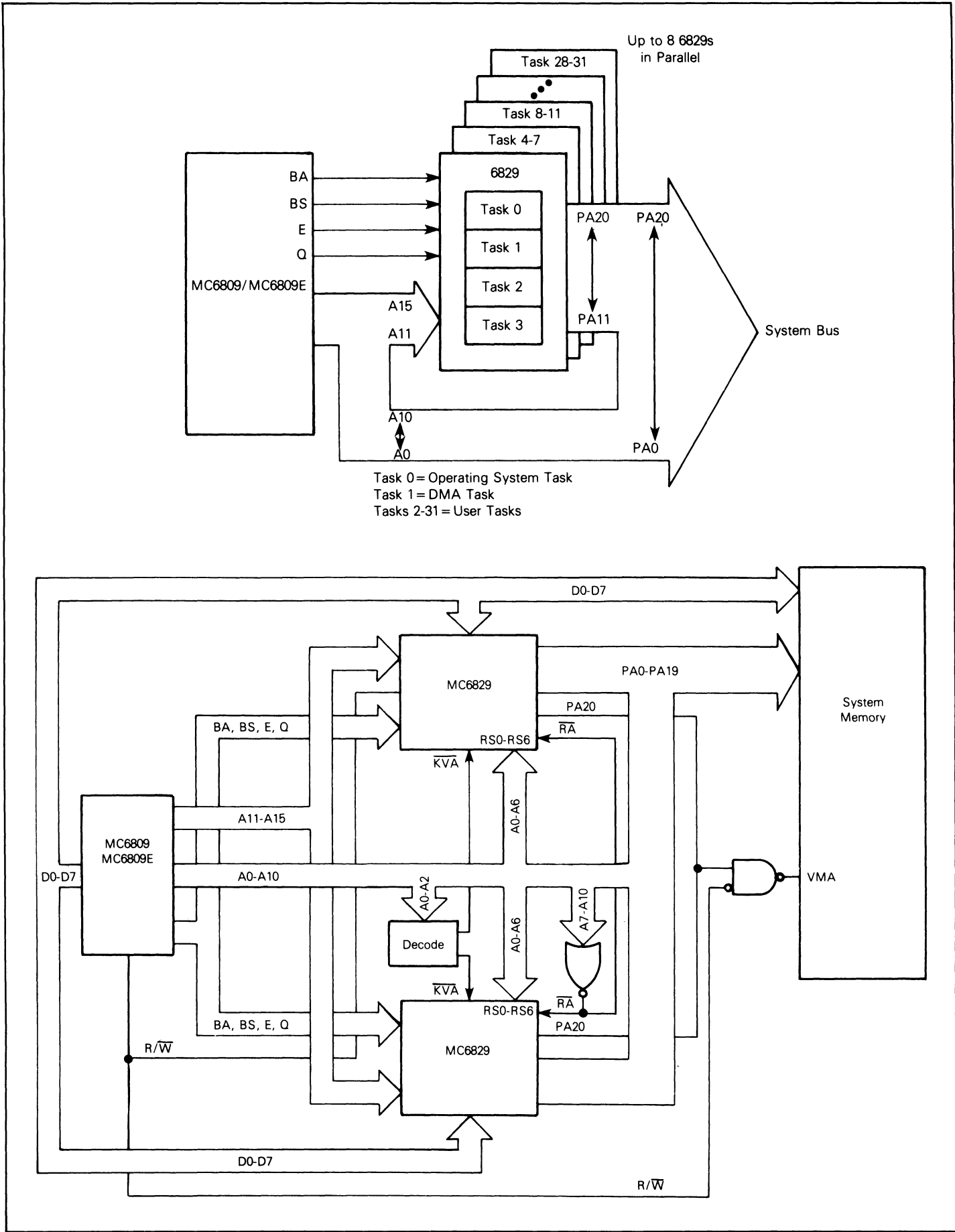


FIGURE 22 – MMU System Configuration

F84E	LDA	86	#\$00	
F850	STA	B7	\$F840	Write MMU0 Key Value Register 1 PA11-PA20= 1 Physical Address= \$1FF850
F853	Next Instruction			PA11-PA20= Contents of Register 3F of MMU0, Task 0 Physical Address= \$1FF853 if 3F contains \$03FF

FIGURE 23 — Program for MMU0 and Key Value Initialization

retrieved. This allows the MPU to pull the proper number of bytes from the stack.

PROGRAMMING

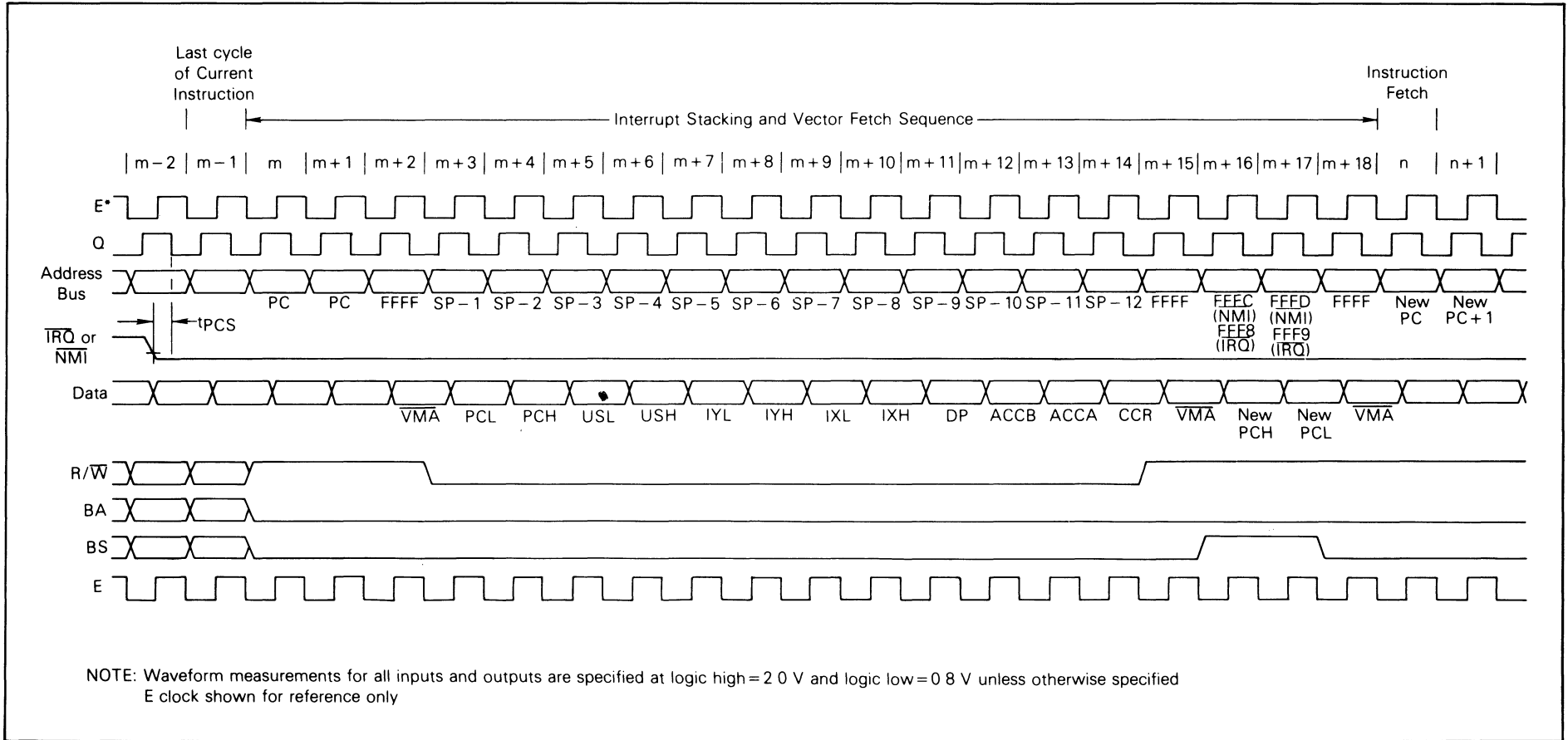
Each of the system MMUs must be initialized before the system can be utilized. Only the upper 2 kilobytes of memory are available until after the initialization is complete. The MMUINIT program first writes key values to MMUs 1 through 7, followed by programming the mapping registers of MMU0, task 0. The key value for MMU0 is then loaded. Table 10 contains a copy of the initial MMU memory map. After each MMU is initialized per Table 10, ASSIST09 is entered.

The ASSIST09 monitor program is a useful monitor that allows tracing through programs in task 0 only (because it uses $\overline{\text{NMI}}$). Also, ASSIST09 contains routines PUNCH and LOAD. These two routines allow the writing and reading of programs contained on the audio cassette tapes. The system can be further configured using ASSIST09. Once the MMUs and peripheral devices are initialized, ASSIST09 would not generally be used except for interrupt vector processing. A complete listing of the MMUINIT (MC6829 MMU INITIALIZATION) program can be found at the back of this application note.

The CONST program (MC6809-MC6829 MMU MONITOR PROGRAM) controls the use of the MMUs. It allows pages to be added and deleted from tasks, examination and change of memory, display of MPU registers, and display of MMU registers for the current task. This program forms the basis of an MMU operating system and a complete command summary is found at the back of this application note.

CONCLUSION

In future systems, memory management will become a topic of increasing importance. RAM densities are sure to increase into the 512 kilobit range in a relatively short time. This means just one RAM chip could fill the entire address space of the current 8-bit processor. Obviously, a system cannot function only from RAM as it would have no restart vectors or boot strap programs. If history repeats itself, the cost per bit in these RAMs will not appreciably increase over the cost per bit of smaller RAMs; this, of course, encourages their use. With the prospect of a single RAM or ROM chip filling an entire memory space, memory management not only becomes desirable but necessary. With the elements presented in this application note, everything is in place to begin large 8-bit MPU system design.

FIGURE 24 — Stack Operation Timing for $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$

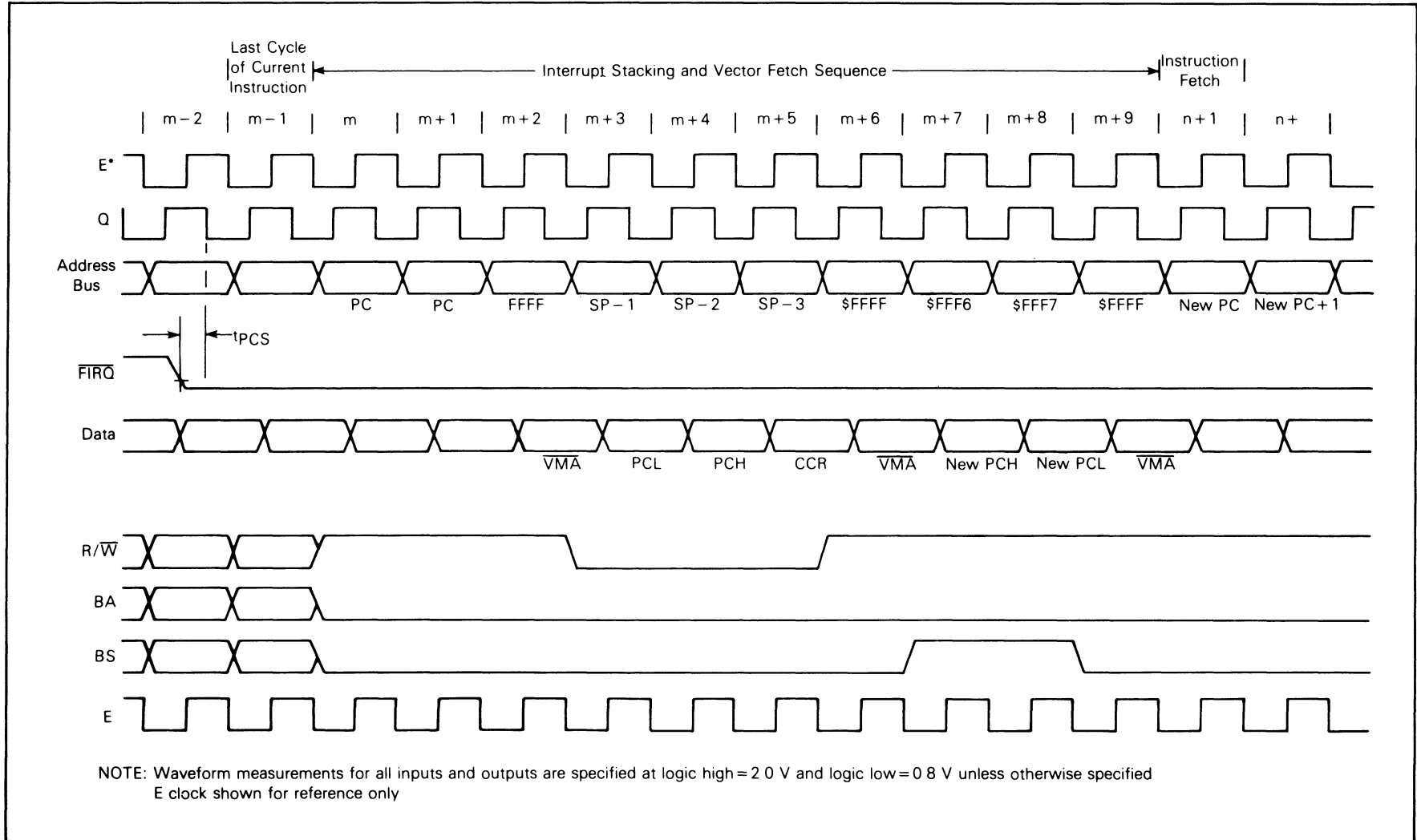


FIGURE 25 — Stack Operation Timing for $\overline{\text{FIRQ}}$

EXIT	LDA	TASK	GET NEXT TASK TO RUN
	STA	OPFRAT	AND PLACE IS IN THE OPERATE KEY
	STS	OSSP	SAVE CURRENT OS STACK POINTER
	ORCC	#F+I	SET F AND I (ENTER CRITICAL SECTION)
	LDS	SAVESP	RESTORE USERS STACK POINTER
	LDA	#1	CAUSE MAP SWITCH 1 CYCLE AFTER WRITE
	STA	FUSF	TO FUSE REGISTER
	RTI		RETURN TO USER TASK
	•		
	•		
	•	MAP SWITCH OCCURS, USER TASK RESUMES	
	•		

FIGURE 26 — Exit Routine from Task 0

TABLE 10 — Initial MMU Memory Map

MMU Register	Data	Physical Address
31	03FF	1FF8xx INITMMU + MMUs
30	03FE	1FF0xx ASSIST09
29	03FD	1FE8xx (Reserved for Peripherals)
28	03FC	1FE0xx DMA, ACIA, PTM, ADLC
27	001B	00D8xx Task 0 Stack RAM
26	001A	00D0xx Task 0 Stack ROM
25	03F9	1FC8xx CONST
24	03F8	1FC0xx CONST
23	0017	00B8xx RAM
22	0016	00B0xx RAM
21	0015	00A8xx RAM
20	0014	00A0xx RAM
19	0013	0098xx RAM
18	0012	0090xx RAM
17	0011	0088xx RAM
16	0010	0080xx RAM
15	000F	0078xx RAM
14	000E	0070xx RAM
13	000D	0068xx RAM
12	000C	0060xx RAM
11	000B	0058xx RAM
10	000A	0050xx RAM
9	0009	0048xx RAM
8	0008	0040xx RAM
7	0007	0038xx RAM
6	0006	0030xx RAM
5	0005	0028xx RAM
4	0004	0020xx RAM
3	0003	0018xx RAM
2	0002	0010xx RAM
1	0001	0008xx RAM
0	0000	0000xx RAM

```

00001
00002
00003          TTL      MC6829 MMU INITIALIZATION
00004
00005A FC00    ORG      $FC00
00006          OPT      ABS,LLE=82,S,CRE
00007          *
00008          *
00009          *
00010A FC00 8E   F841   A      LDX      #$F841   MMU #1 ADDRESS
00011A FC03 86   01     A      LDA      #$01
00012A FC05 A7   80     A NEXT STA      0,X+   WRITE KEY VALUE TO MMU
00013A FC07 4C           INCA           NEXT MMU
00014A FC08 81   08     A      CMPA     #$08   LAST MMU?
00015A FCOA 26   F9     FC05 BNE     NEXT
00016          *
00017          * ALL MAPPING REGISTERS FOR MMU 0 ARE INITIALIZED
00018          *
00019A FC0C 8E   03FF   A      LDX      #$03FF
00020A FCOF BF   F83E   A      STX      $F83E   PHYS ADDR $1FF800-$1FFFFFF REG 31
00021A FC12 30   1F           DEX
00022A FC14 BF   F83C   A      STX      $F83C   PHYS ADDR $1FF000-$1FF7FF REG 30
00023A FC17 30   1F           DEX
00024A FC19 BF   F83A   A      STX      $F83A   PHYS ADDR $1FE800-$1FEFFF REG 29
00025A FC1C 30   1F           DEX
00026A FC1E BF   F838   A      STX      $F838   PHYS ADDR $1FE000-$1FE7FF REG 28
00027A FC21 8E   001B   A      LDX      #$001B
00028A FC24 BF   F836   A      STX      $F836   PHYS ADDR $00D800-$00DFFF REG 27
00029A FC27 30   1F           DEX
00030A FC29 BF   F834   A      STX      $F834   PHYS ADDR $00D000-$00D7FF REG 26
00031A FC2C 30   1F           DEX
00032A FC2E 108E 03F9   A      LDY      #$03F9
00033A FC32 10BF F832   A      STY      $F832   PHYS ADDR $1FC800-$1FCFFF REG 25
00034A FC36 31   3F     A      LEAY     -1,Y
00035A FC38 10BF F830   A      STY      $F830   PHYS ADDR $1FC000-$1FC7FF REG 24
00036A FC3C 30   1F           DEX
00037A FC3E 30   1F           DEX
00038A FC40 BF   F82E   A      STX      $F82E   PHYS ADDR $00B800-$00BFFF REG 23
00039A FC43 30   1F           DEX
00040A FC45 BF   F82C   A      STX      $F82C   PHYS ADDR $00B000-$00B7FF REG 22
00041A FC48 30   1F           DEX
00042A FC4A BF   F82A   A      STX      $F82A   PHYS ADDR $00A800-$00AFFF REG 21
00043A FC4D 30   1F           DEX
00044A FC4F BF   F828   A      STX      $F828   PHYS ADDR $00A000-$00A7FF REG 20
00045A FC52 30   1F           DEX
00046A FC54 BF   F826   A      STX      $F826   PHYS ADDR $009800-$009FFF REG 19
00047A FC57 30   1F           DEX
00048A FC59 BF   F824   A      STX      $F824   PHYS ADDR $009000-$0097FF REG 18
00049A FC5C 30   1F           DEX
00050A FC5E BF   F822   A      STX      $F822   PHYS ADDR $008800-$008FFF REG 17
00051A FC61 30   1F           DEX
00052A FC63 BF   F820   A      STX      $F820   PHYS ADDR $008000-$0087FF REG 16
00053A FC66 30   1F           DEX
00054A FC68 BF   F81E   A      STX      $F81E   PHYS ADDR $007800-$007FFF REG 15
00055A FC6B 30   1F           DEX
00056A FC6D BF   F81C   A      STX      $F81C   PHYS ADDR $007000-$0077FF REG 14
00057A FC70 30   1F           DEX
00058A FC72 BF   F81A   A      STX      $F81A   PHYS ADDR $006800-$006FFF REG 13

```

```

00059A FC75 30 1F DEX
00060A FC77 BF F818 A STX $F818 PHYS ADDR $006000-$0067FF REG 12
00061A FC7A 30 1F DEX
00062A FC7C BF F816 A STX $F816 PHYS ADDR $005800-$005FFF REG 11
00063A FC7F 30 1F DEX
00064A FC81 BF F814 A STX $F814 PHYS ADDR $005000-$0057FF REG 10
00065A FC84 30 1F DEX
00066A FC86 BF F812 A STX $F812 PHYS ADDR $004800-$004FFF REG 9
00067A FC89 30 1F DEX
00068A FC8B BF F810 A STX $F810 PHYS ADDR $004000-$0047FF REG 8
00069A FC8E 30 1F DEX
00070A FC90 BF F80E A STX $F80E PHYS ADDR $003800-$003FFF REG 7
00071A FC93 30 1F DEX
00072A FC95 BF F80C A STX $F80C PHYS ADDR $003000-$0037FF REG 6
00073A FC98 30 1F DEX
00074A FC9A BF F80A A STX $F80A PHYS ADDR $002800-$002FFF REG 5
00075A FC9D 30 1F DEX
00076A FC9F BF F808 A STX $F808 PHYS ADDR $002000-$0027FF REG 4
00077A FCA2 30 1F DEX
00078A FCA4 BF F806 A STX $F806 PHYS ADDR $001800-$001FFF REG 3
00079A FCA7 30 1F DEX
00080A FCA9 BF F804 A STX $F804 PHYS ADDR $001000-$0017FF REG 2
00081A FCAC 30 1F DEX
00082A FCAE BF F802 A STX $F802 PHYS ADDR $000800-$000FFF REG 1
00083A FCB1 30 1F DEX
00084A FCB3 BF F800 A STX $F800 PHYS ADDR $000000-$0007FF REG 0
00085A FCB6 7E F037 A JMP $F037

```

```

00086 *
00087 * THE START ADDRESS FOR THE MMU MONITOR IS $C008. IT IS
00088 * PERMISSIBLE TO JUMP DIRECTLY TO THE MMU MONITOR FROM
00089 * THIS JUMP INSTRUCTION. HOWEVER, ASSIST09 MUST BE IN
00090 * PLACE BECAUSE THE INTERRUPT VECTORS IN THIS PROGRAM
00091 * ARE BASED ON ASSIST09.
00092 *
00093 * BOTH THE MMU MONITOR AND ASSIST09 WILL WORK ONLY IN
00094 * TASK 0
00095 *

```

```

00096A FFF0 ORG $FFF0
00097A FFF0 F7D4 A FDB $F7D4 RESERVED VECTOR
00098A FFF2 F7D8 A FDB $F7D8 SWI3 VECTOR
00099A FFF4 F7DC A FDB $F7DC SWI2 VECTOR
00100A FFF6 F7E0 A FDB $F7E0 FIRQ VECTOR
00101A FFF8 F7E4 A FDB $F7E4 IRQ VECTOR
00102A FFFA F7E8 A FDB $F7E8 SWI VECTOR
00103A FFFC F7EC A FDB $F7EC NMI VECTOR
00104A FFFE FC00 A FDB $FC00
00105 END

```

```

TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

```

```

FC05 NEXT 00012*00015

```

```

00001A C000          ORG    $C000
00002              TTL    MC6809-MC6829 MMU MONITOR PROGRAM
00003              OPT    ABS,LLE=82,S,CRE
00004
00005              *
00006              *           MANIFEST CONSTANTS
00007              *
00008              *           REGISTER OFFSETS FROM STACK TOP
00009              *
00010          0000    A COFF EQU    0           CONDITION CODES
00011          0001    A AOFF EQU    1           A
00012          0002    A BOFF EQU    2           B
00013          0001    A DOFF EQU    1           D SAME AS A
00014          0003    A DPOFF EQU   3           DPR
00015          0004    A XOFF EQU    4           X
00016          0006    A YOFF EQU    6           Y
00017          0008    A UOFF EQU    8           U
00018          000A    A POFF EQU   10          PC
00019              *
00020              *           CONDITION CODE BITS
00021              *
00022          0080    A E     EQU   %10000000  ENTIRE FLAG
00023          0040    A F     EQU   %01000000  FIRQ BIT
00024          0020    A H     EQU   %00100000  HALF CARRY
00025          0010    A I     EQU   %00010000  INTERRUPT MASK
00026          0008    A N     EQU   %00001000  NEGATIVE
00027          0004    A Z     EQU   %00000100  ZERO
00028          0002    A V     EQU   %00000010  OVERFLOW
00029          0001    A C     EQU   %00000001  CARRY
00030              *
00031              *           °NOT' CONDITION CODE BITS
00032              *
00033          007F    A NE     EQU   $FF-E
00034          00BF    A NF     EQU   $FF-F
00035          00DF    A NH     EQU   $FF-H
00036          00EF    A NI     EQU   $FF-I
00037          00F7    A NN     EQU   $FF-N
00038          00FB    A NZ     EQU   $FF-Z
00039          00FD    A NV     EQU   $FF-V
00040          00FE    A NC     EQU   $FF-C
00041              *
00042              *           THE INTERRUPT VECTORS
00043              *
00044          FFF0    A NOVEC EQU   $FFF0      RESERVED VECTOR
00045          FFF2    A SWI3VC EQU  $FFF2      SOFTWARE INTERRUPT 3
00046          FFF4    A SWI2VC EQU  $FFF4      SOFTWARE INTERRUPT 2
00047          FFF6    A FIRQVC EQU  $FFF6      FAST INTERRUPT
00048          FFF8    A IRQVEC EQU  $FFF8      EXTERNAL INTERRUPT
00049          FFFA    A SWIVEC EQU  $FFFA      SOFTWARE INTERRUPT (SYS CALL)
00050          FFFC    A NMIVEC EQU  $FFFC      NON-MASKABLE INTERRUPT
00051          FFFE    A RESVEC EQU  $FFFE      PROCESSOR RESTART VECTOR
00052              *
00053              *           CHARACTER CONSTANTS
00054              *
00055          0000    A EOS     EQU   $00        END OF STRING
00056          0007    A BEEP   EQU   $07        BELL (@G)
00057          0008    A BS     EQU   $08        BACKSPACE (@H)
00058          0009    A TAB    EQU   $09        TAB (@I)

```

```

00059      000A  A LF      EQU    $0A    LINE FEED (ⓐJ)
00060      000C  A FF      EQU    $0C    FORM FEED (ⓐL)
00061      000D  A CR      EQU    $0D    CARRIAGE RETURN (ⓐM)
00062      001B  A ESC     EQU    $1B    ESCAPE (ⓐ[)
00063      0020  A BLANK  EQU    $20    ASCII BLANK
00064      007F  A RUBOUT EQU    $7F    RUBOUT (DEL)
00065      *
00066      *          SYS CALL NAMES
00067      *
00068      0000  A GETC   EQU    0      GET NEXT CHARACTER FROM STANDARD
00069      0001  A PUTC   EQU    1      PUT CHARACTER ON STANDARD OUTPUT
00070      *
00071      *          END OF CONSTANTS
00072      *
00073      *
00074      *          EQUATES FOR 6829 REGISTERS
00075      *
00076      F800  A MMU    EQU    $F800  START OF MMU REGISTERS
00077      F840  A MMU0   EQU    MMU+$40  KEY VALUE FOR MMU0
00078      F847  A MMU7   EQU    MMU+$47  KEY VALUE FOR MMU7
00079      F848  A SBIT   EQU    MMU+$48  SYSTEM/USER FLAG (BIT 0)
00080      F849  A FUSE   EQU    MMU+$49  COUNT-DOWN FUSE
00081      F84A  A ACCESS EQU    MMU+$4A  ACCESS KEY
00082      F84B  A OPERAT EQU    MMU+$4B  OPERATE KEY
00083      *
00084      0800  A PSIZE  EQU    $800   PAGE SIZE
00085      0020  A NTASK  EQU    32    NUMBER OF TASKS IN SYSTEM
00086      0020  A NPAGE  EQU    32    NUMBER OF PAGES PER TASK
00087      0400  A MAXPAG EQU    $400   MAXIMUM NUMBER OF PAGES IN PHYSI
00088      0020  A MAXTAS EQU    32    MAXIMUM NUMBER OF TASKS IN SYSTE
00089      0200  A FAULT  EQU    $200   PAGE FAULT PAGE (FIRST PAGE OF N
00090      E008  A CONSOL  EQU    $E008  EXORCISOR ACIA
00091      D7A0  A STACKP EQU    $D7A0  CONVENIENT STACK START
00092      D7B0  A SPTAB  EQU    $D7B0  ADJOINS BOTTOM OF STACK
00093      D7AF  A CURTAS EQU    $D7AF  CURRENTLY EXAMINED TASK
00094      *
00095      *          MEMORY MANAGEMENT UNIT MONITOR
00096      *
00097      *          COMMANDS ARE:
00098      *
00099      *          T          EXAMINE/CHANGE CURRENT TASK NUMBER
00100      *          D          DISPLAY CPU AND MMU REGISTERS FOR CURRE
00101      *          R          DISPLAY REGISTERS
00102      *          M          EXAMINE/CHANGE MEMORY
00103      *          S          EXAMINE/CHANGE STACKP POINTER
00104      *          G          START/CONTINUE TASK EXECUTION
00105      *          +          ADD RAM PAGE
00106      *          -          REMOVE RAM PAGE
00107      *          Z          ZAP MMU REGISTERS TO FAULT
00108      *          ?          COMMAND SUMMARY
00109      *
00110      *-----
00111      *
00112      0004  A FREE    EQU    4      ADDRESS OF FREE MMU REGISTER
00113      1000  A FREEPG  EQU    $1000  ADDRESS IN MAP 0 OF FREE PAGE
00114      *
00115      *          SYS    MACR
00116      *          SWI

```

```

00117                                FCB @0
00118                                ENDM
00119A C000                          ORG    $C000
00120A C000 8E    C008    A START    LDX    #$C008    SWI VECTOR VALUE OF SWI
00121A C003 BF    D7D0    A          STX    $D7D0    ASSISTO9 SWI VECTOR LOCATION
00122A C006 20    03    C00B          BRA    INIT
00123A C008 16    04C9 C4D4          LBRA   SWIH    JUMP TO THE SWI HANDLER
00124                                *
00125                                *          INITIALIZE ONLY TASK 0'S SAVED STACKP POINTER
00126                                *
00127A C00B 10CE D7A0    A INIT    LDS    #STACKP  SETUP STACK FOR MONITOR
00128A C00F 10FF D7B0    A          STS    SPTAB   REWRITE TASK 0'S SAVED STACKP AD
00129                                *
00130A C013 CE    E008    A          LDU    #CONSOL
00131A C016 17    0588 C5A1          LBSR   INZACI  INITIALIZE THE TERMINAL
00132                                *
00133A C019 7F    D7AF    A          CLR    CURTAS  START BY EXAMINING TASK 0
00134A C01C 7F    F84B    A          CLR    OPERAT  AND WE'RE RUNNING FROM ZERO TOO
00135                                *
00136A C01F 17    0469 C48B MAIN    LBSR   CRLF    PRINT ON NEXT LINE
00137A C022 86    5F      A          LDA    #PROMPT
00138A C024                          SYS    PUTC    PRINT PROMPT
00139A C026                          SYS    GETC    AND WAIT FOR INPUT
00140A C028 84    7F      A          ANDA   #$7F   MASK PARITY
00141A C02A 17    0456 C483          LBSR   PUTS    PRINT A BLANK
00142A C02D 17    03E5 C415          LBSR   MAPUP   CONVERT LOWER TO UPPER CASE
00143A C030 30    8C 1C          LEAX   <CTAB,PCR  TABLE OF VALID COMMANDS
00144A C033 6D    84      A LOOP    TST    ,X      END OF TABLE TEST
00145A C035 27    12    C049          BEQ    HUH     COMMAND NOT FOUND
00146A C037 A1    84      A          CMPA   ,X      MATCH COMMAND
00147A C039 27    04    C03F          BEQ    LOOP2   FOUND IT
00148A C03B 30    03      A          LEAX   3,X     GO TO NEXT ENTRY
00149A C03D 20    F4    C033          BRA    LOOP
00150A C03F EC    01      A LOOP2   LDD    1,X     PICKUP OFFSET FROM TABLE
00151A C041 30    8D 3FBB          LEAX   0,PCR
00152A C045 AD    8B      A OFFSET  JSR    D,X     GO TO ROUTINE
00153A C047 20    D6    C01F          BRA    MAIN
00154                                *
00155A C049 86    3F      A HUH     LDA    #'?    UNRECOGNIZED COMMAND
00156A C04B                          SYS    PUTC
00157A C04D 20    D0    C01F          BRA    MAIN    LOOP AROUND
00158                                *
00159                                *          CTAB --- TABLE OF COMMAND CHARACTERS
00160                                *
00161                                *
00162A C04F    52      A CTAB   FCB    'R      DISPLAY REGISTERS
00163A C050    02C5   A      FDB    REGS-OFFSET
00164A C052    2B     A      FCB    '+'     ADD PAGE TO TASK
00165A C053    01B9   A      FDB    ADDPAG-OFFSET
00166A C055    44     A      FCB    'D     DISPLAY CURRENT TASK REGISTERS
00167A C056    0396   A      FDB    DISPLA-OFFSET
00168A C058    2D     A      FCB    '-'     REMOVE PAGE FROM TASK
00169A C059    01E5   A      FDB    REMOVE-OFFSET
00170A C05B    54     A      FCB    'T     CHANGE CURRENT TASK
00171A C05C    020C   A      FDB    TASK-OFFSET
00172A C05E    5A     A      FCB    'Z     ZAP TASK REGISTERS
00173A C05F    037E   A      FDB    ZAP-OFFSET
00174A C061    4D     A      FCB    'M     EXAMINE/CHANGE MEMORY

```

```

00175A C062 0222 A FDB MEMORY-OFFSET
00176A C064 47 A FCB 'G BEGIN/CONTINUE EXECUTION OF TASK
00177A C065 035B A FDB EXECUT-OFFSET
00178A C067 53 A FCB 'S EXAMINE/CHANGE STACKP POINTER
00179A C068 175B A FDB STACKP-OFFSET
00180A C06A 24 A FCB '$ DISPLAY PAGE (256 BYTES) IN HEX
00181A C06B 032D A FDB HEXDUM-OFFSET
00182A C06D 3F A FCB '?' ASKING FOR HELP
00183A C06E 002C A FDB HELP-OFFSET
00184A C070 00 A FCB 0 END OF TABLE
00185 *
00186 * HELP --- PRINT LEGAL COMMANDS SUMMARY
00187 *
00188A C071 30 8C 04 HELP LEAX <HELPIIN,PCR
00189A C074 17 0420 C497 LBSR PSTRNG
00190A C077 39 RTS
00191 *
00192 * HELPIINFO --- TEXT EMITTED FOR HELP COMMAND
00193 *
00194A C078 0D A HELPIIN FCB CR,LF
00195A C07A 4C A FCC /LEGAL COMMANDS ARE:/
00196A C08D 0D A FCB CR,LF
00197A C08F 54 A FCC :T DISPLAY/CHANGE CURRENT TASK:
00198A C0B1 0D A FCB CR,LF
00199A C0B3 44 A FCC /D DISPLAY MMU REGISTERS FOR CURRENT
00200A C0E0 0D A FCB CR,LF
00201A C0E2 4D A FCC :M EXAMINE/CHANGE MEMORY:
00202A C0FE 0D A FCB CR,LF
00203A C100 52 A FCC :R EXAMINE TASK'S REGISTERS:
00204A C11F 0D A FCB CR,LF
00205A C121 53 A FCC :S EXAMINE/CHANGE STACKP POINTER:
00206A C145 0D A FCB CR,LF
00207A C147 24 A FCC :$ DISPLAY A PAGE IN HEX:
00208A C163 0D A FCB CR,LF
00209A C165 47 A FCC :G BEGIN/CONTINUE TASK:
00210A C17F 0D A FCB CR,LF
00211A C181 2B A FCC /+ ADD A PAGE/
00212A C192 0D A FCB CR,LF
00213A C194 2D A FCC /- DELETE A PAGE/
00214A C1A8 0D A FCB CR,LF
00215A C1AA 5A A FCC /Z ZAP PAGES BACK TO DEFAULT/
00216A C1CA 0D A FCB CR,LF
00217A C1CC 3F A FCC /? THIS MESSAGE/
00218A C1DF 00 A FCB EOS
00219 *
00220A C1E0 50 A PHYMSG FCC /PHYSICAL PAGE=/
00221A C1EE 00 A FCB EOS
00222A C1EF 20 A LOGMSG FCC / LOGICAL PAGE=/
00223A C1FD 00 A FCB EOS
00224 *
00225 * ADD PAGE XXX TO CURRENT TASK AT PAGE PP (A XXX
00226 *
00227 * XXX = $000 TO MAXPAG
00228 * PP = $00 TO NPAGE
00229 *
00230A C1FE 30 8C DF ADDPAG LEAX PHYMSG,PCR PROMPT FOR PHYSICAL PAGE#
00231A C201 17 0293 C497 LBSR PSTRNG
00232A C204 17 024E C455 LBSR GET3HX ASK FOR PAGE

```

```

00233A C207 25 20 C229 BCS ADDX
00234A C209 1083 0400 A CMPD #MAXPAG
00235A C20D 24 1A C229 BHS ADDX PAGE NUMBER OUT OF RANGE
00236A C20F 1F 02 A TFR D,Y SAVE IT IN Y
00237 *
00238A C211 30 8C DB LEAX LOGMSG,PCR PROMPT FOR LOGICAL PAGE#
00239A C214 17 0280 C497 LBSR PSTRNG
00240A C217 17 0227 C441 LBSR GET2HX ASK FOR TASK PAGE
00241A C21A 25 0D C229 BCS ADDX
00242A C21C 81 20 A CMPA #NPAGE CHECK OUT OF RANGE PAGE
00243A C21E 24 09 C229 BHS ADDX
00244 *
00245 * NOW PUT PHYSICAL PAGE NUMBER IN TASK'S MAP
00246 *
00247A C220 8D 25 C247 BSR WINDOW GET CURRENT TASK WINDOW
00248A C222 8E F800 A LDX #MMU
00249A C225 48 ASLA COMPUTE MMU OFFSET
00250A C226 10AF 86 A STY A,X PUT PAGE IN MMU
00251A C229 39 ADDX RTS
00252 *
00253 * REMOVE A PAGE FROM A TASK
00254 *
00255 * THE LOGICAL PAGE NUMBER
00256 *
00257A C22A 30 8C C2 REMOVE LEAX LOGMSG,PCR ASK FOR LOGICAL PAGE #
00258A C22D 17 0267 C497 LBSR PSTRNG
00259A C230 17 020E C441 LBSR GET2HX
00260A C233 25 11 C246 BCS REMX QUIT IF NOT HEX
00261A C235 81 20 A CMPA #NPAGE OR IF PAGE OUT OF RANGE
00262A C237 24 0D C246 BHS REMX
00263A C239 8D 0C C247 BSR WINDOW
00264A C23B 8E F800 A LDX #MMU
00265A C23E 48 ASLA
00266A C23F 108E 0200 A LDY #FAULT AN EMPTY PAGE
00267A C243 10AF 86 A STY A,X
00268A C246 39 REMX RTS
00269 *
00270 * WINDOW --- SET ACCESS KEY TO CURRENT TASK
00271 *
00272A C247 34 02 A WINDOW PSHS A
00273A C249 B6 D7AF A LDA CURTAS
00274A C24C B7 F84A A STA ACCESS
00275A C24F 35 82 A PULS A,PC
00276 *
00277 * TASK --- DISPLAY/CHANGE CURRENT TASK BEING EXAM
00278 *
00279A C251 B6 D7AF A TASK LDA CURTAS PICKUP CURRENT TASK
00280A C254 17 025E C4B5 LBSR PUT2HX DISPLAY IT
00281A C257 17 0229 C483 LBSR PUTS PRINT A BLANK AND
00282A C25A 17 01E4 C441 LBSR GET2HX ASK FOR NEW TASK NUMBER
00283A C25D 25 07 C266 BCS TASKX NON-HEX TASK#
00284A C25F 81 20 A CMPA #NTASK
00285A C261 24 03 C266 BHS TASKX TASK OUT OF RANGE
00286A C263 B7 D7AF A STA CURTAS ADDRESS TASK GIVEN
00287A C266 39 TASKX RTS
00288 *
00289 * MEMORY --- EXAMINE/CHANGE MEMORY
00290 *

```



```

00291A C267 17 01F8 C462 MEMORY LBSR GET4HX GET START ADDRESS
00292A C26A 25 38 C2A4 BCS MEMX BAD ADDRESS
00293A C26C 1F 01 A TFR D,X MOVE POINTER TO X
00294A C26E 17 021A C48B MLOOP LBSR CRLF START NEW LINE
00295A C271 1F 10 A TFR X,D MOVE POINTER TO D AND
00296A C273 17 0255 C4CB LBSR PUT4HX PRINT CURRENT ADDRESS
00297A C276 17 020A C483 LBSR PUTS SEPARATE WITH A BLANK
00298A C279 F6 D7AF A LDB CURTAS PICKUP CURRENT TASK#
00299A C27C 8D 27 C2A5 BSR FUBYTE PICKUP THE BYTE FROM USER MAP
00300A C27E 17 0234 C4B5 LBSR PUT2HX AND PRINT IT
00301A C281 17 01FF C483 LBSR PUTS ANOTHER SPACE
00302A C284 17 01BA C441 LBSR GET2HX TRY TO GET NEW BYTE
00303A C287 25 06 C28F BCS MMOVE NON-HEX, MIGHT BE SPECIAL
00304A C289 8D 22 C2AD BSR SUBYTE PLACE BYTE IN USER MAP
00305A C28B 30 01 A LEAX 1,X UPDATE POINTER TO NEXT BYTE
00306A C28D 20 DF C26E BRA MLOOP INCREMENTED ADDRESS
00307A C28F 81 2E A MMOVE CMPA #AGAIN RE-EXAMINE SAME?
00308A C291 27 DB C26E BEQ MLOOP
00309 *
00310A C293 81 0D A CMPA #FWD ADVANCE TO NEXT?
00311A C295 26 04 C29B BNE MEM2
00312A C297 30 01 A LEAX 1,X ADD 1 TO X
00313A C299 20 D3 C26E BRA MLOOP
00314A C29B 81 5E A MEM2 CMPA #BACK GO BACK ONE?
00315A C29D 26 04 C2A3 BNE MEM3
00316A C29F 30 1F A LEAX -1,X SUBTRACT ONE FROM X
00317A C2A1 20 CB C26E BRA MLOOP
00318A C2A3 12 MEM3 NOP
00319A C2A4 39 MEMX RTS NONE OF THE ABOVE
00320 *
00321 * FUBYTE --- FETCH USER BYTE (SIMULATE LDA ,X OF
00322 * BYTE ADDRESS IS IN X, TASK TO USE IS IN
00323 * RETURNS WITH BYTE IN A. OTHER REGS UNCH
00324 *
00325A C2A5 34 10 A FUBYTE PSHS X
00326A C2A7 8D 0C C2B5 BSR GETPAG PLACE USER PAGE IN FREEPG
00327A C2A9 A6 84 A LDA ,X PICKUP BYTE
00328A C2AB 35 90 A PULS X,PC AND RETURN TO CALLER
00329 *
00330 * SUBYTE --- SET USER BYTE (SIMULATE STA ,X OF TA
00331 * BYTE ADDRESS IS IN X, BYTE IN A, TASK T
00332 * REGISTERS UNCHANGED ON EXIT
00333 *
00334A C2AD 34 10 A SUBYTE PSHS X
00335A C2AF 8D 04 C2B5 BSR GETPAG PLACE USER PAGE IN FREEPG
00336A C2B1 A7 84 A STA ,X SAVE IT IN RIGHT PLACE
00337A C2B3 35 90 A PULS X,PC AND RETURN
00338 *
00339 * GETPAG --- POINT TO USER BYTE
00340 *
00341 * X HAS USER ADDRESS, B HAS TASK #
00342 * RETURNS WITH X POINTING TO BYTE IN
00343 * OS MAP. D UNCHANGED, ACCESS KEY IS LEFT WITH T
00344 * TASK NUMBER IN B.
00345 *
00346A C2B5 34 26 A GETPAG PSHS D,Y SAVE SOME REGISTERS
00347A C2B7 F7 F84A A STB ACCESS SETUP WINDOW TO TASK
00348A C2BA 1F 10 A TFR X,D MOVE POINTER INTO ACCUMULATOR

```

```

00349A C2BC 47          ASRA          FIND PHYSICAL PAGE #
00350A C2BD 47          ASRA
00351A C2BE 84         3E          A          ANDA          #%00111110 MASK ALL BUT PAGE #
00352A C2C0 108E F800  A          LDY          #MMU
00353A C2C4 10AE A6          A          LDY          A,Y          PICKUP PAGE
00354A C2C7 7F F84A      A          CLR          ACCESS NOW TALK TO OS MAP
00355A C2CA 10BF F804      A          STY          MMU+FREE °FREE' OS PAGE
00356A C2CE 1F 10          A          TFR          X,D          NOW POINT TO OFFSET
00357A C2D0 84 07          A          ANDA          #%111          MASK HIGH BITS OF ADDRESS
00358A C2D2 8E 1000       A          LDX          #FREEPG POINT TO PAGE BEGIN
00359A C2D5 30 8B          A          LEAX         D,X          ADD OFFSET
00360A C2D7 35 A6          A          PULS         D,Y,PC      RESTORE REGISTERS AND RETURN
00361          *
00362          *          REGS --- DISPLA TASK'S REGISTERS
00363          *
00364          *          EFHINZVC A-XX B-XX DPR-XX
00365          *          X-XXXX Y-XXXX U-XXXX P-XXXX
00366          *          S-XXXX
00367          *
00368A C2D9 45          A RSTRNG FCC /EFHINZVC/THE NAMES OF THE BITS
00369          *
00370A C2E1 20          A MSSTR FCC / A-? B-? DPR-?/
00371A C2F0 0D          A          FCB CR,LF
00372A C2F2 58          A          FCC /X-?? Y-?? U-?? P-??/
00373A C305 0D          A          FCB CR,LF
00374A C307 53          A          FCC /S-/
00375A C309 00          A          FCB EOS
00376          *
00377          *
00378          *          FIRST, PRETTY PRINT THE CCR
00379          *
00380A C30A F6 D7AF A REGS LDB CURTAS
00381A C30D 17 0258 C568 LBSR GETUS GET TOP OF USER STACKP
00382A C310 30 84 A LEAX COFF,X ADD IN CCR OFFSET
00383A C312 8D 91 C2A5 BSR FUBYTE PICKUP CONDITION CODES
00384A C314 34 02 A PSHS A STORE ON STACKP
00385A C316 30 8C CO LEAX RSTRNG,PCR PICKUP STRING START
00386A C319 C6 08 A LDB #8 LOOP COUNTER
00387A C31B A6 80 A REGS3 LDA ,X+ PICKUP APPROPRIATE CHARACTER
00388A C31D 69 E4 A ROL ,S GET LEADING BIT
00389A C31F 25 02 C323 BCS REGS2 PRINT BIT NAME
00390A C321 86 2E A LDA #' BIT OFF CHARACTER
00391A C323 REGS2 SYS PUTC PRINT . OR CHARACTER
00392A C325 5A DECB LOOP ON ALL 8 BITS
00393A C326 26 F3 C31B BNE REGS3
00394A C328 32 61 A LEAS 1,S DELETE TEMPORARY
00395          *
00396A C32A F6 D7AF A LDB CURTAS
00397A C32D 17 0238 C568 LBSR GETUS
00398A C330 30 01 A LEAX AOFF,X POINT TO START OF REGISTERS
00399A C332 31 8C AC LEAY MSSTR,PCR GET FORMATTING STRING
00400A C335 A6 A0 A REGS6 LDA ,Y+ PICKUP FORMAT CHAR
00401A C337 81 00 A CMPA #EOS IF AT END, WRAP UP WITH STACKP P
00402A C339 27 12 C34D BEQ REGS4
00403A C33B 81 3F A CMPA #'? INSERT HEX NUMBER?
00404A C33D 26 0A C349 BNE REGS5
00405A C33F 17 FF63 C2A5 LBSR FUBYTE GET THE BYTE
00406A C342 30 01 A LEAX 1,X AND ADVANCE TO NEXT

```

```

00407A C344 17 00FA C441 LBSR GET2HX
00408A C347 20 EC C335 BRA REGS6 GET NEXT CHAR
00409A C349 REGS5 SYS PUTC JUST PRINT THE CHAR
00410A C34B 20 E8 C335 BRA REGS6
00411A C34D 17 0218 C568 REGS4 LBSR GETUS
00412A C350 1F 10 A TFR X,D FINALLY,
00413A C352 17 0176 C4CB LBSR PUT4HX PRINT USER STACKP POINTER
00414A C355 39 RTS
00415 *
00416 * STACK --- EXAMINE CHANGE USER STACK POINTER
00417 *
00418A C356 F6 D7AF A STACK LDB CURTAS GET CURRENT TASK NUMBER
00419A C359 17 020C C568 LBSR GETUS PICKUP SP IN X
00420A C35C 1F 10 A TFR X,D MOVE TO D
00421A C35E 17 016A C4CB LBSR PUT4HX
00422A C361 17 011F C483 LBSR PUTS
00423A C364 17 00FB C462 LBSR GET4HX ASK FOR NEW VALUE
00424A C367 25 08 C371 BCS STACKX
00425 *
00426 * PUT NEW STACKP POINTER IN PLACE
00427 *
00428A C369 1F 01 A TFR D,X MOVE TO PLACE PUTUS EXPECTS IT
00429A C36B F6 D7AF A LDB CURTAS GET CURRENT TASK
00430A C36E 17 0201 C572 LBSR PUTUS REPLACE IT
00431A C371 39 STACKX RTS
00432 *
00433 * HEXDUM --- DUMP A PAGE IN HEXADECIMAL
00434 *
00435A C372 17 00CC C441 HEXDUM LBSR GET2HX AS FOR PAGE NUMBER :
00436A C375 25 28 C39F BCS HEXX BAD PAGE ADDRESS
00437A C377 6F E2 A CLR ,-S INITIALIZE BYTE COUNT
00438A C379 5F CLRB ZERO LOW BYTE OF ADDRESS
00439A C37A 17 010E C48B HEX3 LBSR CRLF
00440A C37D 1F 01 A TFR D,X MOVE ADDRESS TO POINTER REG.
00441A C37F 17 0149 C4CB LBSR PUT4HX PRINT CURRENT ADDRESS
00442A C382 17 00FE C483 HEX2 LBSR PUTS THEN A BLANK
00443A C385 F6 D7AF A LDB CURTAS
00444A C388 17 FF1A C2A5 LBSR FUBYTE PICKUP THE BYTE
00445A C38B 30 01 A LEAX 1,X BUMP POINTER
00446A C38D 17 00B1 C441 LBSR GET2HX PRINT THE BYTE
00447A C390 6C E4 A INC ,S
00448A C392 A6 E4 A LDA ,S
00449A C394 85 0F A BITA #%1111 START NEW LINE WHEN COUNT MOD 16
00450A C396 26 EA C382 BNE HEX2
00451A C398 1F 10 A TFR X,D
00452A C39A 5D TSTB WHILE LOW BYTE != 0 KEEP PRINTIN
00453A C39B 26 DD C37A BNE HEX3
00454A C39D 32 61 A LEAS 1,S DROP TEMP
00455A C39F 39 HEXX RTS
00456 *
00457 * EXECUT --- BEGIN/CONTINUE EXECUTION OF TASK
00458 *
00459A C3A0 B6 D7AF A EXECUT LDA CURTAS
00460A C3A3 B7 F84B A STA OPERAT SETUP FOR TASK SWITCH
00461A C3A6 17 019F C548 LBSR RETURN SIMULATE RTI
00462 *
00463 * WHEN TASK QUILTS, IT RETURNS HERE
00464 *

```

```

PAGE 009 MMB .SA:1 MC6809-MC6829 MMU MONITOR PROGRAM

00465A C3A9 7F F84B A CLR OPERAT SWITCH BACK TO MONITOR
00466A C3AC B6 D7AF A LDA CURTAS
00467A C3AF 17 008F C441 LBSR GET2HX SHOW WHICH TASK QUIT
00468A C3B2 30 8C 04 LEAX <XITMSG,PCR
00469A C3B5 17 00DF C497 LBSR PSTRNG
00470A C3B8 39 RTS TASK TERMINATED, RETURN TO MONIT
00471 *
00472A C3B9 20 A XITMSG FCC / STOPPED./
00473A C3C2 00 A FCB EOS
00474 *
00475 * ZAP --- SET ALL REGISTERS FOR CURRENT TASK TO F
00476 *
00477A C3C3 B6 D7AF A ZAP LDA CURTAS CAN'T ZAP TASK 0 !
00478A C3C6 27 12 C3DA BEQ ZAPX
00479A C3C8 B7 F84A A STA ACCESS BRING IT INTO THE WINDOW
00480A C3CB 8E F800 A LDX #MMU START OF REGS
00481A C3CE 108E 0200 A LDY #FAULT PAGE FAULT PAGE
00482A C3D2 10AF 81 A ZAP2 STY ,X++
00483A C3D5 8C F840 A CMPX #MMU+NPAGE+NPAGE END OF MMU REGISTERS
00484A C3D8 26 F8 C3D2 BNE ZAP2
00485A C3DA 39 ZAPX RTS
00486 *
00487 * DISPLA --- DISPLAY MMU REGISTERS FOR CURRENT TA
00488 *
00489 *
00490A C3DB 17 00AD C48B DISPLA LBSR CRLF START ON A NEW LINE
00491A C3DE 30 8C 2C LEAX <TASMSG,PCR
00492A C3E1 17 00B3 C497 LBSR PSTRNG PRINT TASK NUMBER
00493A C3E4 B6 D7AF A LDA CURTAS
00494A C3E7 8D 58 C441 BSR GET2HX PRINT TASK #
00495A C3E9 17 009F C48B LBSR CRLF START NEW LINE
00496 *
00497A C3EC 8E F800 A LDX #MMU POINT TO START OF REGS
00498A C3EF 34 10 A DISP2 PSHS X SAVE POINTER
00499 *
00500 * FINALLY PRINT THE PAGE NUMBER
00501 *
00502A C3F1 17 FE53 C247 LBSR WINDOW
00503A C3F4 35 10 A PULS X RESTORE POINTER TO REGS.
00504A C3F6 EC 81 A LDD ,X++ PICKUP PAGE AND ADVANCE TO NEXT
00505A C3F8 17 00C7 C4C2 LBSR PUT3HX PRINT PAGE ADDRESS
00506A C3FB 17 0085 C483 LBSR PUTS
00507A C3FE 1F 10 A TFR X,D CHECK IF TIME TO PRINT CR
00508A C400 C5 07 A BITB #%111 CR EVERY 4 REGISTERS (EACH 8 BYT
00509A C402 26 03 C407 BNE DISP5
00510A C404 17 0084 C48B LBSR CRLF
00511A C407 8C F840 A DISP5 CMPX #MMU+NPAGE+NPAGE CHECK IF AT END OF REGS
00512A C40A 26 E3 C3EF BNE DISP2
00513A C40C 39 RTS
00514 *
00515A C40D 54 A TASMSG FCC /TASK # /
00516A C414 00 A FCB EOS
00517 *
00518 005F A PROMPT EQU ' PROMPT FOR INPUT
00519 002E A AGAIN EQU ' RE-EXAMINE SAME BYTE
00520 000D A FWD EQU CR GO TO NEXT BYTE
00521 005E A BACK EQU '@ GO BACK ONE BYTE
00522 *

```

```

00523          *          MONIO --- MONITOR CONSOLE I/O ROUTINES
00524          *
00525          *
00526          *          MAPUP --- CONVERT a-z TO A-Z
00527          *
00528          *          CHARACTER TO CONVERT IS IN A, ONLY CHARACTERS F
00529          *          ARE CHANGED
00530          *
00531A C415 81 61      A MAPUP CMPA #'a CHECK BOUNDS
00532A C417 25 06     C41F BLO  NOMAP CHARACTER <A
00533A C419 81 7A     A     CMPA #'z
00534A C41B 22 02     C41F BHI  NOMAP CHARACTER >Z
00535A C41D 80 20     A     SUBA #'a-'A PERFORM MAP DOWN
00536A C41F 39          NOMAP RTS
00537          *
00538          *          GETNYB --- GET NYBBLE IN A
00539          *
00540          *          TRY TO GET ONE HEX CHARACTER. IF 0-9 OR A-F,
00541          *          CONVERT TO BINARY. OTHERWISE RETURN CHARACTER
00542          *          IN A WITH C-BIT SET.
00543          *
00544A C420          GETNYB SYS  GETC  GET ONE CHARACTER
00545A C422 8D F1     C415 BSR  MAPUP MAP TO UPPER CASE ONLY
00546A C424 34 02     A     PSHS A SAVE IT IN CASE IT'S NOT HEX
00547A C426 80 30     A     SUBA #'0
00548A C428 2B 13     C43D BMI  NOTHEX WAS LESS THAN 0
00549A C42A 81 09     A     CMPA #9
00550A C42C 2F 0A     C438 BLE  GOTIT IS BETWEEN 0-9
00551A C42E 80 07     A     SUBA #'A-'9-1
00552A C430 81 09     A     CMPA #9
00553A C432 2F 09     C43D BLE  NOTHEX WAS BETWEEN 9 AND A
00554A C434 81 0F     A     CMPA #9
00555A C436 2E 05     C43D BGT  NOTHEX WAS GREATER THAN F
00556A C438 32 61     A GOTIT LEAS 1,S DON'T NEED SAVED CHAR
00557A C43A 1C FE     A     ANDCC #NC TURN OFF CARRY
00558A C43C 39          RTS
00559A C43D 1A 01     A NOTHEX ORCC #C SET CARRY
00560A C43F 35 82     A     PULS A,PC RETURN TYPED CHARACTER
00561          *
00562          * GET2HX --- GET ASCII CHARACTERS AND CONVERT TO BINARY
00563          * RESULT IS RETURNED IN A, NO OTHER REGISTERS ARE CHANG
00564          * IF A NON-HEX CHARACTER IS TYPED, THE C BIT IS SET
00565          * AND THE CHARACTER TYPED IS RETURNED IN A. OTHERWISE
00566          * THE C BIT IS CLEARED.
00567          *
00568A C441 8D DD     C420 GET2HX BSR  GETNYB GET HIGH NYBBLE
00569A C443 25 0F     C454 BCS  GET22
00570A C445 48          ASLA          MOVE IT TO THE HIGH NYBBLE
00571A C446 48          ASLA
00572A C447 48          ASLA
00573A C448 48          ASLA
00574A C449 34 02     A     PSHS A SAVE IT
00575A C44B 8D D3     C420 BSR  GETNYB GET LOW NYBBLE
00576A C44D 25 03     C452 BCS  GET23 BAD SECOND NYBBLE
00577A C44F AA E0     A     ORA  ,S+ COMBINE HIGH AND LOW NYBBLES
00578A C451 39          RTS
00579A C452 32 61     A GET23 LEAS 1,S DROP HIGH NYBBLE
00580A C454 39          GET22 RTS

```

```

00581          *
00582          *      GET3HX --- GET ASCII CHARACTERS AND CONVERT TO
00583          *
00584          *      RESULT IS RETURNED IN D, NO OTHER REGISTERS ARE
00585          *      IF A NON-HEX CHARACTER IS TYPED, THE C-BIT IS S
00586          *      AND THE CHARACTER TYPED IS RETURNED IN A. OTHE
00587          *      THE C-BIT IS CLEARED.
00588          *
00589A C455 8D   C9   C420 GET3HX BSR   GETNYB
00590A C457 25   15   C46E          BCS   GET42
00591A C459 1F   89           A     TFR   A,B     MOVE HIGH NYBBLE TO B
00592A C45B 8D   E4   C441          BSR   GET2HX
00593A C45D 25   0F   C46E          BCS   GET42
00594A C45F 1E   89           A     EXG   A,B     SWAP HIGH FOR LOW
00595A C461 39           GET32  RTS
00596          *
00597          *      GET4HX --- GET ASCII CHARACTERS AND CONVERT TO
00598          *
00599          *      RESULT IS RETURNED IN D, NO OTHER REGISTERS ARE
00600          *      IF A NON-HEX CHARACTER IS TYPED, THE C-BIT IS S
00601          *      AND THE CHARACTER TYPED IS RETURNED IN A. OTHE
00602          *      THE C-BIT IS CLEARED.
00603          *
00604A C462 8D   DD   C441 GET4HX BSR   GET2HX
00605A C464 25   08   C46E          BCS   GET42
00606A C466 1F   89           A     TFR   A,B     MOVE HIGH BYTE TO B
00607A C468 8D   D7   C441          BSR   GET2HX
00608A C46A 25   02   C46E          BCS   GET42
00609A C46C 1E   89           A     EXG   A,B     SWAP HIGH FOR LOW
00610A C46E 39           GET42  RTS
00611          *
00612          *      GETRNG --- GET START AND END ADDRESSES
00613          *      RETURN START IN X AND END+1 IN Y
00614          *      C-BIT SET IF A NON-HEX DIGIT WAS TYPED;
00615          *
00616A C46F 8D   F1   C462 GETRNG BSR   GET4HX
00617A C471 25   0F   C482          BCS   GETRX   BAD START ADDRESS
00618A C473 1F   01           A     TFR   D,X
00619A C475 8D   0C   C483          BSR   PUTS   ACKNOWLEDGE FIRST ADDRESS AS GOO
00620A C477 8D   E9   C462          BSR   GET4HX
00621A C479 25   07   C482          BCS   GETRX   BAD END ADDRESS
00622A C47B C3   0001  A     ADDD  #1     BUMP END ADDRESS
00623A C47E 1F   02           A     TFR   D,Y
00624A C480 1C   FE           A     ANDCC #NC   CLEAR C-BIT TO SAY NO ERROR
00625A C482 39           GETRX  RTS
00626          *
00627          *      PUTS --- PRINT A BLANK ON THE CONSOLE
00628          *
00629          *
00630          *      *ALL REGISTERS UNCHANGED
00631          *
00632A C483 34   02           A     PUTS   PSHS   A
00633A C485 86   20           A     LDA    #BLANK
00634A C487           SYS   PUTC
00635A C489 35   82           A     PULS  A,PC   RESTORE AND RETURN
00636          *
00637A C48B 34   02           A     CRLF  PSHS   A     SAVE A
00638A C48D 86   0D           A     LDA    #CR

```

```

00639A C48F          SYS   PUTC
00640A C491 86     0A     A     LDA   #LF
00641A C493          SYS   PUTC
00642A C495 35     82     A     PULS  A,PC   RESTORE AND RETURN
00643                *
00644                * PSTRNG --- PRINT STRING
00645                *
00646                * POINTER TO STRING START IS IN X, STRING
00647                * IS TERMINATED BY EOS CHARACTER
00648                * NO REGISTERS ARE CHANGED
00649                *
00650A C497 34     12     A PSTRNG PSHS  A,X     SAVE STUFF
00651A C499 A6     80     A PSTR2  LDA   ,X+   PICKUP CHARACTER
00652A C49B 81     00     A     CMPA  #EOS   AT END OF STRING?
00653A C49D 27     04     C4A3    BEQ   PSTR9  YES, QUIT
00654A C49F          SYS   PUTC   PRINT IT
00655A C4A1 20     F6     C499    BRA   PSTR2  AND CONTINUE
00656A C4A3 35     92     A PSTR9  PULS  A,X,PC  RESTORE AND RETURN
00657                *
00658                * PUTNYB --- PRINT LOWER NYBBLE OF A IN HEX
00659                *
00660A C4A5 34     02     A PUTNYB PSHS  A     SAVE A
00661A C4A7 84     0F     A     ANDA  #$F   CLEAR HIGH GARBAGE
00662A C4A9 8B     30     A     ADDA  #'0   ADD ASCII OFFSET
00663A C4AB 81     39     A     CMPA  #'9   CHECK IF >9
00664A C4AD 23     02     C4B1    BLS   PUTNY2 NOW PRINT
00665A C4AF 8B     07     A     ADDA  #'A-'9-1 MOVE UP TO A-F
00666A C4B1          PUTNY2 SYS   PUTC
00667A C4B3 35     82     A     PULS  A,PC   RESTORE AND RETURN
00668                *
00669                * PUT2HX --- PRINT A IN HEX ON TERMINAL
00670                *
00671                * NUMBER TO BE PRINTED IS IN A, NO REGISTERS ARE CHANGE
00672                *
00673A C4B5 34     02     A PUT2HX PSHS  A     SAVE A FOR LOW NYBBLE
00674A C4B7 47          ASRA
00675A C4B8 47          ASRA
00676A C4B9 47          ASRA
00677A C4BA 47          ASRA
00678A C4BB 8D     E8     C4A5    BSR   PUTNYB  PRINT ONE HEX DIGIT
00679A C4BD 35     02     A     PULS  A     PICKUP VALUE AGAIN
00680A C4BF 8D     E4     C4A5    BSR   PUTNYB  PRINT LOW NYBBLE
00681A C4C1 39          RTS     RETURN
00682                *
00683                * PUT3HX --- PRINT D IN HEX ON TERMINAL
00684                *
00685                * NUMBER TO BE PRINTED IS IN D, NO REGISTERS ARE CHANGE
00686                *
00687A C4C2 8D     E1     C4A5  PUT3HX BSR   PUTNYB  PRINT HIGH NYBBLE FIRST
00688A C4C4 1E     89     A     EXG   A,B    NOW PRINT LOW NYBBLE
00689A C4C6 8D     ED     C4B5    BSR   PUT2HX
00690A C4C8 1E     89     A     EXG   A,B    SWITCH BYTES BACK
00691A C4CA 39          RTS
00692                *
00693                * PUT4HX --- PRINT D IN HEX ON TERMINAL
00694                *
00695                * NUMBER TO BE PRINTED IS IN D, NO OTHER REGISTERS
00696                * ARE CHANGED

```

```

00697
00698A C4CB 8D E8 C4B5 PUT4HX BSR PUT2HX PRINT HIGH BYTE FIRST
00699A C4CD 1E 89 A EXG A,B NOW PRINT LOW BYTE
00700A C4CF 8D E4 C4B5 BSR PUT2HX
00701A C4D1 1E 89 A EXG A,B SWITCH BYTES BACK
00702A C4D3 39 RTS
00703
00704 * SWIH --- SOFTWARE INTERRUPT HANDLER
00705 *
00706 * SWIH IS ENTERED VIA THE SWI INSTRUCTION AND IS
00707 * FOR TRANSFERRING CONTROL TO A ROUTINE DETERM
00708 * BYTE THAT FOLLOWS THE SWI OPCODE. THIS BYTE IS
00709 * INDEX INTO THE DISPATCH TABLE FOR STAND
00710 * FUNCTIONS. THE RETURN ADDRESS OF THE CALLING
00711 * ADJUSTED BY SWIH BEFORE THE NAMED ROUTINE IS C
00712 * CALLED ROUTINE IS ENTERED AS IF IT WERE DIREC
00713 * FROM AN SWI INSTRUCTION (EXCEPT THAT THE MACHIN
00714 * WILL NOT BE CORRECT).
00715 *
00716 * NOTE: THIS ROUTINE CONVERTED FROM A NON-MMU SY
00717 *
00718A C4D4 8E D7B0 A SWIH LDX #SPTAB SAVE TASKS STACK POINTER
00719A C4D7 B6 F84B A LDA OPERAT
00720A C4DA 48 ASLA
00721A C4DB 10EF 86 A STS A,X SAVE IT OFF
00722A C4DE 10FE D7B0 A LDS SPTAB PICKUP MONITOR'S STACK
00723 *
00724 * FETCH TASK'S PC
00725 *
00726A C4E2 F6 F84B A LDB OPERAT GET TASK THAT INTERRUPTED
00727A C4E5 17 0080 C568 LBSR GETUS GET STACK POINTER
00728A C4E8 30 0B A LEAX POFF+1,X LOW BYTE OF PC
00729A C4EA 17 FDB8 C2A5 LBSR FUBYTE
00730A C4ED 34 02 A PSHS A SAVE
00731A C4EF 30 1F A LEAX -1,X BACKUP TO HIGH BYTE OF PC
00732A C4F1 17 FDB1 C2A5 LBSR FUBYTE
00733A C4F4 34 02 A PSHS A PC NOW ON STACK
00734A C4F6 35 10 A PULS X RESTORE TO X
00735A C4F8 17 FDAA C2A5 LBSR FUBYTE GET SWI FUNCTION CODE
00736A C4FB 34 02 A PSHS A SAVE
00737A C4FD 30 01 A LEAX 1,X BUMP USER PC
00738A C4FF 34 10 A PSHS X SAVE FOR REPLACEMENT
00739A C501 8D 65 C568 BSR GETUS
00740A C503 30 0A A LEAX POFF,X POINT TO HIGH BYTE
00741A C505 35 02 A PULS A GET HIGH BYTE
00742A C507 17 FDA3 C2AD LBSR SUBYTE REPLACE HIGH
00743A C50A 30 01 A LEAX 1,X POINT TO LOW BYTE
00744A C50C 35 02 A PULS A GET LOW BYTE
00745A C50E 17 FD9C C2AD LBSR SUBYTE REPLACE LOW
00746 *
00747 * NOW DO THE REAL WORK
00748 *
00749A C511 35 02 A PULS A RESTORE FUNCTION CODE
00750A C513 81 03 A CMPA #NCALLS THROW OUT BAD NUMBERS
00751A C515 24 31 C548 BHS RETURN CALL OUT OF RANGE
00752A C517 30 8C 05 LEAX <SWITAB,PCR GET TABLE START ADDRESS
00753A C51A 48 ASLA TABLE IS 2 BYTES/ENTRY
00754A C51B EC 86 A LDD A,X PICKUP OFFSET FROM TABLE

```



```

PAGE 014 SWIH .SA:1 MC6809-MC6829 MMU MONITOR PROGRAM
00755A C51D 6E 8B A JMP D,X GO TO ROUTINE
00756 *
00757 * SWITAB --- TABLE OF SYSTEM CALL ADDRESSES
00758 *
00759A C51F 0006 A SWITAB FDB UGETC-SWITAB #0 USER GETC
00760A C521 0017 A FDB UPUTC-SWITAB #1 USER PUTC
00761A C523 0028 A FDB UQUIT-SWITAB #2 USER QUIT
00762 *
00763 0003 A NCALLS EQU 3 NUMBER OF ENTRIES IN THE TABLE
00764 *
00765 *
00766A C525 CE E008 A UGETC LDU #CONSOL GET CONSOLE ADDRESS
00767A C528 8D 53 C57D BSR RAWGTC
00768A C52A F6 F84B A LDB OPERAT DESTINATION TASK#
00769A C52D 8D 39 C568 BSR GETUS GET USERS STACK POINTER
00770A C52F 30 01 A LEAX AOFF,X ADD OFFSET FOR A REGISTER
00771A C531 17 FD79 C2AD LBSR SUBYTE WRITE CHARACTER INTO STACK
00772A C534 20 12 C548 BRA RETURN SWITCH BACK TO TASK
00773A C536 CE E008 A UPUTC LDU #CONSOL
00774A C539 F6 F84B A LDB OPERAT DESTINATION TASK#
00775A C53C 8D 2A C568 BSR GETUS POINT TO STACK
00776A C53E 30 01 A LEAX AOFF,X ADD A REG OFFSET
00777A C540 17 FD62 C2A5 LBSR FUBYTE FETCH THE CHARACTER
00778A C543 8D 40 C585 BSR RAWPTC PRINT IT
00779A C545 20 01 C548 BRA RETURN
00780A C547 39 UQUIT RTS RETURN FROM MONITOR CALL TO STAR
00781 *
00782 * RETURN --- RETURN TO INTERRUPTED TASK
00783 *
00784 * THE INTERRUPTED TASK NUMBER IS IN THE OPERAT
00785 * THE TASK WAS ZERO (I.E. THE MONITOR ITSELF) A
00786 * IS EXECUTED. IF NOT, THEN THE STACK POINTER I
00787 * TO WHAT IT WAS WHEN THE INTERRUPT OCCURRED AND
00788 * TO THAT TASK IS CAUSED BY WRITING TO THE 6
00789 * REGISTER.
00790 *
00791 *
00792A C548 B6 F84B A RETURN LDA OPERAT PICKUP INTERRUPTED TASK#
00793A C54B 27 1A C567 BEQ SIMPLE WAS JUST THE MONITOR
00794A C54D 48 ASLA
00795A C54E 8E D7B0 A LDX #SPTAB FIND OLD STACK POINTER
00796A C551 10FF D7B0 A STS SPTAB SAVE MONITOR'S STACK POINTER
00797A C555 C6 01 A LDB #1 SETUP FOR WRITE TO FUSE
00798A C557 1A 50 A ORCC #I+F ENTER CRITICAL SECTION
00799A C559 10EE 86 A LDS A,X PICKUP USER TASK STACK POINTER
00800A C55C F7 F849 A STB FUSE START THE SWITCH
00801A C55F 3B RTI AND AWAY WE GO.....
00802A C560 12 NOP
00803A C561 12 NOP
00804A C562 12 NOP
00805A C563 12 NOP
00806A C564 12 NOP
00807A C565 20 FE C565 BRA * SOMETHING SERIOUSLY WRONG HERE
00808A C567 3B SIMPLE RTI JUST AN RTI FOR MONITOR
00809 *
00810 * GETUS --- GET USERS STACK POINTER IN X (TASK#
00811 *
00812A C568 34 04 A GETUS PSHS B SAVE B

```

```

00813A C56A 58          ASLB
00814A C56B 8E   D7B0   A   LDX   #SPTAB   POINT TO STACK SAVE AREA
00815A C56E AE     85     A   LDX   B,X     PICKUP APPROPRIATE POINTER
00816A C570 35   84     A   PULS  B,PC    RETURN WITH SP IN X
00817          *
00818          *   PUTUS --- MAKE X THE STACK POINTER FOR TASK IN
00819          *
00820A C572 34   24     A PUTUS PSHS  B,Y
00821A C574 58          ASLB
00822A C575 108E D7B0   A   LDY   #SPTAB
00823A C579 AF     A5     A   STX   B,Y     PUT NEW POINTER IN PLACE
00824A C57B 35   A4     A   PULS  B,Y,PC
00825          *
00826          *   RAW ACIA I/O SUBROUTINES
00827          *
00828          0000   A ACIACR EQU   0       CONTROL REGISTER
00829          0001   A ACIADR EQU   1       DATA REGISTER
00830          0001   A .RDRF EQU  %00000001 RECEIVER FULL FLAG
00831          0002   A .TDRE EQU  %00000010 TRANSMITTER EMPTY FLAG
00832          *
00833          *   RAWGTC --- GET A CHARACTER FROM THE 'CONSOLE AND
00834          *
00835          *   U CONTAINS THE ADDRESS OF THE ACIA CONTROL REGI
00836          *   CHARACTER IS RETURNED IN A, ALL OTHER REGISTERS
00837          *   GETC FALLS INTO PUTC IN ORDER TO ECHO THE CHARA
00838          *
00839A C57D A6   C4     A RAWGTC LDA   ACIACR,U
00840A C57F 85   01     A       BITA  #.RDRF  WAIT FOR RECEIVER FULL
00841A C581 27   FA     A       BEQ   RAWGTC
00842A C583 A6   41     A       LDA   ACIADR,U PICKUP RECEIVED CHARACTER
00843          *
00844          *   RAWPTC --- SEND THE CHARACTER IN A OUT
00845          *
00846          *   U CONTAINS THE ADDRESS OF THE ACIA CONTROL REGI
00847          *   ALL REGISTERS RETURN UNCHANGED.
00848          *   THERE IS SPECIAL TREATMENT OF LF FOR SLOW TERMI
00849          *
00850          *
00851A C585 34   02     A RAWPTC PSHS  A       SAVE IT
00852A C587 A6   C4     A RPUTC2 LDA   ACIACR,U
00853A C589 85   02     A       BITA  #.TDRE  WAIT FOR EMPTY ACIA
00854A C58B 27   FA     A C587   BEQ   RPUTC2
00855A C58D 35   02     A       PULS  A       PICKUP CHARACTER
00856A C58F A7   41     A       STA   ACIADR,U AND SEND IT ON ITS WAY
00857A C591 81   0A     A       CMPA  #LF    FUDGE FOR SLOW BANTAMS!
00858A C593 27   01     A C596   BEQ   PRDLY
00859A C595 39          RTS
00860A C596 34   02     A PRDLY PSHS  A       SAVE IT
00861A C598 4F          CLRA
00862A C599 8D   EA     C585   BSR   RAWPTC
00863A C59B 8D   E8     C585   BSR   RAWPTC
00864A C59D 8D   E6     C585   BSR   RAWPTC
00865A C59F 35   82     A       PULS  A,PC
00866          *
00867          *   INZACI --- INITIALIZE AN ACIA
00868          *
00869          *   U CONTAINS THE ADDRESS OF THE ACIA CONTROL REGI
00870          *

```

```

00871A C5A1 34 02 A INZACI PSHS A SAVE
00872A C5A3 86 03 A LDA #00000011 MASTER RESET ACIA
00873A C5A5 A7 C4 A STA ACIACR,U
00874A C5A7 86 15 A LDA #00010101 8DATA,1STOP,DIVIDE BY 16
00875A C5A9 A7 C4 A STA ACIACR,U INITIALIZE CONTROL REG
00876A C5AB 35 82 A PULS A,PC RESTORE AND RETURN
00877 *
00878 * RAWEMP --- CHECK IF A CHARACTER IS READY
00879 *
00880 * U CONTAINS THE ADDRESS OF THE CONTROL REGISTER,
00881 * IS TESTED TO SEE IF A CHARACTER HAS BEEN RECEIV
00882 *
00883 * RETURNS WITH A=0 IF NO CHARACTER PRESENT, OTHER
00884 *
00885A C5AD A6 C4 A RAWEMP LDA ACIACR,U PICKUP STATUS BYTE
00886A C5AF 84 01 A ANDA #.RDRF A CHARACTER PRESENT?
00887A C5B1 27 02 C5B5 BEQ RAWEXT NO QUIT WITH A=0
00888A C5B3 86 01 A LDA #1
00889A C5B5 39 RAWEXT RTS
00890 END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000

```

```

0001 .RDRF 00830*00840 00886
0002 .TDRE 00831*00853
F84A ACCESS 00081*00274 00347 00354 00479
0000 ACIACR 00828*00839 00852 00873 00875 00885
0001 ACIADR 00829*00842 00856
C1FE ADDPAG 00165 00230*
C229 ADDX 00233 00235 00241 00243 00251*
002E AGAIN 00307 00519*
0001 AOFF 00011*00398 00770 00776
005E BACK 00314 00521*
0007 BEEP 00056*
0020 BLANK 00063*00633
0002 BOFF 00012*
0008 BS 00057*
0001 C 00029*00040 00559
0000 COFF 00010*00382
E008 CONSOL 00090*00130 00766 00773
000D CR 00061*00194 00196 00198 00200 00202 00204 00206 00208 00210 00212
00214 00216 00371 00373 00520 00638
C48B CRLF 00136 00294 00439 00490 00495 00510 00637*
C04F CTAB 00143 00162*
D7AF CURTAS 00093*00133 00273 00279 00286 00298 00380 00396 00418 00429 00443
00459 00466 00477 00493
C3EF DISP2 00498*00512
C407 DISP5 00509 00511*
C3DB DISPLA 00167 00490*
0001 DOFF 00013*
0003 DPOFF 00014*
0080 E 00022*00033
0000 EOS 00055*00218 00221 00223 00375 00401 00473 00516 00652
001B ESC 00062*
C3A0 EXECUT 00177 00459*
0040 F 00023*00034 00798

```

```

0200 FAULT 00089*00266.00481
000C FF 00060*
FFF6 FIRQVC 00047*
0004 FREE 00112*00355
1000 FREEPG 00113*00358
C2A5 FUBYTE 00299 00325*00383 00405 00444 00729 00732 00735 00777
F849 FUSE 00080*00800
000D FWD 00310 00520*
C454 GET22 00569 00580*
C452 GET23 00576 00579*
C441 GET2HX 00240 00259 00282 00302 00407 00435 00446 00467 00494 00568*00592
00604 00607
C461 GET32 00595*
C455 GET3HX 00232 00589*
C46E GET42 00590 00593 00605 00608 00610*
C462 GET4HX 00291 00423 00604*00616 00620
0000 GETC 00068*00139 00544
C420 GETNYB 00544*00568 00575 00589
C2B5 GETPAG 00326 00335 00346*
C46F GETRNG 00616*
C482 GETRX 00617 00621 00625*
C568 GETUS 00381 00397 00411 00419 00727 00739 00769 00775 00812*
C438 GOTIT 00550 00556*
0020 H 00024*00035
C071 HELP 00183 00188*
C078 HELPIN 00188 00194*
C382 HEX2 00442*00450
C37A HEX3 00439*00453
C372 HEXDUM 00181 00435*
C39F HEXX 00436 00455*
C049 HUH 00145 00155*
0010 I 00025*00036 00798
C00B INIT 00122 00127*
C5A1 INZACI 00131 00871*
FFF8 IRQVEC 00048*
000A LF 00059*00194 00196 00198 00200 00202 00204 00206 00208 00210 00212
00214 00216 00371 00373 00640 00857
C1EF LOGMSG 00222*00238 00257
C033 LOOP 00144*00149
C03F LOOP2 00147 00150*
C01F MAIN 00136*00153 00157
C415 MAPUP 00142 00531*00545
0400 MAXPAG 00087*00234
0020 MAXTAS 00088*
C29B MEM2 00311 00314*
C2A3 MEM3 00315 00318*
C267 MEMORY 00175 00291*
C2A4 MEMX 00292 00319*
C26E MLOOP 00294*00306 00308 00313 00317
C28F MMOVE 00303 00307*
F800 MMU 00076*00077 00078 00079 00080 00081 00082 00248 00264 00352 00355
00480 00483 00497 00511
F840 MMUO 00077*
F847 MMU7 00078*
C2E1 MSSTR 00370*00399
0008 N 00026*00037
00FE NC 00040*00557 00624
0003 NCALLS 00750 00763*

```

```
007F NE      00033*
00BF NF      00034*
00DF NH      00035*
00EF NI      00036*
FFFC NMIVEC  00050*
00F7 NN      00037*
C41F NOMAP   00532 00534 00536*
C43D NOTHEX  00548 00553 00555 00559*
FFFO NOVEC   00044*
0020 NPAGE   00086*00242 00261 00483 00483 00511 00511
0020 NTASK   00085*00284
00FD NV      00039*
00FB NZ      00038*
C045 OFFSET  00152*00163 00165 00167 00169 00171 00173 00175 00177 00179 00181
00183
F84B OPERAT  00082*00134 00460 00465 00719 00726 00768 00774 00792
C1E0 PHYMSG  00220*00230
000A POFF    00018*00728 00740
C596 PRDLY   00858 00860*
005F PROMPT  00137 00518*
0800 PSIZE   00084*
C499 PSTR2   00651*00655
C4A3 PSTR9   00653 00656*
C497 PSTRNG  00189 00231 00239 00258 00469 00492 00650*
C4B5 PUT2HX  00280 00300 00673*00689 00698 00700
C4C2 PUT3HX  00505 00687*
C4CB PUT4HX  00296 00413 00421 00441 00698*
0001 PUTC    00069*00138 00156 00391 00409 00634 00639 00641 00654 00666
C4B1 PUTNY2  00664 00666*
C4A5 PUTNYB  00660*00678 00680 00687
C483 PUTS    00141 00281 00297 00301 00422 00442 00506 00619 00632*
C572 PUTUS   00430 00820*
C5AD RAWEMP  00885*
C5B5 RAWEXT  00887 00889*
C57D RAWGTC  00767 00839*00841
C585 RAWPTC  00778 00851*00862 00863 00864
C30A REGS    00163 00380*
C323 REGS2   00389 00391*
C31B REGS3   00387*00393
C34D REGS4   00402 00411*
C349 REGS5   00404 00409*
C335 REGS6   00400*00408 00410
C22A REMOVE  00169 00257*
C246 REMX    00260 00262 00268*
FFFE RESVEC  00051*
C548 RETURN  00461 00751 00772 00779 00792*
C587 RPUTC2  00852*00854
C2D9 RSTRNG  00368*00385
007F RUBOUT  00064*
F848 SBIT    00079*
C567 SIMPLE  00793 00808*
D7B0 SPTAB   00092*00128 00718 00722 00795 00796 00814 00822
C356 STACK   00418*
D7A0 STACKP  00091*00127 00179
C371 STACKX  00424 00431*
C000 START   00120*
C2AD SUBYTE  00304 00334*00742 00745 00771
FFF4 SWI2VC  00046*
```

```
FFF2 SWI3VC 00045*
C4D4 SWIH 00123 00718*
C51F SWITAB 00752 00759*00759 00760 00761
FFFA SWIVFC 00049*
0009 TAB 00058*
C251 TASK 00171 00279*
C266 TASKX 00283 00285 00287*
C40D TASMMSG 00491 00515*
C525 UGETC 00759 00766*
0008 UOFF 00017*
C536 UPUTC 00760 00773*
C547 UQUIT 00761 00780*
0002 V 00028*00039
C247 WINDOW 00247 00263 00272*00502
C3B9 XITMSG 00468 00472*
0004 XOFF 00015*
0006 YOFF 00016*
0004 Z 00027*00038
C3C3 ZAP 00173 00477*
C3D2 ZAP2 00482*00484
C3DA ZAPX 00478 00485*
```


Motorola reserves the right to make changes to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.



MOTOROLA *Semiconductor Products Inc.*

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721 • A SUBSIDIARY OF MOTOROLA INC.