RCA MICROPROCESSORS 1800

# DESIGN IDEAS BOOK

## for the CDP1802 COSMAC Microprocessor

BMP 802

# DESIGN IDEAS BOOK

# FOR THE CDP1802

# COSMAC MICROPROCESSOR

## ACKNOWLEDGEMENT

We would like to express our thanks to all contributors

D. Block
R. Boon
J. Johnson
J. Nizet
J. Paradise
J. Pintaske
D. Shand
J. Strahler
H. Tweddle

and especially M-C. Vandecaasbeek for her considerate typing work.

RCA Applications Team - Europe.

CONTENTS

CONTENTS (cont'd) - (2)

# INTRODUCTION

New and integrated circuits, particularly microprocessors, are often thought difficult to understand. This is a myth, not true, they are complex but not difficult to understand as they operate in a logical way. Understanding microprocessors does require study, explanations and guidelines and the purpose of this book is to provide such information to help you become really familiar with these devices and especially the CDP1802, a CMOS microprocessor.

Happy reading and good luck.

RCA Applications Team - Europe.

# LIST OF 1800 SERIES DEVICES

| | | | | |
|---|---|---|---|---|
| CDP | 1802 | CMOS | Microprocessor | |
| CDP | 1804 | CMOS | 1 Chip Microcomputer | |
| CDP | 1805 | CMOS | Microprocessor high performance | |
| CDP | 1821 | CMOS | RAM 1K x 1 | |
| CDP | 1822 | CMOS | RAM 256 x 4 | |
| CDP | 1823 | CMOS | RAM 128 x 8 | |
| CDP | 1824 | CMOS | RAM 32 x 8 | |
| CDP | 1825 | CMOS | RAM 1K x 4 | |
| CDP | 1831 | CMOS | ROM 512 x 8 | (on chip decoding) |
| CDP | 1832 | CMOS | ROM 512 x 8 | |
| CDP | 1833 | CMOS | ROM 1K x 8 | (on chip decoding) |
| CDP | 1834 | CMOS | ROM 1K x 8 | |
| CDP | 1835 | CMOS | ROM 2K x 8 | (on chip decoding) |
| CDP | 18U42 | CMOS | EPROM 256 x 8 | |
| CDP | 1851 | CMOS | PIO with interrupt conditions | |
| CDP | 1852 | CMOS | Input-Output port | |
| CDP | 1853 | CMOS | 1 of 8 decoder | |
| CDP | 1854 | CMOS | UART up to 500K bit/sec | |
| CDP | 1855 | CMOS | Multiply/divide unit | |
| CDP | 1856 | CMOS | Bus buffer for memories | |
| CDP | 1857 | CMOS | Bus buffer for I/O | |
| CPD | 1858 | CMOS | Latch-decoder for memories | |
| CDP | 1859 | CMOS | Latch-decoder for memories | |

| | | | |
|---|---|---|---|
| CDP | 1861 | CMOS | Video display controller |
| CDP | 1862 | CMOS | Colour generator controller |
| CDP | 1863 | CMOS | Programmable frequency generator |
| CDP | 1864 | CMOS | TV interface in PAL |
| CDP | 1866 | CMOS | 4 bit latch decoder |
| CDP | 1867 | CMOS | 4 bit latch decoder |
| CDP | 1868 | CMOS | 4 bit latch decoder |
| CDP | 1869 | CMOS | Video interface address and sound |
| CDP | 1870 | CMOS | Video interface color generator |
| CDP | 1871 | CMOS | ASCII keyboard encoder |
| CDP | 1872 | CMOS | Input port high speed |
| CDP | 1873 | CMOS | 1 of 8 binary decoder high speed |
| CDP | 1874 | CMOS | Input port high speed |
| CDP | 1875 | CMOS | Output port high speed |
| CDP | 1876 | CMOS | Video interface color generator RGB |

This list shows only CDP 1800 series devices other parts are MWS series for memories and the whole CD4000 series.

## SPECIALITIES USING THE 1800 SERIES

. Small systems even without RAM

  - Subroutines calls do not need RAM, on chip registers might be enough memory.

. In some cases even interrupt is possible without RAM

. Very fast interrupt response saves only what is needed

. Optimize addressing modes

. Arithmetic on program counters

. Several stack pointers

. Multiple data pointers

. Some program counters for fast subroutine response

. Easy to implement interpreters

. Special "macro instructions" possible

. CMOS intelligence without power consumption

. CDP 1802 structural advantages

. DMA transfer built in

. Complete system without selection logic for RAM and ROM

. CMOS ROMs for sequential CMOS systems

. Software interrupts using SEP

. 16 bit I/O in one instruction

. Power reduction by use of WAIT line

. Power reduction using frequency reduction

# SHORT DESCRIPTION

## of CDP1802 CMOS Microprocessor

One possibility to understand this Microprocessor could be to set up a wishlist and transfer it into CMOS afterwards :

1. A sixteen bit program counter is needed to address 64K.

2. The accumulator together with the ALU (arithmetic and logic unit) manipulates the data, 8 bit (1 byte) wide.

3. Add instructions (e.g. FF + FF) give an overflow in the accumulator (D-register), so a DF (data flag) is needed.

4. A stack pointer points to a free location in RAM to store and load from memory via program control (16 bit long).

5. One pointer is normally not enough, 16 could be the optimum (all 16 bit long).

6. To address one of these registers, 4 bit are needed (N-designator).This gives the possibility for memory operations to define for example : load into register from RAM and use register 7 as the address pointer, then store it, using register 9 as pointer to another address.

7. The instruction to be executed is 8 bit long; 4 bit are N definition and the other 4 bit are the instruction, being loaded into an I register.

8. The disadvantage of always defining a register could be overcome by defining and X-register with special instructions using this designator. This X can be changed as needed during program flow.

9. This designator usage as for the X pointer could be used as well for something completely different. As defined before all registers are 16 bit long.

   Introducing a 4 bit P register pointer adds the special feature of multiple program counters. One instruction would change from one program counter to another by just changing the contents of P with one instruction.

10. After reset P is set to Ø and program execution starts at ØØØØ with register Ø as program counter.

11. To have after reset a defined X pointer, it is set to Ø as well.

12. A serial input and output is very useful for a serial data link to a terminal without a lot of external hardware. This is done via a Q output, a flip flop and a flag input.

13. One flag input line only is very often not enough. For a minimum system four input flags all together should be sufficient. The Qoutput could even be used to multiplex this to 8 input lines.

14. Interrupt is a needed feature, but the problem is to define how much to put on the chip. The easiest way is to have one line with a very fast response and to optimise externally with additional hardware, for example using the EF lines to distinguish between 4 interrupts for a less complex system already.

15. The P register definition allows easily to have a fast interrupt response. This instruction is modified and automatically puts 1 in the P-register and 2 in the X-register, then starting program execution from the location where R1 points to.

16. To allow or not allow an interrupt a IE-flip-flop is needed (interrupt enable).

17. The very fast interrupt response changes the program counter to 1 but it has to be remembered somehow, how X and P were before the interrupt. This is done via the T-register. If an interrupt occurs, the contents of X, P is automatically transferred into the T register, before being changed to 2, 1.

    A save of the T-register on stack is normally the first instruction of the interrupt program.

18. A Microprocessor has to communicate via I/O parts. If the ports are in memory, even for a minimum system, decoding logic is needed; as well the transfer has to go through the CPU (as it can only address one device at the same time (memory - CPU - I/O). With some special I/O lines this can be overcome and a direct transfer from memory to I/O is possible without involving the CPU, except for addressing. Three lines (NØ, N1, N2) can address up to 8 devices (8 x IN, 8 x OUT). For a small system 3 I/O devices can be selected directly, even without additional logic.

19. This structure leads to an included DMA channel. As the I/O transfer is directly to memory (or from), register Ø is used to point to a memory location, and the DMA channel works both ways (input and output).

    Using cycle stealing, an S2 (DMA) cycle is inserted between normal fetch and execution. DMA-IN means that the byte on the data bus is stored at the location where RØ points to and the pointer is incremented to be prepared for the next DMA-IN. DMA-OUT takes the byte where RØ points to and puts it on the data bus to be stored in an output part, incrementing itself after the transfer.

    (The DMA output feature is used together with the video controllers CDP1861, CDP1864).

20. As all these lines together add up to more than 40 lines, a reduction has to be made. The 16 lines of the address bus can be reduced to 8 lines by sending out the high byte together with a reference pulse and then the low byte.

    This sequential addressing leads to the possibility of internal ROM address selection. As the whole address is on the address bus, the high byte can be latched in the ROM and decoded. This allows a 64K ROM system without any external decoding logic. The mask for the on chip decoding logic defines the address area of each ROM.

Figure 1 shows the pinout of the CDP1802 and figure 4 the internal structure.

Some of the pin functions need further explanations :

. CLOCK (1) and $\overline{\text{XTAL}}$ (39) form a crystal generator gate
. $\overline{\text{WAIT}}$ (2) and $\overline{\text{CLEAR}}$ (3) define the mode of the CPU

1. RUN is the normal mode of the CPU
2. PAUSE means freeze the CPU, for example for slow memories or lower power consumption.
3. RESET to have a defined start condition (RØ = 0000, X = 0, P = 0, Q = 0, IE = 1)
4. LOAD uses the DMA channel to load data, but stopping program execution.

. Q (4) is the flip-flop under program control
. SC1 (5), SC0 (6) represent the state of the CPU (See Fig. 3)
. $\overline{\text{MRD}}$ (7) memory read output
. BUS 0-7 (8-15) - the data bus
. $V_{CC}$ (16) I/O voltage, separated from internal voltage $V_{DD}$
. NØ-N2 (17-19) I/O address lines
. $V_{SS}$ (20) ground
. $\overline{\text{EF1}}$-$\overline{\text{EF4}}$ (21-24) input flag lines to be tested under program control
. MA0-7 (25-32) multiplexed address bus
. TPB (33) reference pulse for I/O transfer
. TPA (34) reference pulse for high order address byte
. $\overline{\text{MWR}}$ (35) memory write
. $\overline{\text{INT}}$ (36) interrupt request line
. $\overline{\text{DMA OUT}}$ (37) DMA output pulse to get a byte from memory, pointed to by RØ
. $\overline{\text{DMAIN}}$ (38) DMA input pulse to store a byte into memory pointed to by RØ
. $V_{DD}$ (40) internal processor voltage, can be higher or equal $V_{CC}$ to work at higher frequency

One advantage of this Microprocessor is its flexibility of changing pointers and program counters with just one instruction.

High or low operating speed is only dependent on the application. As this Microprocessor is designed in static CMOS logic, the user sets the clock frequency(up to 6.4MHz for example). But the low current consumption at low frequencies might also be important. This processor can easily modify its own clock frequency, for example using the Q output to decide between two clock frequencies with some external devices.

Figure 5 shows a block diagram of a CDP1802 system. Figure 6 gives the timing relations.

Appendix A shows the instruction set of the CDP1802.

For more information about CDP1802, see data sheet of CDP1802 or MPM201 user manual.


Figure 1 : PIN OUT

| CLEAR | WAIT | MODE |
|---|---|---|
| L | L | Load |
| L | H | Reset |
| H | L | Pause |
| H | H | Run |

Figure 2 : CDP1802 modes

| State Type | State Code Lines | |
|---|---|---|
| | SC1 | SC0 |
| S0 (Fetch) | L | L |
| S1 (Execute) | L | H |
| S2 (DMA) | H | L |
| S3 (Interrupt) | H | H |

Figure 3 : State Codes

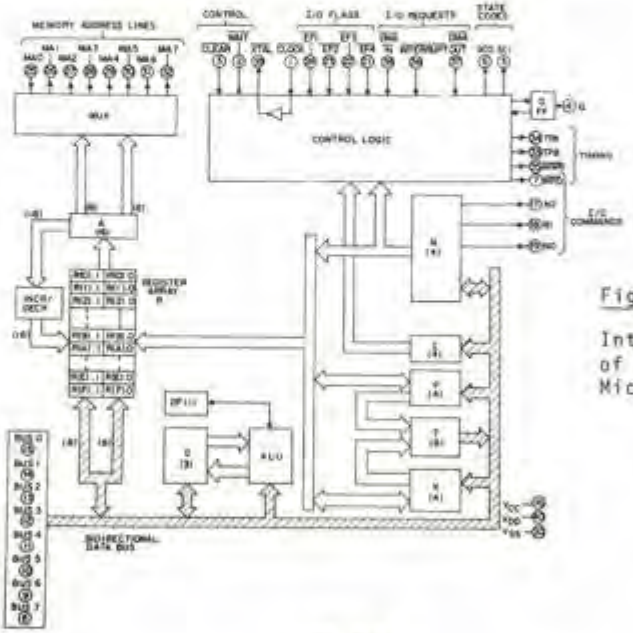



Figure 4 :

Internal Structure of the CDP1802 Microprocessor

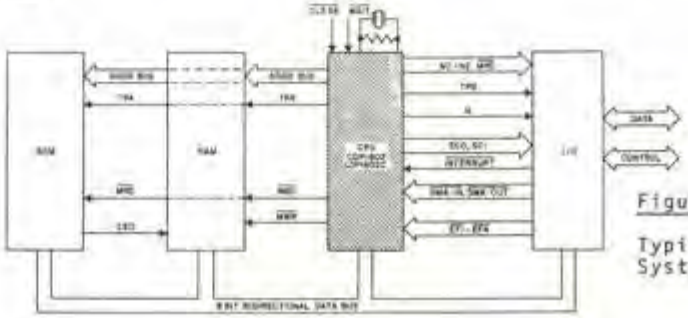
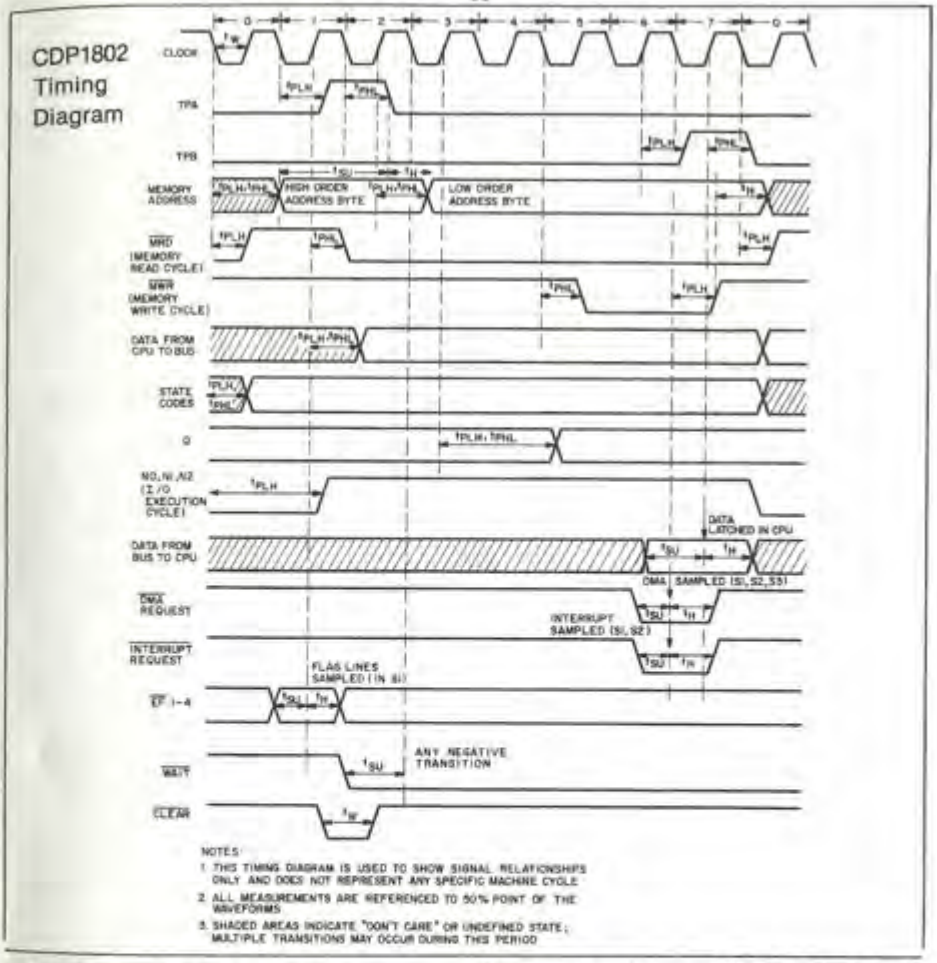



Figure 5 :

Typical CDP1800 System




CDP1802 INSTRUCTION EXECUTION TIMES

1-Byte instructions require 2 Machine Cycles: 1 Fetch, 1 Execute.

2-Byte instructions also require 2 Machine Cycles: 1 Fetch, 1 Execute.

3-Byte instructions require 3 Machine Cycles: 1 Fetch, 2 Executes.

| INSTRUCTION SIZE | | | % OF TOTAL REPERTOIRE | INSTRUCTION TIME @ f = | | |
|---|---|---|---|---|---|---|
| Bytes of Op. Code | Bytes of Address or Data | Total Bytes of Memory | | 6.4 MHz | 3.2 MHz | 2 MHz |
| 1 | 0 | 1 | 93 | 2.5 µs | 5 µs | 8 µs |
| 1 | 1 | 2 | | | | |
| 1 | 2 | 3 | 7 | 3.75 µs | 7.5 µs | 12 µs |
| Average Instruction Execution Time | | | – | 2.6 µs | 5.2 µs | 8.3 µs |

Figure 6 : CDP1802 Timing Diagram

## COSMAC INSTRUCTION SET

In order to better understand the instruction set of the CDP1802, some explanations are necessary (see APP. A) :

1. As the CDP1802 has 16 registers, many instructions exist 16 times, for example the first instruction ØN means LOAD via N, with N = 0...F. This means that one of the 16 registers (N) is used as pointer (16 bit) to memory and the information (data) at this location is loaded into the D-register. This "N" is valid for 10 instructions and so uses already 160 of the 256 possible op-codes.

2. A second type of instructions is referenced to register X. 73 means STXD (store via X and decrement). Here the 4 bit X-designator (0-F) points to one of the 16 registers. The contents of this register (16 bit) points to one of the 65K addresses. At this location, the information of the D-register is stored and afterwards this register is automatically decremented (stack operations) to point again to a free location.

3. All registers can be incremented and decremented.

   This gives easy-to-use counters :

   a) Initialize register
   b) Decrement register
   c) Take half of it in D
   d) Is it 00 ?
   e) If ØØ exit, if not back to decrement

4. For logic and arithmetic operations, there are two classes of instructions :

   a) D-register and the immediate byte after the instruction (ADI)

   b) D-register and the byte where the X-register points to (ADD)

5. Branches are always within a page, means they are absolute, it is not possible to branch over page boundaries.

6. For jumps outside a page, long branches are used.

7. A special control instruction is IDL. This stops the program counter and it waits for DMA or INTERRUPT.

8. I/O transfer is always between memory and I/O, this means :

   a) Initialize memory pointer
   b) Set X to this register
   c) Output or input

9. For further explanations, see MPM-201 user manual.

## STATIC - WHAT DOES IT MEAN FOR YOUR DESIGN

Power saving is a characteristic which is featured in many present Microprocessor system designs, both as a necessity of the specification. For example in portable systems or in critical power applications - telephone, etc... and as a useful selling point with low consumption and smaller unit size being an advantage over competitors.

The COSMAC CMOS Microprocessor and memory series can easily offer this characteristic by its static feature. In addition to power savings due to CMOS, making it applicable in such diverse applications such as intelligent telephones, portable data systems, fire hazard equipment.

The COSMAC series is based around the CDP1802 8-bit processor and the CDP1804 single-chip 8-bit Microcomputer. The Microprocessors have an unique architectural feature of stack pointer and program counter designators, which allow the SP and PC to be redefined in one instruction to any of two or one 16-bit register pairs.

Both processors feature on-chip interrupt,DMA, and I/O select encoding lines, four testable flag (EF) inputs and one setable and testable output (Q). Additionally the 1804 has an on-chip 8-bit counter timer, 64 bytes RAM and 2K ROM.

The COSMAC family also consists of I/O-devices:multiply divide unit 1855, peripheral I/O 1851, video interface devices - 1869, 1870, 1861, 1862, 1864, programmable frequency generator 1863, eight bit I/O port 1852, UART device 1854 and memory, and I/O support devices.

The memory family consists of CMOS ROM's, EPROM and RAM's.

As CMOS devices, in addition to the static feature they offer high noise immunity, lower self heating and therefore high reliability.

What is static ?

Static operation enables a device to function from DC to the devices upper frequency limit. This characteristic is due to static memory cells being employed in all registers. As a result the device requires no minimum refresh rate as in the case of dynamic cells and it can be stopped without data loss.

When one considers that with CMOS the major component of power consumption is dynamic and that this is proportional to frequency then one advantage of static operation is obviously seen that of power consumption.

"BY REDUCING FREQUENCY ONE CAN REDUCE OPERATIONAL POWER CONSUMPTION"

As data is retained independant of the clock in a static device, it can be switched off by stopping the clock and switched on to initiate processing at the same point. This has advantages in easy device control.

The capability of varying the operational speed from DC to maximum has many advantages, such as in interfacing with slower devices, memories and peripherals. The operation speed in addition to the hardware and software can be tailored to a particular application.

In the application shown in Figure 1 a COSMAC system derives a pulse by entering the subroutine on testing for an active flag input (EF1, 2, 3, 4) on occurence of a second flag input e.g. (EF2) the clock speed is modified, therefore, the pulse length produced will be changed. The design therefore minimizes software by a simple hardware alteration, and, therefore, can be used in a memory/software critical development.

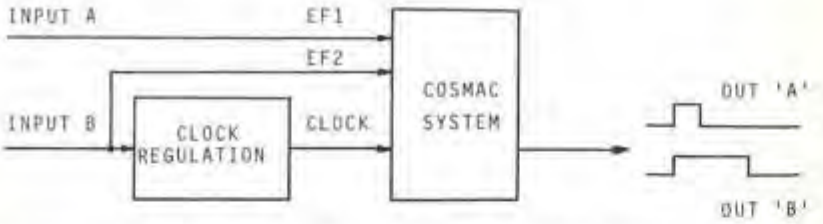



Figure 1 : COSMAC System Regulated by Changing Clock

In timing applications the technique is useful due to COSMAC's instruction set - the majority of instructions are two machine cycles long. *

Strict timing is achieveable by instruction counting and by clock adjustment.

Using a technique unique to COSMAC and well known to 1800 series users, it is possible to implement an automatic software counter. When a DMA request is asserted in COSMAC, a DMA machine cycle occurs, after setting the status line SC0 and resetting SC1 and incrementing one of the sixteen 1802 registers R0, as the DMA pointer.Data is input or output from memory pointed to by R0 dependent on whether request was DMA in or DMA out. By using SC1, the status line which is active during the execute machine cycle, to request DMA the register R0 is incremented every three machine cycles, if two machine cycle instructions are used exclusively. By adjusting the frequency, and using external circuitry to disable DMA request, this simple technique can be used in many timing applications to get a software clock, by connecting the DMA-out with SC1.

* 80% of 1802 instructions are two machine cycle, the remainder are three machine cycle.

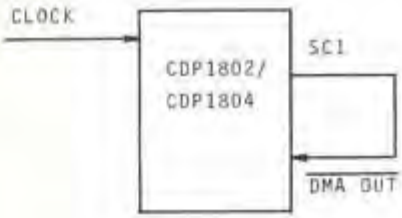



Figure 2 : Automatic Software Counter

After reset R0 is program counter and DMA pointer.

As DMA increments the program counter only every second, instruction is executed. This means that the initialization cycle looks like

```
ADD  DATA    MNEMONIC          COMMENT
00   XX                    .. DMA
01   F8 00   LDI#00        .. get  00 in D-register
03   XX                    .. DMA
04   B3      PHI R3        .. transfer to R3.1
05   XX                    .. DMA
06   F8 0C   LDI#0C        .. get  0C in D-register
08   XX                    .. DMA
09   A3      PLO R3        .. transfer to R3.0
0A   XX                    .. DMA
0B   D3      SEP R3        .. set PC to R3
0C                         .. now program starts
                           .. running in R3
                           .. and R0 is only
                           .. used for DMA
```

Static characteristic also allows the Microprocessor to share the same clock as required in the other part of the system or in fact be tailored to the external system. This is shown in Figure 3.

A system is shown in Figure 3 implementing a data communication interface using the same clock for the CPU and the baud rate input. With COSMAC a simple low frequency interface can be constructed omitting a down counter and crystal by using a simple cheap RC oscillator constructed from a single CMOS schmitt-trigger. This, therefore, reduces system costs.

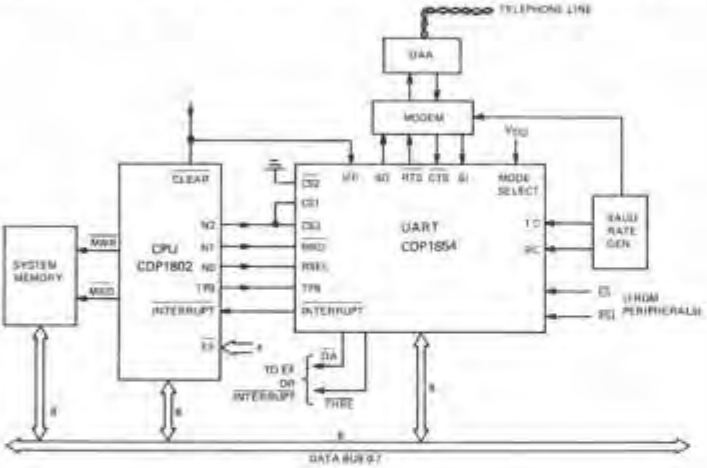



Figure 3 : Data Communication Interface Using CDP1854

The main advantage of a static Microprocessor system however is in power saving, in power critical applications.

A COSMAC operating at 1MHz at a typical operating power of 10mW can be reduced to almost quiescent power levels by reducing the clock frequency. The implementation of such a system is shown in Figure 4 where it must operate at a low power in an idle state, while waiting for an interrupt.

In COSMAC an idle state is indicated by a constant SC1 high, which is used to switch the oscillator down to 50KHz. The advantage of this system is that the processor is still running and an interrupt or DMA request is detectable, automatically returning SC0 to a low state and the clock to its normal operation frequency.

TABLE 1

POWER CONSUMPTION OF A TYPICAL COSMAC SYSTEM AT A CLOCK FREQUENCY OF 1MHz AND 200KHz

Typical power consumption at 5V

| | AT 1MHz | AT 200KHz |
|---|---|---|
| 1 x CPU (CDP1802) | 4mW | 0.2mW |
| 1 x RAM (MWS5101) | 0.75mW | 0.2mW |
| 1 x ROM (CDP1833) | 10mW | 2mW |
| All together | 14.75mW | 2.4mW |

TABLE 2

Static consumption

| | AT 5V (max.) | AT 2V (data re |
|---|---|---|
| 1 x CPU (CDP1802) | 50uA | 0.5uA |
| 1 x RAM (MWS5101) | 50uA | 2uA |
| 1 x ROM (CDP1833) | 50uA | 1uA |
| Sum of static current consumption | 150uA | 3.5uA |

The COSMAC range of Microprocessor components can be incorporated in designs using the techniques above due to static properties of CMOS giving the Microprocessor user, in addition to other CMOS features, an extremely powerful design tool.

These advantages of the COSMAC family have led to its usage in many fields of Microprocessor applications - portable equipment, fire hazard equipment, telecoms and automotive - where it is very successful in applications such as ignition control, information systems, transmission control etc...

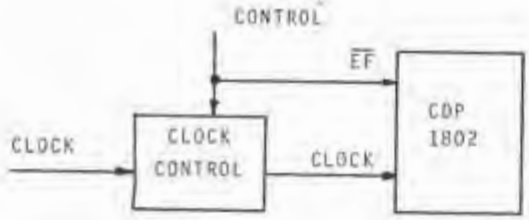



Figure 4 :
General Clock Control

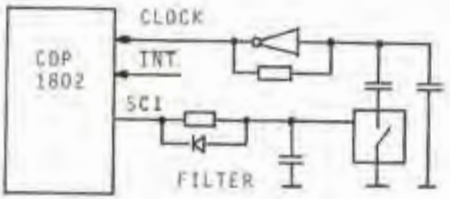



Figure 5 :
Automatic IDLE System

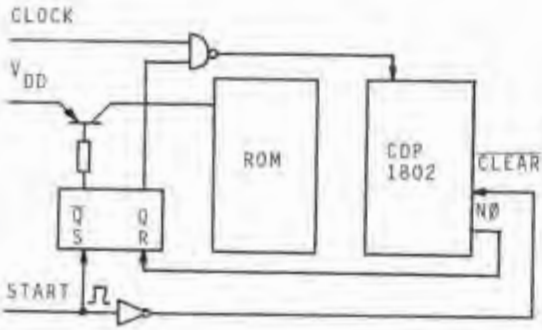



Figure 6 :
IDLE System Stopping the Clock

## OSCILLATOR DESIGN CONSIDERATIONS FOR THE CDP1802

Despite the widespread use of crystal-controlled oscillators for Microprocessors, crystal selection may still pose problems for many designers. Most of these problems can be minimized by an understanding of the various properties and specifications needed to define an oscillator circuit for the CDP1802 Microprocessor.

### Clock frequency and accuracy

Quartz crystal oscillators will provide frequency stability better than 0.01%. However, many Microprocessor applications do not require an exact clock frequency - therefore, the use of RC or LC type oscillators may be a wise cost-effective choice. If a crystal is to be used, there are two basic low-cost types to consider : parallel resonant AT cut quartz crystals typically ranging from 0.8MHz to 6MHz, and low frequency tuning fork type crystal available in several clock frequencies from 10KHz to 240KHz, including the popular 32.768KHz digital watch frequency. Statek Corp., for instance, specializes in low-cost tuning fork crystals, and they have numerous free applications notes to aid in design .

### The oscillator circuit

Figure 1 shows the basic crystal oscillator circuit for the CDP1802. The 15 megohm resistor is used to bias the gate in its linear region so that it behaves like an amplifier. Capacitors $C_S$ and $C_T$ provide the required capacitive loading for

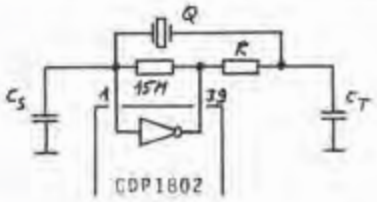



Figure 1 : Basic Crystal Oscillator

the crystal and act as high-frequency filters to avoid overtone oscillations. The values of $C_S$ and $C_T$ can be calculated using the following equations found in ICAN-6086 :

$$C_T = \frac{4C_L}{1 - 5f\ R_e C_L}, \text{ and}$$

$$C_S = \frac{4C_L}{3 + 5f\ R_e C_L}$$

$R_e$ is the equivalent resistance of the crystal. Figure 2 shows the approximate relation of $R_e$ to frequency for AT type crystals.

$C_L$ is the load capacitance for the crystal, generally a standard value set by the crystal vendor between 10 and 32pF.

f is the frequency in Hz.

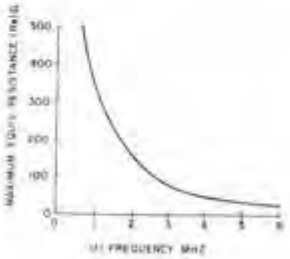



Figure 2 :

Equivalent Resistance Versus Frequency

The actual value $C_S$ used should be about 4pF less than the calculated value to allow for the amplifier input capacitance.

R can usually be 0 ohm unless low power drain or stability during variable $V_{DD}$ voltage is important. ICAN-6086 and ICAN-6539 explain in detail the need for and calculation of R.

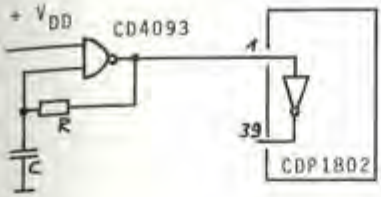



Figure 3 :

CD4093 as
Clock Oscillator

LC type -- A parallel-resonant LC circuit may be used as the frequency determining network for the CDP1802. (See Figure 4) The frequency and component values have the following relationship :

$$f = \frac{1}{2\pi\sqrt{LC}}$$

The high-impedance secondary of a small 455 KC IF transformer similar to those found in most portable transistor radios makes an excellent LC oscillator (the C is built in). Furthermore, it is tunable (about ± 10%) using the slug.

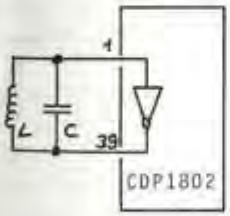



Figure 4 : Clock Oscillator Using LC Circuit

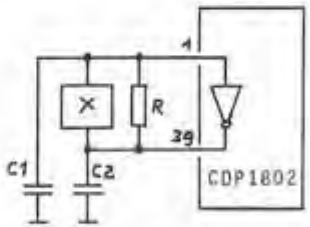



Figure 5 : Ceramic Resonator

Ceramic resonators

Made of piezo-electric material, ceramic resonators behave similar to crystals, requiring two capacitors and a bias resistor. Frequency tolerances of ± 1% are typical minimums. Values of C1, C2 and R depend on the device used.

Crystal specifications :

Frequency and tolerance -- ± 0.01% from -20 to 100°C is common. tighter tolerances down to ± 0.001% are available at additional cost (check with vendor).

Mode of oscillation -- Fundamental, parallel resonance.

Load capacitance ($C_L$) -- Typically between 10pF and 32pF - choose a stock value the vendor has. Higher values of $C_L$ will improve frequency stability, but lower values will decrease oscillation power consumption.

Maximum equivalent resistance ($R_e$) -- This number is related to the frequency. Most vendors will supply crystals with $R_e$ values lower then the curve shown in Figure 2.

Max. drive level -- The crystal should be able to dissipate 5 milliwatts of power. If the crystal cannot handle this level, frequency drift or even damage to the crystal may result. Crystals with lower drive capability, such as the tuning fork type, can be used if R is increased to reduce the drive level. ICAN-6086 gives details on how to compute R, also Statek has numerous free Application Notes about this.

Can types -- HC33 and HC18 are popular types.

Alternative oscillator types :

RC type -- A simple RC oscillator is shown in Figure 3 using a CD4093. The approximate frequency (f) is determined as follows :

$$f \approx \frac{1.25}{RC}, \text{ at } V_{DD} = 5V$$

Unlike the CDP1802, the CDP1804 Microprocessor uses a Schmitt inverter for the oscillator amplifier so that it can be used directly for the RC oscillator.

## A MINIMUM SYSTEM

To show the high density of CMOS integration, see this block diagram which includes all devices to have a minimum system, with 25 I/O lines.

1. The clock frequency of the CPU (1) is generated via a crystal; it could be replaced by a clock generator (1/4 CD4093). To get a clean POWER ON RESET 1/4 CD4093 is used. EF1 to 4 and the Q line are 5 I/O's already.

2. EPROMS CDP18U42 or 57U58 can be used, as well NMOS EPROMS are possible but this would need a bigger power supply.For production CMOS ROMS would be used

   Another possibility is to use UT4 (CDPR512), a monitor program in ROM, giving the possibility to control the CPU (see data sheet CDP1832).

3. The selection logic allows the following RAM's

   CDP1824 (32 x 8)
   CDP1823 (128 x 8)
   2 x CDP1822 (256 x 8)
   2 x CDP1825 (1K x 8)

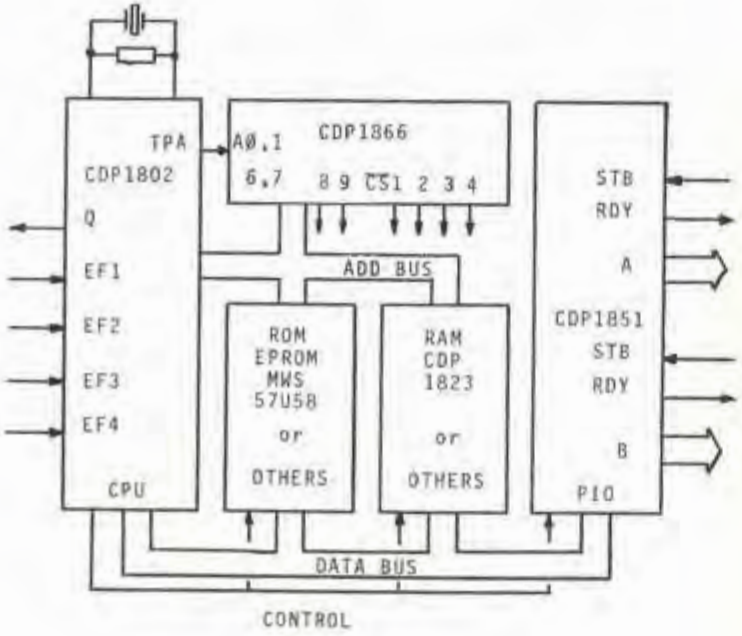
   The CDP1825 (MWS5114) should be enough memory for this minimum system.

4. The PIO (CDP1851) adds up to 20 I/O lines to this mini-system, they can be used as input, output, bidirectional or bit programmable.

5. A CDP1866 is used to demultiplex the address bus. Bit 0, 1, 6, 7 are latched and provide address bits 8 and 9; with 10 address lines a 1K EPROM can be addressed.

To select RAM or ROM, address bits 6/7 are used. This puts RAM at 0000 and ROM at 8000 or vice versa. As CDP1866 has 4$\overline{CS}$ outputs, two other RAMs or ROMs can be addressed.

The power consumption of the complete system, running at 2MHz is less than 10mA.




MINIMUM SYSTEM CIRCUIT DIAGRAM

## MINIMAL CDP1802 KEY OR SWITCH SCAN SYSTEM USING ADDRESS LINES

A keyboard scan can be implemented in a minimal CDP1802 system, by using only address lines as key scan and flag lines to monitor key depression.

Address lines are valid during the execute cycle of a "branch on EF" instruction at the point of flag test, as RP contents. A simple system can then be designed by utilising the upper four address lines in a key board matrix multiplex. A system is shown in Figure 1.

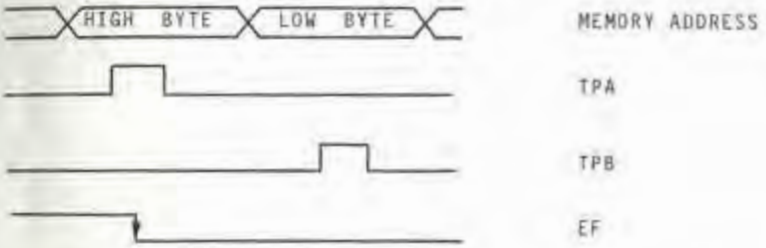



Figure 1 : Sampling of EF-Lines

No address overlap is ensured by placing memory below address FFF HEX and making the upper address lines don't care states in the mask of the ROM.

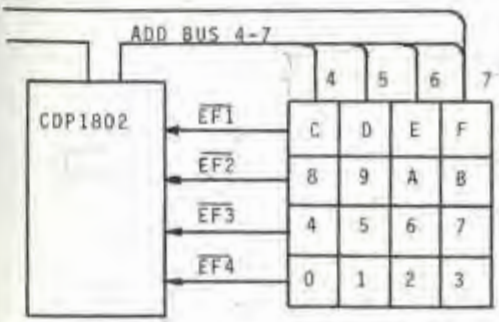



Figure 2 :

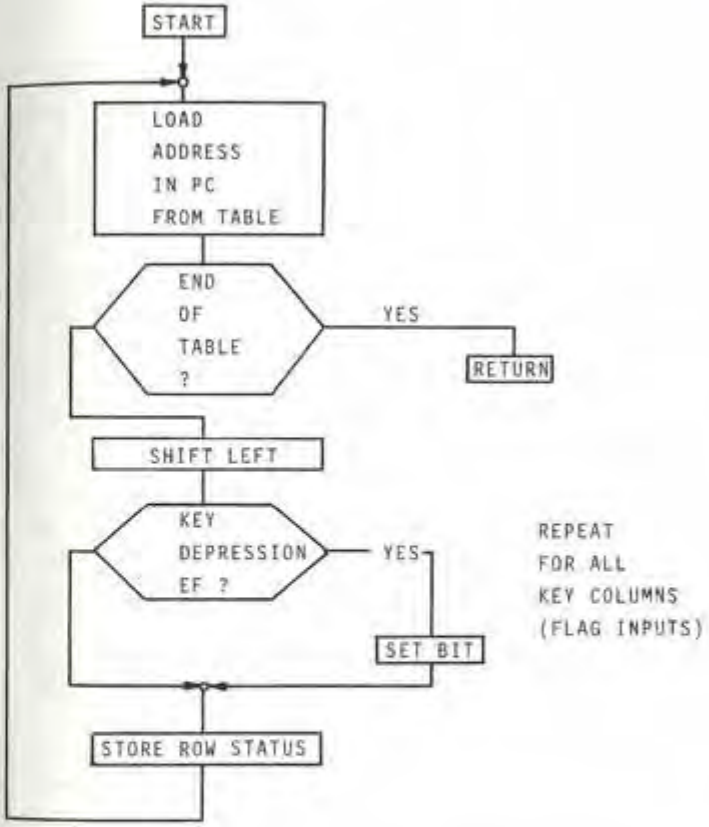
Hardware for Keyboard Scan Matrix

A key input routine can then be placed in memory and accessed with no dependence on MA7, 6, 5, 4. The key address lines are set by loading the program counter while continually accessing the routine. The routine would typically consist of key input - by bit set on valid EF condition, debounce, roll-over by software techniques.

Address line isolation is achieved by diodes on each line.

It is possible by using this technique, together with some 16 scratch pad registers as data storage of the 1802, to construct a two chip solution for simple key or switch scan and serial output application, such as remote portable data, control & monitor systems.

## Key scan software - flow chart




A flow chart of key scan is shown. Entry is by a subroutine call exit by "return" with a map of the key scan.

This routine only performs the function of scanning and storing a map of keys, debounce and coding functions must be done elsewhere.

Key scan software - code

```
RESCAN:  SEX   ROW TBL          ..  ROW SCAN FETCH FROM
         LDXA                   ..  TABLE AND LOADED INTO
         PHI   PC               ..  PROGRAM COUNTER
         ANI  #FØ               ..  END OF TABLE ?
         BNZ   TEST             ..  IF SO RETURN OTHERWISE
         RET                    ..  IF KEY DEPRESSION (FLAG IS SET)
TEST:    BN1   *+2              ..  THEN SET BIT OTHERWISE
         ADI  #01               ..  SHIFT + CLEAR NEXT BIT
         SHL                    ..  REPEAT FOR ALL KEY
         BN2   *+2              ..  COLUMNS (EF2, EF3, EF4)
         ADI  #01
         SHL
         BN3   *+2
         ADI  #01
         SHL
         BN4   *+2
         ADI  #01
         SEX   KEYPTR           ..  STORE MAP OF KEY
         STXD                   ..  ROW STATUS IN MEMORY
         BR    RESCAN           ..  BRANCH TO BEGINNING

DC             FIRADR, FINADR - 10, FIRADR - 30
                              FIRADR - 70, FIRADR - E0
                              .. ROW SCAN TABLE
                              .. FIRADR = 111ØY
                              .. WHERE Y IS ROM ADDRESS SELECT
```

The data table consists of four values setting the complement of each address line in turn and E.O.T. marker.

Note that due to ROM addressing loading the contents of this table into the program counter does not affect program flow.

## DMA INPUT - TO READ A PAPER TAPE

The DMA function in the CDP1802 works as follows :

A DMA pulse initiates a cycle stealing. The normal CPU operation is fetch-execute and again fetch new instruction and execute it with 8 clock cycles for the fetch and 8 or 16 clock cycles for the execute (2 or 3 cycle instructions).

Nearly all instructions are 2 cycles long, a 3 cycle instruction would be a LONG BRANCH or a NOP. If the CPU sees a DMA input for example, it will finish the current instruction and before the new fetch it will execute a DMA input :

RØ (register Ø) points to a byte in memory and the data coming from the input part is stored at the location where RØ points to, then this register is automatically incremented to point to the next free location.

There are some designs where this feature can be used :

1. The whole system is in RAM and after switch on the program is loaded via the DMA channel (CPU in LOAD mode).

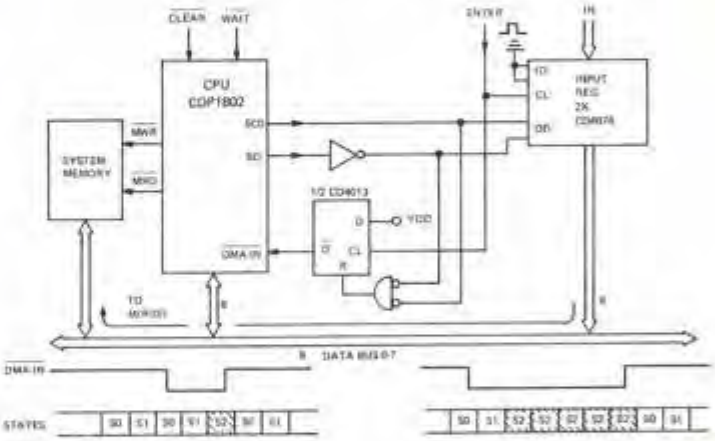
2. Fast data transfer from floppy disk.

3. This feature can easily be used as well to load a program from a paper tape.

   Nine LED's, 9 photo sensors and 9 Schmitt Triggers are all that is needed.

Eight of the sensors look at the data holes and the 9th (the sprocket hole) latches the current data into the CDP1852. Automatically the service request line initiates a DMA cycle (see Figure 1) and stores the information where R0 points to.

There are even hand-held paper tape readers on the market, their outputs are simply connected to the data and clock inputs of the CDP1852. (See MPM 201 page 79).




ADDITIONAL CIRCUITS FOR DMA INPUT

## DMA OUTPUT - CHECK THE MEMORY

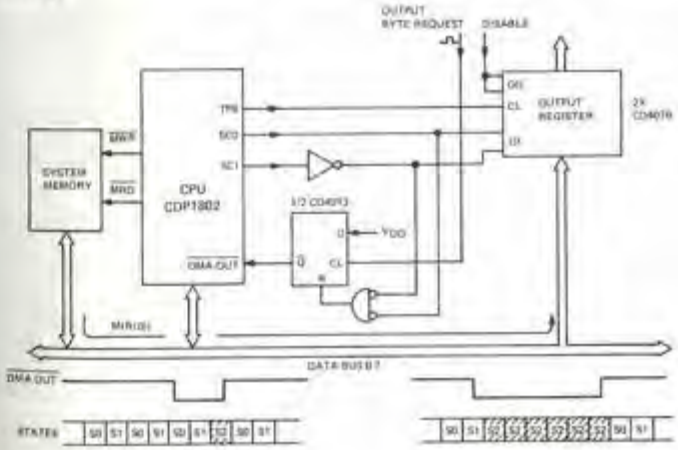
A low signal on the $\overline{\text{DMA-OUT}}$ line causes the same cycle stealing as described (S2 cycle) but now to output data.

Again register Ø points to a memory location and the data byte stored at this address is transfered to the output latch.

This is used for fast data transfer, either to another CPU or possibly to a floppy disk.

Another use is to step through the memory to verify data. Two LED displays with decoders are connected to the output register.

The CPU is reset and then put in the LOAD mode. RØ is now pointing to address 0000 and a $\overline{\text{DMA OUT}}$ pulse gets the data from this address into the output register and appears on the LED display. Further $\overline{\text{DMA OUT}}$'s will step through the whole memory.




ADDITIONAL CIRCUITS FOR DMA OUTPUT

See MPM-201, Page 80

## INTERRUPT MECHANISM IN THE CDP1802

The interrupt mechanism permits an external signal to interrupt normal program execution and transfer control to a program designed to handle the interrupt condition. This function is especially needed at power down to save data into the battery backup RAM or to respond very quickly to an exceptional event.

Here the structure of the CDP1802 shows its advantages. In its instruction set exists the SEP instruction to change from the current program counter register to any other. This instruction is just modified and gives a very fast response. For an interrupt, it is a SEP 1 instruction. So automatically, register 1 becomes the new program counter and register 2 stack pointer; the next instruction is fetched from the location where R1 points to.

As the interrupt program would not know which register was the PC of the interrupted program, the values of X and P are automatically saved in a T-register and further interrupts are inhibited (interrupt enable is reset).

The first two instructions of the interrupt program normally save the T-register on the stack and the D-register as well.

Now the IE can be set to 1 again to allow nested interrupts.

A program resumes from the interrupt normally using the RET instruction, which takes X and P from the stack and puts them in the registers, so the program counter is changed again and comes back to the interrupted program.

A short example demonstrates how to write interrupt routines for the CDP1802. RØ is the current PC, X = Ø.
Register 1 is initialized to point to address 0401, register 2 is loaded with free RAM area (40FF).

When the interrupt occurs, R1 is program counter and R2 is X pointer. As this routine does not know if the stack was empty, at first it is decremented. All registers needed, and only what is needed,is saved (R7 might not be necessary) and Q is set. There is a shorter version of this program.

```
EXIT : RET
ENTRY: DEC R2
       SAV
       SEQ
       BR EXIT
```

The reason for having this RET instruction in front of the rest of the program gives the advantage that after the return this routine is ready to be executed without being initialized again. The same kind of subroutine is used in SEP technique.

To toggle the interrupt enable flip flop, the RET and DIS instruction are used.

For example : IE = 0 (means, interrupts are disabled)
P = 1
X = 2

The sequence

```
SEX R1          .. Go to immediate mode (X = P)
RET             .. Do return with next byte (21)
.#21            .. Immediate byte
```

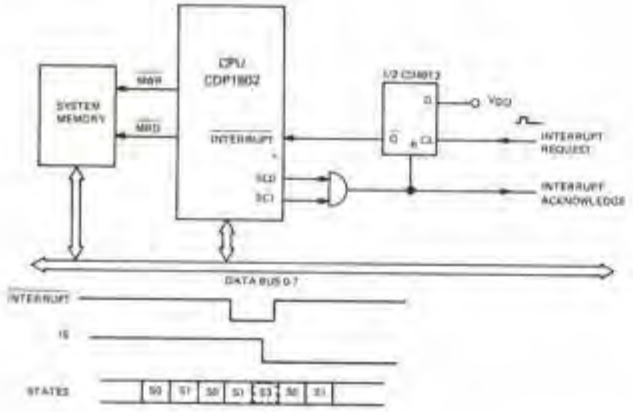
Will put the byte 21 in X and P, not changing the program execution but setting IE = 1.

To disable the IE flip-flop it would be

```
SEX R1          .. Go to immediate byte
DIS             .. Disinstruction with 21
.#21            .. Immediate byte
```

As X and P are equal, the immediate byte is used and not the byte on the stack.

Reference MPM-201 page 64, page 80.



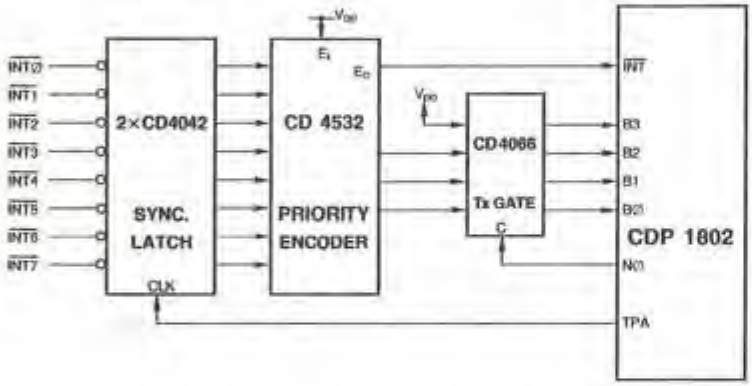

TYPICAL CIRCUIT FOR IMPLEMENTATION OF INTERRUPT OPERATION

```
0000 ?
0000 ?
0000 ?
0000 ?
0000 ?
0000 ?
0000 ?
0000 ?
0000 ?
0000 7400?
0002 F8B4B1?
0005 F8B4A1?
0008 F8400B?
000B F8FFA2?
000E 7000?
0010 ?
0010 7A?
0011 00?
0012 7B?
0013 00?
0014 3044?
0016 ?
0016 ?
0016 ?
0016 ?
0016 ?
0400 70?
0401 22?
0402 78?
0403 22?
0404 73?
0405 7A?
0406 73?
0407 8773?
0409 9773?
0408 ?
040B ?
040B 7B?
040C ?
040C ?
040C 12?
040D 42B7?
040F 42A7?
0411 42FE?
0413 42?
0414 3000?
0416 ?
0000
```

```
0001
0002          ..EXAMPLE PROGRAM
0003          ..INTERRUPT ROUTINE
0004
0005          ..RAM AT 04FF
0006
0007
0008          ..MAIN PROGRAM
0009
0010          ORG #0000
0011          DIS,#00                    ..DISABLE INTERRUPTS
0012 INIT:    LDI #04; PHI R1            ..INIT INTERRUPT
0013          LDI #01; PLO R1            ..REGISTER R1=0401
0014          LDI #40; PHI R2            ..AND AS WELL THE
0015          LDI #FF; PLO R2            ..STACKPTR FOR IT
0016          RET,#00                    ..ENABLE INTERRUPTS
0017
0018 START:   REQ                        ..RESET Q
0019          IDL                        ..WAIT FOR INTERRUPT
0020          SEQ                        ..SET Q
0021          IDL                        ..WAIT FOR INTERRUPT
0022 STOP:    BR STOP                    ..WAIT FOR RESET
0023
0024
0025          ..INTERRUPT ROUTINE
0026
0027          ORG #0400
0028 EXIT:    REF                        ..RETURN TO MAIN
0029 ENTRY:   DEC R2                     ..POINT TO FREE RAM
0030          SAV                        ..SAVE T REGISTER
0031          DEC R2                     ..DECREMENT POINTER
0032          STXD                       ..SAVE D REGISTER
0033          SHRC                       ..GET DF IN D
0034          STXD                       ..AND STORE AWAY
0035          GLO R7; STXD               ..SAVE REGISTER R7
0036          GHI R7; STXD               ..
0037          ..                         ..AND OTHERS
0038          ..                         ..IF NEEDED
0039          SEQ                        ..SET Q TO SHOW
0040          ..                         ..INTERRUPT HAPPENED
0041          ..                         ..START RESTORING
0042          INC R2                     ..INCREMENT STACKPTR
0043          LDA R2; PHI R7             ..GET R7 BACK
0044          LDA R2; PLO R7             ..
0045          LDA R2; SHR                ..AS WELL OF
0046          LDA R2                     ..AND D
0047          BR EXIT                    ..BRANCH TO EXIT
0048          END
```

## 8-LEVEL INTERRUPT VECTORING SCHEME

The more powerful I/O devices in the 1800 series, such as the 1851 PIO and the 1854A UART, have internal interrupt request latches and masking logic, which are under software control. Therefore most multi-level interrupt tasks can be implemented efficiently without the need for a sophisticated interrupt controller : a simple priority encoder will do the job.




The circuit shown here consists of a CD4532 8-level priority encoder which generates a 3-bit code representing the highest-priority interrupt input present. When any interrupt is asserted an interrupt request is passed on to the CPU. In response to the interrupt the CPU inputs the 3-bit vector via the 1857 tristate buffer. Control is then passed to one of the CPU registers 8 to F in a way analogous to the 'SEP' subroutine call. In this way each interrupting device is vectored to its own service routine, whose starting address is stored in the appropriate CPU register during initialization.

In most applications the incoming interrupts must be synchronized as shown to the TPA signal, to avoid the input of spurious codes during transitions in the 4532.

```
INTGO   :   DIS              .. switch to service routine
ENTRY   :   DEC 2 ; SAV      .. push X and P
            DEC 2 ; STXD     .. push D
            INP VECTOR       .. vector carries register number
            BR INTGO         .. prepare for next interrupt
```

The housekeeping software is minimal. During initialization, R1 must be initialized to the address "ENTRY".
In response to the interrupt the routine first pushes X, P and D into the stack, then inputs the vector code from the 4532. After branching to "INTGO" to preserve the R1 starting address control is passed to the appropriate register via the "DIS" instruction. This leaves interrupts disabled to allow the servicing routine to complete any further housekeeping needed before re-enabling further requests with a "RET".

The software inputs the word INP VECTOR and it goes automatically on the stack.

With the "DIS" instruction the contents of this stack is put in X and P and a new program counter is chosen according to the input word.

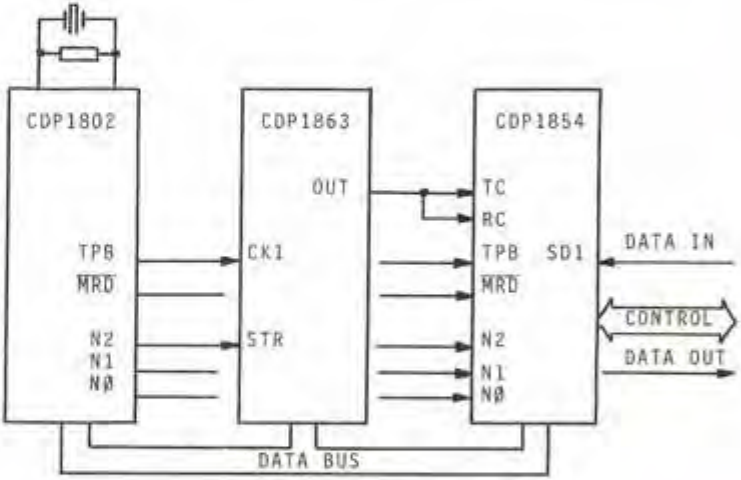
Since the four "push" instructions are necessary even in a single-level system, the overhead incurred by vectoring in this way is only three instructions, i.e. 9.6 uS with a 5MHz clock.

Reference : 1802 user manual MPM-201 B, page 64 : interrupt service.

## CDP1863 used as a baud-rate generator for the CDP1854A UART

The 1854 UART needs a clock input frequency 16 times the baud-rate required. By using the 1863 programmable frequency generator to produce this clock the baud rate may be set by software.

Suggested circuit :




I/O Port Assignments :

OUT 2 Load UART transmit register
OUT 3 Load UART control register
INP 2 Read UART receive register
INP 3 Read UART status register
OUT 4 Load 1863 count value

1863 Counter Values :

Using a standard CPU clock frequency of 2.4576MHz the following rates are available :

| 1863 Count | Baud Rate |
|---|---|
| Ø | 2400 |
| 1 | 1200 |
| 3 | 600 |
| 7 | 300 |
| 11 | 200 |
| 15 | 150 |
| 21 | 110 |
| 31 | 75 |
| 47 | 50 |

All rates are accurate except for the 110 value, which is actually 109,091, an error of 0.83%.

For calculating the code required for other rates or for other clock frequencies the following formula applies :

$$\text{Baud rate} = F_{XTAL} \times \frac{1}{8} \times \frac{1}{8} \times \frac{1}{16} \times \frac{1}{N+1}$$

Where N is the value to be loaded into the 1863.

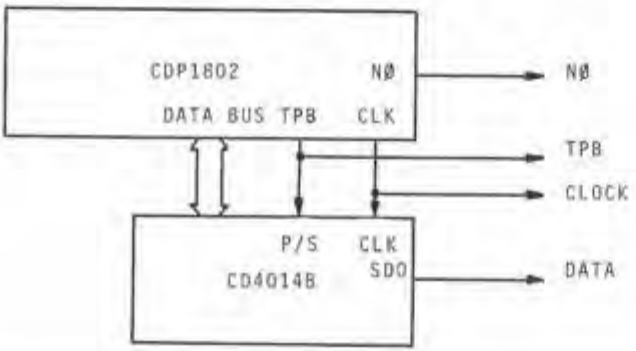
References :

CDP1854A, datasheet, file 1193
CDP1863 datasheet, file 1179
UART application note, ICAN 6632

## SYNCHRONOUS SERIAL OUTPUT FOR THE 1802

In systems having output circuitry which is physically remote from the processor itself, cost and complexity may be reduced by transmitting output data in serial form instead of via a complete parallel bus structure. Such an approach does not necessarily need extensive serial communications protocols, and may often be realized with an inexpensive shift-register.

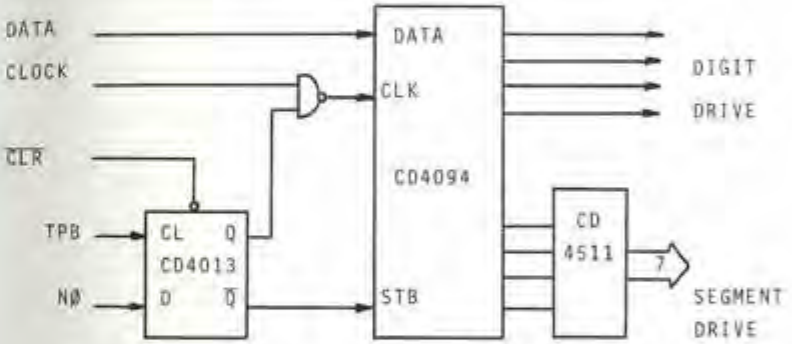
As an example, connecting a CD4014 to the bus, clock and TPB lines of the 1802 as shown results in a continuous serial representation of the databus on the output line. This makes use of the fact that each 1802 machine cycle has eight clock pulses and every eighth one is framed by the TPB pulse. Thus, the CPU clock provides the shift-clock and TPB enables synchronous parallel loading of the register each machine cycle.




This configuration can form the basis of a number of different synchronous serial output arrangements. It should be noted that worst-case dynamic characteristics of the CD4014 will restrict the maximum clock frequency to about 3.6MHz (at 10 volts).

To reduce loading on the interface signals, especially clock and TPB, a driver such as CDP1856 is normally needed to buffer these lines.

A typical application of this interface would be the implementation of a remote 4-digit multiplexed LED display. The receiving register is a CD4094 which provides the eight outputs bits needed (4 for digit-drive, and 4 for segment-drive via a CD4511 decoder/driver). A CD4013 dual flip-flop and a CD4011 UB gate complete the interface by generating a gated clock and a strobe signal for the CD4094.




Note that the 4011 and 4013 are not completely utilized and the spare halves may be used to interface a second 4094 with no additional components. Note also that the 4094 register has 3-stateable outputs and so may be used to interface to another data-bus if required.

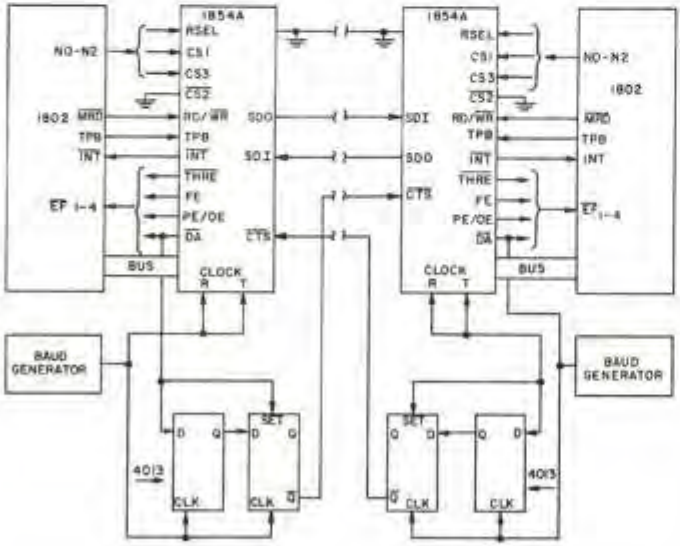
References :

CD4014B datasheet, file 1043
CD4094B datasheet, file 869

## INTERFACING TWO COSMAC SYSTEMS USING 1854A UART's

Asynchronous full duplex data communication between two independent COSMAC systems can be achieved using two 1854A UART's wired as shown. Baud rate generators must be the same frequency for both systems.

When the UART receives a serial data character it generates a data available ($\overline{DA}$) and an interrupt request signal ($\overline{INT}$). Until the CPU responds by reading the data the $\overline{DA}$ line inhibits word transmission from the other UART, and vice versa.



Internal timing of the 1854A is such that a pulse at its $\overline{CTS}$ input at the time that $\overline{DA}$ of the receiving UART is activated will cause the stop bit to be reset early. To avoid errors, the 4013 flip-flops provide a 2 clock period delay between $\overline{DA}$ (low) and $\overline{CTS}$ (high).

If the 1802 clock is used to drive the UART clocks the flip-flops can be eliminated and replaced with a simple inverter since the CPU will take many more than 2 clock periods from when it is flagged to when it services (reads) the UART.

# PROGRAMMABLE MULTIPLY AND DIVIDE UNIT

## 1. Features

. Fast multiplication and division of unsigned binary numbers.

. No additional hardware needed, uses N-lines.

. For more I/O, two level I/O or memory mapping is used. In memory, four memory locations needed.

. Multiply algorithm: shift right and add.

. Divide algorithm: shift left and subtract.

. Performs 8 x 8 multiply, 8 / 8 divide, 16 / 8 divide.

. Expandable, up to 4 circuits in parallel without additional logic for 32 x 32 multiply, 32 / 32 divide, 64 / 32 divide.

. Speed example 8 x 8 multiply

| | |
|---|---|
| Software at 3MHz | 917 µS |
| Hardware at 3MHz | 67 µS |

. Multiply time at 5V 5 µS
10V 2,5µS

## 2. Functional description (See Figure 1)

The CDP1855 consists of 3 registers to achieve the multiply and divide : X, Y, Z.

A control register (See Figure 2) is loaded to reset registers, start the multiply and divide, reset internal counters, set the number of MDU devices and select the divided clock rate. A status byte (Figure 3) indicates an overflow (for example, divide by 0).

The X, Y, Z registers are loaded with the operands and via the control word the multiply or divide is started.

At power on the device must be cleared to avoid data bus problems. This also resets the internal sequence counters and the shift pulse generator.

The sequence counters are needed for addressing if more than one device is used. The 1855 has two program inputs to be set as a device number (CN0, CN1). If registers are read or written, this number is compared with an internal counter and automatically incremented. Only the device with the same number is selected. So after a R/W the next device is addressed. With this type of selection no further address lines are needed if more than one device is used.

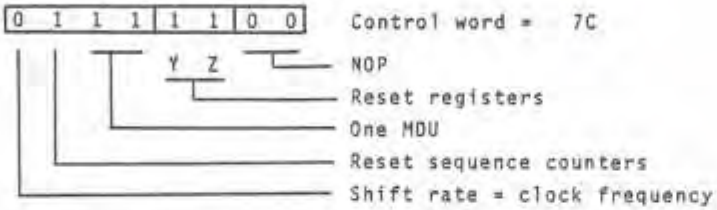
The first instruction has to load the control register with the number of MDU's to set up internal control lines.

The X, Y, Z registers in a cascaded system can be accessed randomly, because internal counters keep track,always starting a sequence with the most significant device.

Before reading results, these counters have to be reset and again they will point to the most significant byte of X, Y, Z.

3. Divide operation (See Figure 4)

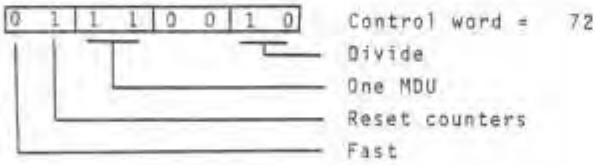
1. Reset counters
   Specify number of MDU
   Reset register
   NOP




2. Transfer dividend — 67 7C
   Load MSB in Y — 66 44
   Load LSB in Z — 65 ZZ
   For dive

3. Transfer divisor in
   Load in X — 64 XX

4. Start the division




5. Read results
   Read quotient in Z — 6D ZZ
   Read remainder in Y — 6E YY
   Read status — 6F 0X

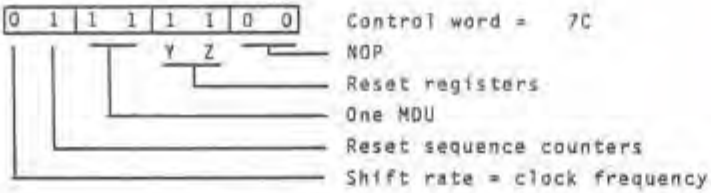
6. Now it is possible to do a further division by resetting Z register, because Y and X are not altered.

7. Example

```
  Y    Z       X       Z     Y
  0 1  6 B  : 3 C  = 0 6 . 0 3          HEX
                          REMAINDER
       363  : 6 0  = 0 6 . 0 3          DEC
```

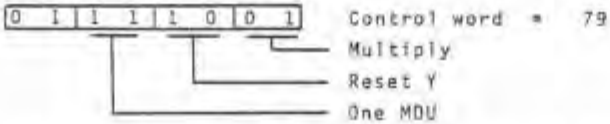
## 4. Multiply operation

1. Reset counters
   Specify number of MDU
   Reset register
   NOP




2. Input multiplicant 1 in X register 64 XX
   Input multiplicant 2 in Z register 65 ZZ

3. Initiate multiplication




   The Y register has to be reset, because its contents is automatically added to the result.

4. Read results

   Read high byte in Y 6E YY
   Read low byte in Z 6D ZZ

5. Example : (Y contains 03)

```
 X       Z       Y     Z
3 C  X  0 6  =  0 1   6 B
                ^
                |
          + 0 3 in Y
                |
                v
6 0 X 0 6 = 360 + 3 = 363
```

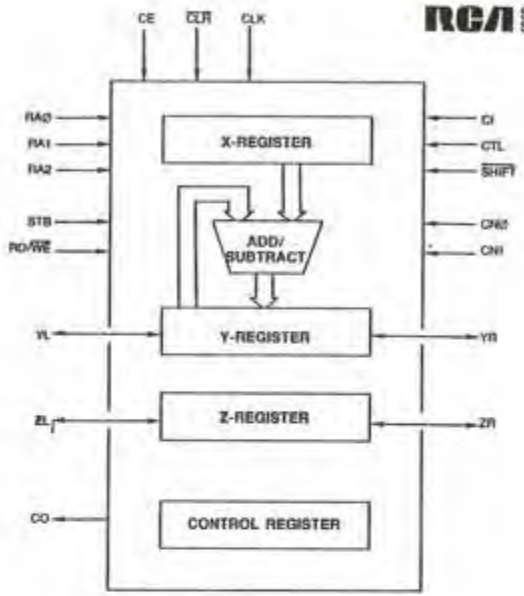
## 5. Programming example

Using 3 MDU's to multiply 20 1F 7C by 72 3C 09

```
MEMORY LOCATION  OP CODE   LINE NO.  LEVEL 1 ASSEMBLY LANGUAGE
0000             F830;     0001      LDI   #30
0002             A2;       0002      PLO   R2       ; ;LOAD 30 INTO R2.0
0003             F800;     0003      LDI   #00
0005             B2;       0004      PHI   R2       ; ;LOAD 00 INTO R2.1 (R2 = 0030
0006             6750;     0005      OUT   7,#50    ; ;LOAD CONTROL REGISTERS
                                                      SPECIFYING
0008             ;         0006                     ; ;THREE MDU's AND RESETTING
                                                      THE
0008             ;         0007                     ; ;SEQUENCE COUNTERS
0008             9420;     0008      OUT   4,#20    ; ;LOAD MSB OF X REGISTER
                                                      WITH 20
000A             941F;     0009      OUT   4,#1F    ; ;LOAD NEXT MSB OF X REG
                                                      WITH 1F
000C             947C;     0010      OUT   4,#7C    ; ;LOAD LSB OF X REGISTER
                                                      WITH 7C.
000E             ;         0011                     ; ;X REGISTER CONTAINS #201F7C
0005             9572;     0012      OUT   5,#72    ; ;LOAD MSB OF Z REGISTER
                                                      WITH 72
0010             953C;     0013      OUT   5,#3C    ; ;LOAD NEXT MSB OF Z REG
                                                      WITH 3C
0012             9509;     0014      OUT   5,#09    ; ;LOAD LSB OF Z REGISTER
                                                      WITH 09;
0014             ;         0015                     ; ;Z REGISTER CONTAINS #723C09
0014             6758;     0016      OUT   7,#58    ; ;LOAD CONTROL REGISTERS
                                                      RESETTING
0016             ;         0017                     ; ;Y REGISTERS AND SEQUENCE
                                                      COUNTERS
0016             ;         0018                     ; ;AND STARTING MULTIPLY
                                                      OPERATION
0016             E2;       0019      SEX   R2
0017             6E60;     0020      INP   6; IRX   ; ;MSB OF RESULTS IS STORED AT
0019             ;         0021                     ; ;LOCATION 0030
0019             6E60;     0022      INP   6; IRX
0018             6E60;     0023      INP   6; IRX
001D             6D60;     0024      INP   5; IRX
001F             6D60;     0025      INP   5; IRX
0021             6D;       0026      INP   5;       ; ;COMPLETE LOADING RESULT INTO
0022             ;         0027                     ; ;MEMORY LOCATIONS 0030 TO 0039
0022             ;         0028                     ; ;RESULTS = 0E558DBA285C
0022             3022;     0029 STOP BR STOP
```

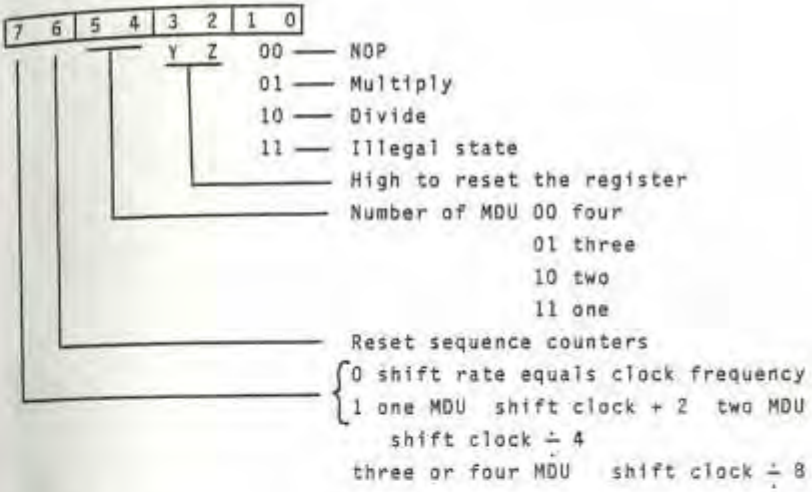
The result of $201F7C_{16} \times 723C09_{16}$ is $0E558DBA285C = 15780812797276_{10}$. It will be stored in memory as follows:

| LOC | BYTE |
|---|---|
| 0030 | 0E |
| 31 | 55 |
| 32 | 8D |
| 33 | BA |
| 34 | 28 |
| 35 | 5C |




**CDP 1855 : MULTIPLY-DIVIDE UNIT**

(Figure 1)

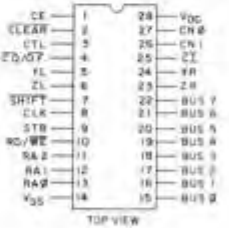



(Figure 2) : Control Register Bit Assignment

Status byte | 0 0 0 0 | 0 0 0 X |

If an overflow occurs during divide 16 : 8 bit, overflow = 1

(Figure 3) : Status Byte

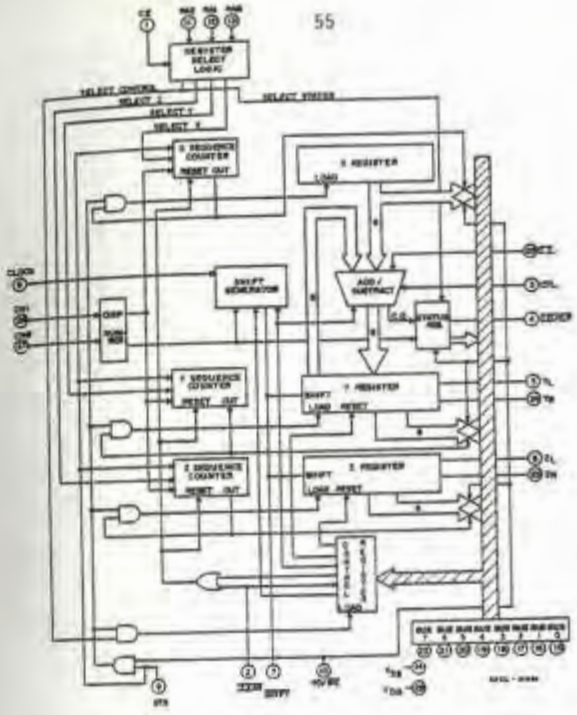



(Figure 4) : Terminal Assignments

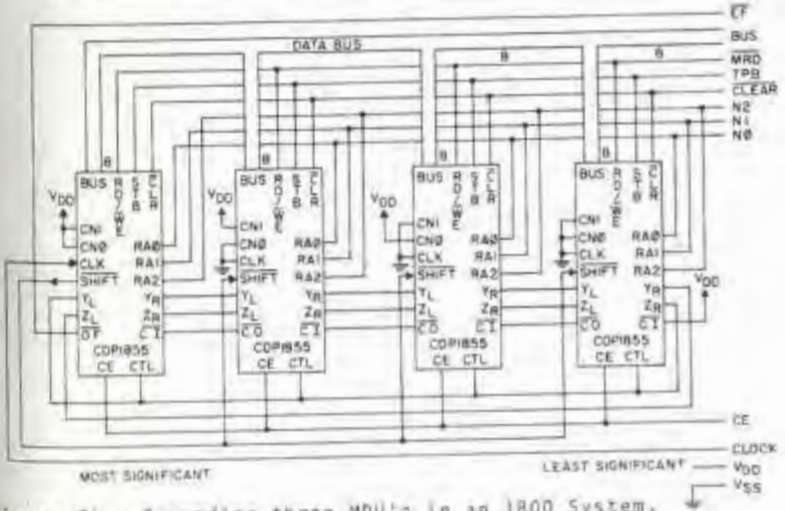
| CE | RA2 | RA1 | RAØ | R/W | STB | REGISTER | FUNCTION X | FUNCTION ÷ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | X | X | X | X | X | | NO ACTION | | | |
| X | 0 | X | X | X | X | | BUS FLOATS | | | |
| 1 | 1 | 0 | 0 | 1 | X | X | MULTIPLIKATOR | DIVISOR | 64 | TO MDU |
| 1 | 1 | 0 | 1 | 1 | X | Z | " | LSB DIVIDEND | 65 | TO MDU |
| 1 | 1 | 1 | 0 | 1 | X | Y | TO BE ADDED ! | MSB DIVIDEND | 66 | TO MDU |
| 1 | 1 | 1 | 1 | 1 | X | CONTROL | | | 67 | TO MDU |
| 1 | 1 | 0 | 0 | 0 | 1 | X | UNCHANGED | UNCHANGED | 6C | TO CPU |
| 1 | 1 | 0 | 1 | 0 | 1 | Z | LSB RESULT | QUOTIENT | 6D | TO CPU |
| 1 | 1 | 1 | 0 | 0 | 1 | Y | MSB RESULT | REMAINDER | 6E | TO CPU |
| 1 | 1 | 1 | 1 | 0 | 1 | STATUS | | 0000 000X | 6F | TO CPU |
| 1 | 1 | X | X | 0 | 0 | NO ACTION | BUS FLOATS | | | |

0 = Low level
1 = High level
X = High or Low level

(Figure 5) : Control Truth-Table



(Figure 6) : Block Diagram of CDP1855




(Figure 7) : Cascading three MDU's in an 1800 System, accessed as I/O Ports

```
0000 ;                  0001 ..
0000 ;                  0002 ..
0000 ;                  0003 ..         JP RCA 8XL
0000 ;                  0004 ..
0000 ;                  0005 ..
0000 ;                  0006 ..         DEMOPROGRAM     1855
0000 ;                  0007 ..         MULTIPLY-DIVIDE-UNIT
0000 ;                  0008 ..
0000 ;                  0009 ..
0000 ;                  0010 .. IT COMES BACK TO CDS AT 0005
0000 ;                  0011 .. TO START UT20 PRESS (CR)
0000 ;                  0012 .. TO WORK WITH UT21 CHANGE TO LBR 0029
0000 ;                  0013
0000 ;                  0014 ..
0000 ;                  0015            ORG #00
0000 F801;              0016            LDI #01
0002 BE;                0017            PHI E
0003 F801;              0018            LDI #01
0005 AE;                0019            PLO E
0006 6148;              0020            OUT 1,#40       .. SET TWO LEVEL IO
0008 DE;                0021            SEP E
0009 C00005;            0022            LBR #0005
000C ;                  0023 ....
000C ;                  0024 ....
000C ;                  0025 ..         MULTIPLY 8 X 8
000C ;                  0026 ..
000C ;                  0027 .. USES THE FOLLOWING MEMORY LOCATIONS:
000C ;                  0028 .. 400 1.MULTIPLIKANT
000C ;                  0029 .. 401 2.MULTIPLIKANT
000C ;                  0030 .. 402 RESULT HIGH BYTE
000C ;                  0031 .. 403 RESULT LOW  BYTE
000C ;                  0032 ..
000C ;                  0033            ORG #0100
0100 D0;                0034 BACK1:     SEP 0
0101 F804;              0035            LDI #04
0103 BF;                0036            PHI F
0104 F800;              0037            LDI #00
0106 AF;                0038            PLO F
0107 EE;                0039            SEX E
0108 6770;              0040            OUT 7,#70
010A EF;                0041            SEX F
010B 64;                0042            OUT 4
010C 65;                0043            OUT 5
010D EE;                0044            SEX E
010E 6779;              0045            OUT 7,#79
0110 EF;                0046            SEX F
0111 6E60;              0047            INP 6;INX
0113 6D;                0048            INP 5
0114 3000;              0049            BR BACK1
0116 ;                  0050 ..
0116 ;                  0051 ..
```

```
0116 ;
0116 ;
0116 ;
0116 ;
0116 ;
0116 ;
0116 ;
0116 ;
0116 ;
0116 ;
0200 D0;
0201 F804;
0203 BF;
0204 F800;
0206 AF;
0207 EE;
0208 677C;
020A EF;
020B 65;
020C 64;
020D EE;
020E 6772;
0210 EF;
0211 6D60;
0213 6E60;
0215 6F;
0216 3000;
0218 ;
0218 ;
0218 ;
0218 ;
0218 ;
0218 ;
0218 ;
0218 ;
0218 ;
0218 ;
0218 ;
0300 D0;
0301 F804;
0303 BF;
0304 F800;
0306 AF;
0307 EE;
0308 677C;
030A EF;
030B 66;
030C 65;
030D 64;
030E EE;
030F 6772;
0311 EF;
0312 6D60;
0314 6E60;
0316 6F;
0317 3000;
0319 ;
```

```
0052 ..        8 X 8 DIVISION
0053 ..
0054 .. USES THE FOLLOWING MEMORY LOCATIONS:
0055 .. 400 DIVIDENT
0056 .. 401 DIVISOR
0057 .. 402 RESULT
0058 .. 403 REMAINDER
0059 .. 404 OVERFLOW
0060 ..
0061          ORG #0200
0062 BACK2:   SEP 0
0063          LDI #04
0064          PHI F
0065          LDI #00
0066          PLO F
0067          SEX E
0068          OUT 7,#7C
0069          SEX F
0070          OUT 5
0071          OUT 4
0072          SEX E
0073          OUT 7,#72
0074          SEX F
0075          INP 5;IRX
0076          INP 6;IRX
0077          INP 7
0078          BR BACK2
0079 ..
0080 ..
0081 ..       16 X 8 DIVISION
0082 ..
0083 .. USES THE FOLLOWING MEMORY AREA:
0084 .. 400 DIVIDENT HIGH BYTE
0085 .. 401 DIVIDENT LOW  BYTE
0086 .. 402 DIVISOR
0087 .. 403 RESULT
0088 .. 404 REMAINDER
0089 .. 405 OVERFLOW
0090 ..
0091          ORG #0300
0092 BACK3:   SEP 0
0093          LDI #04
0094          PHI F
0095          LDI #00
0096          PLO F
0097          SEX E
0098          OUT 7,#7C
0099          SEX F
0100          OUT 6
0101          OUT 5
0102          OUT 4
0103          SEX E
0104          OUT 7,#72
0105          SEX F
0106          INP 5;IRX
0107          INP 6;IRX
0108          INP 7
0109          BR BACK3
0110
```

## USING THE MDU

COSMAC systems using the CDP1855 MDU can perform an 8 bit x 8 bit multiply in 8% of the machine time required by systems using a software multiply algorithm. Programs for both methods are explained below :

### Software method

Software implementation of an 8 x 8 multiply algorithm, based on a method of add and shift right, is listed in program 1. Normal overhead of initializing the CPU registers is done in the main program. Since 9 shifts are required to complete the 8 x 8 multiply operation, 18 instructions of the subroutine must be repeated nine times, resulting in a total of 344 machine cycles. Assuming a clock frequency of 3MHz, the time required to complete the operation is 917 uSec.

### Hardware method using the CDP1855

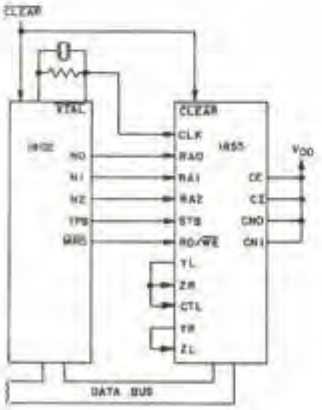



Figure 1 :

Connections Between Microprocessor and Multiply/ Divide Unit

Figure 1 shows the pin connections for using the 1855 with the 1802/1804. A subroutine for performing an 8 x 8 multiply is listed in program 2. The overhead of register initializing is about the same as in the 1st program, however, the actual 13 instruction subroutine must be executed only once since the MDU does all the shifting and adding.

Again assuming a clock frequency of 3MHz, the time required to load two 8 bit numbers, multiply, and read the 16 bit result is only 69 uSec.

Step-by-step operation of the MDU

1. Load the control register by executing an OUT 7 instruction followed by the control byte, in this case F0 (11110000). Logic highs on bits 4 and 5 define the number of cascaded MDU's as one. Bit 6 resets the internal sequence counters, and bit 7 selects the optional clock prescaler, which causes the shift frequency to be 1/2 of the clock frequency. This is required when the clock frequency is higher than 1.5MHz at 5V operation or 3MHz at 10V operation.

2. Load the X register by executing an OUT 4 instruction followed by one of the 8 bit operands.

3. Load the Z register by executing an OUT 5 instruction followed by the other 8 bit operand. The order of steps 2 and 3 can be interchanged if desired.

4. Load the control register as in step 1 but this time with F9 (11111001). Bit 3 resets register Y to zero and the code of 01 on bits 1 and 0 cause the multiply operation to begin.

5. The MDU will automatically perform the 8 x 8 multiply and store the most significant half of the 16 bit answer in register Y and the lesser significant half in the Z register. The time required to perform the actual multiply operation is 8N + 1 shifts, where N = the number of MDU's, in this case one. Therefore, it takes 9 shift cycles, and since the prescaler is being used, this translates to 18 clock pulses of the CPU.

   To provide enough time for the multiply operation, software should be arranged so that one 2-cycle instruction occurs between the multiply command and the reading of the answer.

6. Read the most significant half of the 16 bit answer from register Y by executing an INP6 instruction.

7. Read the lesser significant half of the 16 bit answer from register Z by executing an INP5 instruction. The order of steps 6 and 7 can be interchanged is desired.

PROGRAM # 1

| M ADDRESS | BYTE | ASSEMBLY PROGRAM | COMMENTS | |
|---|---|---|---|---|
| 0000 | F800 | LDI 00 | R0 = Prog. counter | Initial-ization |
| 0002 | B7 | PHI R7 | R7 = Product register | |
| 0003 | B8 | PHI R8 | R7.1 = 00 | |
| 0004 | F8FF | LDI FF | | |
| 0005 | A8 | PLO R8 | R8 = 00FF | |
| 0007 | F803 | LDI 03 | | |
| 0009 | AE | PLO RE | | |
| 000A | F802 | LDI 02 | | |
| 000C | BE | PHI RE | RE = Sub. Prog. counter | |
| 000D | DE | SEP RE | Call subroutine | |
| 000E | (data) | (data) | 8 bit Multiplicand | |
| 000F | (data) | (data) | 8 bit multiplier | |
| 0202 | D0 | SEP R0 | Return to main prog. | Sub-routine |
| 0203 | F809 | LDI 09 | | |
| 0205 | A6 | PLO R6 | R6 = Loop counter | |
| 0206 | FE | SHL | Presets DF = 0 | |
| 0207 | 40 | LDA R0 | Load M'cand | |
| 0208 | 58 | STR R8 | into 00FF | |
| 0209 | 40 | LDA R0 | Load M'lier | |
| 020A | A7 | PLO R7 | into R7.0 | |
| 020B | E8 | SEX R8 | | |
| 020C | 97 | GHI R7 | | |
| 020D | 76 | SHRC | Shift low byte | |
| 020E | B7 | PHI R7 | | |
| 020F | 87 | GLO R7 | | |
| 0210 | 76 | SHRC | Shift high byte | |
| 0211 | A7 | PLO R7 | | |
| 0212 | 3B17 | BNF 17 | Branch if DF = 0 | Loop 9 times |
| 0214 | 97 | GHI R7 | | |
| 0215 | F4 | ADD | Add M'lier to R7.1 | |
| 0216 | B7 | PHI R7 | Return sum to R7.1 | |
| 0217 | 26 | DEC R6 | Index loop count | |
| 0218 | 86 | GLO R6 | | |
| 0219 | 3202 | BZ 02 | Loop count = 97 | |
| 021A | A6 | PLO R6 | | |
| 021B | 300C | BR0 0C | Continue looping | |

PROGRAM # 2 *

| M ADDRESS | BYTE | ASSEMBLY PROGRAM | COMMENTS |
|---|---|---|---|
| 0000 | F801 | LDI 01 | Load 0101 |
| 0002 | BE | PHI E | into |
| 0003 | AE | PLO E | RE of the CPU |
| | | | |
| 0004 | DE | SEP E | Go to subroutine at 0101 |
| 0005 | | | Main program continues |
| | | | |
| 0100 | D0 | SEP 0 | Return to main program |
| 0101 | F802 | LDI 02 | Load |
| 0103 | BF | PHI F | RF of the CPU |
| 0104 | F800 | LDI 00 | with |
| 0106 | AF | PLO F | 0200 |
| 0107 | EE | SEX E | |
| 0108 | 67F0 | OUT 7, F0 | Load MDU control register |
| 010A | EF | SEX F | |
| 010B | 64 | OUT 4 | Load X register of MDU |
| 010C | 65 | OUT 5 | Load Z register of MDU |
| 010D | EE | SEX E | |
| 010E | 67 F9 | OUT 7, F9 | Load MDU control register |
| 0110 | EF | SEX F | |
| 0111 | 6E | INP 6 | Read Y register of MDU |
| 0112 | 60 | IRX | |
| 0113 | 6D | INP 5 | Read Z register of MDU |
| 0114 | 3000 | BR 00 | Branch to 0100 |
| | | | |
| 0200 | (data) | | 8 bit Multiplicand loaded in X |
| 0201 | (data) | | 8 bit Multiplier loaded in Z |
| 0202 | (data) | | Storage space for result, high byte |
| 0203 | (data) | | Storage sapce for result, low byte |

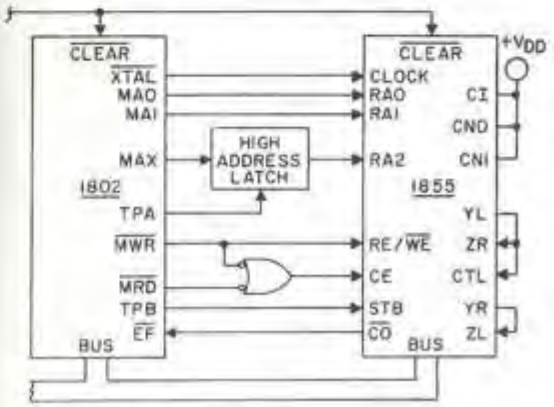
* Based on program examples submitted by J. Pintaske, RCA Brussels.

## MEMORY MAPPING THE MDU

The current CDP1855 data sheet does not currently show how to use the device in a memory mapped configuration; however, it does mention the use of $\overline{MWR}$ for RE/$\overline{WE}$, and address lines or functions of address lines for RA0, 1 and 2. This will work provided certain non-memory operations, such as PHI, GLO, etc...., do not cause spurious selection of the MDU when the contents of the internal registers appear on the address bus.

To prevent the MDU from responding to spurious addresses, and thereby avoiding bus contention, the $\overline{MWR}$ and $\overline{MRD}$ signals must be OR'd to produce a valid CE signal. The IRX instruction is a special case; it should not be used when R (X) is pointing to any of the MDU register addresses, since this instruction not only presents the contents of R (X) on the address bus, but also forces $\overline{MRD}$ low, which enables the MDU and causes inadvertent advancing of the sequence counters.

The required "OR'd" CE signal, as well as latched high order address bits, are usually available in most memory intensive systems using an 1866 or 1867 decoder, or similar hardware.



## FASTER MDU THROUGHPUT USING "DMA METHOD"

When using the MDU in the normal configuration shown in the CDP1855 data sheet, a large portion of the throughput time is consumed by loading and unloading data and control bytes. Throughput speed can be increased by 45% by using a DMA in/out technique to load and unload the MDU. Figure 1 shows the hardware needed to : sequentially address the MDU; control the timing of read/write; and issue DMA in/out signals to the CPU.

With the normal I/O mapped MDU, the CPU must execute 6 I/O instructions to facilitate a multiply/divide operation. The "DMA method" requires only one I/O instruction , leaving more unused I/O mapping space for larger systems.

### Speed comparison

Table 1 shows a comparison of multiply time (not including loading and unloading the stack), and system throughput frequency (including loading and unloading the stack) for 4 different multiply methods.

1. The "software method" using only software (no MDU)
2. The "normal hardware method" using the MDU addressed by the CPU
3. The "DMA method" using the MDU as shown in Figure 1
4. The "theoretical minimum DMA method" using the MDU; assuming multiplication by a constant number. Special hardware would avoid reloading X and Y registers. A pulse at the $\overline{\text{CLEAR}}$ pin would replace loading the last control byte (FC).

TABLE 1

| MULTIPLY METHOD | MULTIPLY TIME IN MICRO SECONDS | SYSTEM THROUGHPUT FREQUENCY |
|---|---|---|
| Software only | 917 | 1 KHz |
| Hardware MDU (normal operation) | 69 | 7.8 KHz |
| Hardware MDU using DMA technique | 29 | 11.4 KHz |
| Theoretical minimum | 21.3 | 14.4 KHz |

Values given are for system operation at 5 volts $V_{DD}$, and 3 MHz clock frequency.

Added hardware (Figure 1)

Both flip-flops are normally in the reset condition, causing no DMA action. A (0111) code is jammed into the CD4516, keeping the MDU de-selected (CE = 0).

When an output 2 instruction occurs, the $N_1$ line goes high, setting flip-flop B, which asserts a $\overline{\text{DMA-OUT}}$ signal at the CPU. Later in the output cycle, during TPB, flip-flop A is set, which removes the preset (JAM) signal from the CD4516.

The CD4516 merely counts up at each TPA pulse, sequentially addressing the MDU. Additional logic decodes the CD4516 output to produce the $\overline{\text{DMA-IN}}$ signal ($\overline{\text{DMA-IN}}$ has priority over $\overline{\text{DMA-OUT}}$). At the end of the address sequence, the carry out ($\overline{\text{CO}}$) restores the flip-flops to their original states; ending the DMA action, and de-selecting the MDU.

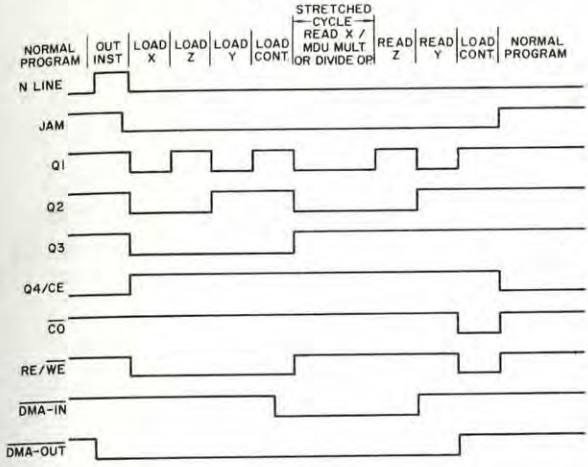
The MDU $\overline{\text{SHIFT}}$ pin is connected to the CPU $\overline{\text{WAIT}}$ pin causing a stretched CPU cycle during the multiply/divide operation, so that the MDU receives at least 18 clock pulses before the answer is read from the Z and Y registers.

Software program

The program starts by intializing register R3, designating it as the program counter, and freeing RØ for the DMA pointer. RX is also set to RØ so that the stack can be loaded/unloaded using INP/OUT instructions.

The program enters the multiply routine, and loops continuously performing successive multiply operations. The looping scheme was chosen to demonstrate the throughput rate of the system (Figure 2) in terms of multiplies/sec.

Although not shown, the software can be modified to facilitate the use of branch and interrupt techniques to get in and out of the multiply loop. With the addition of a SEP instruction, the multiply operation can become a one-shot subroutine.




TIMING DIAGRAM

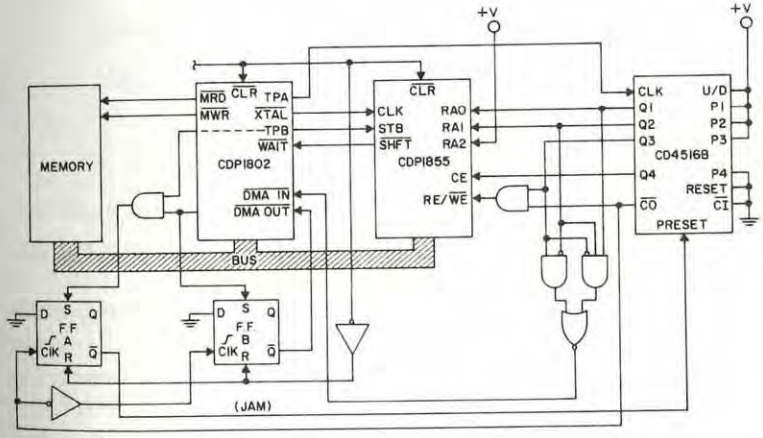



Figure 1

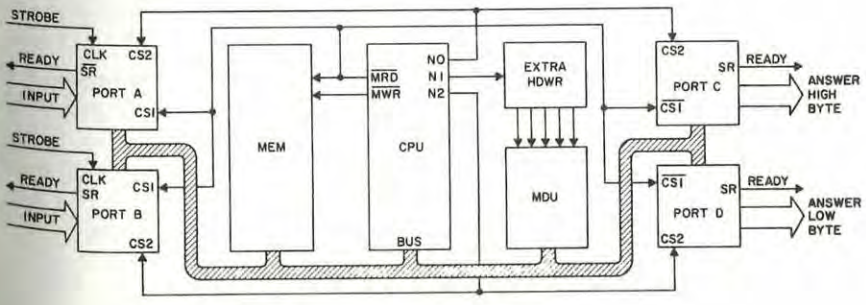



Figure 2

PROGRAM FOR MULTIPLY USING "DMA METHOD"

| M ADDRESS | BYTE | ASSEMBLY PROGRAM | COMMENTS |
|---|---|---|---|
| 0000 | F800 | LDI 00 | R0 = Prog. Counter |
| 0002 | B3 | PHI R3 | |
| 0003 | F807 | LDI 07 | |
| 0005 | A3 | PLO R3 | R3 = 0007 |
| 0006 | D3 | SEP R3 | |
| 0007 | E3 | SEX R0 | R0 = RX |
| 0008 | F807 | LDI 07 | |
| 000A | A0 | PLO R0 | |
| 000B | F802 | LDI 02 | |
| 000D | B0 | PHI R0 | |
| 000E | F8FC | LDI FC | Put FC in Stack |
| 0010 | 50 | STR R0 | at 0207 |
| 0011 | F803 | LDI 03 | |
| 0013 | A0 | PLO R0 | |
| 0014 | F8F1 | LDI F1 | Put F1 in Stack |
| 0016 | 73 | STXD | at 0203 |
| 0017 | F800 | LDI 00 | Put 00 in Stack |
| 0019 | 73 | STXD | at 0202 |
| 001A | F801 | LDI 01 | |
| 001C | A0 | PLO R0 | |
| 001D | 69 | INP 9 | Port A data to Stack 0201 |
| 001E | 20 | DEC R0 | |
| 001F | 6A | INP C | Port B data to Stack 0200 |
| 0020 | 20 | DEC R0 | |
| 0021 | 62 | OUT 2 # | Initiate DMA and multiply |
| 0022 | F805 | LDI 05 | action |
| 0024 | A0 | PLO R0 | |
| 0025 | 61 | OUT 1 | STACK 0205 to Port C |
| 0026 | 64 | OUT 4 | Stack 0206 to Port D |
| 0027 | 301A | BR routine | |

| | | |
|---|---|---|
| 01FF | | Dummy byte |
| 0200 | | Load X register |
| 0201 | | Load Z register |
| 0202 | STACK | Load Y register |
| 0203 | | Load control register |
| 0204 | | Read X register |
| 0205 | | Read Z register |
| 0206 | | Read Y register |
| 0207 | | Load control register |

\# Only one instruction required to activate a complete MDU operation

## DIGITAL FILTER USING THE CDP1855

Figure 1 shows a simple low-pass filter composed of a resistor and a capacitor. The output voltage can be described by the differential equation :

1. $$\frac{dVO}{dt} + \frac{VO(t)}{RC} = \frac{VI(t)}{RC}$$

By using Euler's method of sampling integration, equation 1 is transformed to :

2. $V_{O(n)} = kV_{I(n)} + m\ V_{O(n-1)}$; where n is any specific sample period,

$$k = \frac{1}{1 + RC/T}, \text{ and } m = \frac{1}{1 + T/RC}$$

T is the sampling time interval in seconds, R is in ohms, and C is in farads.

Equation 2 can be implemented with a recursive digital filter that uses multiply, add, and delay functions as shown in Figure 2. Figure 3 shows two multiply operations of the CDP1855. The second operation adds the result of the first operation to its product. The feedback delay is obtained by a read-in and then read-out of a scratch pad memory cell.

The program listing shows a routine used to do real time recursive digital filtering, where the incoming analog signal is digitized by an A/D converter, processed by the digital filter, and reconstructed by a D/A converter (Figure 4).

The beginning of the program initializes CPU registers, and loads constant data into the stack area. The program then moves into a continuous loop of data input, data processing (MDU action), and data output.

The sample rate (T) is the time it takes the program to execute the routine loop of 15 instructions.

If the CPU clock is 3.2 MHz, the time required to complete one loop is 75 uSec, or a sampling frequency of about 13 KHz. Based on the Nyquist criterion of at least two samples per cycle, the input signal frequency should be limited to 6.5 KHz.

Example

Assume that Figure 1 has R = 30K Ohms, and C = .1 uF, and that T = 75 uSec;

then $k = \frac{1}{1 + 300/75} = .2 = .00110011$ Binary = 33 hex,

and $m = \frac{1}{1 + 75/300} = .8 = .11001100$ Binary = CC hex.

Assume now that the input signal goes from 0 to a full scale voltage of 1 volt, and remains at 1 volt for 10 sample periods. Using equation 2 the corresponding output voltages are listed below, and if plotted would produce a response curve similar to that of the analog RC circuit.

| Sample time n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Input voltage $V_I$ | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Output voltage $V_0$ | 0 | .20 | .36 | .49 | .59 | .67 | .74 | .79 | .83 | .86 | .90 |

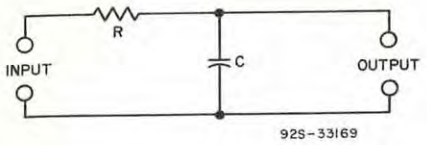



Figure 1

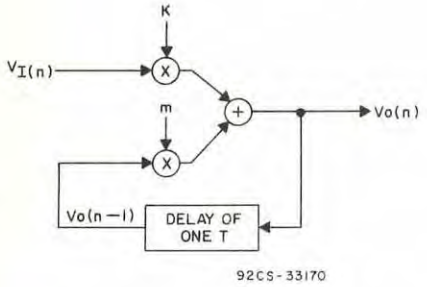



Figure 2

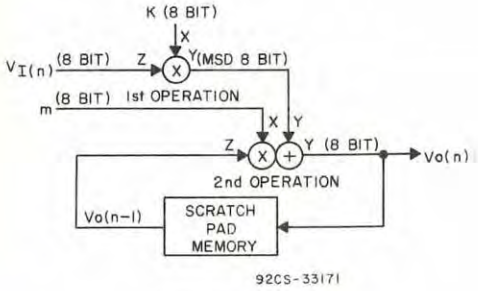



Figure 3

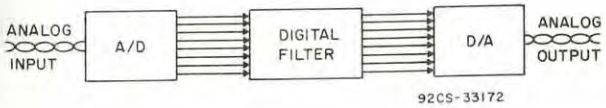



Figure 4

| M. Address | Byte | Assembly Program | Comments |
|---|---|---|---|
| 0 0 0 0 | F8 04 | LDI 04 | RP = R0 |
| 0 0 0 2 | AB | PLO RB | |
| 0 0 0 3 | F8 05 | LDI 05 | |
| 0 0 0 5 | AA | PLO RA | |
| 0 0 0 6 | F8 01 | LDI 01 | |
| 0 0 0 8 | BA | PHI RA | RA = 0105 |
| 0 0 0 9 | BB | PHI RB | RB = 0104 |
| 0 0 0 A | EA | SEX A | RX = RA |
| 0 0 0 B | F8 FD | LDI FD | Load data |
| 0 0 0 D | 73 | STXD | in Stack 0105 |
| 0 0 0 E | F8 00 | LDI 00 | Initialize storage space |
| 0 0 1 0 | 73 | STXD | data to zero (0104) |
| 0 0 1 1 | F8 CC | LDI CC | Load data |
| 0 0 1 3 | 73 | STXD | in stack 0103 |
| 0 0 1 4 | F8 F9 | LDI F9 | Load data |
| 0 0 1 6 | 73 | STXD | in stack 0102 |
| 0 0 1 7 | 73 | STXD | Decrement RX again |
| 0 0 1 8 | F8 33 | LDI 33 | Load data |
| 0 0 1 A | 5A | STR A | in stack 0100 |
| 0 0 1 B Routine | 64 | OUT 4 | Load X register |
| 0 0 1 C | 69 | INP 9 | READ $V_I$ from A/D |
| 0 0 1 D | 66 | OUT 6 | Load Z register |
| 0 0 1 E | 67 | OUT 7 | Load control register |
| 0 0 1 F | 7A | REQ | Dummy instruction during multiply |
| 0 0 2 0 | 64 | OUT 4 | Load X register |
| 0 0 2 1 | 66 | OUT 6 | Load Z register |
| 0 0 2 2 | 67 | OUT 7 | Load control register |
| 0 0 2 3 | 7A | REQ | Dummy instruction during multiply |
| 0 0 2 4 | 6A | INP A | READ Y register |
| 0 0 2 5 | 5B | STR B | STORE in stack at 0104 |
| 0 0 2 6 | 61 | OUT 1 | Load D/A with same data |
| 0 0 2 7 | F8 00 | LDI 00 | |
| 0 0 2 9 | AA | PLO RA | Set RX back to 0100 |
| 0 0 2 A | 30 1B | BR Routine | Branch to "Routine" |

| | | | |
|---|---|---|---|
| 0 1 0 0 | 33 | | K constant value |
| 0 1 0 1 | (data) | | $V_I$ data from A/D |
| 0 1 0 2 | F9 | STACK | Control byte, reset Y and multiply |
| 0 1 0 3 | CC | AREA | m constant value |
| 0 1 0 4 | (data) | | Storage space for result $V_O$ |
| 0 1 0 5 | FD | | Control byte, multiply and add Y |

# MULTIPLE INTERRUPT WITH CDP1851

## Introduction

The flexibility of CDP1802 register designation allows easy implementation of a vectored interrupt system.

By using software and the unique register designation capabilities of the CDP1802 interrupt,service routines can be positioned at random in memory.

In the CDP1802 the interrupt mechanism consists of X, P designators being saved in a temporary register T,these designators being subsequently loaded by 2, 1 respectively. Interrupts are automatically disabled.

In an interrupt vectored system the interrupt routine pointed to by R1 consists of software, which stacks the T-register and other registers which have to be preserved, and a software vectored system.

A software vectoring system is one which computes, with I/O interrupt status information, the address of the interrupt routine.

This status information in a minimal software system is obtained from an input port such as the CDP1851.

In the interrupt system implemented by CDP1851 shown, several unique features of the device have been used.

The interrupt system allows for multiple interrupt requests by use of the CDP1851 operating in a bit programmable port mode, port A, B set as inputs. An interrupt masking system can be implemented by using the CDP1851 interrupt logical masking feature. Interrupts are generated on the logical condition of port input lines.

By use of a SEP instruction to the interrupt P.C. the service routine is executed. During the service routine the interrupt on the device can be reset and interrupts allowed if necessary. On occurence of a second interrupt the interrupt entry would be processed again, the previous contents of R(N) stacked and T-register and the new value, from the interrupt status, loaded. Execution of the second interrupt service routine terminates by setting P to R1 and returning to the interrupt entry exit routine.

In the interrupt exit routine, the previous interrupt is restored and P set to this value by the use of a return instruction. A schematic of the process for two levels is shown in Figure 3.

The interrupt can be enabled and disabled in the priority system both in the CDP1802 and CDP1851, the latter by writing a control word.

Interrupts can be reset by one of two ways; firstly by clearing the source of the interrupt by, for example, a decode of the CDP1802 state code lines SCØ, SC1; alternatively, by changing the interrupt mask of the PIO (CDP1851) and resetting the interrupt source by outputting a control word to the peripheral.

This system, by using only one port of the PIO, leaves the other port available for I/O interfacing.

System expansion

The number of interrupt levels may be increased by using the other port of the PIO in conjunction with a CD4532 in a similar way to the 8 level priority system.

As the CDP1851 interrupt lines $\overline{\text{AINT}}$, $\overline{\text{BINT}}$ are open drain NMOS, a WIRED OR configuration is possible making such a system feasible.

The logical condition can be set for any of AND, OR, NOR, NAND. With the system shown the encoded output of the line encoders,together with interrupt mask setting capability of the CDP1851,can implement a level system with an 'OR' condition or a selective system with 'AND' condition.

Interrupt level can be obtained by an input read from the port and software in the interrupt service routine,loading the appropriate register with the address of the routine to be serviced; defining the other bits in the port as output and setting these as zero,allows the bits to be read as zero during an input.

An eight level priority system

A system with the capability of handling up to eight levels of interrupts is shown in Figure 1. It consists of a CDP1851 and a CD4532, eight bit priority encoder, the encoded output of which is connected to D3 to D7 of one of the CDP1851 ports.

The system allows each interrupt service routine up to 32 bytes which can be expanded by use of branch instructions.

The software required to implement the system is shown in Figure 2A, 2B for 1802 and 1804 CPUs. One additional register is used to permit nesting of interrupts.

The interrupt response time at 6.4 MHz is 15 uSec .

The software consists of an interrupt ENTRY and EXIT routine, to generate interrupt vectors and return from subroutines, and interrupt service subroutines.

The interrupt program counter high order bits are loaded in initialisation, therefore giving the capability of positioning these routines anywhere in memory. Entry to a service routine is done by inputting interrupt status to the interrupt program counter low order after firstly stacking it.

The relative priorities of the interrupts are maintained by software. The source of the interrupt i.e. $\overline{\text{AINT}}$, $\overline{\text{BINT}}$ being obtained by polling the status word of the PIO.

This system allows for 16 interrupts. A system which cascades CD4532's can be implemented but more circuit elements are required than the before mentioned 16 level system.

Alternative systems

For small interrupt systems by altering the priority input to bits DØ1 to D3 and loading this information into the P designator a faster interrupt response, of 15 uSecs at 2 MHz can be obtained (Figure 1).

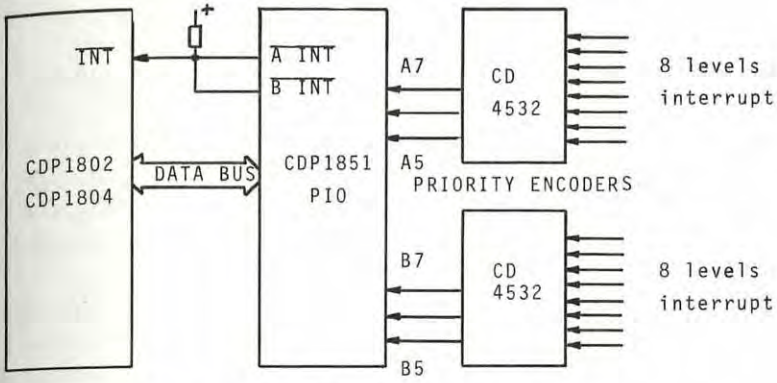


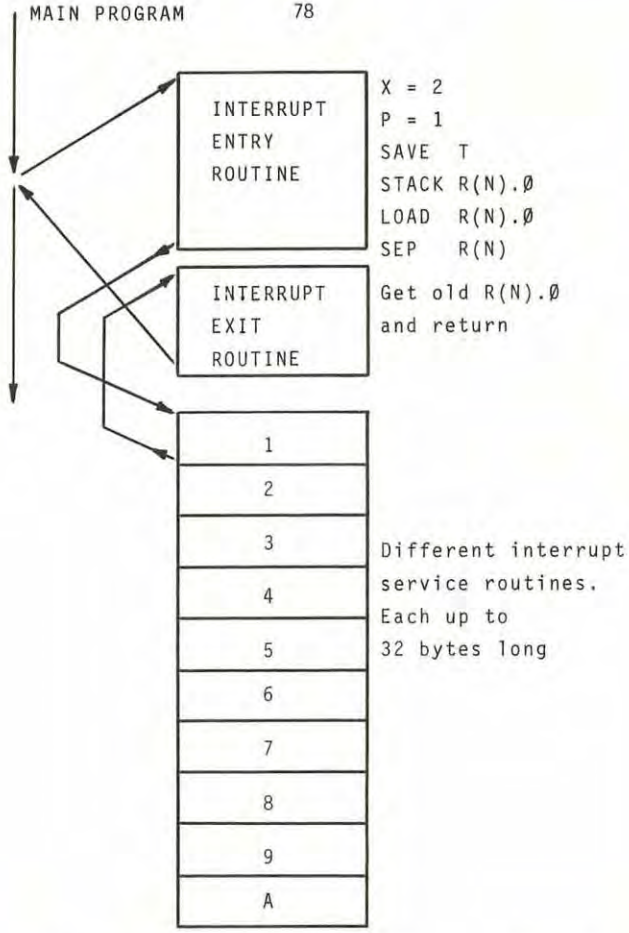

Figure 1 : Multiple Interrupt Masking System

Figure 2a : Interrupt Entry (1802 and 1804 MNEMONICS)

```
1802                        1804
                            BXL EXTINT      .. Test if ext. - int.
                          ..
                          ..
                          ..
DEC R2              EXTINT: DEC R2          .. SAVE
SAV                         SAV             .. X,P on stack
GLO RN                      RSXD            .. Stack PC
STXD                        ..
INP INTST                   INP INTINST     .. Get interrupt states
PLO RN                      PLO RN          .. Load INT PC
SEP RN                      SEP RN          .. And execute program
```

Figure 2b : Interrupt Exit (1802 and 1804 MNEMONICS)

```
LDXA                        RLXA            .. Destack interrupt
PLO RN                      ..
RET                         RET             .. Return to
                                            .. Interrupt routine
```




INTERRUPT NESTING FOR 1 LEVEL OF INTERRUPT

| | |
|---|---|
| 4FF4 | $X_M, P_M$ |
| 4FF3 | $R_1(N).Ø$ |
| 4FF2 | 2, 1 |
| 4FF1 | $R_2(N).Ø$ |
| 4FF0 | 2, 1 |
| 4FEF | $R_3(N).Ø$ |

STACK CONTENTS FOR 3 LEVELS OF INTERRUPT
(Interrupted interrupt service routines)

# VIS - A COMPLETE VIDEO INTERFACE SYSTEM

## 1. INTRODUCTION

RCA's 1800 series is a fully CMOS Microprocessor family consisting of CPUs, RAMs, ROMs, EPROMs, PIO, UART, MDU,interface chips and low resolution video controllers.

As the need for displaying data increases, RCA adds to this line a high resolution Video Interface System (VIS). This system consists of CDP1869 and CDP1870, two CMOS LSI devices to generate all the necessary signals needed for visualisation on the screen; in addition a sound generator and white noise are included with volume control on the chip.

This set of devices interfaces directly with minimum additional hardware to the CPU CDP1802 but is designed as a completely independant I/O. No refresh signals are needed that might stop the processor for synchronisation during display time. "Predisplay" signals the CPU this,one display line before the refresh starts and no further access to VIS memories is possible until the display of a frame has been finished. Then again internal multiplexers allow access to the VIS memories in order to change character memory - dot and colour information or page memory - character allocation on the screen. During display time the processor is completely free for other I/O or computing.

## 2. CAPABILITIES

In short the capabilities of VIS are :

- Programmable resolution

  40 characters and 24 rows
  20 characters and 12 rows

- Character size NTSC 6 dots wide by 8 dots high
  PAL 6 dots wide by 9 dots high

- Up to 256 different characters are possible

- Character definition in

  RAM, ROM, or even mixed

- Up to 8 character colours

- Up to 8 background colours

- Hardware scrolling

- Sound generation of 8 octaves

  128 frequencies for each octave

- Noise generation - 8 ranges

- Volume control for sound and noise -

  16 steps each

- Chroma Luma generation on chip

- Clock signal for CPU provided

All features are under software control.

## 3. MEMORY SYSTEMS for Video Interface Systems

Two systems are possible, if a video system is designed :

a) Full bit map approach
b) Character display

In a full bit map approach each memory bit represents a pixel or dot on the screen. This type is ideal to display curves and pictures but it needs a large amount of memory. For example a display of 40 characters per line and 24 rows with a character size of 6 x 8 in bit map would be represented by

6 bit per character line x
8 character lines x
40 x 24 = 960 x 6 x 8 = 48K bit built by a memory of 8K and 6 bit wide.

A second choice is to divide the screen into a surface of characters. For example a definition of 24 rows x 40 characters would give the same resolution. These character positions on the screen would be addressed by a page memory, so it would be 960 bytes long to define 40 x 24 character positions.

In the character display system it has to be defined how many different characters on the screen should be possible. With a maximum of 256 characters this is represented by 8 bits.

The page memory was 960 locations long. In this memory it is defined, which of these 256 characters is where. It means that it has to be 8 bit wide.

Each character has a size of 6 x 8 bits, which means 256 characters need

256 x 6 x 8 = 12K bits or 2K byte

So for page memory 1K is needed, for the character memory a further 2K.

Memory comparison :

| Bit map | Character map |
|---|---|
| 8K byte | 3K byte |

This comparison looks even better for a system that needs less than 256 characters, for 128 characters it would change from 3K to 2K.

Another reason to look at character mapping is for example in terminal applications. To have a character on the screer it is only necessary to store one byte in page memory; for bit map, we would have to write all the bits again and again for any character. Even in games there are fixed characters for objects, which can be represented by a set of fixed characters and the motion capability of the character approach is much faster.

## 4. DESIGN GOALS

RCA's system was developed after examination of both methods. The following needs lead automatically to the design approac chosen.

4.1. Full CMOS

The whole system has to be in CMOS in order to offer the traditional CMOS advantages : in particular the possibility of battery operation, which offers low power and wide power supply range.

4.2. Minimum component count

The system has to be optimized for package count for the complete system, not only for the video signal generation.

4.3. Minimum memory

As discussed before, the bit map approach needs a large amount of memory. To minimize this, the character map method has to be chosen, but if possible at least parts of the bit map advantages should be included.

4.4. Completely independant I/O

The CPU has only to be involved in changing the memories or updating internal registers of VIS and should not be stopped during display time (as is necessary with a conventional DMA approach). This allows full CPU processing time for other tasks.

4.5. Maximum resolution via RF

Since the system has to be used for general applications and should not be restricted to high resolution displays a maximum has to be found for best video bandwidth usage.

4.6. Programmable resolution

A high resolution is needed to display large amounts of data. Educational applications use a lower resolution because the distance between screen and observer is greater.

4.7. Graphics and motion capability

Many applications will need only the display of a fixed character set, but the possibility of smooth motion on the screen has also to be included for applications requiring an "animated" display. Even semigraphic pictures should be allowed.

4.8. External synchronisation

Applications in TV need the possibility of overlaying the video signal of the television and VIS in order to display, for example, time and channel or received text information.

4.9. Colour

The display of colour or at least of different shades of grey is a must in many applications. The system has to be applicable to both colour systems PAL and NTSC; in addition the direct colour information in RGB should be available.

### 4.10. Audio generation

Many applications need a programmable sound. This is particularly true for the games area, but also in the industrial area it is useful to have special sounds for different alarms; even for a terminal several tones could allow audible prompts.

## 5. DESCRIPTION OF THE CIRCUITS

In order to include most of the external hardware (multiplexers, colour generation...) it was necessary to choose a 2 chip solution,where the CDP1869 solves the address manipulation and the CDP1870 the video signal generation (an alternative version of the CDP 1870, the CDP1876, provides RGB logic outputs in place of chrominance and luminance).

### 5.1. CDP1869 (see figure 1) address generation and audio

The CDP1802 MICROPROCESSOR has an eight bit multiplexed address bus and generates two timing pulses TPA and TPB, to define the presence of the high and low byte respectively. Therefore an 8 bit latch (1) is needed on chip to keep the high byte and implement an internal 16 bit address bus.

The control circuit (2) recognizes special instructions to set resolution, sound and noise, change memory addressing, character scanning, provide control signals for memories and data bus buffer.

The selection of these instructions is done via the three I/O selections of the CDP1802; it also generates a decoded select signal for the CDP1870.

The address multiplexer (3) decides which information goes to the address lines of the page memory. During display time it selects the character positions on the screen (PMA 0...9) and scans the character lines (CMA 0...2). During non-display time the CPU memory bus is directly connected to the page memory and it appears as an extension on the CPU memory fixed at F800 to FFFF. As only 960 bytes of this memory are used (max 40 x 24) in a 1K system 64 bytes are free for stack or other purposes.

The address counter (4), either defines the positions on the screen,or is used for character RAM loading during non-display time.

The home address register (5) defines the leftmost character of the first line on the screen. If it is set to 0000 the display shows an unshifted image of the page memory contents. Loaded with multiples of 40 it allows line by line scrolling. In the low resolution (20 x 12) it would allow the display of 4 different pages on the screen.

The sound and noise generator (6) consists of 4 different parts :

a) Sound generator

It is designed as a three bit prescaler and a seven bit down counter that are loaded via a command byte from the output signal and are automatically loaded with the same byte again. The resolution is 3 octaves and 128 different frequencies within each octave.

b) Envelope generator for sound

A 4 bit R - 2R - ladder network with a latch defines the amplitude of the sound signal.

c) Noise generator

Eight ranges of white noise are provided. The result is an explosion type sound effect useful in TV game systems.

d) Envelope generator for noise

A 4 bit R - 2R - ladder network with a latch defines the amplitude of the noise signal. Sound and noise signals can be mixed as needed.

5.2. CDP1870 (see figure 2) video signal generation

The control section (1) provides signals for the data bus multiplexer to allow character memory access from the CPU, latch signals for the control instruction, and special signals to synchronize CDP1869 and CDP1870.

One input is used to switch the internal logic from PAL standard to NTSC standard.

The data bus multiplexer (2) is needed for character memory access, if it is in RAM. It has an 8 bit wide input. Six bits (CDB 0...5) are used for character (dot) information (a character line is 6 dots long). The next two bits (CCB0, CCB1) are used for character colour bit information.

With these,4 colours out of 8 are defined per character line. Another input line (PCB) is the page colour bit and expands the colour capabilities to 8 colours and is normally connected to the page memory (see figure 3).

If character memory is in RAM, it has to be initialized after power on. With this multiplexer the character memory is switched to the CPU during non-display time.

The dot oscillator (3) provides the clock for the timing generator (4). Here nearly all the signals for synchronisation are generated. A special signal (Predisplay) tells the CPU that one display line later the multiplexers are switched and a refresh cycle starts. Connected to the interrupt signal it can decide between updating the VIS and working on other parts of the program. In this way the CPU does not waste time in waiting for the end of the display refresh.

Some other signals are needed to synchronize the two chips, for example, increment page memory counter. The dot frequency is about 5.6 MHz. This signal is devided by two and may be used as clock frequency for the CPU.

Vertical and horizontal timing appears on COMPSYNC.

The parallel to serial shift register (5) latches one line of a character and shifts it out to the luminance and chrominance logic (6) where the dot information is combined with the colour information of the character.

COMPSYNC, LUMINANCE and CHROMINANCE are combined outside the chip and give a complete video signal.

There are two bond options of the CDP1870. For a terminal or built in TV game it would be useless to have the complete colour video signal and seperate it afterwards. In this case the CDP1876 should be used, colour luminance and chrominance are replaced by RED GREEN and BLUE.

## 6. EXAMPLE OF A CRT TERMINAL

A complete system which could be a game or a terminal is shown in figure 3. It uses a CDP1804 (1) as processor, a one chip device with RAM and ROM on it,or is replaced by a CDP1802 with seperate RAM and ROM. The CDP1869 creates sound and noise through the amplifier (3) and addresses the page memory (4) with PMA0...9 (character position on the screen), scans the lines of the character memory with CMA0...2, and generates some signals for buffer or RAM selection; also it gets signals for synchronization from CDP1870 (7).

The page memory (4) is loaded and read through the buffer/ separator (5).

The character RAM (6) feeds character data bits (CDB) and colour bits (CCB) to the CDP1870 where the colour and sync signals are generated.

One crystal is needed for dot generation (DOT) and provides the clock for the CPU. Predisplay signals if VIS refreshes the screen or CPU can talk to VIS memories.

The second crystal is used for the colour burst.

Chrominance, luminance and synch pulses are combined and this signal is connected to a monitor or a modulator.

Another device, the CDP1871,a keyboard encoder,scans the key contacts and is used as input.

## 7. INSTRUCTIONS TO CONTROL CDP1869 AND CDP1870

The CDP1802 CPU has a special feature to output information. An internal register is specified and loaded with the address of the byte to be sent. This feature is used to programme the VIS devices.

For the CDP1869, which is only connected to the address bus the address itself is the information - it does not use the data.

For the CDP1870, the data at the memory location specified by this register is the information.

One instruction (OUT 3) controls the CDP1870 (8 bits) for

- Resolution (20 or 40 characters)
- Character colour control
- Character format control
- Display on or off
- Background colour definition

Three instructions (OUT 4...7) control the CDP1869

OUT 4 sets

- Octave
- Frequency within the octave
- Sound amplitude
- Sound on and off

OUT 5 programs

- o White noise range
- o White noise amplitude
- o Resolution (12 or 24 lines)
- o Page length (960 or 1920 characters)
- o Resolution (6 x 8, 6 x 16 in NTSC, 6 x 9 in PAL)
- o NTSC or PAL system

OUT 6 selects

- o A fixed address for page or character memory access

OUT 7 enables

- o Set home address register for scrolling

## 8. SOFTWARE SUPPORT

The most time-consuming after-design task is normally to set up all the necessary software modules to manipulate the data on the screen.

For this reason RCA provides a VIS INTERPRETER. It consists of a 3K program with 86 different subroutines that can be called via a one byte instruction. It includes memory manipulation as well as colour definition, setting of sound and noise plus all other control instructions. In addition the interpreter permits interruption for the execution of machine code subroutines.

## 9. HARDWARE SUPPORT

RCA produces a line of "MICROBOARDS" - board - level micro-computer products.

It consists of CPU boards with RAM and ROM, memory boards and I/O boards.

A new board in this range will be a VIS BOARD with the complete I/O system. Together with a CPU board and an ASCII keyboard it represents a complete video system with output for a colour monitor.

## 10. SUMMARY

The main advantage of VIS is indicated by the name "SYSTEM". It was not designed to be a new interface device but rather to be a complete interface system with a maximum of flexability. As such, it makes minimal demands on the CPU, leaving it available for the execution of other routines.

Once initialized it generates the display completely independant of the CPU. Together with the CPU it gives a small package count together with a very low power consumption and high temperature range completely in CMOS.

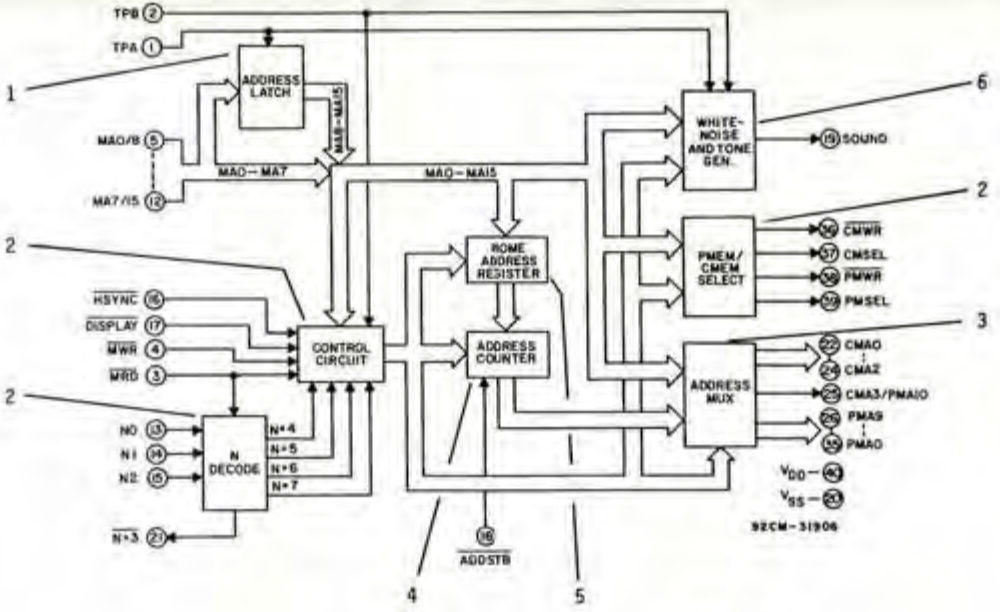

TPB 2
TPA 1
ADDRESS LATCH
MA8-MA15
MA0/8 5
MA7/15 12
MA0-MA7
MA0-MA15
WHITE-NOISE AND TONE GEN.
19 SOUND
PMEM/ CMEM SELECT
56 CMWR
57 CMSEL
58 PMWR
59 PMSEL
ROME ADDRESS REGISTER
ADDRESS COUNTER
CONTROL CIRCUIT
HSYNC 16
DISPLAY 17
MWR 4
MRD 3
N0 13
N1 14
N2 15
N DECODE
N+4
N+5
N+6
N+7
N+3 21
ADDRESS MUX
22 CMA0
24 CMA2
23 CMA3/PMA10
26 PMA9
25 PMA0
$V_{DD}$ 40
$V_{SS}$ 20
18 ADDSTB
92CM-31906
1
2
2
2
3
4
5
6

FIGURE 1

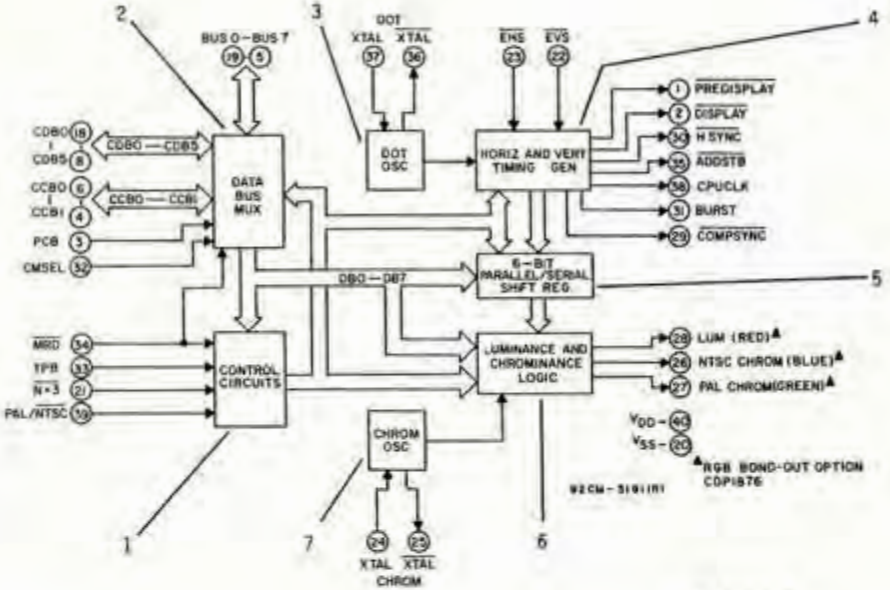

2
3
4
5
1
7
6
BUS 0 – BUS 7
DOT XTAL
XTAL
EHS
EVS
CDB0 – CDB5
CCB0 – CCB1
PCB
CMSEL
DATA BUS MUX
DOT OSC
HORIZ AND VERT TIMING GEN
PREDISPLAY
DISPLAY
H SYNC
ADDSTB
CPUCLK
BURST
COMPSYNC
DB0 – DB7
6-BIT PARALLEL/SERIAL SHIFT REG
MRD
TPB
N÷3
P&L/NTSC
CONTROL CIRCUITS
LUMINANCE AND CHROMINANCE LOGIC
LUM (RED)▲
NTSC CHROM (BLUE)▲
PAL CHROM(GREEN)▲
VDD
VSS
▲RGB BOND-OUT OPTION CDP1876
CHROM OSC
XTAL
CHROM
92CM-5191(R)

FIGURE 2

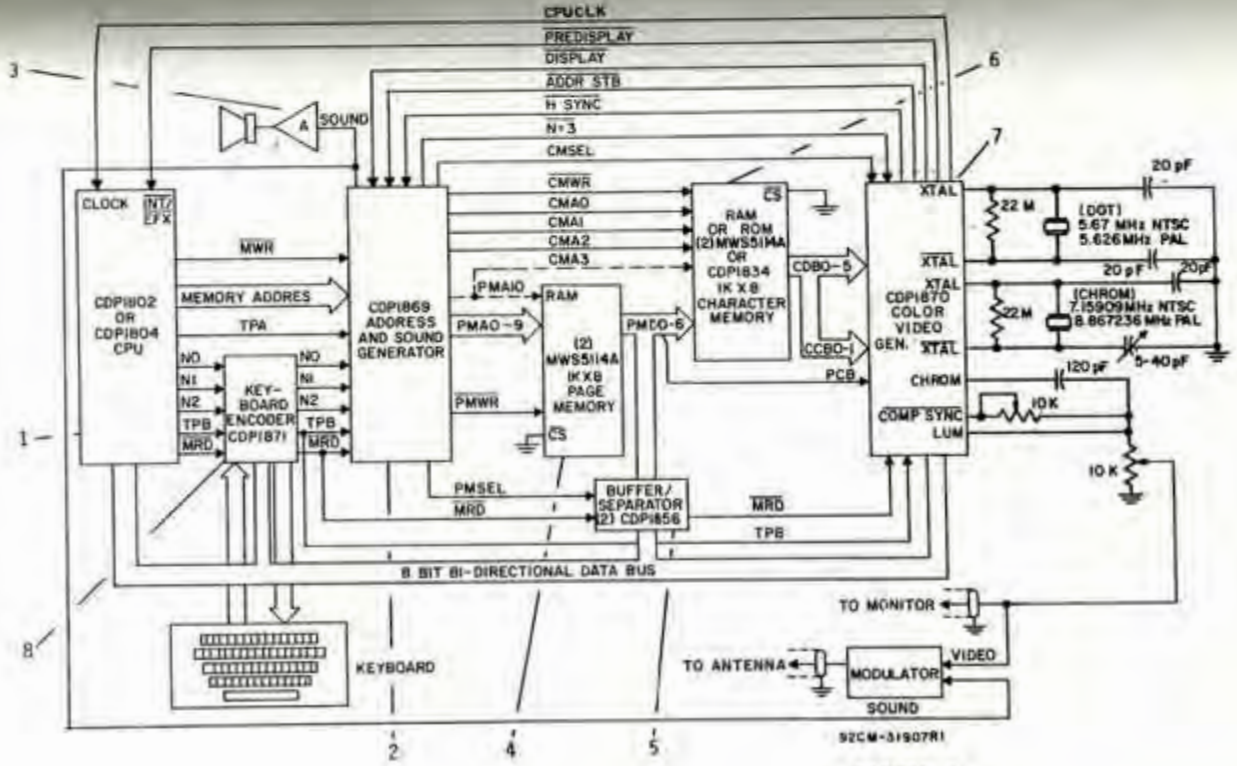
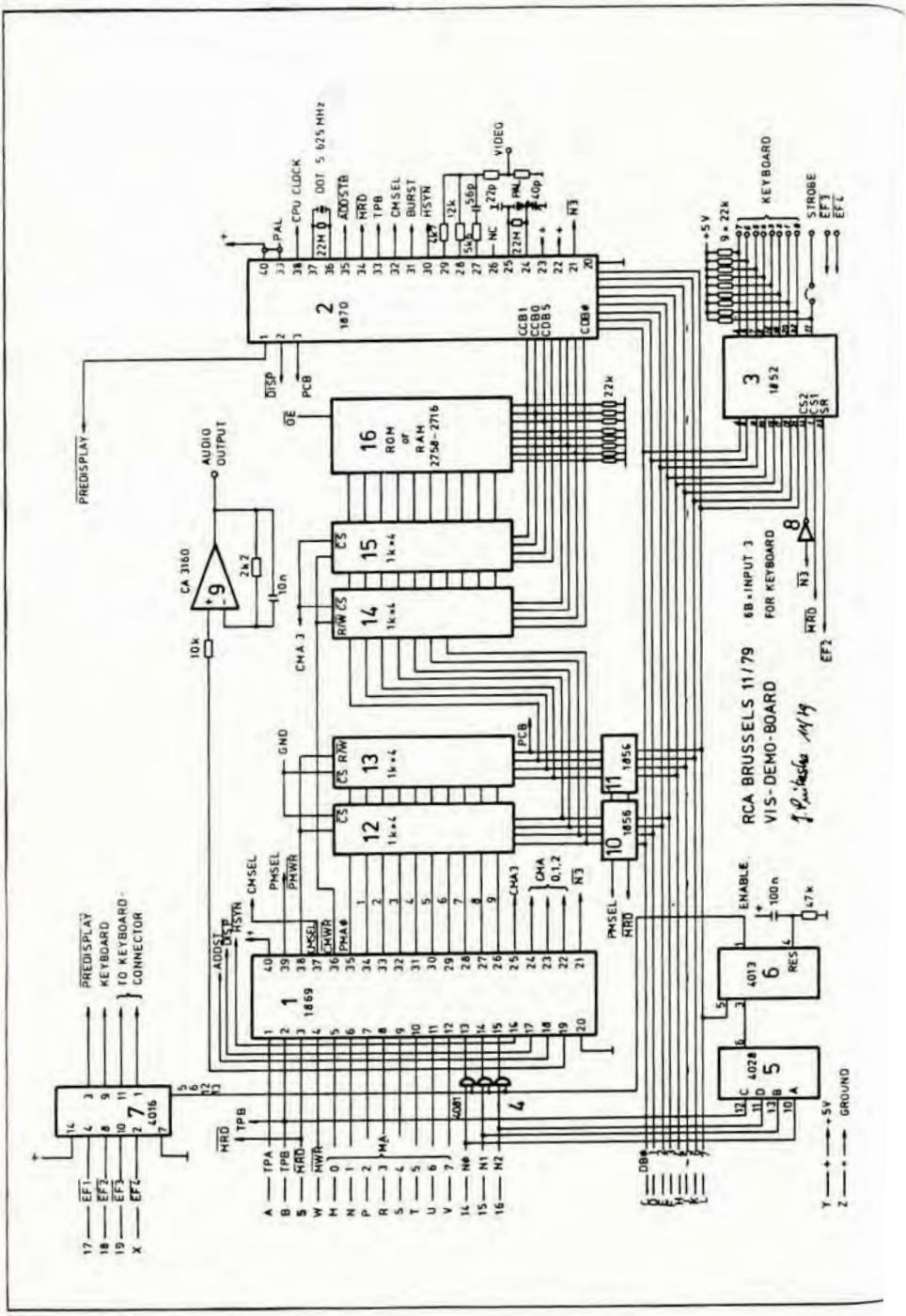

CPUCLK
PREDISPLAY
DISPLAY
ADDR STB
H SYNC
N=3
CMSEL
CMWR
CMA0
CMA1
CMA2
CMA3
SOUND
CLOCK
NTSC PAL
CDP1802 OR CDP1804 CPU
MWR
MEMORY ADDRES
TPA
N0
N1
N2
TPB
MRD
KEY-BOARD ENCODER CDP1871
CDP1869 ADDRESS AND SOUND GENERATOR
PMA10
PMA0-9
PMWR
RAM
(2) MWS5114A 1K X 8 PAGE MEMORY
CS
PMD0-6
RAM OR ROM (2) MWS5114A OR CDP1834 1K X 8 CHARACTER MEMORY
CDB0-5
CCB0-1
PCB
XTAL
CDP1870 COLOR VIDEO GEN.
CHROM
COMP SYNC
LUM
22 M
20 pF
(DOT) 5.67 MHz NTSC 5.626MHz PAL
22M
(CHROM) 7.15909MHz NTSC 8.867236 MHz PAL
5-40 pF
120 pF
10K
PMSEL
BUFFER/ SEPARATOR (2) CDP1856
8 BIT BI-DIRECTIONAL DATA BUS
KEYBOARD
TO MONITOR
TO ANTENNA
MODULATOR
VIDEO
SOUND
92CM-31907R1

FIGURE 3


RCA BRUSSELS 11/79
VIS-DEMO-BOARD
PREDISPLAY
KEYBOARD
TO KEYBOARD CONNECTOR
AUDIO OUTPUT
CA 3160
CPU CLOCK
PAL
VIDEO
KEYBOARD
STROBE
GROUND
ENABLE
RES
INPUT 3
FOR KEYBOARD

# CDP1802 MICROPROCESSOR IN TELEPHONES

The ability of microprocessor based systems to allow one design with software program changes to meet differing functional requirements rapidly and without need for changes to the integrated circuit combined with the practical feasibility of system expansion are features and benefits attractive to many system applications.

A microprocessor realized in CMOS technology combines these benefits with those of very low power consumption and low voltage operating capability. In a sub set application a CMOS microprocessor allows operation of the telephone line current during dialling and "on hook" for number storage and other functions e.g. clock. This eliminates the need for a mains power supply which gives savings in space, cost and installation time.

Conceptually the system design proposed is modular so as to allow expansion from a minimum system offering for example : repertory dialling, repeat-last-number to a more complex sub set with additional functions for example : clock, timer,LCD display and cost of call.

The design assumes the use of a "key block" for dialling and offers either pulse output dialling as a direct output from the microprocessor or DTMF (Dual Tone) dialling utilizing the RCA CMOS DTMF device CD22859. Analogue circuitry for line interfacing, ringing etc. are independent of the microprocessor system.

In this system, the key block decoding and DTMF control are performed using a complex input/output CMOS device, the CDP1851.

Alternative decoding methods ,for example the one utilized in the minimum system ,are possible and have to be optimized for a specific design.

The LCD display in the demonstration unit uses 4 x CD4056 devices; alternative driving devices for production designs are possible e.g. the RCA CA22105.

A timer device, the CD4536, is utilized in the clock or cost-of-call function : it simply generates an interrupt with the appropriate time period which activates a programmed clock routine.

This system can be further extended by additional I/O and memory devices to perform extra functions such as intercom.

## Performance Characteristics

| | | |
|---|---|---|
| a) | Program length | 600 bytes |
| b) | RAM size | 128 bytes |
| c) | Operating frequency | 1MHz nominal<br>Can be reduced to 50KHz in idle condition (off hook) |
| d) | Power consumption | 2 mA at 5V/1MHz<br>200 microamps at 5V/50KHz |
| e) | Quiescent consumption | 5 microamps at 2.4V |

RCA has developed several techniques for reducing the operating and quiescent power consumption which are not utilized in the demonstration set. These include :

- Automatic frequency reduction in idle state
- System clock removal
- Power switch-off for ROM

FUNCTIONS

. Keyboard decoding

. Pulse of DTMF

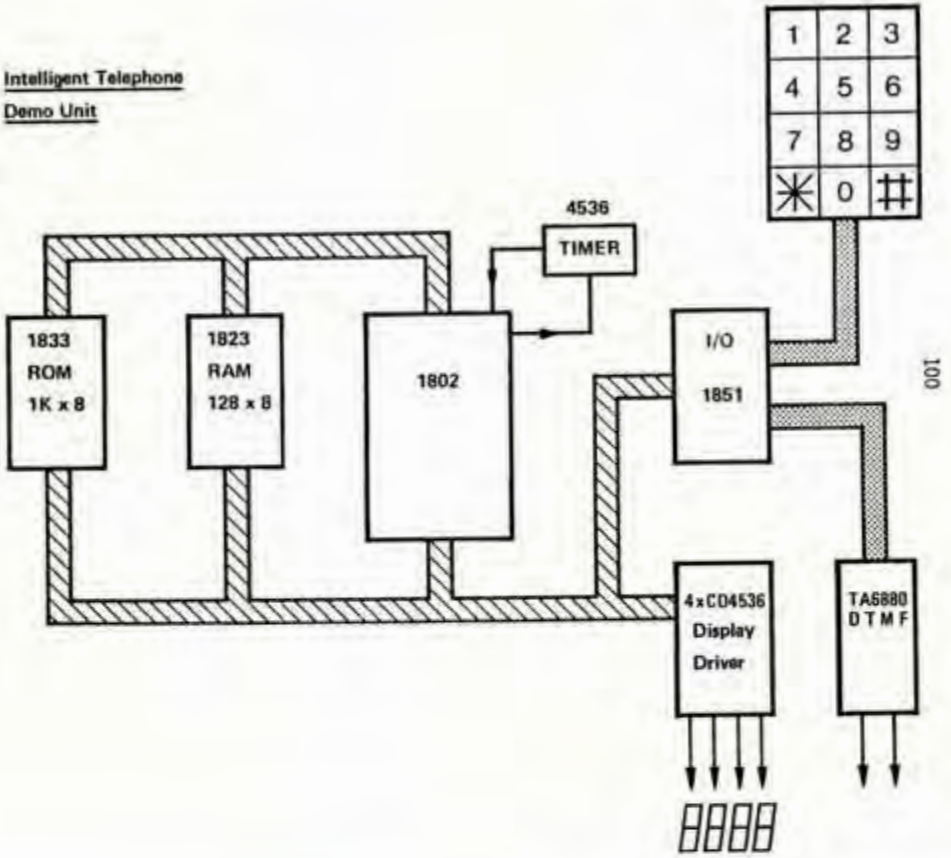
. Repeat last number

. Repertory dialling
  10 numbers/15 digits

. Clock 4 digit LCD

. Timer or cost of call

**Intelligent Telephone**
**Demo Unit**



## MINIMUM INTELLIGENT TELEPHONE

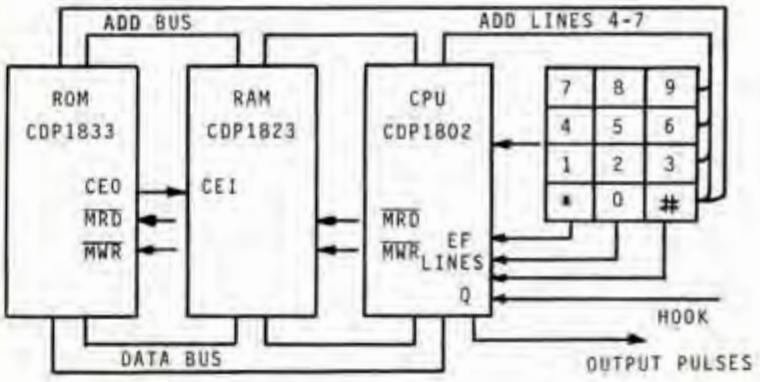
To get a minimum solution for an intelligent telephone some of the possibilities could be cut out and the rest optimized.

1. Using the Q line would give a direct toggle output

2. Using keyboard between address bus and EF lines would need no external hardware.

3. CDP1823 RAM has 128 bytes of RAM. 8 bytes per number would allow 10 numbers, 16 digits long, the rest for program use.

4. The program is about 1/2K long (CDP1831) and the CDP1831 includes a chip enable output to select the RAM.



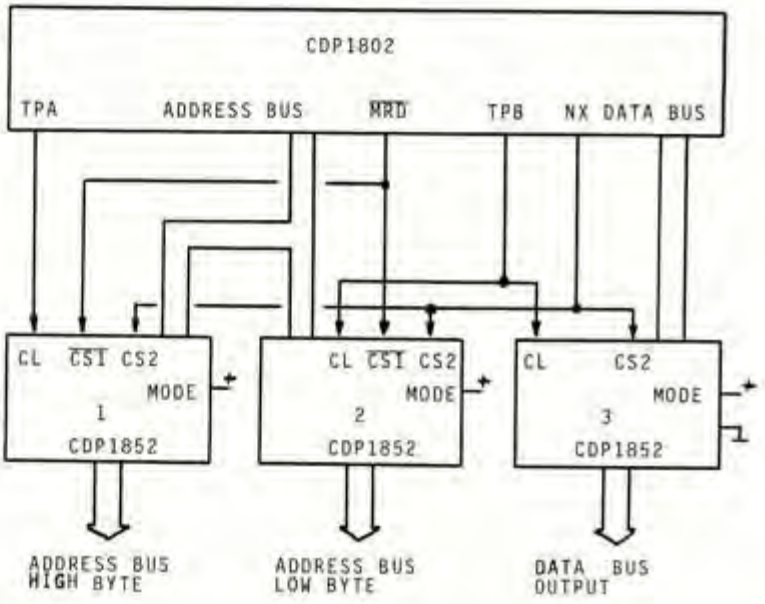
## 16 BIT OUTPUT

The CDP1802 transfers the address as two bytes : high byte and low byte. This feature allows a 16 bit data transfer from register to a 16 bit output port. In this case not the data is used as information but the address,or better the contents of one of the registers.

It is done via an output instruction.

Normally, the CDP1802 addresses the memory via the X-register and the data stored at this location is stored in one of 7 output devices.

But in this case, the port on the data bus does not exist. Only the address bus is latched means high byte using TPA and low byte using TPB. As 'chip enable' the N lines are used together with $\overline{MRD}$. REF. ICAN-6562 "Register Based Output"



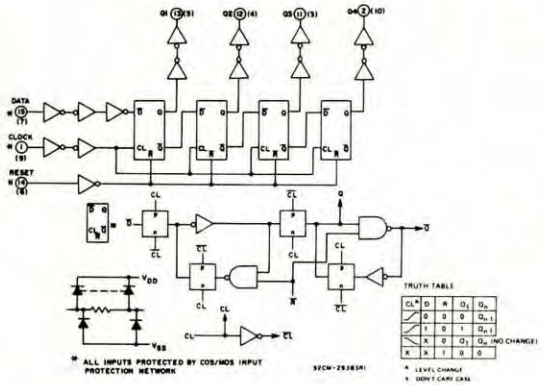
## LED INTERFACE

### As one memory location or as one output

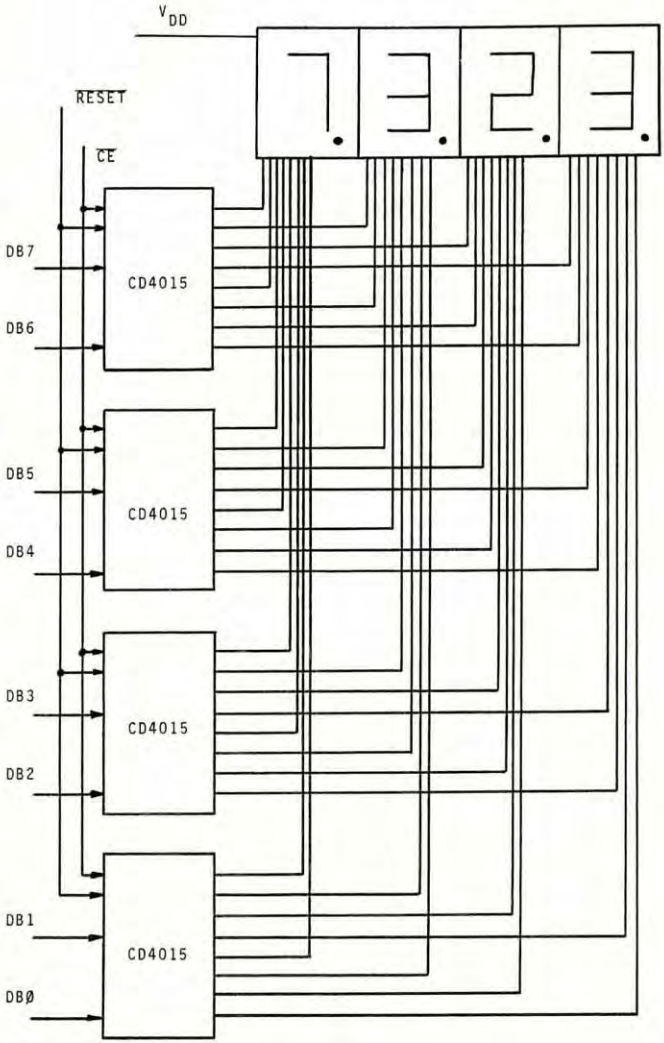
To have a LED interface to the Microprocessor system, many possibilities can be used. One especially easy to use is having a shift register as one output port. As the data is transferred from memory directly to I/O only a sequence of output instructions are necessary to transfer any number of data to this output.

Connected to drivers any segment configuration can be displayed. Each byte can be connected to decoder circuits to use each byte as two digits.

With 4 CD4015, 4 LED displays can be addressed directly. This chain can be expanded to any number and has, compared with multiplexing, the advantage that once transferred the data is stable.




LOGIC DIAGRAM OF CD4015

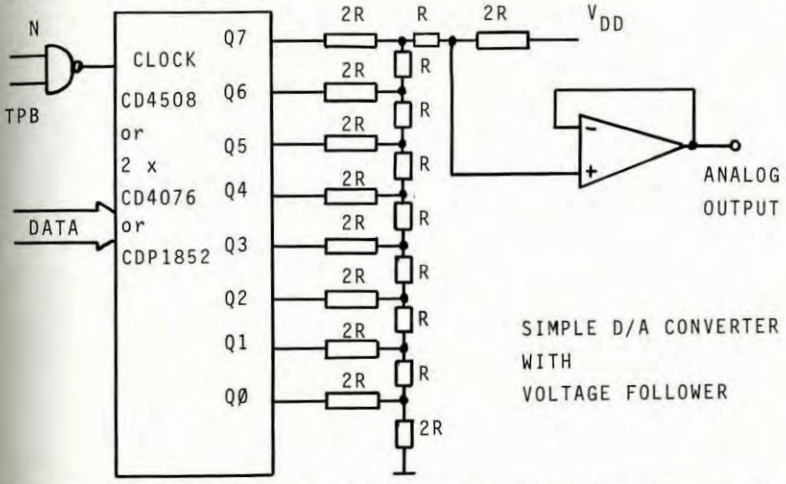



OUTPUT CIRCUITS FOR A 4 DIGIT LED DISPLAY USING ONE OUTPUT INSTRUCTION AND SHIFT REGISTERS

# D-A-CONVERSION

Very often a simple analog output is needed and the resolution of 5% will be enough. In a lot of the cases even speed is not the most important feature . Under these conditions, a very cheap analog interface is possible, completely in CMOS.

One CD4508 8 bit latch or 2 CD4076 4 bit latch together with some resistors are needed.




SIMPLE D/A CONVERTER
WITH
VOLTAGE FOLLOWER

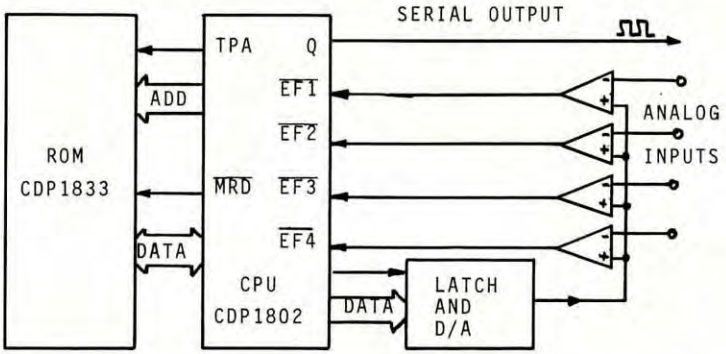
This circuit can be used either as I/O as shown here or with a different selection logic - it is also suitable for memory space.

For higher resolution and different levels a CD4054 is used instead to provide level shifting (See ICAN-6289).

If this 8 bit converter is "expanded" to a 9 bit converter, where the highest bit never changes, the output voltage changes between OV and $V_{DD/2}$,this means even single supply is possible.

## A-D-CONVERSION

Analog to digital conversion can be achieved the same way as using the D/A converter and compare it with the analog input. Four analog inputs can be built this way, using the EF lines as input to the Microprocessor. In a minimum data aquisition system the Q line would send the serial information (using the UT4 routines) to a central unit.




This system shows the feature of the CDP1802 to work completely without RAM, as internal registers can be used for storing.

Different algorithms are possible for this type of A to D converter using software :

1. COUNTER

   The software counter starts at 00, and increments and outputs this to the latch. As the EF lines change, the value is reached and stored in one half of a free register.

2. UP-DOWN-COUNTER

Here the counter looks at the EF line and counts up or down depending on high or low.

3. SUCCESSIVE APPROXIMATION

The D to A value works like a weighing scale and puts the output to half if bigger adds the next (quarter) until done 8 times.

| | | |
|---|---|---|
| a. | Set output to half scale | 1000 0000 |
| b. | If bigger, store in a register, | otherwise substract |
| c. | Try with | 0100 0000 |
| d. | Then with | 0010 0000 |
| e. | | 0001 0000 |
| f. | | 0000 1000 |
| g. | | 0000 0100 |
| h. | | 0000 0010 |
| k. | | 0000 0001 |

So after eight times,the whole conversion is done. But with this system the analog input has to be stable during the conversion period, otherwise "out of range" could happen.

## CA3162

### A three digit A/D interface for COSMAC

This circuit, together with one CD4016 quad transmission gate lets the CDP1802 understand analog inputs with an accurracy of 3 digit and a conversion rate up to 96Hz.

The input range is +999mV to -99mV and an extra overrange indication is built in.

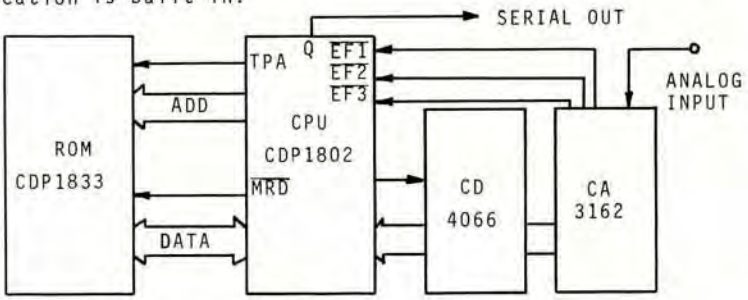
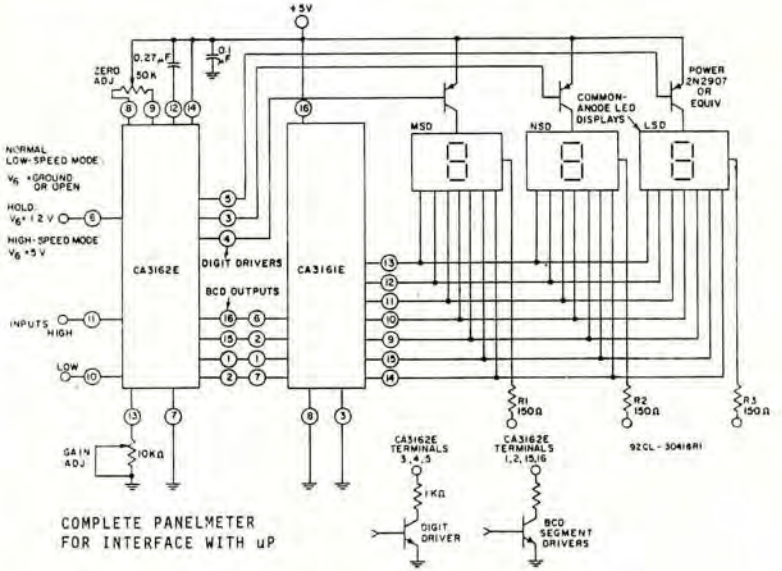
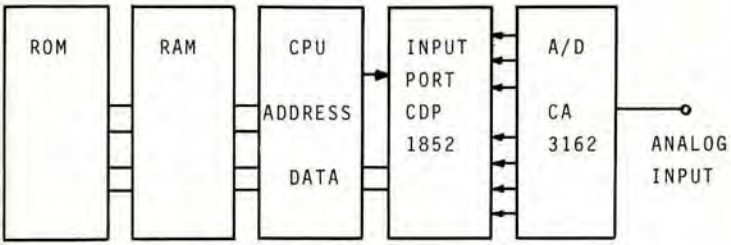



Figure 1 : Minimum Interface



Another approach would be to connect the CA3162 completely to the data bus via an input port CDP1852.




SUBROUTINE TO INPUT DATA IN MEMORY

```
EXIT:   SEP 5           .. return to calling program
ENTRY:  LDI #4F         .. set up memory pointer
        PHI 8           .. for input of data
        LDI #F0         .. 4FF0
        PLO 8           ..
        SEX 8           .. set X pointer to 8
MSD:    INP 4           .. get digit
        ANI #80         .. select MSD
        BNZ  MSD        .. wait for low
        INC 8           .. LSD comes next
        INC 8           .. so 2 x INC
LSD:    INP 4           .. input
        ANI #20         .. select bit
        BNZ  LSD        .. wait for it
        DEC 8           .. point to NSD now
NSD:    INP 4           .. get word
        ANI #40         .. look for bit
        BNZ  NSD        .. wait for bit
        DEC 8           .. point to
        DEC 8           .. a free location
        BR  EXIT        .. go to EXIT
```

With minor program changes it can use internal registers as memory locations, this is possible as the input instruction stores data in M (R(X)) and in the D-register.

```
      .. Subroutine to input data in registers
EXIT: SEP 5           .. Go back to calling program
MSD:  INP 4           .. Get the 7 lines of CA3162
      ANI #80         .. Look only at bit 7
      BNZ   MSD       .. Wait for low
      PLO 9           .. Store in low half of R9
LSD:  INP 4           .. Next digit
      ANI #20         .. Low digit comes faster
      BNZ   LSD       .. Wait for low
      PLO 7           .. Store it in low half of R7
NSD:  INP 4           .. Now NSD
      ANI #40         .. Mask bit 6
      BNZ   NSD       .. Wait for low
      PLO 8           .. Store in low half of R8
      BR  EXIT        .. Branch to exit
```

The information is stored in R7, R8, R9.

This subroutine exits with a SEP5 instruction, means it can be called via SCRT.

The memory locations used are :

```
4FF0      MSD
4FF1      NSD
4FF2      LSD
```

For more information see data sheet CA3162.

## SIMPLE CONTROL INTERFACE for CDP1802

The COSMAC Microprocessor CDP1802 has an internal oscillator that works with a crystal connected between CLOCK and $\overline{XTAL}$ terminals. If desired, however, an external oscillator may be used and fed into the CLOCK input. If an external oscillator is used, no connection is required at the $\overline{XTAL}$ terminal. (Note : care must be taken not to load the $\overline{XTAL}$). Any type of single-phase clock may be used so long as the rise and fall times of the clock pulse are less than 15 microseconds. Each machine cycle consists of eight clock pulses, and each instruction requires two or three machine cycles. Thus, with a 6.4MHz clock frequency, a machine cycle of 1.25 microseconds could be achieved, and instructions would be executed in 2.5 to 3.75 microseconds depending on the instruction.

During normal operation, the $\overline{CLEAR}$ and $\overline{WAIT}$ lines are both held high. A low level on the $\overline{CLEAR}$ line will put the machine into the reset mode with I,N,X,P,Q, Data bus = 0, and IE = 1. Actually, X,P, and R(0) are reset during a special S1 cycle (not available to the programmer) immediately following transition from the reset mode to any of the other modes (load, run, or pause). The clock must be running to effect this cycle.

If the $\overline{CLEAR}$ and $\overline{WAIT}$ lines are both held low, the machine enters the load mode. This mode allows input bytes to be sequentially loaded into memory beginning at M(0000). Input bytes can be supplied from a keyboard, tape reader, etc... by way of the DMA facility. This feature permits direct program loading without the use of external "bootstrap" programs in ROM's.

If the $\overline{WAIT}$ line is brought low (with $\overline{CLEAR}$ high), the CPU stops operation cleanly on the next negative-going transition of the clock (Pause mode). Output signals are held at their values indefinitely. This state is useful for several purposes.

Using the $\overline{WAIT}$ line, the CPU can be easily single-stepped for debugging purposes or, if stopped early in the machine cycle, the CPU can be held off the data bus to allow for multiprocessor systems, etc... Also, the $\overline{WAIT}$ line can be used as a data ready signal from a slow memory or peripheral, or signals TPA and TPB can be streched. When the $\overline{WAIT}$ line is returned high, the machine resumes running on the next negative-going transition of the clock input. The $\overline{WAIT}$ signal does not inhibit the on-chip crystal oscillator. DMA's and Interrupts are not acknowledged in the Pause mode.

Figure 1 shows one circuit using standard devices from the CD4000 series for controlling the run and load modes of the CDP1802. Note the power-on reset feature. For design details, refer to ICAN-6581 "Power-On Reset/Run Circuits for the RCA CDP1802 COSMAC Microprocessor". To load and start a program the sequence of operations would be as follows : first, depress the reset and then the load buttons. The CPU is now ready to load by means of the DMA channel. When loading is completed, depressing the reset and then the run buttons will start program execution at M(0000) with R(0) as the program counter (after one machine cycle). If a DMA request is present when the run switch is turned on , the machine will go into the DMA state immediately with R(0) as the program counter. The user should therefore inhibit DMA externally until the program has changed to a program counter different from R(0). Interrupts, however, are disabled until the first instruction or DMA request is executed. This delay allows the programmer to place instruction 71 and 00 in the first two memory bytes to inhibit interrupts until he is ready for them. The combined effect of the two bytes is to set IE = 0. Interrupts must not occur, however, when the machine is in the load mode because they will force the machine into an anomalous running state.

Another circuit that can be used for single-stepping the Microprocessor (one machine cycle per switch depression) is shown in Figure 2. This capability is often useful as a debugging aid.

Figure 3 provides a summary of the modes discussed, the control levels, and the characteristic features of these modes. It is evident that the run mode can be entered from either the reset or the pause mode.

For more information, see MPM 201 page 70.

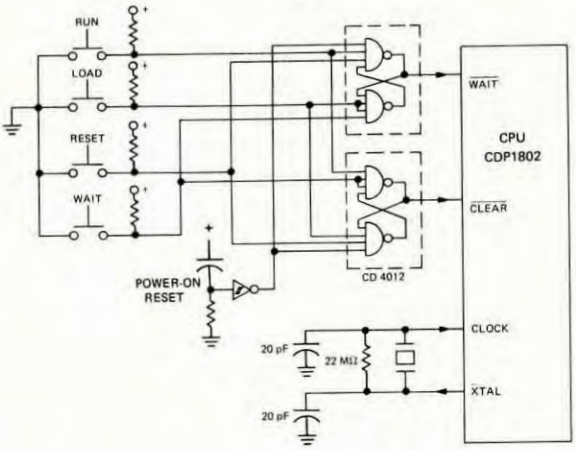



Figure 1 : Simple Control Interface of CDP1802 Microprocessor

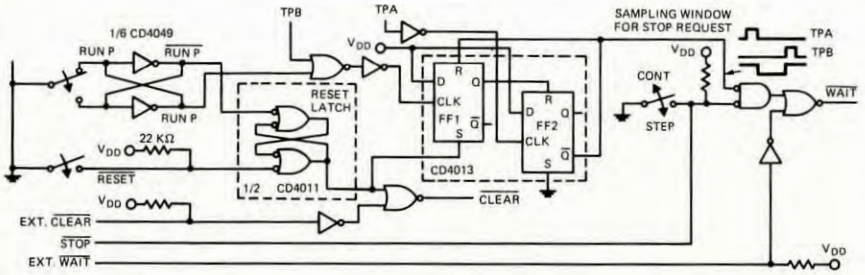



Figure 2 : Circuit for Single-Stepping the CDP1802

| MODE | CLEAR | WAIT | OPERATION |
|---|---|---|---|
| RESET | 0 | 1 | I, N, X, P = 0, R(0) = 0, Q = 0, BUS = 0, IE = 1; TPA and TPB are suppressed; CPU in S1. |
| RUN | 1 | 1 | CPU starts running one machine cycle after CLEAR is released. Execution starts at M(0000), or an S2 cycle follows if DMA was asserted. Internal sampling of interrupt is inhibited during initialization cycle. |
| RESET | 0 | 1 | As above. |
| LOAD | 0 | 0 | CPU in IDLE. An I/O device can load memory without "bootstrap" loader. |
| PAUSE | 1 | 0 | Clock: ⎍ ← stops internal operation. CPU outputs held indefinitely. Permits stretching of machine cycle to match slow devices or memory cycles. DMA and INTERRUPTS not acknowledged. |
| RUN | 1 | 1 | Clock: ⎍ Resume operation |

Figure 3 : Thruth Table for Mode Control

# UNDERSTANDING THE CDP1851 PROGRAMMABLE I/O

## Overview

The CDP1851 is a general purpose programmable I/O device, having 20 I/O pins which may be used in several different modes of operation. (See table 2).

The I/O lines are grouped into two sections, A and B, each having 10 lines; 8 data and 2 handshaking lines (ready and strobe).

In essence, the CPU programs the PIO by asserting the proper address (if memory mapped) or the proper N-line code (if I/O mapped) on the PIO control lines, and outputs a sequence of control bytes on the data bus to the control register of the PIO. The control bytes contain information to define port mode, interrupt enable/disable, I/O bit assignment, bit masking, etc... (See codes A-P, Table 1).

The CPU transfers data bytes to and from each port by asserting codes S, T, U or V, given in Table 1. Modes may be combined so that their functional definition can be tailored to almost any I/O requirement.

## Modes of Operation

### a. Normal Input/Output Mode

Ports A and B can be separately programmed to be 8 bit input or output parts with handshaking control lines, ready and strobe.

b. Bi-Directional Mode

Port A can be programmed to be a bi-directional port. This configuration provides a means for communicating with a peripheral device or structure on a single 8 bit bus for both transmitting and receiving data. Handshaking signals are provided to maintain proper bus flow discipline. The handshaking lines for port A are used in the normal manner for input control. The handshaking lines for port B are used by port A for output control; consequently, port B I/O lines must be in the bit programmable mode where handshaking is not used.

c. Bit-programmable Mode

Ports A and B can be separately bit programmed so that each individual line can be designated as an input or output line.

An additional feature of the bit programmable mode is that the four handshaking lines, A RDY, A STROBE, B RDY, and B STROBE can be individually programmed as input or output lines (See code K, Table 1).

In the input, output, and bi-directional modes the STROBE lines give the PIO the trigger signals needed to generate interrupt requests. However, in the bit programmable mode the handshake lines are not used to carry strobe and ready signals, but carry I/O data if programmed to do so.

Interrupt

Interrupt requests are generated differently depending on the port mode.

a. Input and Output Modes

The falling edge of the strobe pulse from the peripheral device sets the $\overline{INT}$ line, and the rising edge of TPB, during the requested read or write operation resets the $\overline{INT}$ signal.

b. Bi-Directional Mode (Port A only)

Set and reset of interrupt requests are done as explained in the input and output modes. However, since the $\overline{A\ INT}$ line is used for both input and output interrupts the CPU must read the status register to determine what condition caused the interrupt request.

c. Bit-Programmable Mode

Interrupt requests can be generated by programming the PIO to re-act to specified logic functions (AND, OR, NAND, or NOR) on selected I/O lines. The CPU must issue two control bytes; the first will select the logic condition, and the second will contain masking information indicating which bit(s) of eight the PIO will monitor for the logic condition. The $\overline{INT}$ signal will exist while the logic condition is present. (See codes I & J, Table 1).

d. Interrupt Enable/Disable

To enable or disable the $\overline{INT}$ lines in all modes, the CPU must issue a control byte for each port (See codes L M N & P, TABLE 1). A and B interrupt status can be read from bit $D_0$ and $D_1$ of the status register to determine which $\overline{INT}$ line is causing the request if they are wired together (OR'd).

Timing

a. Input Data Transfer

Refer to input port sequential timing diagram. Assume an I/O mapped I/O system similar to Figure 1. The peripheral presents data to the I/O port and outputs a strobe pulse to the PIO. The strobe pulse causes three things to happen :

1. The READY signal is reset inhibiting further transmission from the peripheral.

2. The input data is latched in the buffer register.

3. The $\overline{INT}$ line is set low signalling the CPU to read the data.

The A and B $\overline{INT}$ lines of the PIO may be wired to the $\overline{INT}$ pin on the CPU to signal program interrupts, or they can be wired to separate $\overline{EF}$ pins where periodic polling of the $\overline{EF}$ pins is required to check for service requests.

In either case, the program will branch to a subroutine and execute an input instruction (INP6 or INP7, see codes S & T, Table 1) which will assert the proper code on the RA0, RA1, and CS pins of the PIO. The PIO will place data onto the system bus so it can be used by the CPU and/or written into memory.

The TPB pulse that occurs during the $\overline{MWR}$ pulse terminates the interrupt request and sets the READY line, indicating to the peripheral that the PIO is ready to accept a new data byte.

b. Output Data Transfer

Refer to output port sequential timing diagram. Assume an I/O mapped I/O system similar to example 1. A strobe pulse from a previous output sequence or a dummy strobe causes the $\overline{INT}$ to go low signalling the CPU to output a data byte to the PIO.

The PIO $\overline{INT}$ line can be wired to the CPU $\overline{INT}$ pin or an $\overline{EF}$ pin as explained in input data transfer above. The program will branch to a subroutine and execute an output instruction (OUT6 or OUT7) which will assert the proper code on the RA0, RA1, and CS pins of the PIO.

Data will be read from memory and placed on the bus and latched into the port buffers on the trailing edge of the TPB. The READY line is also set at this time. The peripheral will transmit a strobe pulse indicating the reading process is completed. The rising edge of the strobe pulse causes the READY signal to reset, and the falling edge sets the interrupt request signalling the CPU to output another data byte.

c. Data Transfer, Bit-Programmable Mode

The CPU loads a data byte to the 8 bit port as in the normal output mode. I/O lines programmed as outputs will accept and latch data bits, however, I/O lines programmed as inputs will ignore the loaded data (See code H, U, and V of Table 1).

The CPU reads the 8 bit port as in the normal input mode. I/O lines are non-latching and therefore input data must be stable while the CPU reads. All 8 I/O lines are read whether they are programmed as input lines or output lines. Data read from the lines programmed as outputs will be data bits latched during the last output cycle (See codes H, S, & T, Table 1).

Examples

Example 1 - Simple Input/Output Mode

Figure 1 shows an I/O space I/O system involving the 1802 CPU, the 1851 PIO, and two peripheral devices both with handshaking. The $\overline{INT}$ lines are individually connected to $\overline{EF}$ lines 1 and 2. Therefore, the CPU is not truly interrupted but must poll the flag lines periodically during the main program.

Step by Step - Refer to flow chart for example 1

1. To begin using this system the device must be cleared which automatically programs both port A and B to the input mode.

2. Port B is set to the output mode by loading the control register with control byte given in code D.

3. Port A interrupt line is enabled by loading control byte given in code L, Table 1.

4. Dummy read port A to raise the ready line.

5. Port B interrupt line is enabled by loading control byte given in code M, Table I. Now the main program can begin running.

6. Some time during or at the end of the main program the EF1 flag is polled. If it is true the program will branch to a subroutine to read port A. The CPU must output the proper N-line code to read A. (See code S Table 1, and input port timing diagram). When this step is completed the flag is again polled to check for more incoming data.

7. If EF1 is false then EF2 is considered. If EF2 is true then the program will branch to a subroutine to load port B with data. The CPU must output the proper N-line code and place the 8 bit data word on the bus. (See code V, Table 1 and output port timing diagram). When this step is completed the CPU returns to the main program.

8. If EF2 is false the CPU returns to the main program.

Example 2

Figure 2 shows an I/O space I/O system involving the 1802 CPU, the 1851 PIO, and two peripheral devices. Peripheral A has a bi-directional bus with handshaking capability for transmit and receive. Peripheral B has a 4 bit receiver and a 4 bit transmitter.

Port A will be programmed for bi-directional mode with interrupt capability.

Port B will be programmed for bit programmable mode with interrupt capability for output data when a logic NOR (all (0's) occurs on the 4 input lines.

Priorities will be as follows :

1. Port A input data, via interrupt and subroutine
2. Port A output data, via interrupt and subroutine
3. Port B output data, via interrupt and subroutine
4. Port B input data, part of main program.

Port A is used to interface with a bi-directional device, therefore the handshaking lines A READY and A STROBE are used for control of incoming data (from peripheral to CPU). B READY and B STROBE handshaking lines are used for output control (from CPU to peripheral).

Port B is used in the bit programmable mode having bits B0, B1, B2 and B3 as outputs (data from CPU to peripheral) and bits B4, B5, B6 and B7 as inputs (data from peripheral to CPU).

The proper coding of the CPU N-lines will select whether the data will be read in, or written out to the PIO, or whether the status register is to be read, or whether the control register is to be loaded with a control byte. (See Table 1).

Step by step

1. To begin using this sytem the device must be cleared which automatically programs both port A and B to the input mode.

2. Since port A is going to be the bi-directional mode, port B must be programmed in the bit programmable mode. Port B is set to the bit mode by loading control byte given in code G, Table 1.

3. Code H control byte must be loaded. This byte determines which bits are input/output.

4. Code I control byte must be loaded which determines the logical condition of bits required to generate an interrupt request from port B (in this case a NOR condition).

5. Code J control byte must be loaded. This byte tells which of the 8 bits in port B are monitored and which are masked for interrupt generation. (In this case all of the input lines B4, B5, B6 and B7 will be monitored).

6. Now port A is set to bi-directional mode by loading code E control byte.

7. Port A interrupt line is enabled by loading the control register with code L control byte.

8. A RDY is raised (input section of port A handshaking line) by doing a dummy read.

9. Port B interrupt line is enabled by loading the control register with code M control byte. Now the main program can begin running.

10. Port B will be read periodically as the main program repeats its loop.

11. When an interrupt request is received by the CPU it will branch to a subroutine, and automatically de-activate the interrupt enable to inhibit further interruptions.

12. The CPU must read the status register by outputting the proper N-line code R. The status register will contain information describing which interrupt has occurred A and/or B, and also indicate whether $\overline{\text{A INT}}$ was caused by an input or an output demand from peripheral A.

    NOTE that $\overline{\text{A INT}}$ and $\overline{\text{B INT}}$ lines are wired (OR'd together (See Figure 2).

13. Several decisions will be made based on the information contained in the status register.

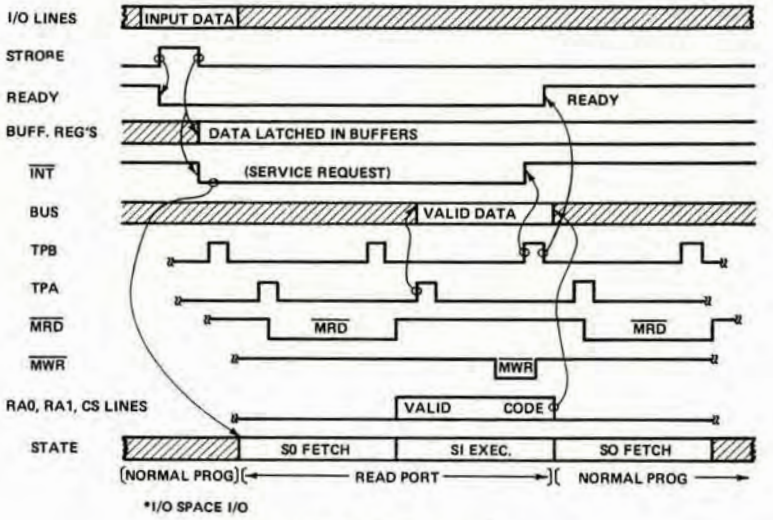
14. To read port A the CPU must output the proper N-line code S.

15. To load port A with data the CPU must output the proper N-line code, and place the 8 bit data word on the bus (See Table 1).

16. To load port B (bits B0, B1, B2 and B3) with data the CPU must output the proper N-line code, and place the 4 bit data on the bus (See Table 1).

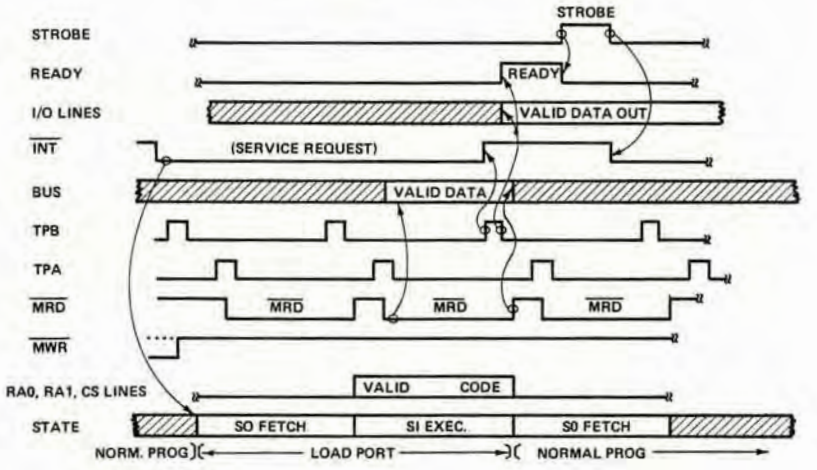
17. Once the interrupt subroutines are completed the CPU returns to the main program and the interrupt enable (CPU) is activated.

INPUT PORT SEQUENTIAL TIMING*




*I/O SPACE I/O

OUTPUT PORT SEQUENTIAL TIMING*




*I/O SPACE I/O

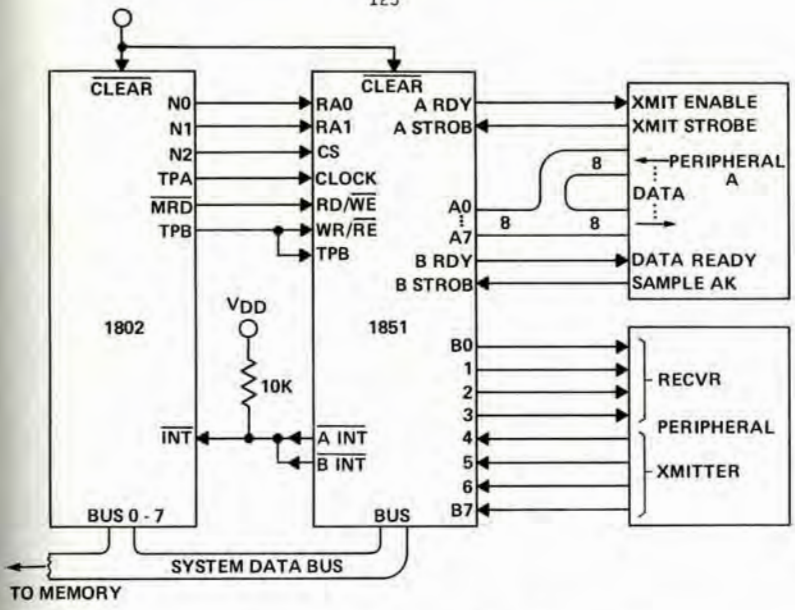



Figure 1 : Hardware For Example 1

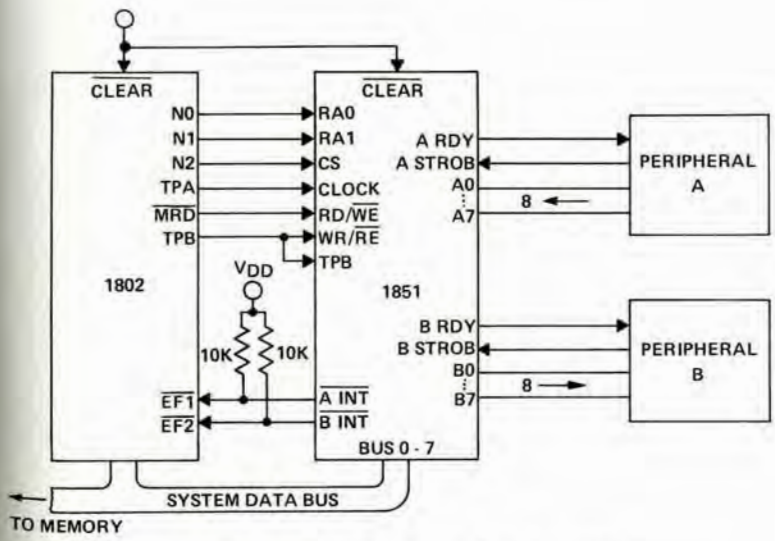
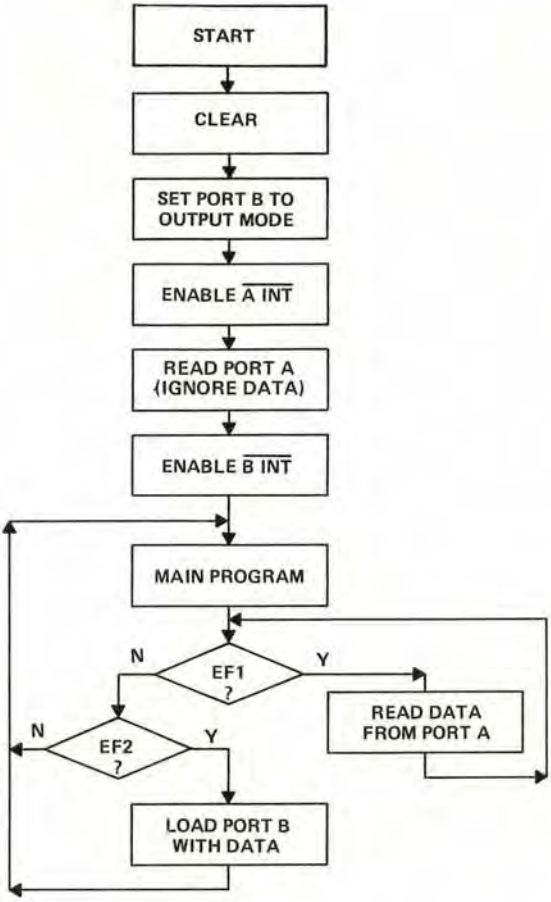
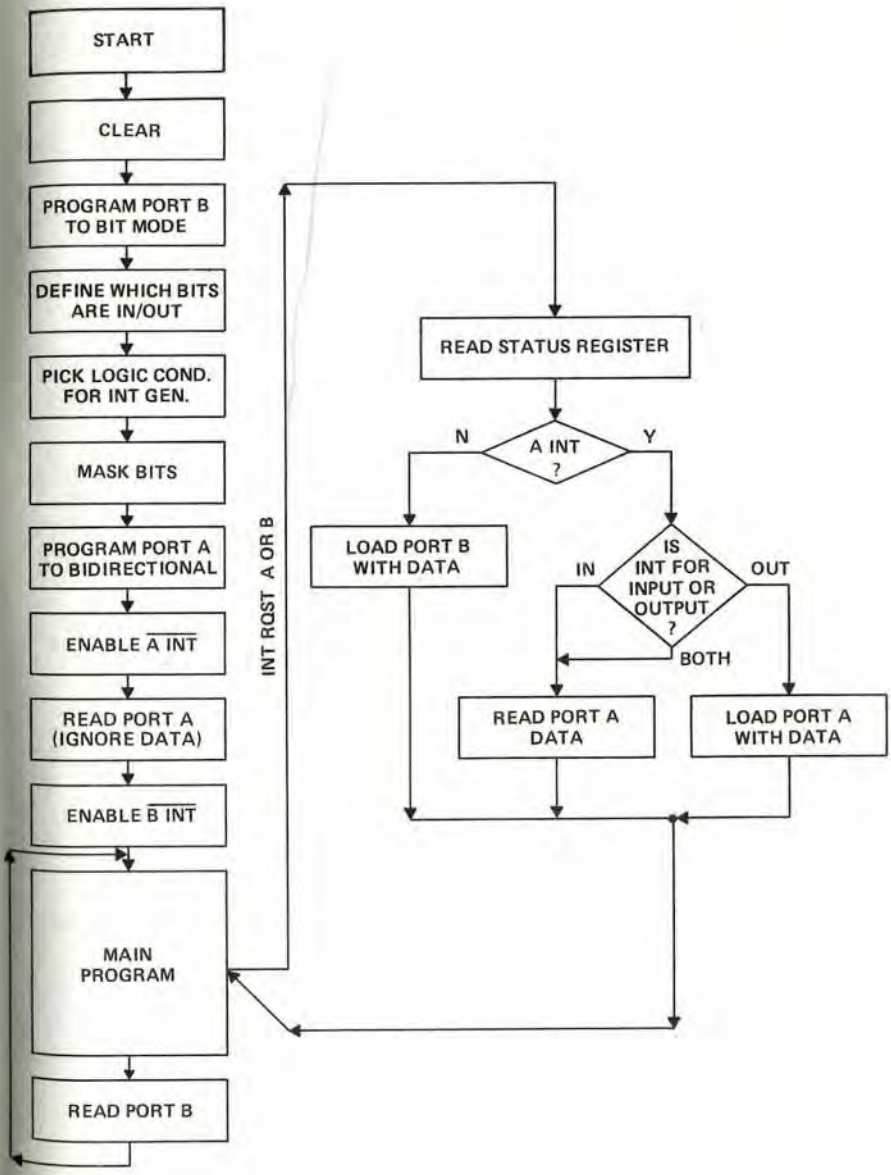



Figure 2 : Hardware For Example 2

FLOWCHART FOR EXAMPLE 1



FLOWCHART FOR EXAMPLE 2



# TABLE 1

## PROGRAMMING CODES

| CODE | DESIRED ACTION | CS | RA1 | RA0 | RD/WE | WR/RE | DATA BUS $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | SET PORT A TO INPUT MODE | 1 | 0 | 1 | 0 | 1 | 0 | 0 | X | 0 | 1 | X | 1 | 1 |
| B | SET PORT B TO INPUT MODE | 1 | 0 | 1 | 0 | 1 | 0 | 0 | X | 1 | 0 | X | 1 | 1 |
| C | SET PORT A TO OUTPUT MODE | 1 | 0 | 1 | 0 | 1 | 0 | 1 | X | 0 | 1 | X | 1 | 1 |
| D | SET PORT B TO OUTPUT MODE | 1 | 0 | 1 | 0 | 1 | 0 | 1 | X | 1 | 0 | X | 1 | 1 |
| E | SET PORT A TO BIDIRECTIONAL (PORT B MUST BE IN BIT MODE FIRST) | 1 | 0 | 1 | 0 | 1 | 1 | 0 | X | 0 | 1 | X | 1 | 1 |
| F | SET PORT A TO BIT - PROG MODE | 1 | 0 | 1 | 0 | 1 | 1 | 1 | X | 0 | 1 | X | 1 | 1 |
| G | SET PORT B TO BIT - PROG MODE | 1 | 0 | 1 | 0 | 1 | 1 | 1 | X | 1 | 0 | X | 1 | 1 |
| H | I/O BIT ASSIGNMENT (A OR B) | 1 | 0 | 1 | 0 | 1 | (0 = INPUT, 1 = OUTPUT) | | | | | | | |
| I | SET LOGICAL CONDITION FOR INTERRUPT GENERATION (BIT - MODE)<br>($D_3$) 0 = PORT A, 1 = PORT B<br>($D_4$) 0 = NO MASK, 1 = MASK BYTE FOLLOWS NEXT | 1 | 0 | 1 | 0 | 1 | 0 | $D_6$ | $D_5$ | $D_4$ | $D_3$ | 1 | 0 | 1 |
| | | | | | | | $D_6$ | $D_5$ | LOGICAL CONDITION | | | | | |
| | | | | | | | 0 | 0 | NAND | | | | | |
| | | | | | | | 0 | 1 | OR | | | | | |
| | | | | | | | 1 | 0 | NOR | | | | | |
| | | | | | | | 1 | 1 | AND | | | | | |
| J | SET MASKING OF BITS (PRECEEDED BY CODE I) | 1 | 0 | 1 | 0 | 1 | (0 = MONITORED, 1 = MASKED) | | | | | | | |
| K | SET RDY AND /OR STROBE LINES TO I/O LINES (BIT - MODE ONLY)<br>($D_1$) 0 = PORT A, 1 = PORT B<br>($D_2$) 0 = NO CHANGE TO RDY LINE FUNCTION, 1 = CHANGE PER BIT $D_6$<br>($D_3$) 0 = NO CHANGE PER BIT $D_7$, 1 = CHANGE PER BIT $D_7$<br>($D_4$) RDY LINE OUTPUT DATA TO BE LOADED<br>($D_5$) STROBE LINE OUTPUT DATA TO BE LOADED<br>($D_6$) RDY LINE USED AS 0 = INPUT LINE, 1 = OUTPUT LINE<br>($D_7$) STROBE LINE USED AS 0 = INPUT LINE, 1 = OUTPUT LINE | 1 | 0 | 1 | 0 | 1 | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | 0 |
| L | ENABLE A $\overline{\text{INT}}$ OUTPUT | 1 | 0 | 1 | 0 | 1 | 1 | X | X | X | 0 | 0 | 0 | 1 |
| M | ENABLE B $\overline{\text{INT}}$ OUTPUT | 1 | 0 | 1 | 0 | 1 | 1 | X | X | X | 1 | 0 | 0 | 1 |
| N | DISABLE A $\overline{\text{INT}}$ OUTPUT | 1 | 0 | 1 | 0 | 1 | 0 | X | X | X | 0 | 0 | 0 | 1 |
| P | DISABLE B $\overline{\text{INT}}$ OUTPUT | 1 | 0 | 1 | 0 | 1 | 0 | X | X | X | 1 | 0 | 0 | 1 |
| R | READ STATUS REGISTER | 1 | 0 | 1 | 1 | 0 | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
| | ($D_0$) $\overline{\text{B INT}}$ STATUS (1 MEANS SET) — ALL MODES<br>($D_1$) $\overline{\text{A INT}}$ STATUS (1 MEANS SET) — ALL MODES<br>($D_2$) 1 = $\overline{\text{A INT}}$ WAS CAUSED BY B STROBE — BIDIRECT MODE A ONLY<br>($D_3$) 1 = $\overline{\text{A INT}}$ WAS CAUSED BY A STROBE — BIDIRECT MODE A ONLY<br>($D_4$) A RDY INPUT DATA — BIT PROGRAMMABLE MODE<br>($D_5$) A STROBE INPUT DATA — BIT PROGRAMMABLE MODE<br>($D_6$) B RDY INPUT DATA — BIT PROGRAMMABLE MODE<br>($D_7$) B RDY INPUT DATA — BIT PROGRAMMABLE MODE | | | | | | | | | | | | | |
| S | READ PORT A | 1 | 1 | 0 | 1 | 0 | (INPUT DATA BYTE) | | | | | | | |
| T | READ PORT B | 1 | 1 | 1 | 1 | 0 | (INPUT DATA BYTE) | | | | | | | |
| U | LOAD PORT A | 1 | 1 | 0 | 0 | 1 | (OUTPUT DATA BYTE) | | | | | | | |
| V | LOAD PORT B | 1 | 1 | 1 | 0 | 1 | (OUTPUT DATA BYTE) | | | | | | | |

TABLE 2

CDP1851 PROGRAMMING MODES

| Mode | (8) Port A Data Pins | (2) Port A Handshaking Pins | (8) Port B Data Pins | (2) Port B Handshaking Pins |
|---|---|---|---|---|
| Input | Accept Input data | Ready Strobe | Accept Input data | Ready Strobe |
| Output | Output Data | Ready Strobe | Output Data | Ready Strobe |
| Bi-directional (Port A only) | Transfer input/output Data | Input Handshaking for Port A | Must be previously set to bit-programmable mode | Output Handshaking for Port A |
| Bit-Programmable | Programmed individually as inputs or outputs | Programmed individually as inputs or outputs | Programmed individually as inputs or outputs | Programmed individually as inputs or outputs |

## SUBROUTINE PROGRAMMING TECHNIQUES

For programming subroutines, three main techniques are used with the CDP1802 in COSMAC systems :

SEP - Set program counter
MARK - Save old contents of X, P
SCRT - Standard call and return technique

1. SEP technique uses the advantages of having several registers, to initialize some of them, and change the P pointer via a SEP instruction to call subroutines. The example SEP technique shows how it is used. After reset, register Ø is automatically program counter. Register 7 and register 8 are initialized to 0020 and 0030 and then with SEP 7 the PC 0 is changed to PC 7. As after an instruction the PC is always incremented, R Ø is pointing now to SEP 8.

   As the P designator is changed to 7 and register 7 has been loaded with 0020, the code at 0020 is the next one to be executed. This is a SEQ and sets the Q line high. The last instruction of this "subroutine" is a branch before the first instruction of the subroutine.

   As this instruction is SEP Ø, the subroutine modifies the P pointer and R Ø is PC again. The SEP 8 executes a jump to a subroutine at 0030 using R8 as program counter. After coming back to R Ø with SEP Ø a BR LOOP closes the program loop. With this technique, no RAM is needed for subroutine calls and it is very fast (one instruction). This technique is used for example in the monitor program UT4. It means, there is no need to have RAM at certain locations (e.g. for stack).

   The disadvantage of this technique is that subroutines may call "deeper" subroutines, there is no free nesting possibility.

2. MARK technique overcomes this problem, but now RAM is needed. Now nesting is allowed to any depth. The MARK technique does the following :

   MARK (X,P) → T; (X,P) → M(R(2))
   then P → X; R(2)-1

   This means in words, that the current contents of the X,P pointers (2 x 4 bit) are stored where R2 (stack pointer) points to and in the temporary register T (for a later save if needed).

   After that X is set to P and the stack pointer is decremented to point again to a free location.

   The use of mark technique also provides another benefit . The calling program may pass data (parameters) to the subroutine via "in-line" data lists. An in-line data list is a block of immediate constant data supplied by the calling program to the subroutine.

   The MARK technique works as follows :

   1. The calling program executes a MARK instruction. This instruction pushes the values of X and P onto the stack pointed to by R(2) and then copies P into X. The current value of X is now the same as the old value of P. P remains the same.

   2. The calling program calls the subroutine via a SEP instruction to a register that points to the subroutine. The execution of the subroutine then begins.

   3. If the calling program has provided inline parameters to the subroutine, then the subroutine picks up these parameters by executing LDXA instructions. This procedure will load the next parameter to the D register and automatically advance the caller's program counter (R(X)) to the next byte. There should be as many LDXA

instructions as there are inline data bytes. Thus, the passing of data is accomplished without the subroutine knowing the old program counter (X designator).

4. After the subroutine performs its task, it returns to the caller by setting X to 2, incrementing R(2) (preparing R(X) for a return), and executing a RET instruction. These instructions will load the contents of memory byte pointed to by R(2) to the X and P registers. Processing continues with the calling program running with its original X and P.

5. Upon return of control from the subroutine to the calling program, the calling program must decrement R(2) to compensate for the increment in the RET instruction.

3. STANDARD CALL AND RETURN technique is the most advanced technique described here. It has several advantages over the preceeding techniques, they are :

1. There is unlimited subroutine nesting capability.
2. There is no confusion over PC assignments.
3. Parameter passing to subroutines is well defined.

SCRT is not without disadvantages, however, they are :

1. Additional time for call and return.
2. It reserves three registers for linkage.

The register assignment for SCRT is as follows :

R2 Stack pointer
R3 Program counter
R4 Call routine PC
R5 Return routine PC
R6 Register contains return location, is used for parameter passing. ("LINK")

The CALL is done using a small subroutine running in R4 :

1. It saves the current contents of R6 in RAM on stack (R2).

2. Copies the current contents of R3 in R6.

3. As two "parameters" the address of the subroutine is loaded and transferred into R3.

4. The last instruction executes a SEP 3 and the subroutine starts in R3.

5. Parameter passing is possible as the address of the calling routine is still in R6.

The RETURN is another short subroutine, running in R5 :

1. Register 6 is put back in R3, as it is the address of the calling routine.

2. X is set to 2 to call back the "old" contents of R6 from stack.

3. An INC is necessary as R2 was pointing to a free location "below" the two R6 bytes.

4. SEP 3 exits to R3 and the calling program is in function again.

All these three programming techniques
SEP
MARK
SCRT
can be used together and combined as needed as they are fully compatible

For further information, see MPM201, page 54.

```
0000 ;                  0001
0000 ;                  0002            ..EXAMPLE PROGRAM
0000 ;                  0003            ..SEP TECHNIQUE
0000 ;                  0004
0000 ;                  0005            ..USES
0000 ;                  0006            ..R0 AS MAIN PC
0000 ;                  0007            ..R7 AS SUB1 PC
0000 ;                  0008            ..R8 AS SUB2 PC
0000 ;                  0009
0000 ;                  0010            ORG #0000
0000 F800B7;            0011 INIT:      LDI #00;PHI R7          ..INIT PROGR
0003 F820A7;            0012            LDI #20;PLO R7          ..PTR FOR SUB1
0006 F800B8;            0013            LDI #00;PHI R8          ..AND PROGRAM
0009 F830A8;            0014            LDI #30;PLO R8          ..PTR FOR SUB2
000C ;                  0015
000C ;                  0016            ..MAINPROGRAM
000C ;                  0017
000C D7;                0018 LOOP:      SEP R7                  ..CALL SEQ
000D ;                  0019                                    ..SUBROUTINE
000D D8;                0020            SEP R8                  ..CALL REQ
000E ;                  0021                                    ..SUBROUTINE
000E 300C;              0022            BR LOOP                 ..BACK TO LOOP
0010 ;                  0023
0010 ;                  0024
0010 ;                  0025            ..SUBROUTINE 1
0010 ;                  0026
0010 ;                  0027            ORG #001F
001F D0;                0028 BACK1:     SEP R0                  ..BACK TO MAIN
0020 7B;                0029 ENTRY1:    SEQ                     ..SET Q HI
0021 F880B9;            0030            LDI #80;PHI R9          ..LOAD CTR REG
0024 29;                0031 TIMER1:    DEC R9                  ..DECR IT
0025 99;                0032            GHI R9                  ..GET IT IN D
0026 3A24;              0033            BNZ TIMER1              ..BRANCH IF NOT
0028 ;                  0034                                    ..YET #00
0028 301F;              0035            BR BACK1                ..OTHERWISE EXIT
002A ;                  0036
002A ;                  0037
002A ;                  0038            ..SUBROUTINE 2
002A ;                  0039
002A ;                  0040            ORG #002F
002F D0;                0041 BACK2:     SEP R0                  ..BACK TO MAIN
0030 7A;                0042 ENTRY2:    REQ                     ..RESET Q
0031 F8FFB9;            0043            LDI #FF;PHI R9          ..LOAD CTR REG
0034 29;                0044 TIMER2:    DEC R9                  ..DECR IT
0035 99;                0045            GHI R9                  ..GET IT IN D
0036 3A34;              0046            BNZ TIMER2              ..IF NOT ZERO JUMP
0038 ;                  0047                                    ..TO TIMER2
0038 302F;              0048            BR BACK2                ..ELSE EXIT
003A ;                  0049            END
0000
```

```
0000 ;              0001          ..EXAMPLE PROGRAM
0000 ;              0002          ..MARK TECHNIQUE
0000 ;              0003
0000 ;              0004          ..R0    PC AT#0000
0000 ;              0005          ..R2    STACKPOINTER
0000 ;              0006          ..R3    MAIN PC
0000 ;              0007          ..R7    SUB1 PC
0000 ;              0008          ..R8    SUB2 PC
0000 ;              0009
0000 ;              0010          ORG #0000
0000 F84FB2;        0011 INIT:    LDI #4F;PHI R2          ..INIT STACKPTR
0003 F8FFA2;        0012          LDI #FF;PLO R2
0006 F800B3;        0013          LDI #00;PHI R3          ..MAIN PC
0009 F830A3;        0014          LDI #30;PLO R3
000C F800B7;        0015          LDI #00;PHI R7          ..SUB1 PC
000F F846A7;        0016          LDI #46;PLO R7
0012 F800B8;        0017          LDI #00;PHI R8          ..SUB2 PC
0015 F850A8;        0018          LDI #50;PLO R8
0018 E2;            0019          SEX R2                  ..FOR MARK
0019 D3;            0020          SEP R3                  ..GO TO MAIN
001A ;              0021
001A ;              0022          ..MAIN PROGRAM
001A ;              0023
001A ;              0024          ORG #0030
0030 C4;            0025 MAIN:    NOP                     ..ANY CODE OF
0031 C4;            0026          NOP                     ..MAINPROGRAM
0032 79;            0027          MARK                    ..SAVE X,P ON STACK
0033 D7;            0028          SEP R7                  ..1:ST LEVEL SUB
0034 22;            0029          DEC R2                  ..COMPENSATE FOR
0035 ;              0030                                  ..INCR OF RET-INSTR
0035 3030;          0031          BR MAIN                 ..BACK TO MAIN
0037 ;              0032
0037 ;              0033          ..SUBROUTINE 1
0037 ;              0034
0037 ;              0035          ORG #003F
003F 70;            0036 EXIT1:   RET                     ..EXIT TO CALLER
0040 78;            0037 SUB1:    SEQ                     ..SET Q HI
0041 C4;            0038          NOP                     ..ANY OTHER CODE
0042 C4;            0039          NOP                     ..OF THIS PROGR
0043 79;            0040          MARK                    ..X,P  SUB1 ON STACK
0044 D8;            0041          SEP R8                  ..2:ND LEVEL SUB
0045 24;            0042          ,#24                    ..PAR FOR SUB2
0046 22;            0043          DEC R2                  ..COMPENSATE FOR
0047 ;              0044                                  ..INCR OF RET-INSTR
0047 E2;            0045          SEX R2                  ..MAKE SURE X=2
0048 12;            0046          INC R2                  ..PREPARE FOR RET
0049 303F;          0047          BR EXIT1                ..BRANCH TO EXIT1
004B ;              0048
004B ;              0049          ..SUBROUTINE 2
004B ;              0050
004B ;              0051          ORG #004F

004F 70;            0052 EXIT2:   RET                     ..EXIT TO CALLER
0050 7A;            0053 SUB2:    REQ                     .. RESET Q
0051 C4;            0054          NOP                     ..ANY CODE
0052 72;            0055          LDXA                    ..GET PAR FROM SUB1
0053 E2;            0056          SEX R2                  ..MAKE SURE X=2
0054 12;            0057          INC R2                  ..PREPARE FOR RET
0055 304F;          0058          BR EXIT2                ..BRANCH TO EXIT2
0057 ;              0059          END
0000
```

```
0000 ;                  0001          ..STANDARD CALL AND RETURN
0000 ;                  0002          ..SUBROUTINES
0000 ;                  0003          ..THEY USE SOME REGISTERS
0000 ;                  0004
0000 ;                  0005          ..R2 STACKPOINTER
0000 ;                  0006          ..R3 MAIN PC
0000 ;                  0007          ..R4 CALL PC
0000 ;                  0008          ..R5 RETURN PC
0000 ;                  0009          ..R6 LINK PC
0000 ;                  0010          ..ARE INITIALIZED
0000 ;                  0011          ..VIA THE MAIN PROGRAM
0000 ;                  0012          ..
0000 ;                  0013
0000 ;                  0014
0000 ;                  0015          .. CALL SUBROUTINE
0000 ;                  0016
0000 ;                  0017          ORG #81E3
81E3 D3;                0018 EXITA:   SEP R3          ..EXECUTE SUBROUTINE
81E4 E2;                0019 CALL:    SEX R2          ..MAKE SURE X=2
81E5 9673;              0020          GHI R6;STXD     ..STORE LINKPOINTER
81E7 8673;              0021          GLO R6;STXD     ..ON STACK
81E9 93B6;              0022          GHI R3;PHI R6   ..FREE R3 AND
81EB 83A6;              0023          GLO R3;PLO R6   ..R6 AS PARAMETER PTR
81ED 46B3;              0024          LDA R6;PHI R3   ..GET SUBROUTINE ADDRESS
81EF 46A3;              0025          LDA*R6;PLO R3   ..THAT HAS TO BE CALLED
81F1 30E3;              0026          BR EXITA        ..BRANCH TO EXITA
81F3 ;                  0027
81F3 ;                  0028
81F3 ;                  0029          .. RETURN SUBROUTINE
81F3 ;                  0030
81F3 ;                  0031          ORG #81F3
81F3 D3;                0032 EXITB:   SEP R3          ..RETURN FROM SUBROUTINE
81F4 96B3;              0033 RETURN:  GHI R6;PHI R3   ..LOAD PARAMETER POINTER
81F6 86A3;              0034          GLO R6;PLO R3   ..BACK IN MAIN PC
81F8 E2;                0035          SEX R2          ..MAKE SURE X=2
81F9 12;                0036          INC R2          ..GO TO STACKED DATA
81FA 72A6;              0037          LDXA;PLO R6     ..AND GET R6 BACK
81FC F0B6;              0038          LDX;PHI R6      ..IN BEFORE FREED R6
81FE 30F3;              0039          BR EXITB        ..AND BRANCH TO EXITB
8200 ;                  0040          END
0000
```

```
0000 ;                  0001
0000 ;                  0002          ..EXAMPLE PROGRAM
0000 ;                  0003          ..STANDARD CALL AND
0000 ;                  0004          ..RETURN ROUTINE TECHNIQUE
0000 ;                  0005          ..
0000 ;                  0006          ..USES R2,R3,R4,R5
0000 ;                  0007
0000 ;                  0008          ORG #0000
0000 F84FB2;            0009 INIT:    LDI #4F;PHI R2  ..INITIALIZE
0003 F8FFA2;            0010          LDI #FF;PLO R2  ..STACKPOINTER
0006 ;                  0011          ..
0006 F881B4;            0012          LDI #81;PHI R4  ..CALL PROGRAM CTR
0009 F8E4A4;            0013          LDI #E4;PLO R4  ..
000C ;                  0014          ..
000C F881B5;            0015          LDI #81;PHI R5  ..RETURN PROGRAM CTR
000F F8F4A5;            0016          LDI #F4;PLO R5  ..
0012 ;                  0017          ..
0012 F800B3;            0018          LDI #00;PHI R3  ..MAIN PROGRAM COUNTER
0015 F830A3;            0019          LDI #30;PLO R3  ..
0018 ;                  0020          ..
0018 E2;                0021          SEX R2          ..REGISTER 2 STACKPOINTER
0019 D3;                0022          SEP R3          ..GO TO MAIN PROGRAM
001A ;                  0023
001A ;                  0024          .. MAIN PROGRAM
001A ;                  0025
001A ;                  0026          ORG #0030
0030 D4;                0027 MAIN:    SEP R4          ..GO TO CALL ROUTINE
0031 0040;              0028          ,#0040          ..AND GIVE SUBROUTINE
0033 ;                  0029                          ..ADDRESS WITH IT
0033 D4;                0030          SEP R4          ..NOW THE SAME
0034 0050;              0031          ,#0050          ..WITH SUB2
0036 3030;              0032          BR MAIN         ..BRANCH BACK TO MAIN
0038 ;                  0033
0038 ;                  0034          .. SUBROUTINE 1
0038 ;                  0035
0038 ;                  0036          ORG #0040
0040 7B;                0037 SUB1:    SEQ             ..SET Q
0041 D5;                0038          SEP R5          ..AND GO TO RETURN ROUTINE
0042 ;                  0039
0042 ;                  0040          .. SUBROUTINE 2
0042 ;                  0041
0042 ;                  0042          ORG #0050
0050 7A;                0043 SUB2:    REQ             ..RESET Q
0051 D4;                0044          SEP R4          ..CALL SUB1 ROUTINE
0052 0040;              0045          ,#0040
0054 D5;                0046          SEP R5          ..AND GO TO RETURN ROUTINE
0055 ;                  0047          END
0000
```

## INTERPRETIVE PROGRAMMING

The manual MPM-201B describes (page 66) a simple interpretive technique which requires that all routines begin and end in the same page of memory. This page restriction can be a limitation in complex programs, where a more general handler operating over the full memory range is more convenient. The handler shown here uses one of the CPU registers (called here "IPC") as its program counter. Registers "PC" (machine-code program-counter) and "PPC" (pseudo program-counter) are as defined in the annual.

```
IEXEC   :   SEP  PC              .. pass control to routine
IFETCH  :   LDA  PPC
            PHI  PC              .. initialize PC.1
            LDA  PPC
            PLO  PC              .. initialize PC Ø
            BR   IEXEC           .. reset IPC
```

Register IPC must be initialized to the address IFETCH, then a "SEP IPC" passes control to the handler, which operates analogously to the fetch cycle at machine-code level. When the pseudo-instruction has been fetched, control is passed to it for the "execute" portion of the interpretive cycle (execute cycle).

The last instruction in each routine is a "SEP IPC", which passes control back to the handler for fetching of the next pseudo instruction.

This simple technique effectively allows the programmer to build a "virtual machine" with an architecture tailored to his application. Consistent use of this philosophy can lead to significant benefits in program documentation, in ease of modification,and speed of development.

The handler does, of course, introduce an execution-time overhead, but it should be noted that this is small (19.2 uS at 5MHz) - and is less than one quarter of the standard 1802 subroutine overhead. It gives a saving of memory space as well, since a subroutine call takes three bytes (the "call" instruction plus the 16-bit address) while a pseudo-instruction needs only two bytes (the address itself).

References : 1802 User Manual, MPM-201B, "Programming Techniques", page 51.

## MACRO FOR BCD TO BINARY CONVERSION

In typical Microprocessor systems numerical inputs are often presented in BCD,but manipulating such data can be done more efficiently if they are converted to binary form. The following short routine for the 1802 performs a 2-digit BCD input from a port and converts the data to binary leaving the result both in D and on the stack. It is presented as a macro definition which functions in exactly the same way as the standard "INP" instruction except that one of the CPU registers ("WKG") is used for temporary storage.

```
MACRO
BCDINP %PORT                    .. macro name
INP %PORT                       .. input to D and M (R(X))
ANI  ØF;PLO WKG                 .. save low digit
LDX;ANI  FØ;SHR                 .. hi digit x 8
STXD;SHR;SHR                    .. hi digit x 2
IRX;ADD;STXD                    .. hi digit x 10
GLO WKG;IRX;ADD                 .. binary in D
STXD;IRX                        .. and on stack
MEND
                                .. WKG = working RAM
                                .. PORT = input port
```

The overhead of this conversion is 18 bytes of memory plus 51.2 uS (at 5MHz clock) execution time. If the value of X is known in advance the overhead can be reduced to 15 bytes and 41.6 uS by replacing the "STXD , IRX" pairs with single "STR" instructions.

If the program contains many of these conversion-inputs, the routine should be rewritten as a "SEP"-type subroutine to save ROM space. In this case the memory overhead will be only one byte per call (the SEP instruction itself). This is a good illustration of the memory economy obtainable by use of the 1802's register-switching feature.

## DETAILED PROGRAM EXAMPLE

### Showing a Traffic Light

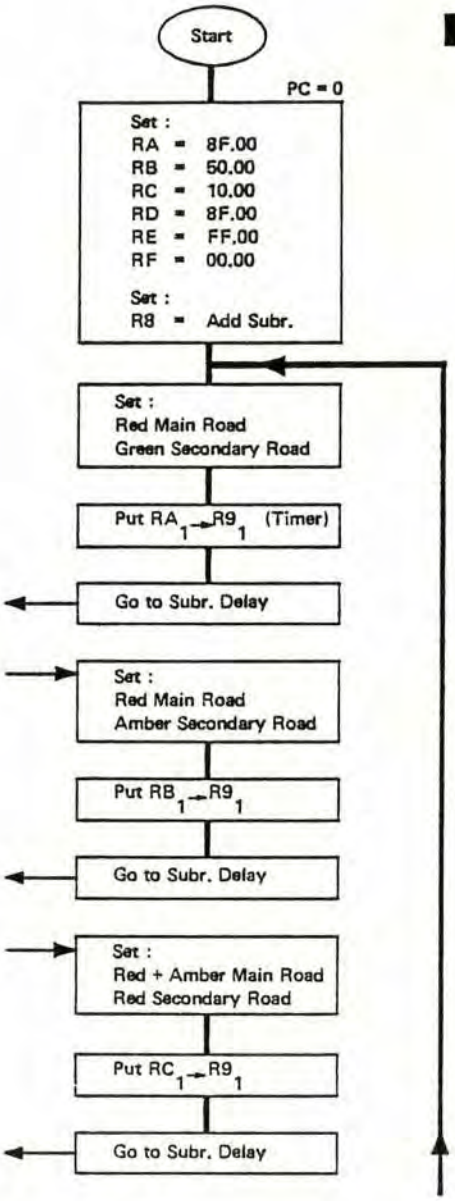
This program shows how to use part of the instruction set. It uses a CDP1852 8-bit output port to switch LED's ON and and OFF. Two EF flag lines give priority to either main or secondary road, and the Q line indicates in which state the program is :

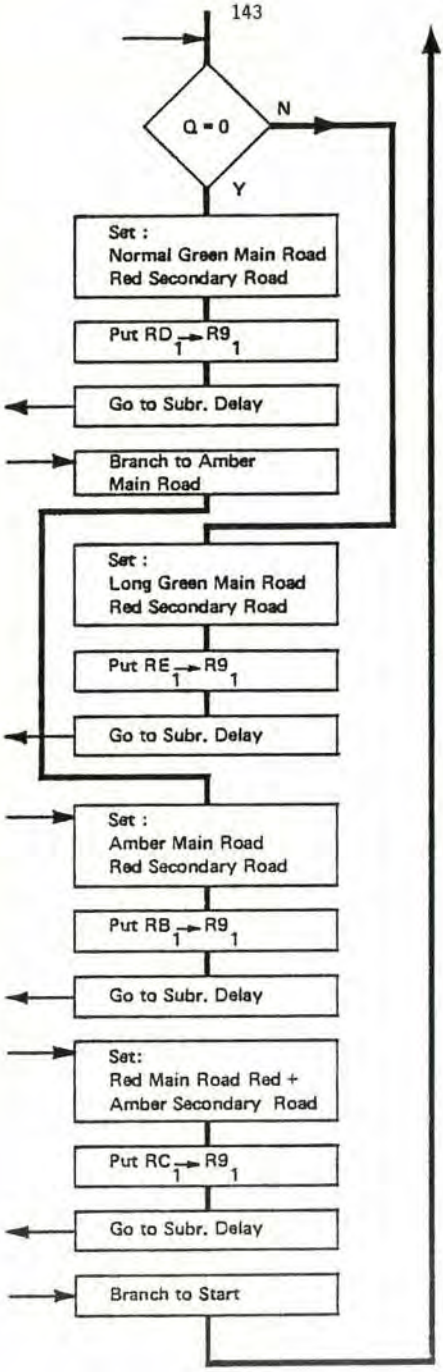
a) Normal
b) Advantage (longer green) main road

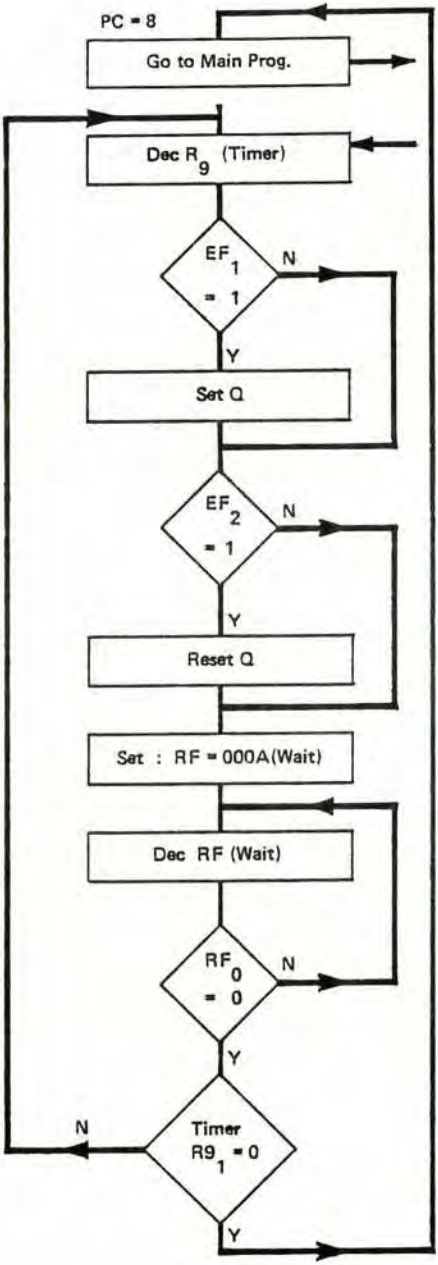
The instructions used in this program are :

```
LDI             load immediate next byte in D register
PHI RN          transfer from D to high byte of RN
PLO RN          transfer from D to low byte of RN
GHI RN          get high byte of RN and put it in D
GLO RN          get low byte of RN and put it in D
SEX RN          set X pointer to RN (for output)
OUT N           output from X where RC points to OUT N
SEP N           set P pointer to N (subroutine call)
DEC N           decrement register N
SEQ             set the Q output high
REQ             set the Q output low
BR  BEGIN       branch immediate
BQ  LGREEN      branch if Q is set
BN1 TEST2       branch if EF1 is not active
BN2 TIMER       branch if EF2 is not active
BNZ LOOP        branch if D register is not 00
```

Depending on the state of the Q flip-flop (LED) the program flow is changed between normal green and long green. This program is not optimized; its intention is to show subroutine calls, software timers use of flag lines, output to external devices. Here one special output -the output immediate- is used. As in this program X and P are the same, the byte just following the output instruction is transferred to the CDP1852.


Start
PC = 0
Set :
RA = 8F.00
RB = 50.00
RC = 10.00
RD = 8F.00
RE = FF.00
RF = 00.00
Set :
R8 = Add Subr.
Set :
Red Main Road
Green Secondary Road
Put RA1→R91 (Timer)
Go to Subr. Delay
Set :
Red Main Road
Amber Secondary Road
Put RB1→R91
Go to Subr. Delay
Set :
Red + Amber Main Road
Red Secondary Road
Put RC1→R91
Go to Subr. Delay

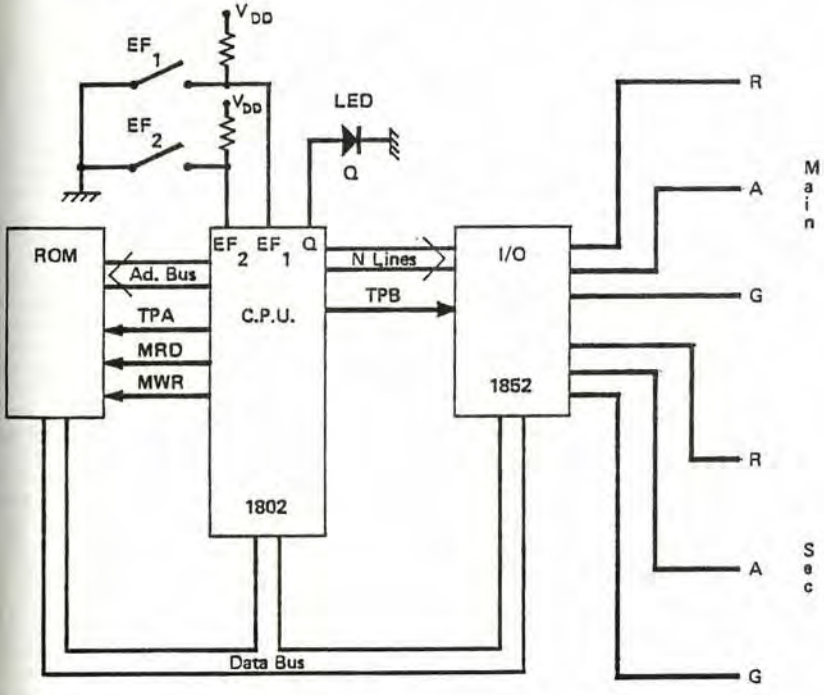

Q = 0
N
Y
Set :
Normal Green Main Road
Red Secondary Road
Put RD1 → R91
Go to Subr. Delay
Branch to Amber
Main Road
Set :
Long Green Main Road
Red Secondary Road
Put RE1 → R91
Go to Subr. Delay
Set :
Amber Main Road
Red Secondary Road
Put RB1 → R91
Go to Subr. Delay
Set:
Red Main Road Red +
Amber Secondary Road
Put RC1 → R91
Go to Subr. Delay
Branch to Start


PC = 8
Go to Main Prog.
Dec R9 (Timer)
EF1 = 1
N
Y
Set Q
EF2 = 1
N
Y
Reset Q
Set : RF = 000A(Wait)
Dec RF (Wait)
RF0 = 0
N
Y
Timer R91 = 0
N
Y

# Traffic Lights Diagram

RCA Solid State

**Traffic Detectors**



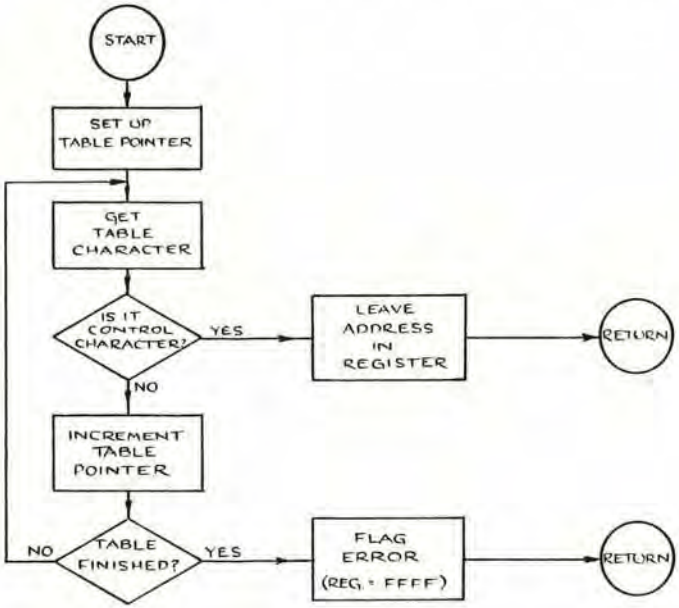
```
0000 ;              0001          ..  EXAMPLE PROGRAM
0000 ;              0002          ..  TRAFFIC LIGHT
0000 ;              0003
0000 ;              0004          ..EF1  USED AS SENSOR MAID ROAD
0000 ;              0005          ..EF2  USED AS SENSOR SECONDARY
0000 ;              0006          ..Q    TO STORE THE STATE AND SHOW IT
0000 ;              0007
0000 ;              0008          ORG #0000              ..AFTER RESET R0 IS PC
0000 F88FBA;        0009 INIT:    LDI #8F;PHI RA         ..LOAD DELAY1 IN RA
0003 F800AA;        0010          LDI #00;PLO RA
0006 ;              0011
0006 F850BB;        0012          LDI #50;PHI RB         ..LOAD DELAY2 IN RB
0009 F800AB;        0013          LDI #00;PLO RB
000C ;              0014
000C F840BC;        0015          LDI #40;PHI RC         ..LOAD DELAY3 IN RC
000F F800AC;        0016          LDI #00;PLO RC
0012 ;              0017
0012 F88FBD;        0018          LDI #8F;PHI RD         ..LOAD DELAY4 IN RD
0015 F800AD;        0019          LDI #00;PLO RD
0018 ;              0020
0018 F8FFBE;        0021          LDI #FF;PHI RE         ..LOAD DELAY 5 IN RE
001B F800AE;        0022          LDI #00;PLO RE
001E ;              0023
001E F800BF;        0024          LDI #00;PHI RF         ..LOAD DELAY6 IN RF
0021 AF;            0025          PLO RF
0022 ;              0026
0022 F800B8;        0027          LDI A.1(SUBRJ);PHI R8  ..SUBROUTINE ADDRESS IN R8
0025 F855A8;        0028          LDI A.0(SUBRJ);PLO R8
0028 ;              0029
0028 E0;            0030 BEGIN:   SEX R0                 ..SET X TO IMMEDIATE MODE
0029 ;              0031
0029 6514;          0032 BYTE1:   OUT 5,#14              ..OUTPUT FIRST BYTE
002B ;              0033
002B 9A;            0034          GHI RA                 ..LOAD TIMER
002C B9;            0035          PHI R9                 ..IN REG 9.1
002D D8;            0036          SEP R8                 ..AND GO TO TIMER
002E ;              0037
002E 6512;          0038 BTYE2:   OUT 5,#12              ..OUTPUT SECOND BYTE
0030 ;              0039
0030 98;            0040          GHI RB                 ..LOAD VALUE 2
0031 B9;            0041          PHI R9                 ..IN REG 9.1
0032 D8;            0042          SEP R8                 ..AND EXECUTE TIMER
0033 ;              0043
0033 6531;          0044 BYTE3:   OUT 5,#31              ..OUTPUT THIRD BYTE
0035 ;              0045
0035 9C;            0046          GHI RC                 ..LOAD TIMER VALUE 3
0036 B9;            0047          PHI R9                 ..IN REG 9.1
0037 D8;            0048          SEP R8                 ..AND GO TO SUB
0038 ;              0049
0038 3141;          0050          BQ LGREEN              ..DECIDE IF LONGGREEN
003A ;              0051                                 ..BRANCH IF Q SET
```

```
003A 6541;          0052 NGREEN: OUT 5,#41          ..OUTPUT FOURTH BYTE (N)
003C ;              0053
003C 9D;            0054         GHI RD             ..TIMER VALUE 48
003D B9;            0055         PHI R9             ..IN R 9.1
003E D8;            0056         SEP R8             ..AND CALL SUB
003F ;              0057
003F 3046;          0058         BR AMBER           ..NOW BRANCH TO AMBER
0041 ;              0059
0041 6541;          0060 LGREEN: OUT 5,#41          ..OUTPUT FOURTH BYTE (L)
0043 ;              0061
0043 9E;            0062         GHI RE             ..LOAD VALUE 48
0044 B9;            0063         PHI R9             ..IN R9.1
0045 D8;            0064         SEP R8             ..AND GOTO SUB
0046 ;              0065
0046 6521;          0066 AMBER:  OUT 5,#21          ..OUTPUT FIFTH BYTE
0048 ;              0067
0048 9B;            0068         GHI RB             ..LOAD TIMER VALUE 5
0049 B9;            0069         PHI R9             ..IN R9.1
004A D8;            0070         SEP R8             ..CALL SUB
004B ;              0071
004B 650D;          0072 BYTE6:  OUT 5,13           ..OUTPUT SIXTH BYTE
004D ;              0073
004D 9C;            0074         GHI RC             ..LOAD TIMER VALUE 6
004E B9;            0075         PHI R9             ..IN R9.1
004F D8;            0076         SEP R8             ..CALL TIMER SUB
0050 ;              0077
0050 3028;          0078         BR BEGIN           ..START AGAIN THE PROGRAM
0052 ;              0079
0052 ;              0080         ORG SUBR-1         ..TIMER SUBROUTINE
0054 D0;            0081 RETURN: SEP R0             ..GO BACK TO CALLING PROGRAM
0055 ;              0082
0055 29;            0083 SUBR:   DEC R9             ..COUNT DOWN TIMER
0056 ;              0084
0056 3C59;          0085 TEST1:  BN1 TEST2          ..IF EF1 NOT SET GO ON
0058 7B;            0086         SEQ                ..ELSE SET Q (LONG GREEN)
0059 ;              0087
0059 3D5C;          0088 TEST2:  BN2 TIMER          ..IF EF2 NOT SET GO ON
005B 7A;            0089         REQ                ..ELSE RESET Q (NORMAL GREEN)
005C ;              0090
005C F80A;          0091 TIMER:  LDI #0A            ..LOAD 0A IN
005E AF;            0092         PLO RF             ..RF.0
005F 2F;            0093 LOOP:   DEC RF             ..AND COUNT DOWN
0060 8F;            0094         GLO RF             ..GET IT IN D AND
0061 3A5F;          0095         BNZ LOOP           ..IF NOT 00 GO BACK
0063 ;              0096
0063 99;            0097         GHI R9             ..ELSE LOOK AT R9.1
0064 3A55;          0098         BNZ SUBR           ..AND IF NOT 00 SUBR AGAIN
0066 ;              0099
0066 C00054;        0100         LBR RETURN         ..IF R9.1 IS 00
0069 ;              0101                            ..BACK TO CALLING PROGRAM
0069 ;              0102         END
```

## CHARACTER SEARCH ROUTINE

Another software example is this search routine.

A table pointer is set up and the contents of this memory location is compared with a byte on the parameter stack using the XOR instruction. The new character is loaded via PNTR in D-register and XORed with the byte on the parameter stack, therefore, the SEX PARSTC is necessary. If it was the searched character, branch to FINISH, otherwise compare with the end of the table. If not yet reached, go to NWCHAR, ERROR means that the end of the table has been found but not the character, so the high byte of PNTR is modified, then the register PNTR is decremented followed by a return to the calling program. This program is written for SCRT,as the return is at the end of the program and the PC for this subroutine has to be initialized, before it can be called.



```
          .. CHARACTER SEARCH ROUTINE
          .. SEARCHES LIST OF SIZE 'LENGTH'
          .. AT LOCATION 'TABLE' FOR CONTROL
          .. CHARACTER FOUND ON STACK

START:    A.1 (TABLE)-->PNTR.1
          A.0 (TABLE)-->PNTR.0      .. INITIALIZE
          SEX PARSTK                .. USE PARAMETER STACK
NWCHAR:   @ PNTRI.XOR.@             .. GET AND COMPARE
          BZ FINISH                 .. END IF MATCH
          PNTR.0. XOR. A (TABLE + LENGTH)  .. TABLE OVER?
          BNZ NWCHAR                .. IF NOT, BRANCH
ERROR:    PHI PNTR                  .. IF SO, FLAG ERROR
FINISH:   DEC PNTR                  .. RESET PNTR
          SEP RETURN                .. GO BACK
          END
```

## MEMORY MOVE ROUTINE

This type of routine could be, for example, used as PROM programmer together with a monostable generator to program different types of EPROMS, e.g. 18U42, 27U58. The monostable streches program execution using the WAIT line (see PROM programmer).

Here it is used to transfer a program from PROM to RAM, load the start address and START the program.

The program starts with PC = Ø and as the program to be executed runs in RØ as well, another register has to be PC for the transfer program. This is done by the first part of the program.

The transfer program runs in R9. Source and destination pointer are loaded and the transfer is done. Then the PROG PC is loaded with the start address and the SEP PROG PC starts the transferred program.

```
M
0000 ;          0001          ..JP RCA BXL
0000 ;          0002          ..
0000 ;          0003          ..MEMORY MOVE PROGRAM
0000 ;          0004
0000 ;          0005          ..
0000 ;          0006          PROGAD=#8800        ..STARTADD OF MOVE PROGRAM
0000 ;          0007          SOURCE=#9000        ..STARTADD OF SOURCE PROGRAM
0000 ;          0008          DESTIN=#0000        ..STARTADD OF DESTINATION
0000 ;          0009          MAXADD=#97FF        ..HIGHEST ADD OF SOURCE PROGRAM
0000 ;          0010          START=#0001         ..STARTADDRESS OF PROGRAM
0000 ;          0011          BEG=PROGAD+7        ..BEGIN OF MOVE PROGRAM
0000 ;          0012          PROGPC=9            ..PROGRAMCOUNTER REGISTER
0000 ;          0013          ..
0000 ;          0014          ORG PROGAD
8800 ;          0015          ..
8800 F888;      0016          LDI A.1(BEG)        ..LOAD R 9 WITH START ADD OF MOVE
8802 B9;        0017          PHI 9
8803 F807;      0018          LDI A.0(BEG)
8805 A9;        0019          PLO 9
8806 D9;        0020          SEP 9               ..SET R 9  AS PROGRAM COUNTER
8807 ;          0021          ..
8807 F896;      0022 BEG:     LDI A.1(SOURCE)     ..LOAD R A  WITH SOURCE
8809 BA;        0023          PHI A
880A F800;      0024          LDI A.0(SOURCE)
880C AA;        0025          PLO A
880D F800;      0026          LDI A.1(DESTIN)     ..LOAD R B  WITH DESTINATION
880F BB;        0027          PHI B
8810 F800;      0028          LDI A.0(DESTIN)
8812 AB;        0029          PLO B
8813 4A;        0030 REPEAT:  LDA A               ..PICK UP A BYTE AND INC R A
8814 5B;        0031          STR B               ..STORE
8815 1B;        0032          INC B
8816 9A;        0033          GHI A
8817 FB98;      0034          XRI A.1(MAXADD)+1   ..END OF TRANSFER ?
8819 3A13;      0035          BNZ REPEAT
881B F800;      0036          LDI A.1(START)      ..LOAD PC FOR MOVED PROGRAM
881D B0;        0037          PHI PROGPC
881E F801;      0038          LDI A.0(START)
8820 A0;        0039          PLO PROGPC
8821 D0;        0040          SEP PROGPC          ..START THE TRANSFERRED PROGRAM
8822 ;          0041          END
0000
```

# DATA BUS CONTENTION DURING CDP1802 REGISTER-TO-REGISTER OPERATIONS

In 1802 based systems using various ROM's (CDP1832, 1834) or EPROM's (2708, 2758, 2716) bus contention problems have been found to occur during internal data transfer operations (GHI, PHI, GLO, PLO). As a result, data is lost in one or more registers.

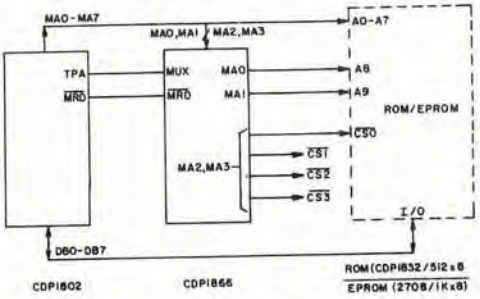
The 1802 generates a valid 16-bit address and a TPA signal during internal operations. If the chip-select signal for the ROM or EPROM is only controlled by higher order address bits, then it is very probable that these memories can be selected and have their output drivers "turned on", creating a bus contention problem with the 1802 data bus drivers.

The solution to this problem is to either gate the chip-select functions with $\overline{MRD}$ externally, or find a spare input on the memories for $\overline{MRD}$. During these register operations $\overline{MRD}$ is held high. (See Table 1 on page 90 of the MPM-201B Manual). See below for specific suggestions.

A. Wiring $\overline{MRD}$ to a spare input

- o CDP1834 (CS1 or CS2) - only if they are "active low"
- o 2716, 2732, 1758 ($\overline{OE}$)

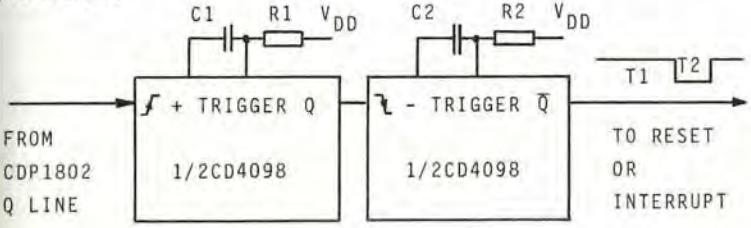
B. Gating $\overline{MRD}$ externally with the chip-select function



# WATCH DOG

By noting the absence of pulses, generated by status reporting statements inserted in a running program, this dual monostable generator recognizes if a pulse is missing and reinitializes the Microprocessor based system or forces it to an interrupt, if glitches on the busses or peripherals stop the program. With the CDP1802 the Q output flip-flop can be easily used to retrigger this monoflop.

A SEQ, REQ instruction in the program creates a pulse, that always retriggers the timer. If the program stops, the second monostable generates a reset or interrupts the processor.




DUAL MONOSTABLE FOR WATCH DOG FUNCTION

The same function could be achieved with counter timer circuits to avoid any capacitor, using clock signal of the processor system (e.g. CD4020).

# UT4 CMOS ROM MONITOR PROGRAM

A CMOS ROM monitor program is necessary for a prototyping system to have an easy possibility to load programs, verify and start execution.

One external flag and the Q output are used for this interface. For additional hardware, see Figure 1 and Figure 2 , it assumes a clock frequency of 2MHz for 100 and 300bd. This monitor program does not need any RAM.

It starts at 8000 (highest address bit high) means after reset this line has to be set high externally to start the monitor program. (See evaluation kit manual MPM203 or MPM 224, as well Microboard brochures).

After being entered it initializes itself and waits for CR or LF. Carriage return sets it to full duplex and line feed to half duplex. The answer is the UT4 prompt. This utility program can even work together with a paper tape reader or punch. For additional information, see MPM224, MPM203.

## ?M Command

To interrogate memory, the user types a command such as

```
?MF5  3
```

and terminates it with CR(carriage return). UT4 responds by printing out the contents of memory beginning at location 00F5: three bytes are printed out as two hex digits each. Each line of output begins with the address, and data is grouped in 2-byte (4-digit) blocks. When necessary, new lines are begun every 16 bytes, with the previous lines ending in semicolons. The user may enter any number of digits to specify the beginning location (leading zeroes are implied, if necessary). If more than four digits are entered, only the last four are used. The number of bytes to be typed out should be in hex.

Again, if more than four digits are entered, only the last four are used. This feature allows the user to correct a mistake. He simply keeps typing, putting in the correct 4-digit values (230024 is effectively 0024).

!M Command

In general, data is entered into memory by means of a command such as

!M2F 434F534D4143

This command enters six bytes (two hex digits each) into memory beginning at location 2F. It is normally terminated by a CR. Once again, the starting location is determined by the last four digits entered. Data is entered into memory after each two hex digits are typed. If the user types an odd number of digits, the last digit is ignored, and the error message ('?') is typed out.

The !M command provides two options that facilitate memory loading. First, a string of data can be extended from line to line by typing in a comma just before the normal CR. (In this case the user must type CR-LF (carriage return-line feed) before he can begin a new line). For example :

!M23 56789ABC,(CR) (LF)

DEF0123456,(CR) (LF)

3047 (CR)

enters 11 successive bytes beginning at location 0023. Between successive hex pairs while data is being entered, any non-hex character except the comma (and semicolon, as will be discussed) is ignored. This arrangement permits arbitrary LF's, spaces (for readability), nulls (generated by the utility program or by a time-share system to give the carriage time to return), etc...

As a second optional form of data entry, a string of input data can be terminated by a semicolon (and a CR). The utility program then expects more data to follow on the next line, but preceded by a new beginning address. The line must have the format of an !M command, but with the initial !M omitted. (The utility program ignores all non-hex characters following !M, which allows the CR, LF, and nulls to be input from the Teletype without disturbing the !M command). Note also that the semicolon feature on input allows non-contiguous memory to be loaded.

## $P Command

A third utility command is $P. For example

$P6C

Starts execution at location 6C with RØ as the program counter (after the user presses CR and the utility program provides a LF). The last-four-digits-in rule applies to the address typed in.

$P always begins with RØ as program counter and X = 0. This arrangement is consistent with the fact that P = 0 and X = 0 after the CPU is RESET. Refer to the CDP1802 data sheet for other actions of RESET.

## Summary of command usage

In summary, after receiving the prompt character, '*' the user may type

?M (address) Δ (count) CR

!M (non-hex) (address) Δ (data) (optional, or ;) CR

(Where the data may have non-hex digits between each hex pair)

or

$P (address) CR

UT4 ignores initial characters until it detects ?, !, or $. Then, inputs which are not compatible with the above formats cause an error message.

Summary of UT4 operating instructions

A further detailed summary of these basic operating instruction is given below, repeating the information just given in a more concise form.

1. After pressing "RUN UTILITY" (start at 8000), the user should press either CR or LF: LF for half duplex, CR for full duplex. This instruction sets up the bit-serial timing and specifies echo or not.

2. UT4 will return * as a prompt.

3. Following *; UT4 ignores all characters until one of ?, $, or ! is typed in.

4. Following ?M or !M, UT4 waits for a hex character. It then assembles an address. If more than four hex digits are typed, only the last four are used. Next, a space is required. Note : Δ denotes a space.

   a. For ?M addr Δ a hex count must follow (again, only the last four digits are kept), and the command is terminated by CR.

   b. For !M addr Δ data must follow. An even number of hex digits is required. Before each hex pair arbitrary filler, except for a CR, comma, or semicolon, is allowed. CR terminates the command, unless it is immediately preceded by a comma or, as is generally the case, by a semicolon.

i. In case of comma CR the user must insert an LF for UT4 to continue to accept data. This procedure is a form of line continuation.

ii. In case of a semicolon all following characters are ignored until the CR is typed. Then the user must again provide an LF, and UT4 continues as if it had received optional filler, then a starting address, then a space, and then data.

iii. The !M command can be followed by as many continuation lines as needed, mixed between the two types if desired, and is finally terminated with a CR not preceded by a comma or semicolon.

5. Command $P must be followed by starting address (last four digits used if more than four are typed in). If no address is entered, 0 is assumed. Program execution begins at this location with RØ as program counter with X set to 0.

6. When a !M or ?M command is accepted and compeleted, UT4 types another prompt character.

7. When UT4 detects bad syntax, it types out a ? and returns the carriage. If a mistake is made when data is entered (by typing in an odd number of digits), all data will have been entered except the last hex digit. Note that the "only-last-four-digits" rule in the address field allows the user to correct an error without retyping the whole command. For example, a mistaken 234 can be corrected by continuing 2340235=0235. A bad command can be aborted by typing in any illegal character except after !M or ?M or between input hex data pairs. In these cases, the user should type any digit and then, for example, a period.

## UT4 register storage feature

UT4 provides for storing in RAM 13 1/2 of the 16 CDP1802 scratch-pad registers. A CDP1824 32-byte RAM has to be provided for this function. The RAM occupies addresses 8C00 - 8C1F. By pressing RESET followed by RUN U, registers R0 - RF are automatically stored in the CDP1824, in numerical order, most significant byte first. R0, R1, and R4.1 are altered in the process.

By using the command

?M8C00 20

The register contents which existed in the Microprocessor at the instant that RESET was pressed preceding the depression of RUN U can be examined. It should be remembered that UT4 uses registers R0, R1, R3, R4.1, R5, and RC - RF. These registers, therefore, will be modified. Should the user wish to continue program execution, he must initialize these registers, by program if necessary. A sample listing is given in Figure 1. It should be recalled that R0, R1, and R4.1 are not correct.

## The bit serial terminal interface

The serial terminal interface is an example of minimizing hardware complexity by the use of software. Further, it illustrates the increased flexibility that can be more readily achieved by software. The CPU receives serial data by sampling $\overline{EF4}$. It transmits serial data via its Q output. Details on the electrical I/O interface are given in the Application Note entitled "Data Terminal Interface Considerations for RCA Microprocessor Evaluation Kit CDP18S020.

The sample character waveform in Figure 4 helps to show what the interface software must do. Each character is framed by a START bit and one or two STOP bits. On input, this signal is tied to $\overline{EF4}$ which is sensed by UT4 at the midpoints of each of the bits. Software assembles the resultant ASCII character. On output, the character is transmitted one bit at a time through the Q output of the CDP1802. (See Figures 2 and 3).

The flexibility obtainable with software is demonstrated by the ability of the program UT4 to sample a character string and adjust its timing so as to cope with terminals of different, even non-standard, character rates. However, it should be noted that while a program is timing either input or output in this manner (i.e., by counting instruction executions), it is completely dedicated to that task and cannot be interrupted except for an occasional DMA service.

```
          RØ   R1              R4.1
?M8C00 2Ø
8C00 DØDØ 8202 2222 3333 9444 5555 6666 7777;
8C10 8888 9999 AAAA BBBB CCCC DDDD EEEE FFFF
```

Figure 1 : Sample Listing Illustrating Register Storage

This utility program can even work together with a paper tape reader or punch. For additional information see MPM 224, MPM203.

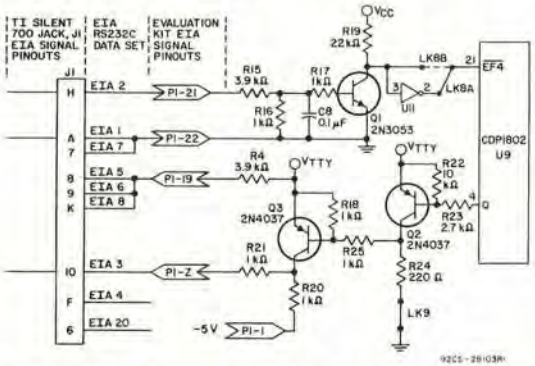



Figure 2 : The EIA RS232 Serial Data Interface For Connecting TI Silent 700 Data Terminal

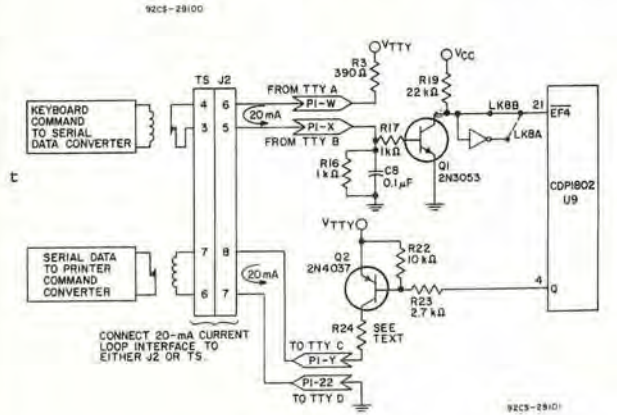



Figure 3 : The 20mA Current Loop Interface To Connect a Teletype In Full Duplex Mode

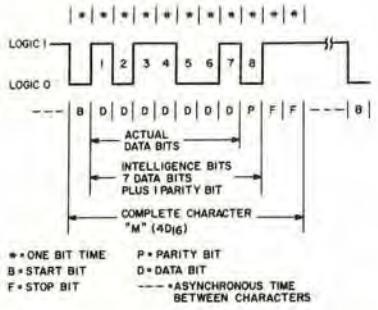



Figure 4 : Data Terminal Bit Serial Output For The Character "M"

# UTILITY PROGRAM UT4 LISTING

```
!M
0000 ;                0001              ORG #8000
8000 ;                0002        .. UT4 IS A UTILITY PROGRAM TO ALTER
8000 ;                0003        .. MEMORY, DUMP MEMORY, AND BEGIN PROGRAM
8000 ;                0004        .. EXECUTION AT A GIVEN LOCATION. THE COMMANDS
8000 ;                0005        .. ACCEPTED ARE $PHHHH (BEGIN EXECUTION AT THE
8000 ;                0006        .. SPECIFIED LOCATION WITH R0 AS PROGRAM
8000 ;                0007        .. COUNTER), !MHHHH DATA (PUT DATA AT SPECIFIED
.
8000 ;                0008        .. LOCATION), AND ?MHHHH HHHH (OUTPUT DATA
8000 ;                0009        .. FROM SPECIFIED LOCATION FOR SPECIFIC COUNT)
8000 ;                0010        .. AT THE BEGINNING OF A COMMAND ALL CHARACTERS
.
8000 ;                0011        .. ARE IGNORED UNTIL A ?,!,OR $ IS
8000 ;                0012        .. ENCOUNTERED. IN THE ?M AND !M COMMANDS NON
8000 ;                0013        .. HEX CHARACTERS ARE IGNORED AFTER M UNTIL A
8000 ;                0014        .. HEX IS READ, THEN THE FIRST NON HEX
8000 ;                0015        .. CHARACTER MUST BE A SPACE . NON HEX
8000 ;                0016        .. CHARACTERS BETWEEN HEX PAIRS OF THE DATA IN
8000 ;                0017        .. THE !M COMMAND ARE IGNORED EXCEPT FOR CR,
8000 ;                0018        .. SEMICOLON, AND COMMA.
8000 ;                0019        .. THE BAUD RATE OF UT4 IS DEPENDENT UPON THE
8000 ;                0020        .. TERMINAL BEING USED. A CR OR LF IS ENTERED
8000 ;                0021        .. AT THE BEGINNING TO SPECIFY THE APPROPRIATE
8000 ;                0022        .. DELAY BETWEEN BITS. UT4 WILL ECHO
8000 ;                0023        .. CHARACTERS IF A CR IS CHOOSEN AS THE
8000 ;                0024        .. TIMING CHARACTER. ECHOING WILL NOT TAKE
8000 ;                0025        .. PLACE IF A LF IS INPUT AS THE TIMING
8000 ;                0026        .. CHARACTER.
8000 ;                0027        .. UI4, AT INITIATION, STORES, ALL REGISTERS
8000 ;                0028        .. BETWEEN 8C00 AND 8C1F IF IT FINDS RAM THERE
8000 ;                0029        .. (BUT R0, R1, AND R4.1 ARE CLOBBERED).
8000 ;                0030           PTR=#00    ..AUXILIARY FOR MAIN ROUTINE
8000 ;                0031             CL=#01   ..CLOBBERED
8000 ;                0032             ST=#02   ..STACK POINTER-ONLY
8000 ;                0033                      ..REFERENCE TO RAM
8000 ;                0034           SUB=#03     ..SUBROUTINE PC
8000 ;                0035              PC=#05   ..MAIN PROGRAM COUNTER
8000 ;                0036         SWITCH=CL     ..DISTINGUSHES BETWEEN ?M AND !M
8000 ;                0037          DELAY=#0C    ..DELAY ROUTINE PROGRAM COUNTER
8000 ;                0038            ASL=#0D    ..HEX ASSYMBLY REG ON INPUT,
8000 ;                0039                       ..AUX FOR HEX OUTPUT
8000 ;                0040         CENTER=ASL    ..USED TO COUNT OUTPUT BYTES
8000 ;                0041           AUX=#0E     ..AUX.1 HOLDS BIT-TIME CONSTANT
8000 ;                0042          CHAR=#0F     ..CHAR.1 HOLDS I/O BYTE
8000 ;                0043        ..
8000 ;                0044        .. ENTER IN R0
8000 C4;              0045           NOP
8001 F88080;          0046           LDI A.1(UT4) ;PHI R0 ..SET PC WHILE
8004 ;                0047                                ..FINGER IS ON
```

```
8004 ;                  0048        ..
8004 ;                  0049        .. THE FOLLOWING WRITES REGISTER CONTENTS INTO
8004 ;                  0050        .. 8C00-8C1F IF IT EXISTS. 8BFE IS ASSUMED NOT
.
8004 ;                  0051        .. TO BE RAM (ELSE ROUTINE OVERRUNS).
8004 F88CB1;            0052            LDI #8C ;PHI CL    ..CL IS CLOBBERED
8007 ;                  0053                               ..BY THIS ROUTIINE
8007 F81EA1;            0054            LDI #1E ;PLO CL    ..SET UP WHERE RF.0
800A ;                  0055                               ..IS TO GO, MINUS 1
800A F8A0B4;            0056            LDI #A0 ;PHI R4    ..R4.1 STORES A MODIFIED
800D ;                  0057                               ..INSTRUCTION
800D E1;                0058            SEX CL
800E F8D051;            0059  LOOP2:    LDI #D0 ;STR CL    ..SET UP SEP INSTRUCTION
.
8011 ;                  0060                               ..FOR RETURN
8011 F3;                0061            XOR                ..CHECK THAT IT WROTE
8012 3A29;              0062            BNZ UT4            ..
8014 21;                0063            DEC CL             ..PREPARE FOR MODIFIED
8015 ;                  0064                               ..INSTRUCTION
8015 94FC70;            0065            GHI R4  ;ADI#70    ..SEE IF IT IS IN THE 90'S
.
8018 331C;              0066            BDF *+#04
801A FC21;              0067            ADI#21             ..IF NO,8N BECOMES 9N
801C FC7F;              0068            ADI#7F             ..IF YES, 9N BECOMES 8(N-1)
.
801E B451;              0069            PHI R4    ;STR CL      ..SET MODIFIED INSTR
8020 ;                  0070                                   ..INTO RAM
8020 F3;                0071            XOR                ..CK THAT IT WROTE
8021 3A29;              0072            BNZ UT4
8023 D1;                0073            SEP CL             ..GO TO EXECUTE INSTR
8024 ;                  0074                               ..(80-9F)
8024 51;                0075            STR CL             ..STORE RESULT IN RAM
8025 2121;              0076            DEC CL  ;DEC CL    ..BACK UP FOR NEXT BYTE
8027 300E;              0077            BR LOOP2
8029 ;                  0078        ..
8029 90B5B3;            0079  UT4:GHI R0      ;PHI PC    ;PHI SUB  ..#80+†PC.1
802C ;                  0080                                   ..AND SUB.1
802C F830A5;            0081            LDI A.0(UT4A)      ;PLO PC   ..
802F D5;                0082            SEP PC
8030 E5;                0083  UT4A:SEX PC
8031 7155;              0084            DIS,#55          ..NOTE PC=5 ASSUMED
8033 ;                  0085                             ..HERE!
8033 6101;              0086            OUT 1,#01      ..SELECT RCA GROUP
8035 F8FEA3;            0087            LDI A.0(TIMALC)    ;PLO SUB      ..READ ONE
8038 ;                  0088                                   ..TO SET TIMER
8038 D3;                0089            SEP SUB
8039 ;                  0090        ..
8039 ;                  0091        ... INITIATION NOW DONE
8039 ;                  0092        ..
8039 F89CA3;            0093  START:LDI A.0(TYPE5D) ;PLO SUB
```

```
803C D30D;          0094          SEP SUB; ,#0D        ..CR=CARRIAGE RETURN
803E D30A;          0095  ST2:SEP SUB; ,#0A  ..LF=LINE FEED
8040 D32A;          0096          SEP SUB; ,#2A      ..* AS PROMPT CHARACTER
8042 F800ADBD;      0097  IGNORE:LDI #00;PLO ASL;PHI ASL      ..PREPARE TO
8046 ;              0098                                       ..INPUT HEX
8046 ;              0099                              ..DIGITS,CLEAR ASL
8046 F83BA3;        0100          LDI A.0(READAH) ;PLO SUB
8049 D3;            0101          SEP SUB         ..INPUT COMMAND
804A FB24;          0102          XRI #24                ..IS IT $ ?
804C 32D6;          0103          BZ DOLLAR
804E FB05;          0104          XRI #05         ..IS IT ! ? (TEST WITH $.XOR.!)

8050 A1;            0105          PLO SWITCH      ..AND SAVE RESULT
8051 CE;            0106          LSZ             ..EQIV. TO BR RDARGS
8052 FB1E;          0107          XRI #1E         ..IS IT ?
8054 ;              0108                            ..?(TEST WITH $.XOR.!.XOR.?)

8054 3A42;          0109          BNZ IGNORE ..IGNORE ALL UNTIL A COMMAND IS
8056 ;              0110                     ..READ
8056 ;              0111      ..
8056 ;              0112      ..THE FOLLOWING IS COMMON FOR ?M AND !M
8056 ;              0113      ..(SWITCH.0 =0 FOR THE LATTER)
8056 ;              0114      ..
8056 D3;            0115  RDARGS:SEP SUB          ..NOTE SUB AT READAH. NOW
8057 ;              0116                          ..READ HEX ARGS
8057 FB4D;          0117          XRI #4D         ..SHOULD BE M
8059 3ACA;          0118          BNZ SYNERR
805B D3;            0119  RD1:SEP SUB
805C 3B5B;          0120          BNF * -#01      ..IGNORE NON HEX CHARS.
805E ;              0121                          ..AFTER M.
805E D3;            0122          SEP SUB
805F 335E;          0123          BDF *-#01       ..READ IN FIRST ARG
8061 ;              0124                          ..(LOCATIONN IN MEMORY)
8061 FB20;          0125          XRI #20         ..NEXT CHAR SHOULD BE A SPACE

8063 3ACA;          0126          BNZ SYNERR
8065 9DB0;          0127          GHI ASL ;PHI PTER
8067 BDA0;          0128          GLO ASL ;PLO PTER  ..PTER NOW POINTS INTO
8069 ;              0129                             ..USER MEMORY
8069 81;            0130          GLO SWITCH         ..LOOK AT SWITCH
806A 32B4;          0131          BZ EX1             ..IF 0 IT WAS !
806C ;              0132                             ..OTHERWISE IT WAS ?
806C ;              0133      ..THE FOLLOWING DOES (?M LOC COUNT) COMMAND
806C ;              0134  ..
806C F800ADBD;      0135          LDI #00 ;PLO ASL ;PHI ASL  ..CLEAR ASL
8070 D3;            0136  RD2:SEP SUB
8071 3370;          0137          BDF RD2            ..READ IN SECOND ARG
8073 ;              0138                             ..(NUMBER OF BYTES)
8073 FB0D;          0139          XRI #0D            ..NEXT CK FOR CR
8075 3ACA;          0140          BNZ SYNERR
```

```
8077 F89CA3;          0141          LDI A.0(TYPE5D) ;PLO SUB     ..TYPE
807A 8DA1;            0142          GLO ASL         ;PLO SWITCH
807C 9DB1;            0143          GHI ASL         ;PHI SWITCH
807E D30A;            0144    LINE:SEP SUB; ,#0A                 ..LF
8080 90BF;            0145    LINE1:GHI PTER          ;PHI CHAR  ..PREPARE LINE
8082 ;                0146                                       ..HEADING
8082 F8AEA3;          0147          LDI A.0(TYPE2)  ;PLO SUB
8085 D3;              0148          SEP SUB                      ..TYPE 2 HEX DIGIT
.S
8086 80BF;            0149          GLO PTER   ;PHI CHAR
8088 F8AEA3;          0150          LDI A.0(TYPE2)  ;PLO SUB
808b D3;              0151          SEP SUB                      ..TYPE
808C D320;            0152          SEP SUB; ,#20                ..SPACE
808E ;                0153       ..
808E 40BF;            0154    TLOOP:LDA PTER          ;PHI CHAR  ..FETCH 1 BYTE FOR
.
8090 ;                0155                                       ..TYPING
8090 F8AEA3;          0156          LDI A.0(TYPE2)  ;PLO SUB
8093 D3;              0157          SEP SUB          ..TYPE 2 HEX
8094 21;              0158          DEC SWITCH
8095 81;              0159          GLO SWITCH
8096 3A98;            0160          BNZ TL3          ..BRANCH IF NOT DONE YET
8098 91;              0161          GHI SWITCH
8099 3239;            0162          BZ START         ..BRANCH IF DONE
809B 80FA0F;          0163    TL3:GLO PTER ;ANI#0F ..IS PTER DIV BY 16
809E 3AA6;            0164          BNZ TL2
80A0 D33B;            0165          SEP SUB; ,#3B ..IF YES TYPE ; THEN
80A2 D30D;            0166          SEP SUB; ,#0D ..CR AND
80A4 307E;            0167          BR LINE
80A6 F6;              0168    TL2:SHR                  ..DIV BY 2?
80A7 338E;            0169          BDF TLOOP        ..IF NO LOOP BACK, ELSE
80A9 308C;            0170          BR TLOOP -#02 ..AND THEN LOOP BACK
80AB ;                0171       ..
80AB ;                0172       ..THE FOLLOWING DOES(!M LOC DATA) COMMAND
80AB ;                0173       ..ENTER AT EX1
80AB ;                0174       ..
80AB ;                0175       ..EFFECT OF THE FOLLOWING IS TO READ IN HEX
80AB ;                0176       ..TERMINATING WITH A CR,IGNORING NON-HEX CHARS.
.
80AB ;                0177       ..PAIRS; EXCEPTIONS: A COMMA BEFORE A CR ALLOWS
.
80AB ;                0178       ..THE INPUT TO CONTINUE ON THE NEXT LINE AND A
80AB ;                0179       ..SEMICOLON ALLOWS AN !M COMMAND TO
80AB ;                0180       ..BE ASSUMED.
80AB ;                0181       ..
80AB D3;              0182    EX3:SEP SUB                 ..INPUT UNTIL A HEX IS READ
.
80AC 3BAB;            0183          BNF EX3
80AE ;                0184       ..
80AE D3;              0185    EX2:SEP SUB                 ..LOOKING FOR SECOND HEX
```

```
80AF ;                  0186                            ..DIGIT
80AF 3BCA;              0187            BNF SYNERR      ..BR IF NOT HEX
80B1 8D50;              0188            GLO ASL   ;STR PTER     ..**SET BYTE**
80B3 10;                0189            INC PTER
80B4 D3;                0190    EX1:SEP SUB                     ..NOTE SUB AT
80B5 ;                  0191                                    ..READAH
80B5 33AE;              0192            BDF EX2                 ..BR IF HEX
80B7 FB0D;              0193            XRI #0D                 ..CHECK IF CR
80B9 3239;              0194            BZ START
80BB FB21;              0195    EX4:XRI #21                     ..ELSE CK IF COMMA
.
80BD ;                  0196                            ..(TEST WITH CR.XOR.,)
80BD 32AB;              0197            BZ EX3      ..IF ELSE BRANCH
80BF FB17;              0198            XRI #17         ..CK FOR SEMICOLON(TEST WITH
.
80C1 ;                  0199                            ..CR.XOR.,.XOR.;)
80C1 3AB4;              0200            BNZ EX1         ..IGNORE ALL ELSE
80C3 D3;                0201            SEP SUB         ..ON SEMI IGNORE AL UNTIL CR
.
80C4 ;                  0202                            ..THEN LOOP BACK
80C4 FB0D;              0203            XRI #0D
80C6 3AC3;              0204            BNZ *-#03
80C8 305B;              0205            BR RD1          ..THEN BRANCH BACK
80CA ;                  0206                            ..FOR IM COMMAND
80CA ;                  0207        ..
80CA F89CA3;            0208    SYNERR:LDI A.0(TYPE5D) ;PLO SUB    ..GENERAL
80CD ;                  0209                                ..RESULT OF
80CD ;                  0210                                ..SYNTACTIC ERROR
80CD D30D;              0211            SEP SUB; ,#0D               ..CR
80CF C081F8;            0212            LBR FSYNER                  ..FINISH ERROR MSG
.
80D2 ;                  0213        ..
80D2 ;                  0214        ..THE FOLLOWING DOES SP HHHH
80D2 ;                  0215        ORG #B0D6
80D6 D3;                0216    DOLLAR:SEP SUB              ..NOTE SUB.0=READAH
80D7 FB50;              0217            XRI #50             ..SHOULD BE P
80D9 3ACA;              0218            BNZ SYNERR
80DB D3;                0219    D1:SEP SUB
80DC 33DB;              0220            BDF D1              ..ASSEMBLE HEX
80DE ;                  0221                                ..STING INTO ASL
80DE FB0D;              0222            XRI #0D             ..FIRST NONHEX
80E0 ;                  0223                                ..MUST BE CR
80E0 3ACA;              0224            BNZ SYNERR
80E2 9DB0;              0225            GHI ASL    ;PHI R0
80E4 8DA0;              0226            GLO ASL    ;PLO R0      ..SET UP NEXT PC
80E6 F89CA3;            0227            LDI A.0(TYPE5D)    ;PLO SUB
80E9 D30A;              0228            SEP SUB; ,#0A           ..LF
80EB E5;                0229            SEX PC
80EC 7000;              0230            RET,#00             ..AND USER PROGRAM
80EE ;                  0231                                ..BEGINS (IN R0)
```

```
80EE ;                  0232                                ..EXIT TO UT4
80EE ;                  0233        ..
80EE ;                  0234        ..
80EE ;                  0235        ..
80EE ;                  0236        ..
80EE ;                  0237        ..SUBROUTINES
80EE ;                  0238        ..
80EE ;                  0239        ..DELAY ROUTINE
80EE ;                  0240        ..DELAY IS 2(1+AUX.1(3+@SUB))
80EE ;                  0241        ..USED BY TYPE, READ, AND TIMALC.
80EE ;                  0242        ..AUX.1 IS ASSUMED TO HOLD A DELAY CONSTANT
80EE ;                  0243        ..=((BIT TIME OF TERMINAL)/
80EE ;                  0244        ..(20*INSTR TIME OF COSMAC))-1.
80EE ;                  0245        ..THIS CONSTANT CAN BE GENERATED
80EE ;                  0246        ..AUTOMATICALLY BY THE TIMALC ROUTINE.
80EE ;                  0247        ..
80EE D3;                0248    DEXIT:SEP SUB
80EF 9EF6AE;            0249    DELAY1:GHI AUX ;SHR  ;PLO AUX     ..SHIFT OUT
80F2 ;                  0250                                      ..ECHO FLAG
80F2 2E;                0251    DELAY2:DEC AUX              ..AUX.0 HOLDS BASIC
80F3 ;                  0252                                ..BIT DELAY
80F3 43FF01;            0253           LDA SUB     ;SMI #01    ..PICK UP A CONSTANT
80F6 3AF4;              0254           BNZ *-#02                ..LOOP AS SPECIFIED
80F8 ;                  0255                                    ..BY CALL
80F8 8E;                0256           GLO AUX                      ..DONE YET ?
80F9 32EE;              0257           BZ DEXIT
80FB 23;                0258           DEC SUB                      ..POINTS SUB
80FC ;                  0259                                  ..AT DELAY POINTER
80FC 30F2;              0260           BR DELAY2
80FE ;                  0261        ..
80FE ;                  0262        ..ROUTINE TO CALUCULATE BYTE TIME AND ECHO
80FE ;                  0263        ..FLAG. WAITS FOR LF (NO ECHO) OR CR(ECHO)
80FE ;                  0264        ..BE TYPED IN. ALSO SETS UP POINTER TO
80FE ;                  0265        ..DELAY ROUTINE.
80FE ;                  0266        ..AUX.1 ENDS UP HOLDING, IN THE MOST
80FE ;                  0267        ..SIGNIFICANT 7 BITS,THE DELAY CONSTANT.
80FE ;                  0268        ..LEAST SIGNIFICANT BIT IS 0 FOR ECHO, 1 FOR
80FE ;                  0269        ..NO ECHO
80FE ;                  0270        ..
80FE 93BC;              0271    TIMALC:GHI SUB     ;PHI DELAY
8100 F800AEAF;          0272           LDI #00     ;PLO AUX          ;PLO CHAR
8104 F8EFAC;            0273           LDI A.0(DELAY1)               ;PLO DELAY
8107 ;                  0274                                  ..DELAY ROUTINE READY
8107 3707;              0275           B4 *                     ..WAIT FOR START BIT
8109 3F09;              0276           BN4 *                    ..WAIT FOR FIRST
810B ;                  0277                                ..NON ZERO DATA BIT
810B F803;              0278           LDI #03                      ..SET UP FOR
810D ;                  0279                                ..10 EXECUTIONS
810D ;                  0280                                ..SO ROUND-OFF IS MINIMAL
810D FF01;              0281    TC2:SMI #01
```

```
810F 3A0D;          0282          BNZ *-#02
8111 8F;            0283          GLO CHAR                    ..LOOK TO SEE
8112 ;              0284                                      ..IF DATA
8112 ;              0285                                      ..CHANGED PREVIOUSLY
.
8112 3A17;          0286          BNZ ZRONE                   ..BR IF IT HAD
8114 3719;          0287          B4 INCR                     ..ELSE LOO FOR
8116 ;              0288                                    ..CHANGE TO 0 NOW
8116 ;              0289                                    ..BRANCH IF NO
8116 1F;            0290          INC CHAR                    .. IF YES SET SWITCH
.
8117 371E;          0291    ZRONE:B4 DAUX                     ..LOOK FOR CHANGE
8119 ;              0292                                      ..TO 1, BR IF YES
8119 1E;            0293    INCR:INC AUX
811A F807;          0294          LDI #07                     ..SET UP FOR
811C ;              0295                                       ..20 INSTR. LOOP
811C 300D;          0296          BR TC2
811E ;              0297       ..AUX.0 NOW HOLDS #LOOPS IN 2 BIT TIMES
811E 2E2E;          0298    DAUX:DEC AUX    ;DEC AUX          ..REDUCE COUNT
8120 ;              0299                                      ..TO BALANCE
8120 ;              0300                                    ..FIXED OVERLOAD
8120 ;              0301                                    ..IN CALLING DELAY
8120 8EF901BE;      0302          GLO AUX ;ORI #01 ;PHI AUX          ..LSB AUX.1=
.1
8124 DC0C;          0303          SEP RC; ,#0C                      ..1.5 BIT
8126 ;              0304                                        ..TIME DEAY
8126 3F2C;          0305          BN4 WAIT  ..BR IF LF=↑NO ECHO, LSB AUX.1=1
8128 ;              0306
8128 9EFAFE;        0307          GHI AUX    ;ANI#FE
812B BE;            0308          PHI AUX      ,..CR=↑ECHO, LBB AUX.1=0
812C DC26;          0309    WAIT:SEP RC; ,#26
812E D5;            0310          SEP R5
812F ;              0311    ..
812F ;              0312    ..
812F ;              0313    ..
812F ;              0314    ..READ ROUTINE--READS 1 BYTE INTO CHAR.1.
812F ;              0315    ..WHEN ENTERED VIA READAH,THEN
812F ;              0316    ..IF INPUT IS A HEX DIGIT ITS HEX VALUE
812F ;              0317    ..IS SHIFTED INTO ASL FROM THE RIGHT
812F ;              0318    ..AND DF=1,ELSE DF=0; CLOBBERS CHAR, AUX.0,(ASL
812F ;              0319    ..ON READAH). LEAVES BYTE IN D (BUT CLOBBERED IF
812F ;              0320    ..SUBR LINKAGE IS USED). LEAVES PC AT READAH
812F ;              0321    ..ENTRY POINT; EXITS TO R5.
812F ;              0322    ..
812F ;              0323    ..WARNING:READ PROCESS HAS NOT FINISHED. DO
812F ;              0324    ..NOT TYPE IMMEDIATELY, OR ELSE ENTER TYPE VIA
812F ;              0325    ..TYPE5D
812F ;              0326    ..
812F ;              0327          ORG #812F
812F FC07;          0328    CKDEC:ADI #07       ..CK FOR ASCII DECIMAL INPUT
```

```
8131 3337;          0329          BDF NFND
8133 FC0A;          0330          ADI #0A
8135 3387;          0331          BDF FND       ..SUB NET 30
8137 FC00;          0332  NFND:ADI #00         ..SETS DF=0
8139 9F;            0333  REXIT:GHI CHAR       ..CHARACTER INTO D
813A D5;            0334          SEP R5
813B F800;          0335  READAH:LDI #00
813D 38;            0336          SKP       ..SKIP OVER TO READ1
813E 83;            0337  READ:GLO SUB      ..CONSTANT WITH A VALUE †0
813F C8;            0338          LSKP
8140 F801;          0339  TTYRED:LDI #01
8142 AF;            0340  READ1:PLO CHAR      ..SET ENTRY FLAG
8143 F880BF;        0341  READ2:LDI #80    ;PHI CHAR       ..INITIALIZE
8146 ;              0342                                  ..INPUT BYTE
8146 ;              0343                                     ..WHEN SHIFTED 80
8146 ;              0344                                     ..IS 1, WILL BE DONE
.
8146 E3;            0345          SEX SUB
8147 8FF6;          0346          GLO CHAR    ;SHR      ..DF=1 -†ENTRY VIA TTYRED
.
8149 3B4D;          0347          BNF TTY1-#02
814B 6780;          0348          OUT 7 ,#80      ..READER ON
814D 3F4D;          0349          BN4 *       ..WAIT FOR END OF LAST DATA BIT
814F 374F;          0350  TTY1:B4 *       ..WAIT FOR PRESENT START BIT
8151 DC02;          0351          SEP RC; ,#02      ..DELAY HALF BIT TIME
8153 374F;          0352          B4 TTY1       ..BR IF NO START BIT
8155 8FF6;          0353          GLO CHAR    ;SHR      ..ENTRY VIA TTYRED?
8157 3B5B;          0354          BNF NOBIT       ..BR IF NO
8159 6740;          0355          OUT 7 ,#40
815B ;              0356  ..
815B E2C4;          0357  NOBIT:SEX R2    ;NOP      ..RESET X, AND DELAY
815D 9EF6;          0358  BIT:GHI AUX    ;SHR      ..ECHO ?
815F 3368;          0359          BDF NOECHO      ..BR IF NO
8161 3766;          0360          B4 OUTBIT       ..IS THE BIT A 1 ?
8163 7B;            0361          SEQ         ..SET Q
8164 3068;          0362          BR NOECHO
8166 7A;            0363  OUTBIT:REQ        ..RESET Q
8167 C4;            0364          NOP         ..DELAY
8168 DC07;          0365  NOECHO:SEP RC; ,#07       ..WAIT ONE BIT TIME
816A C4C4;          0366          NOP    ;NOP       ..MORE DELAY
816C 9FF6BF;        0367          GHI CHAR    ;SHR    ;PHI CHAR      ..SHIFT
816F ;              0368                                  ..THE INPUT CHAR
816F 3378;          0369          BDF NEXT      ..BR IF INPUT FINISHED
8171 ;              0370                        ..D=CHAR.1
8171 F980;          0371          ORI#80
8173 3F5B;          0372          BN4 NOBIT       ..BR IF INPUT WAS A ZERO
8175 BF;            0373          PHI CHAR
8176 305D;          0374          BR BIT        ..CONTINUE LOOP
8178 ;              0375  ..
8178 ;              0376  ..
```

```
8178 7A;          0377  NEXT:REQ      ..OUTPUT THE STOP BIT
8179 3243;        0378          BZ READ2       ..BR IF D=0, =↑CHAR.1
817B ;            0379                         ..IS A NULL
817B 8F;          0380          GLO CHAR       ..CK ENTRY FLAG
817C 3A39;        0381          BNZ REXIT       ..BR IF ENTRY WAS VIA READ
817E 9F;          0382          GHI CHAR
817F FF41;        0383          SMI#41       ..CK FOR ASCII HEX
8181 3B2F;        0384          BNF CKDEC       ..(AT TOP OF ROUTINE)
8183 FF06;        0385          SMI#06       ..CK FOR A THRU F
8185 3337;        0386          BDF NFND
8187 ;            0387  ..
8187 ;            0388  ..
8187 FEFEFEFE;    0389  FND:SHL    ;SHL    ;SHL    ;SHL
818B FC08FE;      0390          ADI#08    ;SHL
818E AE;          0391  FND1:PLO AUX       ..READY TO SHIFT INTO RD
818F 8D7EAD;      0392          GLO ASL    ;SHLC    ;PLO ASL      ..SHIFT
8192 ;            0393                                          ..LOW HALF
8192 9D7EBD;      0394          GHI ASL    ;SHLC    ;PHI ASL      ..SHIFT
8195 ;            0395                                          ..HIGH HALF
8195 8EFE;        0396          GLO AUX    ;SHL
8197 3A8E;        0397          BNZ FND1       ..BR IF NOT FINISHED
8199 3039;        0398          BR REXIT
819B ;            0399  ..TYPE ROUTINE--TYPES 1 BYTE FROM @R5!,@R6!,
819B ;            0400  ..OR CHAR.1,OR TYPES A BYTE AS TWO HEX DIGITS
819B ;            0401  ..FROM CHAR.1 FOLLOWS A LINE FEED BY SIX NULLS.
819B ;            0402  ..USES 2 AUXILIARY REGS-AUX AND CHAR-PLUS
819B ;            0403  ..RAM LOCATION @ST.EXITS READY TO TYPE 1 BYTE
819B ;            0404  ..FROM @R5!. EXITS TO R5
819B ;            0405  ..WHEN ENTERED AT TYPE5D,PAUSES TO ALLOW AN
819B ;            0406  ..EARLIER READ TO COMPLETE.
819B ;            0407  ..
819B ;            0408  ..AUX.0 HOLDS OUTPUT CHAR (AT FIRST), THEN
819B ;            0409  ..THE DELAY CONSTANT BETWEEN BITS. CHAR.0 HOLDS
819B ;            0410  ..THE NUMBER OF BITS (11) IN ITS LOWER DIGIT,
819B ;            0411  ..AND IN ITS UPPER DIGIT HOLDS A CODE--
819B ;            0412  ..      0 FOR BYTE OUTPUT
819B ;            0413  ..      1 FOR FIRST HEX OUTPUT
819B ;            0414  ..      2 FOR LST NULL OUTPUT
819B ;            0415  ..      8 FOR LF OUTPUT
819B ;            0416  ..
819B ;            0417          ORG #819C
819C DC17;        0418  TYPE5D:SEP RC; ,#17      ..3 BIT TIME DELAY
819E 38;          0419          SKP       ..SKP TO TYPE5D
819F D5;          0420  TEXIT:SEP R5
81A0 4538;        0421  TYPE5:LDA R5    ;SKP      ..ENTRY FOR UT4
81A2 ;            0422                            ..SKIP TO TYPE
81A2 4638;        0423  TYPE6:LDA R6    ;SKP      ..ENTRY FOR G.P.
81A4 ;            0424                            ..IMMED TH
81A4 9F;          0425  TYPE:GHI CHAR
81A5 AE;          0426  TY1:PLO AUX      ..SAVE BYTE FOR LATER
```

```
81A6 FB0A;          0427          XRI#0A      ..IS IT LINE FEED ?
81A8 3ABF;          0428          BNZ TY2
81AA F88B;          0429          LDI#8B      ..(# OF BITS)+(#OF NULLS
81AC ;              0430                      ..TO FOLLOW LF+1)
81AC 30C1;          0431          BR TY3
81AE 9F;            0432    TYPE2:GHI CHAR      ..UT4 ENTRY
81AF F6F6F6F6;      0433    TY4:SHR   ;SHR    ;SHR    ;SHR      ..SHIFT FIRST
81B3 ;              0434                                        ..HEX TO RIGHT
81B3 FCF6;          0435          ADI#F6      ..CONVERT TO HEX
81B5 3BB9;          0436          BNF *+#04     ..IF A OR MORE
81B7 FC07;          0437          ADI#07      ..ADD NET 37
81B9 FFC6AE;        0438          SMI#C6    ;PLO AUX      ..ELSE ADD NET 30
81BC F81B;          0439          LDI#1B      ..10+(# OF BITS)
81BE C8;            0440          LSKP      ..EQUIV. TO BR TY3
81BF ;              0441    ..
81BF F80B;          0442    TY2:LDI#0B      ..(# OF BITS TO OUTPUT)
81C1 AF;            0443    TY3:PLO CHAR      ..SAVE MAIN TALLY VALUE
81C2 ;              0444    ..
81C2 ;              0445    ..
81C2 7B;            0446    BEGIN:SEQ       ..START BIT
81C3 8E;            0447          GLO AUX       ..GET CHAR TO BE TYPED
81C4 AD;            0448          PLO RD      ..SAVE THE CHAR.
81C5 ;              0449                        ..(AUX.0 CLOBBERED)
81C5 DC07;          0450    PREBIT:SEP RC; ,#07   ..WAIT ONE BIT TIME
81C7 ;              0451              ..RETURN FROM DELAY WITH D=0
81C7 2F;            0452                DEC CHAR      ..DEC THE BIT COUNTER
81C8 F5;            0453          SD      ..SET DF=1
81C9 8D76AD;        0454          GLO RD    ;SHRC    ;PLO RD      ..SHIFT
81CC ;              0455                                      ..OUTPUT CHAR
81CC 33D1;          0456          BDF OUT1B       .. BR IF THE BIT IS A 1
81CE 7B;            0457          SEQ       ..ELSE SET Q TO ZERO
81CF 30D3;          0458          BR OUT1B+#02
81D1 7A;            0459    OUT1B:REQ       ..SET Q TO 1
81D2 C4;            0460          NOP       ..DELAY
81D3 8FFA0F;        0461          GLO CHAR    ;ANI#0F       ..FINISHED TYPING ?
81D6 C4C4;          0462          NOP     ;NOP        ..DELAY(14 INSTR.LOOP)
81D8 3AC5;          0463          BNZ PREBIT        ..BR IF NOT FINISHED
81DA 8FFCFB;        0464    NXCHAR:GLO CHAR     ;ADI#FB
81DD AF;            0465          PLO CHAR      ..SET UP FOR NEXT CHAR
81DE 3B9F;          0466          BNF TEXIT       ..BUT EXIT IF NO MORE
81E0 FF1B;          0467          SMI#1B      ..TEST FOR ALTERNATIVES
81E2 329F;          0468          BZ TEXIT      ..IF JUST TYPED LST NULL
81E4 3BEA;          0469          BNF HEX2      ..IF JUST TYPED FIRST HEX
81E6 ;              0470                      ..JUST TYPED LF OR NULL--
81E6 F800;          0471          LDI#00      ..PREPARE TO TYPE NULL
81E8 30F5;          0472          BR HX22
81EA ;              0473    ..
81EA 9FFA0F;        0474    HEX2:GHI CHAR     ;ANI#0F     ..GET 2ND HEX DIGIT
81ED FCF6;          0475          ADI#F6      ..CONVERT TO HEX
81EF 3BF3;          0476          BNF *+#04       ..IF A MORE
```

```
81F1 FC07;          0477          ADI#07      ..ADD NET 37
81F3 FFC6;          0478          SMI#C6      ..ELSE ALL NET 30
81F5 AE;            0479  HX22:PLO AUX       ..STORE CHAR AWAY
81F6 30C2;          0480          BR BEGIN
81F8 ;              0481  ..
81F8 D30A;          0482  FSYNER:SEP SUB; ,#0A      ..LF
81FA D33F;          0483          SEP SUB; ,#3F      ..?
81FC C08039;        0484          LBR START
81FF ;              0485          END
0000
```

## TABLE 1

### UT4 REGISTER UTILIZATION

| REGISTER NAME | REGISTER NUMBER | FUNCTION and COMMENTS |
|---|---|---|
| PTER | R0 | Altered by UT4 while storing registers. |
| CL | R1 | |
| SUB | R3 | Program counter for all READ, all TYPE, and TIMALC routines. |
| PC | R5 | Program counter for UT4, which calls the routines above. |
| DELAY | RC | Program counter for the DELAY routine. Points to DELAY1 in memory. |
| ASL | RD | Assembled into by READAH (input hex digits). |
| AUX | RE | AUX.1 holds time constant and echo bit. AUX.0 is used by all READ and TYPE routines and by TIMALC. |
| CHAR | RF | CHAR.1 holds input/output ASCII character. CHAR.0 is used by all READ and TYPE routines and by TIMALC. |

## TABLE 2

### ENTRY POINTS FOR UT4 SUBROUTINES

| ENTRY NAME | ABSOLUTE ADDRESS | FUNCTION and COMMENTS |
|---|---|---|
| READ | 813E | Input ASCII → CHAR.1, D (if non-standard linkage). |
| READAH | 813B | Same as READ. If hex character, DIGIT → ASL (see text). |
| TTYRED | 8140 | Same as READ. Controls paper tape reader (see text). |
| TYPE5D | 819C | 1.5-bit delay. Then TYPE5 function. |
| TYPE5 | 81A0 | Output ASCII character at M(R5). Then increment R5. |
| TYPE6 | 81A2 | Output ASCII character at M(R6). Then increment R6. |
| TYPE | 81A4 | Output ASCII character at CHAR.1. |
| TYPE2 | 81AE | Output hex digit pair in CHAR.1. |
| TIMALC | 80FE | Read input character and set up control byte in AUX.1. Initialize RC to point to DELAY1. |
| DELAY1 | 80EF | Delay, as function of M(R3) (see text). Then R3+1 → R3. |

NOTES:

1. All routines except DELAY use R3 as program counter, exit with SEP5, and alter registers X, D, DF, AUX, and CHAR.
2. DELAY routine uses RC as program counter, exits with SEP3 after incrementing R3, and alters register X, D, DF, and AUX.
3. READ and READAH exit with R3 pointing back at READAH.
4. All five TYPE routines exit with R3 pointing at TYPE5.
5. As indicated in Table 3-I, ASL = RD, AUX = RE, and CHAR = RF.

## OUTPUT ROUTINE USING UT4

The monitor program UT4 includes all the software to use the "software" UART with EF4 and Q line,not only for the monitor program, but as well these subroutines can be called by user programs. This is very useful,for example, to send messages to the terminal during program execution.

Description of the program :

1. At the first instruction,interrupts have to be disabled as the whole timing is via counters that cannot be interrupted (line 9).

2. The TYPE subroutine comes back with a SEP 5 instruction, which means it has to be called from a program running in R5 as PC (10-12).

3. A DELAY counter has to be initialized (14-15).

4. Timing constant and echo bit have to be loaded (17-18).

5. The text pointer has to be prepared, pointing to the text bytes at add #0036 (20-21).

6. A call of subroutine at #819C gets the timing right (23-25).

7. Now the output routine starts with preparing R3 to 81A4 (27).

8. Now the first ASCII character is loaded via R6 in D and then to RF.1, where the subroutine TYPE gets the byte from (28).

9. Branch now if this byte in D is #00, which means end of string (29,43).

10. If it is not#00, do a SEP 3 and call the TYPE routine (30).

11. TYPE exits with a SEP 5 which means the program continues at location #002B and branches back to OUTPUT (31).

12. If the string has been sent a delay routine is initialized and when it has finished, the same string is sent again.

```
0000 ;                  0001          ..  EXAMPLE PROGRAM
0000 ;                  0002          ..  OUTPUT A STRING
0000 ;                  0003          ..  USING UT4 ROUTINES
0000 ;                  0004
0000 ;                  0005          CONST1=#80EF
0000 ;                  0006          CONST2=#2800
0000 ;                  0007
0000 ;                  0008          ORG #0060
0000 7400;              0009 INIT:    DIS,#00                 ..DISABLE INTERRUPTS
0002 F800B5;            0010          LDI A.1(WRITE);PHI R5   ..LOAD MAIN PC R5
0005 F809A5;            0011          LDI A.0(WRITE);PLO R5   ..WITH START ADDRESS
0008 D5;                0012          SEP R5                  ..CHANGE PC FROM 0 TO 5
0009 ;                  0013
0009 F880BC;            0014 WRITE:   LDI A.1(CONST1);PHI RC  ..INIT DELAY POINTER
000C F8EFAC;            0015          LDI A.0(CONST1);PLO RC
000F ;                  0016
000F F828BE;            0017          LDI A.1(CONST2);PHI RE  ..INIT TIME CONSTANT
0012 F800AE;            0018          LDI A.0(CONST2);PLO RE  ..AND ECHO BIT
0015 ;                  0019
0015 F800A6;            0020          LDI A.1(TEXT);PHI R6    ..PREPARE TEXT POINTER
0018 F836A6;            0021          LDI A.0(TEXT);PLO R6
0018 ;                  0022
0018 F884B3;            0023          LDI #84;PHI R3          ..LOAD R3 WITH 849C TO
001E F89CA3;            0024          LDI #9C;PLO R3          ..DO A TYPESD FIRST
0021 D300;              0025          SEP R3,#00              ..EXECUTE TYPESD
0023 ;                  0026
0023 F8A4A3;            0027 OUTPUT:  LDI #A4;PLO R3          ..PREPARE R3 FOR TYPE
0026 46BF;              0028          LDA R6;PHI RF           ..GET BYTE IN RF.1 AND D
0028 322D;              0029          BZ DELAY                ..IF IT IS 00 GO TO DELAY
002A D3;                0030          SEP R3                  ..OTHERWISE SEND IT
002B 3023;              0031          BR OUTPUT               ..BACK AGAIN
002D ;                  0032
002D F8FF;              0033 DELAY:   LDI #FF                 ..LOAD DELAY CONSTANT
002F B7;                0034          PHI R7                  ..IN R7
0030 27;                0035 LOOP:    DEC R7                  ..DEC THIS REGISTER
0031 97;                0036          GHI R7                  ..GET R7.1 IN D-REGISTER
0032 3A30;              0037          BNZ LOOP                ..IF NOT 00 DEC AGAIN
0034 3009;              0038          BR WRITE                ..GO TO NEXT CHARACTER
0036 ;                  0039
0036 0D0A524341204252;0040 TEXT:    ,X'0D0A',T'RCA BRUSSELS'
003D 5255535354454C53;0040
0044 0D0A535452494E47;0041          ,X'0D0A',T'STRING OUTPUT'
004B 4720544F55545055;0041
0052 54;                0041
0053 0D0A5553494E4720;0042          ,X'0D0A',T'USING UT4 ROUTINES'
005A 2055543420524F55;0042
0061 5554494E4553;      0042
0067 00;                0043          ,X'00'                  ..END OF STRING
0068 ;                  0044          END
0000
```

# INTERRUPT ENTRY AND EXIT

Compatible with SCRT

## A. Interrupt Entry

When the 1802 re-ponds to an interrupt,X and P are set to 2 and 1 respectively. Thus, the interrupt handler always starts at the location loaded into R1 during initialization. It is convenient for most purposes to use R1 to execute a short housekeeping routine which is analogous to the "CALL" routine of SCRT. Such a routine is shown in Fig. 1.

Using R2 as stack pointer, the handler first pushes X, P, D, DF and R3 onto the stack. It is also necessary to save R4 and R5, since an interrupt may occur during subroutine calls or returns, when one of these registers is active. To reduce interrupt response time only the low bytes of R4 and R5 are saved, thus assuming that the CALL and EXIT routines do not cross page boundaries. (This requirement is already satisfied by SCRT, because of the use of short branches in the CALL and EXIT routines - again to reduce execution time).

It is not necessary to save R6, since this register ("link") will be saved automatically by any SCRT subroutine call used in the interrupt service routine. In fact, the information in R6 is very useful in debugging since by reading R6 the programmer can easily idetify not only which subroutine was active at the time of the interrupt but also the location of the call to that subroutine.

To allow use of SCRT subroutines during interrupts, R4 and R5 are re-initialized and finally the starting address of the service routine itself is loaded into R3, which becomes program counter after execution of the SEP 3 instruction.

After passing control to R3, R1 is left reset to the interrupt entry point (by the "branch-to-stop" instruction). Since interrupts are automatically disabled on interrupt response, the interrupt handler can never be interrupted, and interrupts remain disabled until the service routine itself re-enables them.

Total execution time of the entry routine is 78.4 uS with a 5MHz clock.

B. Interrupt Exit

At the end of any interrupt service, the machine status must be restored for return to the interrupted program. This is done by a routine analogous to the "EXIT" of SCRT. This routine needs a dedicated pointer, and it is often convenient to use RØ for this task in applications not using DMA. However, for generality it is suggested that the next "available" register, R7, should be used for the return-from-interrupt function in Figure 1.

The routine shown assumes that X = 2 on entry and that interrupts are disabled during its operation. It restores R5, R4, R3, DF and D and then branches to the RET instruction, which restores X and P and re-enables interrupts.

To pass control to the interrupt exit routine, it is necessary to disable interrupts, set X to 2 and set P to 7. A two-instruction sequence is needed to do this : -

```
SEX 3 ; DIS, # 27
```

This sequence is analogous to the SEP 5 used for subroutine exits under SCRT, and may be conveniently defined as a macro if several interrupt routines are used in the application.

Including the exit instructions, total time overhead is 54.4 uS with a 5MHz clock.

```
0000 ;              0001          .. EXAMPLE PROGRAM
0000 ;              0002          .. INTERRUPT HANDLER
0000 ;              0003          .. RUNS IN R4 (ENTRY)
0000 ;              0004          .. AND IN  R7 (EXIT)
0000 ;              0005
0000 ;              0006          CALL=#84E4          ..ADDRESS CALL ROUT.
0000 ;              0007          RET=#84F4           ..ADDRESS RET  ROUT.
0000 ;              0008          INTSRV=#0400        ..ADD OF SERVICE ROUT.
0000 ;              0009
0000 ;              0010          ORG #0200
0200 D3;            0011 TOP1:    SEP R3              ..GO TO SERVICE ROUTINE
0201 22;            0012 ENTRY:   DEC R2              ..GO TO FREE STACK
0202 78;            0013          SAV                 ..SAVE  X,P
0203 22;            0014          DEC R2              ..DEC POINTER
0204 73;            0015          STXD                ..SAVE D REGISTER
0205 76;            0016          SHRC                ..GET DF IN D
0206 73;            0017          STXD                ..STORE ON STACK
0207 83;            0018          GLO R3              ..SAVE THE
0208 73;            0019          STXD                ..PROGRAM COUNTER
0209 93;            0020          GHI R3              ..REGISTER R3
020A 73;            0021          STXD
020B 84;            0022          GLO R4              .. SAVE LOW PART
020C 73;            0023          STXD                .. OF R4
020D 85;            0024          GLO R5              .. AND AS WELL
020E 73;            0025          STXD                .. R5.0
020F F8E4;          0026          LDI A.0(CALL)       ..INIT
0211 A4;            0027          PLO R4              ..R4.0
0212 F8F4;          0028          LDI A.0(RET)        ..AND
0214 A5;            0029          PLO R5              ..R5.0
0215 F804;          0030          LDI A.1(INTSRV)     ..LOAD NOW ADDRESS
0217 B3;            0031          PHI R3              ..OF INTERRUPT
0218 F800;          0032          LDI A.0(INTSRV)     ..SERVICE ROUTINE
021A A3;            0033          PLO R3              ..IN R3
021B 3000;          0034          BR TOP1             ..AND BRANCH TO TOP1
021D ;              0035
021D ;              0036
021D ;              0037
021D 70;            0038 TOP2:    RET                 ..GET X,P BACK
021E E2;            0039 INEXIT:  SEX R2              ..SET X=2 FIRST
021F 60;            0040          IRX                 ..PREPARE STACKPTR
0220 72A5;          0041          LDXA;PLO R5         ..RESTORE REGISTERS
0222 72A4;          0042          LDXA;PLO R4
0224 72B3;          0043          LDXA;PHI R3
0226 72A3;          0044          LDXA;PLO R3
0228 72F6;          0045          LDXA;SHR            ..GET DF BACK
022A 72;            0046          LDXA                ..AND D AS WELL
022B 301D;          0047          BR TOP2             ..BRANCH TO TOP2
022D ;              0048          END
0000
```

## CASSETTE INTERFACE USING UT4

A simple cassette interface can be connected to UT4 using its routines to load and store data on a normal cassette.

The terminal and the cassette interface are in parallel.

The command

?MØØØØ 1000 (CR)

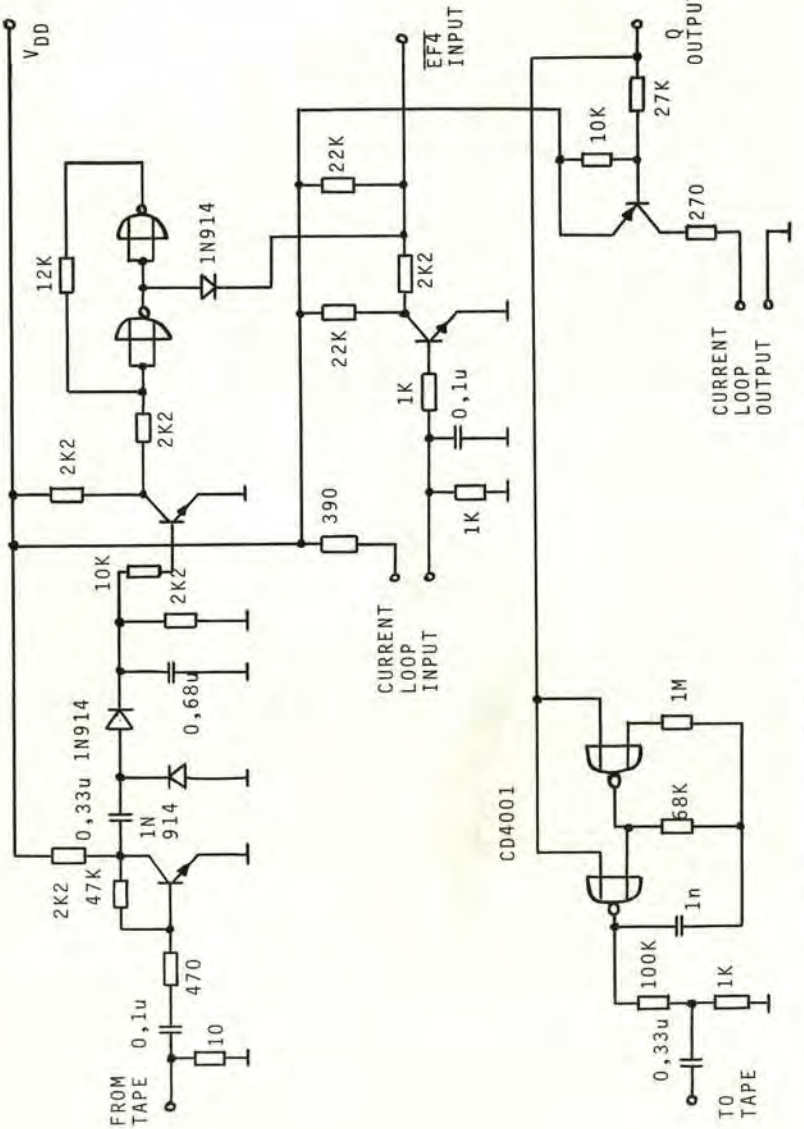
will write 4000 bytes from memory to the terminal, starting at address ØØØØ. The Q output switches an oscillator, consisting of a 1/2 CD4001, on and off. This signal is fed through a voltage divider and then recorded on the tape. (Switch tape to RECORD just before the (CR)).

To get this information back from tape into memory, type via terminal.

!M
and then switch tape recorder to play.

The signal is amplified, rectified and two other gates of the CD4001, shape it and feed it to EF4 where it is ored with the terminal signal.

After having stored the file, UT4 comes back with * and the cassette recorder has to be switched off.




CASSETTE INTERFACE AND CURRENT LOOP INTERFACE USING THE SAME SOFTWARE

# REAL TIME CLOCK WITHOUT HARDWARE

A real time clock function can be included in COSMAC programs without additional hardware. Connecting SC1 (STATE CODE 1) and $\overline{\text{DMA OUT}}$ (see Fig. 1) results in an automatic increment of RØ after every instruction. The fetch-execute is modified to fetch-execute-DMA. The clock frequency is chosen to give a complete wrap-around of RØ in whatever time period is required for the application. For example, a crystal of 1.572864MHz gives 1 second. If a less convenient frequency is used, software techniques can be used to adjust the timing by add and substract.

How does it work :

Normally the SC1 line is low. At the moment when the DMA line is sampled internally, it is therefore sensed as low. So after an execute, a DMA cycle is inserted. Now the SC1 goes high (DMA cycle) and the $\overline{\text{DMA OUT}}$ is high as well. As the DMA request is thus removed, a normal fetch starts, but the DMA pointer has incremented itself. There is no output port for the DMA data as the output function itself is not used : only the increment of RØ is important.

Software implications :

1. Instruction execution time is degraded by a factor of 1,5 because instructions effectively take 3 cycles (fetch - execute - DMA).

2. 3 cycle instructions and interrupts cannot be used in such an application because they distroy the relationship of the DMA count to real-time.

3. Initialization must be interspersed with NOP's or any other instruction while RØ remains program counter. As RØ is PC and DMA pointer, the sequence of the program is :

```
NOP             .. (DMA)
GHI RØ          .. Load register
NOP             .. (DMA)
PHI R3          .. R3 with
NOP             .. (DMA)
LDI #ØB         .. The start location
NOP             .. (DMA)
PLO R3          .. Of the main program
NOP             .. (DMA)
SEP 3           .. And SEP R3
```

As DMA has priority after the reset first a DMA is executed, incrementing RØ pointing now to the GHI RØ. After execution of this instruction another DMA is done. This means that only odd instructions are executed as long as RØ is DMA pointer and PC. Here NOP was chosen as filler, but any other instruction works as well.

4. The main program is reponsible for keeping track of the value in RØ and updating a clock register - which might be simply a seconds counter, or could be a packed BCD storage of real time in seconds, minutes and hours, even including date, month and year.

   This idea is especially useful in data logging applications with very significant saving in hardware and therefore reduced system cost.

For more information, see Application Note ICAN-6677 "Software Control of Microprocessor-Based Real Time Clock"

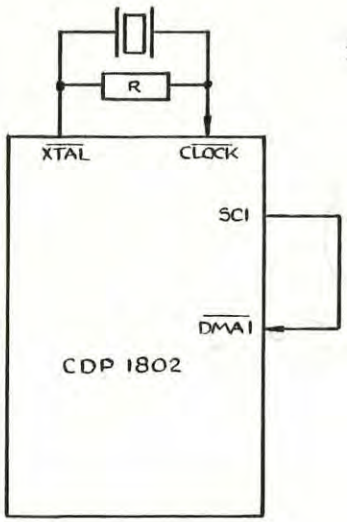



Figure 1 : Hardware For Real Time Clock

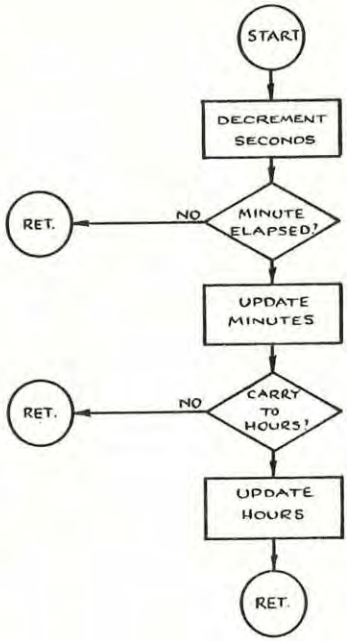



Figure 2 : Flowchart To Update The Clock

## BCD Real-Time Clock

```
SECOND = R9       .. SECONDS COUNTER
CLOCK  = R8       .. POINTS TO STORAGE LOCATION
DISP   = #ø2øø    .. STORAGE LOCATION
MAIN   = R3       .. MAIN PROGRAM COUNTER
STACK  = R2       .. STACK POINTER

EXTCLK: STR CLOCK              .. STORE HOURS
        A.ø (DISP)→CLOCK.ø     .. RESET 'CLOCK'
GOBACK: SEP MAIN               .. EXIT POINT
ENTCLK: DEC SECOND             .. ENTRY POINT
        GLO SECOND
        BNZ GOBACK             .. MINUTE ELAPSED?
        6ø→SECOND.ø            .. RELOAD 'SECOND'
        SEX STACK
MINUTE: @CLOCK.AND.#øF         .. GET LSD MINUTES
        ADI 247
        BDF MIN10              .. CHECK FOR CARRY
        SMI 246                .. IF NO CARRY ADD 1
MIN1ø:  STR STACK              ..  AND STORE
        @CLOCK.AND.#Fø         .. GET MSD MINUTES
        BNF MINFIN             .. IF NO DF DONE
        ADI 176
        BDF MINFIN             .. CHECK FOR CARRY
        SMI 160                .. IF NO CARRY ADD 1
MINFIN: &.OR.@→@CLOCK          .. COMBINE AND STORE
        BNF EXTCLK             .. IF NO DF DONE
HOURS:  INC CLOCK              .. POINT TO HOURS
        @CLOCK+221
        BDF EXTCLK             .. IF MIDNIGHT EXIT
        BR MINUTE              .. USE SAME CODE
        END
```

# PROM PROGRAMMER FOR CDP18U42

The CMOS EPROM CDP18U42 is the first of a series of UV erasable CMOS PROMS for the COSMAC microprocessor. The content is 2K bits organized as 256 x 8 bit. Normally these EPROMS are programmed using the facilities of the development system. Its programmer package performs sophisticated program and verify functions in addition to saving data files and printing program lists.

However, since the CDP18U42 is relatively easy to program, the user may design and build a low cost PROM programmer circuit to perform the basic programming functions. Two circuits are suggested in Figures 1 and 2.

The design of Figure 1 uses LED's to display data and addresses. Two modes of operation are available : PROGRAM and VERIFY. In the program mode, data is entered in binary using eight toggle switches at the address indicated by the address LED's. After releasing the ADVANCE switch, a timing cycle is initiated in which the CDP18U42 is put in the program mode and the program voltage (20V) is applied to the $V_{DD}$, $V_{SAT}$ pins. At the end of this timing cycle the address counter is incremented by one and the CDP18U42 is ready to accept the next data byte.

After programming is completed, the CDP18U42 may be checked by placing the PGM/VERIFY switch in the VERIFY position. In this mode the CDP18U42 is selected in the read mode and the program voltage circuit is disabled. After depressing the RESET switch, each data byte may be read on the DATA OUTPUT LED's, starting at address #00, using the ADVANCE switch.

The design Figure 2 uses a monitor program together with a terminal and UT4 to program the CDP18U42.

To program a location, input

!MXX YY (CR)

and to verify, type

?MXX YY (CR)

In this circuit, the location of the EPROM is programmed directly.

Adding a RAM, and using a small MEMORY MOVE routine, programs can be typed in the RAM and after verification the program is transferred from the RAM into the EPROM using a MEMORY MOVE routine and coming back to the monitor via LBR 8000.

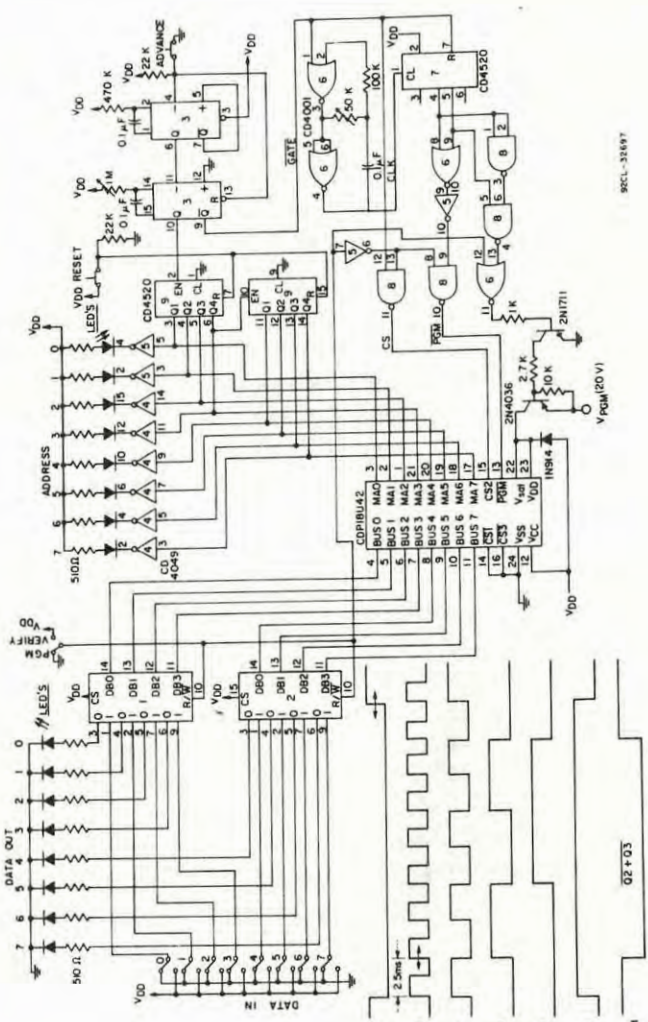



Figure 1 : PROM Programmer Using Switches

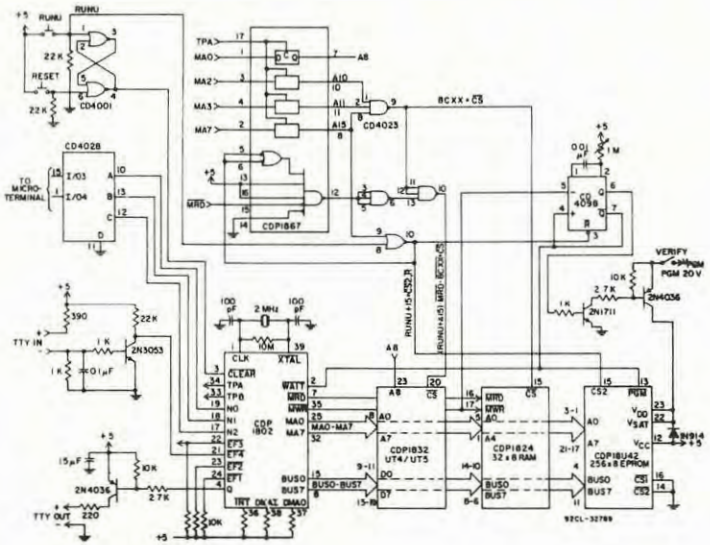
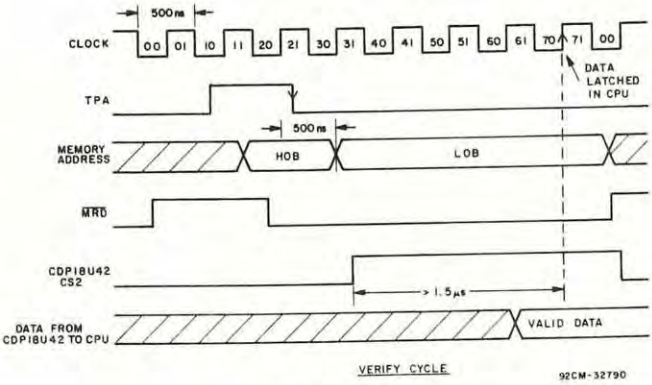
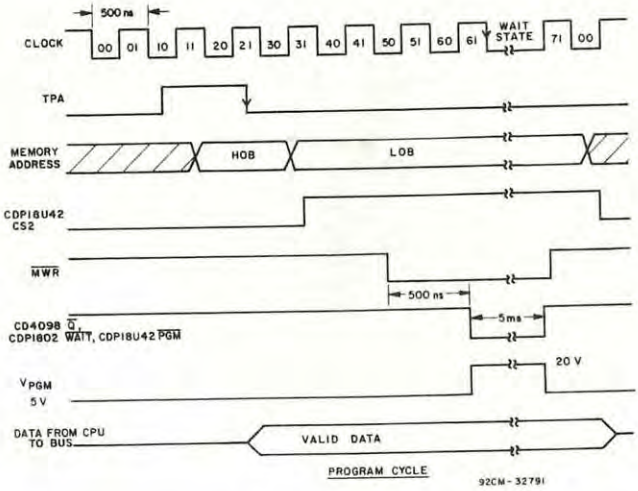



Figure 2 : PROM Programmer Using Serial Interface and UT4

CDP1802-Based PROM Programmer Circuit Timing




Verify Cycle




Program Cycle

## USING THE 1802 SCRATCHPAD TO STORE RAM VARIABLES

Small systems with modest RAM requirements can sometimes be implemented without external RAM by using a portion of the 1802 scratchpad register array to store variable data. Since the scratchpad can be configured for 16-bit addresses or 8-bit data, a typical small system could allocate 8 registers for pointer addressing, and still leave 8 registers for up to 16 bytes of "RAM" storage.

A difficulty arises when attempting to perform an arithmetic or ALU operation on register data - or an output instruction -: these operations require M(R(X)) and the D register as operands . Register to D manipulations can be performed with PHI, GHI, PLO, and GLO instructions, but R(X) cannot point to an internal register to complete the operation.

The problem can be solved if one page of ROM is available for use as a lookup table. With this method, one register operand becomes the lower order table pointer address, while the other operand is transferred to D. The lookup table contains sequential bytes from ØØ to FF, and when the arithmetic or ALU operation is performed, the table contents and D are operated upon, with the result in the D register.

EXAMPLE

- Register data is stored in R(9).1 and R(B).0
- An XOR instruction is to be performed M(R(X)) + D → D
- Lookup table is located in locations Ø3ØØ - Ø3FF
- R(7) is dedicated as lookup table pointer
- R(7).1 has already been initialized to Ø3

```
MACRO:
GHI R9 . . . Get first operand into D
PLO R7 . . . Use first operand to lookup table value
GLO RB . . . Get second operand into D
XOR    . . . Exclusive-OR D with contents of table
                address


if  R(9).1 = AA and R(B).0 = FF
then R(7) would point to address Ø3AA
     Contents of Ø3AA = AA
     M(R(X)) + D = AA + FF = AA
     D will contain AA when operation is complete
```

## DEBUGGING AID

One of the most useful tools for debugging is a short program that allows to have an interactive look at the state of the CPU. As the CDP1802 has 16 internal 16 bit registers plus D, DF, X, P, T it is rather difficult to see why something is wrong in a program if you cannot have a look at them.

The most useful tool for this purpose in the RCA MICRO-MONITOR. It consists of a CDP1802 microprocessor system, controlling the target system.

For a minimum system and to understand how a debug program works, this small aid explains how to do it completely in software.

As this is a program as well, it needs a program counter and a data pointer.

It has to have a "hard break" . This means that it has to be possible to get out of a loop. The easiest way is to RESET the microprocessor.

In this state, RØ starts to be program counter and starts at ØØØØ, so this mini debug is located at ØØØØ.

As data pointer R8 is chosen.

The "soft break" is done by using the DØ instruction.

Using this debug program does not allow the usage of RØ, R8, and only programs in RAM can be debugged. If RØ is modified, the "soft break" does not work, only RESET.

If R8 is modified, the D register is lost but the registers R1 to R7 are still saved together with DF.

At locations, where a break is wanted, a SEP Ø is placed overwriting the program byte at this location, so the program has to be in RAM.

As many "soft breaks" as needed can be used during a debug session.

This debug can easily be changed to save more registers or use others for the DEBUG itself.

```
0000! ;                 0001        ..CP RCA BXL
0000  ;                 0002        ..SMALL DEBUG PROGRAM
0000  ;                 0003        ..SAVES SOME REGISTERS, D, DF
0000  ;                 0004        ..USING SOFT BREAKPOINT "D0"
0000  ;                 0005        ..OR RESET AS HARD BRAKE
0000  ;                 0006        ..R0 IS USED AS PC
0000  ;                 0007        ..R8 IS DATA POINTER
0000! ;                 0008        ..FOR UT4 BUT NOT RESTRICTED
0000  ;                 0009        ..RUNS IN RAM OR ROM
0000  ;                 0010        ..TO TEST PROGRAMS IN RAM
0000  ;                 0011
0000  ;                 0012
0000  ;                 0013        ..          #0000
0000  ;                 0014        ..
0000  ;                 0015        ..
0000  ;                 0016        ..    BRANCH TO SAVE          RESET OR HARD BRAKE
0000  ;                 0017
0000  ;                 0018
0000  ;                 0019        ..
0000  ;                 0020        ..                            ENTRY PROGRAM FROM UT4
0000  ;                 0021        ..    RESTORE REGISTERS
0000  ;                 0022        ..    START EXECUTION
0000  ;                 0023        ..                            GO TO USER PROGRAM
0000  ;                 0024
0000  ;                 0025
0000  ;                 0026        ..                  P         FOUND D0 IN USER PROG
0000  ;                 0027        ..
0000  ;                 0028        ..    SAVE SOME REGISTERS
0000  ;                 0029        ..    D AND DF
0000  ;                 0030        ..
0000  ;                 0031        ..
0000  ;                 0032        ..    GO TO MONITOR PROGRAM
0000  ;                 0033        ..    AT #8000  P=0  X=8
0000  ;                 0034
0000  ;                 0035
0000  ;                 0036
0000  ;                 0037
0000  ;                 0038
0000  ;                 0039        ..    MEMORY USAGE   FOR 1802601  BOARD
0000  ;                 0040        ..    4FE0  WANTED X,P
0000  ;                 0041        ..    4FE1  D REGISTER
0000  ;                 0042        ..    4FE2  R1
0000  ;                 0043        ..    4FE4  R2
0000  ;                 0044        ..    4FE6  R3
0000  ;                 0045        ..    4FE8  R4
0000  ;                 0046        ..    4FEA  R5
0000  ;                 0047        ..    4FEC  R6
0000  ;                 0048        ..    4FEE  R7
0000  ;                 0049        ..    4FF0  DF    SHOWS IF D IS OK
0000  ;                 0050
0000  ;                 0051
```

```
0000 ;              0052          ..THIS SHORT DEBUG PROGRAM
0000 ;              0053          ..SAVES AND RESTORES PART OF THE
0000 ;              0054          ..CDP 1802 STATE USING
0000 ;              0055          ..D0 AS SOFTWARE BREAKPOINT
0000 ;              0056
0000 ;              0057          ORG #0000
0000 C00035;        0058          LBR SAVE                ..THIS IS AN ENTRY AFTER RESET
0003 ;              0059                                  ..OR HARD BRAKE
0003 ;              0060          ..RESTORE REGISTERS AND START EXECUTION
0003 ;              0061          ..OF USER PROGRAM WITH $P3(CR)
0003 ;              0062          ..WITH X,P FROM LOCATION #4FE0
0003 ;              0063
0003 F808;          0064          LDI #08                 ..A SHORT DELAY
0005 B8;            0065          PHI R8
0006 28;            0066 LOOP:    DEC R8
0007 98;            0067          GHI R8
0008 3A06;          0068          BNZ LOOP
000A ;              0069
000A F84FB8;        0070          LDI #4F;PHI R8          ..LOAD DATA POINTER
000D F8E2A8;        0071          LDI #E2;PLO R8          ..R8 WITH #4FE2
0010 E8;            0072          SEX R8                  ..SET X TO 8
0011 ;              0073
0011 72B1;          0074          LDXA;PHI R1             ..RESTORE REGISTERS
0013 72A1;          0075          LDXA;PLO R1             ..R1 TO R7
0015 72B2;          0076          LDXA;PHI R2
0017 72A2;          0077          LDXA;PLO R2
0019 72B3;          0078          LDXA;PHI R3
001B 72A3;          0079          LDXA;PLO R3
001D 72B4;          0080          LDXA;PHI R4
001F 72A4;          0081          LDXA;PLO R4
0021 72B5;          0082          LDXA;PHI R5
0023 72A5;          0083          LDXA;PLO R5
0025 72B6;          0084          LDXA;PHI R6
0027 72A6;          0085          LDXA;PLO R6
0029 72B7;          0086          LDXA;PHI R7
002B 72A7;          0087          LDXA;PLO R7
002D ;              0088
002D F076;          0089          LDX;SHRC                ..GET DF BACK
002F ;              0090
002F F8E1A8;        0091          LDI #E1;PLO R8          ..SET ADD OF D
0032 F0;            0092          LDX                     ..AND LOAD
0033 ;              0093
0033 28;            0094          DEC R8                  ..POINT TO X,P
0034 70;            0095          RET                     ..DIS TO DISABLE INTERRUPTS
0035 ;              0096
0035 ;              0097          ..NOW THE USER PROGRAM IS RUNNING
0035 ;              0098          ..WITH THE SUGGESTED X,P FROM 4FE0
0035 ;              0099          ..IT COMES BACK TO DEBUG VIA A SEP 0
0035 ;              0100          ..IT STORES PART OF THE 1802 REGISTERS
0035 ;              0101          ..AND BRANCHES TO #0000
0035 ;              0102          ..0I4 WAITS NOW FOR CR OR LF
```

```
0035 ;              0103
0035 ;              0104          ..AS R8 AND R0 HAVE NOT BEEN
0035 ;              0105          ..CHANGED HOPEFULLY,A SEP 0
0035 ;              0106          ..IN THE USER PROGRAM WILL
0035 ;              0107          ..START THE SAVE ROUTINE
0035 ;              0108          ..WITH R8 POINTING TO #4FF1
0035 ;              0109
0035 E0;            0110 SAVE:    SEX R0                    ..X TO IMMEDIATE
0036 7180;          0111          DIS,#80                   ..TO DIS INTERRUPTS
0038 58;            0112          STR R8                    ..SAVE D
0039 88;            0113          GLO R8                    ..CHECK DATA POINTER
003A F8E1;          0114          XRI #E1
003C 3A43;          0115          BNZ REPAIR                ..IF CHANGED GO TO REPAIR
003E 98;            0116          GHI R8
003F F84F;          0117          XRI #4F
0041 324C;          0118          BZ DOK
0043 ;              0119
0043 F84FB8;        0120 REPAIR:  LDI #4F;PHI R8            ..SET DATA POINTER
0046 F8F0A8;        0121          LDI #F0;PLO R8
0049 58;            0122          STR R8                    ..STORE F0 AT DF
004A 3052;          0123          BR SDF
004C ;              0124
004C F8F0A8;        0125 DOK:     LDI #F0;PLO R8            ..SET R8 TO DF LOCATION
004F F800580;       0126          LDI #00;STR R8            ..STORE D IS OK
0052 ;              0127
0052 E8;            0128 SDF:     SEX R8
0053 7EF4;          0129          SHLC;OR                   ..SAVE DF
0055 73;            0130          STXD
0056 ;              0131
0056 8773;          0132          GLO R7;STXD               ..SAVE R4 TO R7
0058 9773;          0133          GHI R7;STXD
005A 8673;          0134          GLO R6;STXD
005C 9673;          0135          GHI R6;STXD
005E 8573;          0136          GLO R5;STXD
0060 9573;          0137          GHI R5;STXD
0062 8473;          0138          GLO R4;STXD
0064 9473;          0139          GHI R4;STXD
0066 8373;          0140          GLO R3;STXD
0068 9373;          0141          GHI R3;STXD
006A 8273;          0142          GLO R2;STXD
006C 9273;          0143          GHI R2;STXD
006E 8173;          0144          GLO R1;STXD
0070 9173;          0145          GHI R1;STXD
0072 ;              0146
0072 C08000;        0147 MONIT:   LBR #8000                 ..BRANCH TO #8000
0075 ;              0148                                    ..U14 WAITS FOR CR OR LF
0075 ;              0149
0075 ;              0150          ..NOW ?M AND !M CAN BE USED TO MODIFY
0075 ;              0151          ..MEMORY AND WHEN USER PROGRAM IS STARTED,
0075 ;              0152          ..THESE VALUES WILL BE TRANSFERRED
0075 ;              0153          ..INTO THE CDP1802
```

```
0075 ;                  0154            ..
0075 ;                  0155            ..IF THE EXIT TO UT4
0075 ;                  0156            ..IS CHANGED A LITTLE BIT
0075 ;                  0157            ..THE STATE IS DISPLAYED
0075 ;                  0158            ..IMMEDIATELY AFTER A D0 IN USER PROGRAM
0075 ;                  0159
0075 ;                  0160
0075 F880BC;            0161            LDI #80;PHI RC          ..SET DELAY POINTER
0078 F8EFAC;            0162            LDI #EF;PLO RC
007B ;                  0163
007B F8288E;            0164            LDI #28;PHI RE          ..TIME CONSTANT
007E F800AE;            0165            LDI #00;PLO RE          ..AND ECHO BIT
0081 ;                  0166
0081 F881B3;            0167            LDI #81;PHI R3          ..PREPARE FOR TYPESD
0084 F800BD;            0168            LDI #00;PHI RD          ..TO TYPE
0087 F811AD;            0169            LDI #11;PLO RD          ..17 BYTES
008A ;                  0170
008A F800B4;            0171            LDI #00;PHI R4          ..TELL IT IS A
008D F811A4;            0172            LDI #11;PLO R4          ..?M SEQUENCE
0090 ;                  0173
0090 F800B5;            0174            LDI #00;PHI R5          ..PREPARE R5
0093 F896A5;            0175            LDI A.0(BACK);PLO R5
0096 ;                  0176
0096 D5;                0177 BACK:      SEP R5                  ..USE R5 AS PC
0097 ;                  0178
0097 F84FB0;            0179            LDI #4F;PHI R0          ..LOAD DISPLAY POINTER
009A F8E0A0;            0180            LDI #E0;PLO R0
009D ;                  0181
009D C0808E;            0182            LBR #808E               ..ENTER UT4
00A0 ;                  0183
00A0 ;                  0184            END
0000
```

RCA
Solid State
Division

# Microprocessor Products
# CDP1802D
# CDP1802CD

File Number 1023

40-Lead Dual-In-Line Ceramic Package (D)

CDP1802D
CDP1802CD H-1847

## COSMAC Microprocessor

*Features:*

- Instruction fetch-execute time of 2.5 or 3.75 μs at $V_{DD}$ = 10 V; 5.0 or 7.5 μs at $V_{DD}$ = 5 V
- Static silicon-gate CMOS circuitry — no minimum clock frequency
- Full military temperature range (−55 to +125°C)
- High noise immunity, wide operating-voltage range
- Single voltage supply
- Low power
- Single-phase clock; optional on-chip crystal-controlled oscillator
- TTL compatible
- On-chip DMA
- Simple control of reset, run, and pause
- 8-bit parallel organization with bidirectional data bus
- Any combination of standard RAM and ROM
- Memory addressing up to 65,536 bytes
- Flexible programmed I/O mode
- Program interrupt mode
- Four I/O flag inputs directly tested by branch instructions
- Programmable output port
- 91 easy-to-use instructions
- 16 x 16 matrix of registers for use as multiple program counters, data pointers, or data registers

The RCA-CDP1802 is an LSI COS/MOS 8-bit register-oriented central-processing unit (CPU) designed for use as a general-purpose computing or control element in a wide range of stored-program systems or products.

The CDP1802 includes all of the circuits required for fetching, interpreting, and executing instructions which have been stored in standard types of memories. Extensive input/output (I/O) control features are also provided to facilitate system design.

The COSMAC architecture is designed with emphasis on the total microcomputer system as an integral entity so that systems having maximum flexibility and minimum cost can be realized. The COSMAC CPU also provides a synchronous interface to memories and external controllers for I/O devices, and minimizes the cost of interface controllers. Further, the I/O interface is capable of supporting devices operating in polled, interrupt-driven, or direct memory-access modes.

The CDP1802D and CDP1802CD are functionally identical. They differ in that the CDP1802D has a recommended operating voltage range of 4-12 volts, and the CDP 1802CD, a recommended operating voltage range of 4-6 volts. These types are supplied in 40-lead dual-in-line ceramic packages (D suffix).

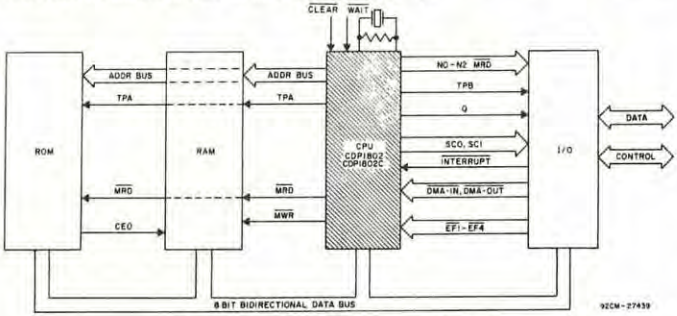



*Fig. 1 — Typical CDP1802 microprocessor system.*



Printed in USA/2-78



*Supersedes preliminary data issued 8/77*

**MAXIMUM RATINGS,** *Absolute-Maximum Values:*

DC SUPPLY-VOLTAGE RANGE, ($V_{CC}$, $V_{DD}$)
(All voltage values referenced to $V_{SS}$ terminal)
$V_{CC} \leq V_{DD}$:
CDP1802D . . . . . −0.5 to +15 V
CDP1802CD . . . . . −0.5 to +7 V
INPUT VOLTAGE RANGE, ALL INPUTS . . . . . −0.5 to $V_{DD}$ +0.5 V
DC INPUT CURRENT, ANY ONE INPUT . . . . . ±10 mA
POWER DISSIPATION PER PACKAGE ($P_D$):
For $T_A$ = −55 to +100°C . . . . . 500 mW
For $T_A$ = +100 to +125°C . . . . . Derate Linearly at 12 mW/°C to 200 mW
DEVICE DISSIPATION PER OUTPUT TRANSISTOR
FOR $T_A$ = FULL PACKAGE-TEMPERATURE RANGE . . . . . 100 mW
OPERATING-TEMPERATURE RANGE ($T_A$) . . . . . −55 to +125°C
STORAGE TEMPERATURE RANGE ($T_{stg}$) . . . . . −65 to +150°C
LEAD TEMPERATURE (DURING SOLDERING):
At distance 1/16 ± 1/32 inch (1.59 ± 0.79 mm) from case for 10 s max. . . . . +265°C

| CHARACTERISTIC | CONDITIONS | | | LIMITS AT INDICATED TEMPERATURES (°C) | | | | | | | UNITS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | VALUES | | | | +25 | | | |
| | $V_O$ (V) | $V_{IN}$ (V) | $V_{CC}$, $V_{DD}$ (V) | −55 | −40 | +85 | +125 | Min. | Typ. | Max. | |
| Quiescent Device Current, $I_L$ Max. | – | – | 5 | – | – | – | – | – | 1 | 100 | μA |
| | – | – | 10 | – | – | – | – | – | 10 | 500 | |
| CDP1802D | – | – | 15 | – | – | – | – | – | – | 1000 | |
| CDP1802CD | – | – | 5 | – | – | – | – | – | – | 500 | |
| Output Low Drive (Sink) Current, $I_{OL}$ Min. | 0.4 | 0,5 | 5 | 1.98 | 1.89 | 1.14 | 0.90 | 1.5 | 2.2 | – | mA |
| (Except $\overline{XTAL}$) | 0.5 | 0,10 | 10 | 3.70 | 3.53 | 2.13 | 1.68 | 2.8 | 5.2 | – | |
| $\overline{XTAL}$ Output $I_{OL}$ Min. | 0.4 | 5 | 5 | 132 | 126 | 76 | 60 | 100 | – | – | μA |
| Output High Drive (Source Current) $I_{OH}$ Min. (Except $\overline{XTAL}$) | 4.6 | 0,5 | 5 | −0.46 | −0.44 | −0.27 | −0.21 | −0.35 | −0.51 | – | mA |
| | 9.5 | 0,10 | 10 | −1.12 | −1.07 | −0.65 | −0.51 | −0.85 | −1.3 | | |
| $\overline{XTAL}$ Output $I_{OH}$ Min. | 4.6 | 0 | 5 | −66 | −63 | −38 | −30 | −50 | – | – | μA |
| Output Voltage Low-Level $V_{OL}$ Max. | – | 0,5 | 5 | 0.05 | | | | – | 0 | 0.05 | V |
| | – | 0,10 | 10 | 0.05 | | | | – | 0 | 0.05 | |
| Output Voltage High Level, $V_{OH}$ Min. | – | 0,5 | 5 | 4.95 | | | | 4.95 | 5 | – | |
| | – | 0,10 | 10 | 9.95 | | | | 9.95 | 10 | – | |
| Input Low Voltage $V_{IL}$ Max. | 0.5,4.5 | – | 5 | 1.5 | | | | – | – | 1.5 | V |
| | 0.5,4.5 | – | 5,10 | 1 | | | | – | – | 1 | |
| | 1,9 | – | 10 | 3 | | | | – | – | 3 | |
| Input High Voltage $V_{IL}$ Min. | 0.5,4.5 | – | 5 | 3.5 | | | | 3.5 | – | – | |
| | 0.5,4.5 | – | 5,10 | 4 | | | | 4 | – | – | |
| | 1,9 | – | 10 | 7 | | | | 7 | – | – | |
| Input Leakage Current $I_{IN}$ Max. | Any Input | 0,15 | 15 | ±1 | | | | – | – | ±1 | μA |
| 3-State Output Leakage Current $I_{OUT}$ Max. | 0,15 | 0,15 | 15 | ±1 | ±1 | ±12 | ±12 | – | $\pm 10^{-4}$ | ±1 | μA |

**RECOMMENDED OPERATING CONDITIONS at $T_A$ = 25°C Unless Otherwise Specified**

*For maximum reliability, nominal operating conditions should be selected so that operation is always within the following ranges:*

| CHARACTERISTIC | CONDITIONS | | LIMITS AT 25°C | | UNITS |
|---|---|---|---|---|---|
| | $V_{CC}$[1] (V) | $V_{DD}$ (V) | CDP1802D | CDP1802CD | |
| Supply-Voltage Range | – | – | 4 to 12 | 4 to 6 | V |
| Input Voltage Range | – | – | $V_{SS}$ to $V_{CC}$ | $V_{SS}$ to $V_{CC}$ | V |
| Maximum Clock Input Rise or Fall Time, $t_r$ or $t_f$ | 4–12 | 4–12 | 1 | 1 | μs |
| Instruction Time[2] (See Fig. 8) | 5 | 5 | 5 | 5 | μs |
| | 5 | 10 | 4 | – | |
| | 10 | 10 | 2.5 | – | |
| Maximum DMA Transfer Rate | 5 | 5 | 400 | 400 | KBytes/sec |
| | 5 | 10 | 500 | – | |
| | 10 | 10 | 800 | – | |
| Maximum Clock Input Frequency, $f_{CL}$[3] | 5 | 5 | DC – 3.2 | DC – 3.2 | MHz |
| | 5 | 10 | DC – 4 | – | |
| | 10 | 10 | DC – 6.4 | – | |

NOTES:
1. $V_{CC} \leqslant V_{DD}$; for CDP1802CD, $V_{DD} = V_{CC} = 5$ volts.
2. Equals 2 machine cycles – one Fetch and one Execute operation for all instructions except Long Branch and Long Skip, which require 3 machine cycles – one Fetch and two Execute operations.
3. Load Capacitance ($C_L$) = 50 pF.

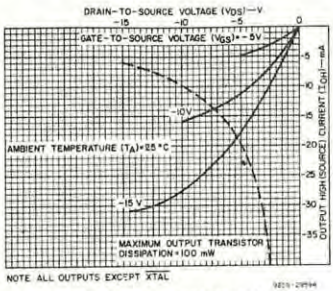



*Fig. 2 – Typical output high (source) current characteristics.*

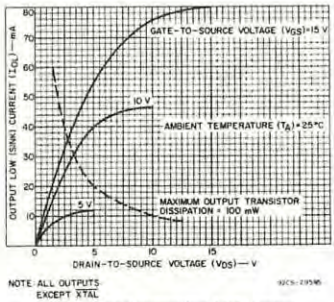



*Fig. 3 – Typical output low (sink) current characteristics.*

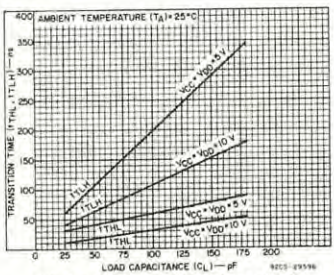



*Fig. 4 – Typical transition time vs. load capacitance.*

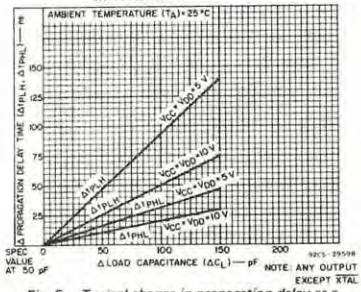



*Fig. 5 – Typical change in propagation delay as a function of a change in load capacitance.*

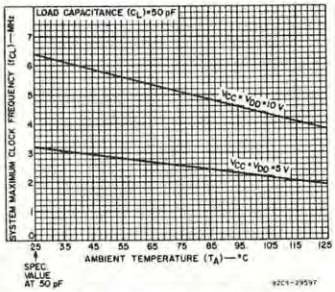



*Fig. 6 — Typical maximum clock frequency as a function of temperature.*

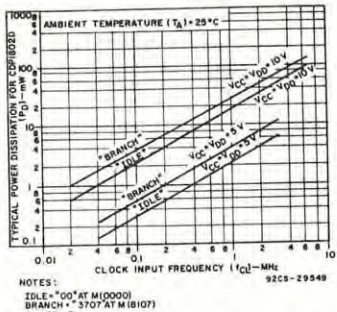



*Fig. 7 — Typical power dissipation as a function of clock frequency for BRANCH instruction and IDLE instruction for CDP1802D.*

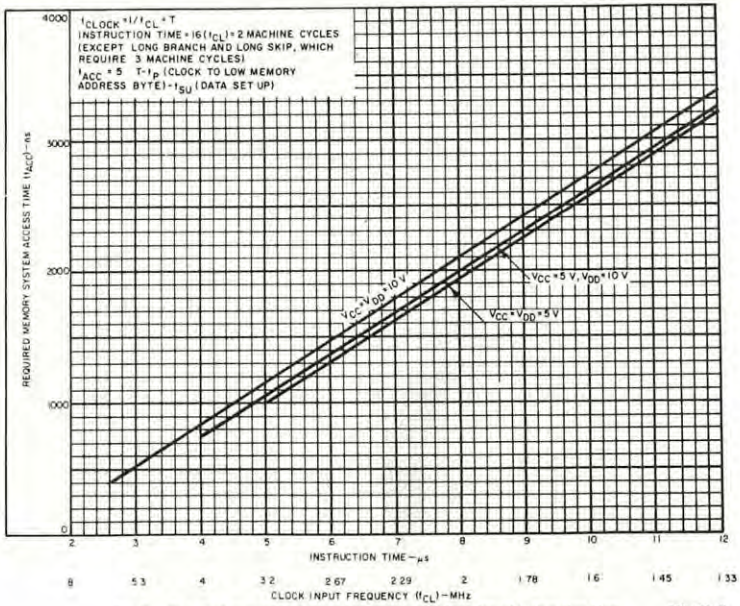



*Fig. 8 — Required memory system address time as a function of instruction time.*

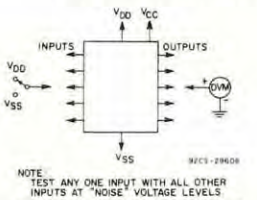



*Fig. 9 — Noise immunity test circuit.*

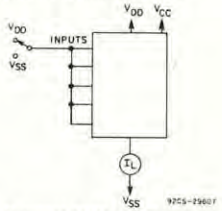



*Fig. 10 — Quiescent-device leakage current test circuit.*

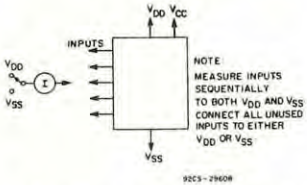



*Fig. 11 – Input leakage current test circuit.*

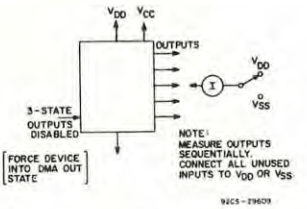



*Fig. 12 – Three-state output leakage (data bus) test circuit.*

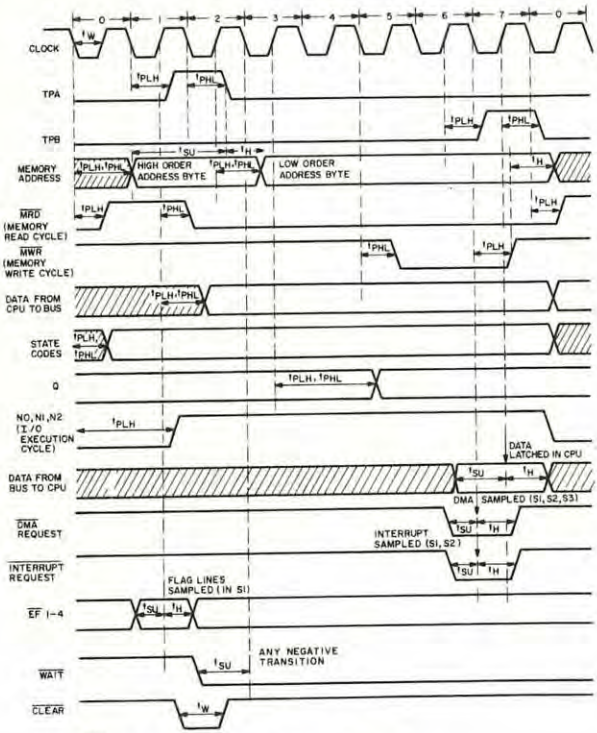



*Fig. 13 – Timing waveforms.*

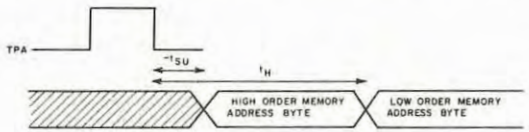
**DYNAMIC ELECTRICAL CHARACTERISTICS at $T_A$ = 25°C, $C_L$ = 50 pF**

| CHARACTERISTIC | | $V_{CC}$ (V) | $V_{DD}$ (V) | LIMITS Min. | Typ. | Max. | UNITS |
|---|---|---|---|---|---|---|---|
| Propagation Delay Time, $t_{PLH}$, $t_{PHL}$: Clock to TPA, TPB | | 5 | 5 | – | 300 | 450 | ns |
| | | 5 | 10 | – | 250 | 400 | |
| | | 10 | 10 | – | 150 | 250 | |
| Clock-to-Memory High-Address Byte | | 5 | 5 | – | 800 | 1200 | ns |
| | | 5 | 10 | – | 600 | 900 | |
| | | 10 | 10 | – | 400 | 600 | |
| Clock-to-Memory Low-Address Byte | | 5 | 5 | – | 300 | 550 | ns |
| | | 5 | 10 | – | 250 | 500 | |
| | | 10 | 10 | – | 150 | 350 | |
| Clock to $\overline{MRD}$, $t_{PLH}$ | | 5 | 5 | – | 300 | 450 | ns |
| | | 5 | 10 | – | 250 | 400 | |
| | | 10 | 10 | – | 150 | 300 | |
| Clock to $\overline{MRD}$, $t_{PHL}$ | | 5 | 5 | – | 300 | 450 | ns |
| | | 5 | 10 | – | 250 | 400 | |
| | | 10 | 10 | – | 150 | 300 | |
| Clock to $\overline{MWR}$, $t_{PLH}$, $t_{PHL}$ | | 5 | 5 | – | 300 | 450 | ns |
| | | 5 | 10 | – | 200 | 300 | |
| | | 10 | 10 | – | 150 | 250 | |
| Clock to CPU DATA to BUS | | 5 | 5 | – | 350 | 600 | ns |
| | | 5 | 10 | – | 300 | 500 | |
| | | 10 | 10 | – | 200 | 400 | |
| Clock to State Code | | 5 | 5 | – | 400 | 600 | ns |
| | | 5 | 10 | – | 200 | 400 | |
| | | 10 | 10 | – | 150 | 300 | |
| Clock to Q | | 5 | 5 | – | 300 | 700 | ns |
| | | 5 | 10 | – | 150 | 400 | |
| | | 10 | 10 | – | 100 | 300 | |
| Clock to N(0-2), $t_{PLH}$ | | 5 | 5 | – | 450 | 800 | ns |
| | | 5 | 10 | – | 300 | 600 | |
| | | 10 | 10 | – | 200 | 400 | |
| High-Order Memory-Address Byte Set Up, $t_{SU}$ (See Note) | f = 4 MHz | 5 | 10 | 0 | – | – | ns |
| | f = 6.4 MHz | 10 | 10 | −50 | – | – | |
| | f = 2 MHz | 5 | 5 | 50 | – | – | |
| | f = 5 MHz | 10 | 10 | 30 | – | – | |
| High-Order Memory-Address Byte Hold $t_H$ | f = 4 MHz | 5 | 10 | 120 | – | – | ns |
| | f = 6.4 MHz | 10 | 10 | 75 | – | – | |
| | f = 2 MHz | 5 | 5 | 200 | – | – | |
| | f = 5 MHz | 10 | 10 | 100 | – | – | |
| Low-Order Memory-Address Hold | f = 4 MHz | 5 | 10 | 100 | – | – | ns |
| | f = 6.4 MHz | 10 | 10 | 50 | – | – | |

**DYNAMIC ELECTRICAL CHARACTERISTICS (cont'd)**

| CHARACTERISTIC | $V_{CC}$ (V) | $V_{DD}$ (V) | LIMITS Min. | Typ. | Max. | UNITS |
|---|---|---|---|---|---|---|
| Set-Up and Hold Times, $t_{SU}$, $t_H$ | 5 | 5 | 0 | −50 | – | |
| Data Set Up | 5 | 10 | 25 | 0 | – | |
| | 10 | 10 | 50 | 0 | – | ns |
| Data Hold | 5 | 5 | 300 | 150 | – | |
| | 5 | 10 | 200 | 100 | – | |
| | 10 | 10 | 150 | 75 | – | |
| $\overline{DMA}$ Set Up | 5 | 5 | 100 | 0 | – | |
| | 5 | 10 | 125 | 25 | – | |
| | 10 | 10 | 150 | 50 | – | ns |
| $\overline{DMA}$ Hold | 5 | 5 | 250 | 150 | – | |
| | 5 | 10 | 200 | 100 | – | |
| | 10 | 10 | 150 | 75 | – | |
| Interrupt Set Up | 5 | 5 | 100 | 0 | – | |
| | 5 | 10 | 125 | 25 | – | |
| | 10 | 10 | 150 | 50 | – | ns |
| Interrupt Hold | 5 | 5 | 250 | 150 | – | |
| | 5 | 10 | 200 | 100 | – | |
| | 10 | 10 | 150 | 75 | – | |
| $\overline{WAIT}$ Set Up | 5 | 5 | 100 | 0 | – | |
| | 5 | 10 | 125 | 25 | – | ns |
| | 10 | 10 | 150 | 50 | – | |
| $\overline{EF1\text{-}4}$ Set Up | 5 | 5 | 100 | 0 | – | |
| | 5 | 10 | 125 | 25 | – | |
| | 10 | 10 | 150 | 50 | – | ns |
| $\overline{EF1\text{-}4}$ Hold | 5 | 5 | 250 | 150 | – | |
| | 5 | 10 | 200 | 100 | – | |
| | 10 | 10 | 150 | 75 | – | |
| Pulse Width, $t_{WL}$ | 5 | 5 | 600 | 300 | – | |
| $\overline{CLEAR}$ Pulse Width | 5 | 10 | 400 | 200 | – | ns |
| | 10 | 10 | 300 | 150 | – | |
| $\overline{CLOCK}$ Pulse Width, $t_{WL}$ | 5 | 5 | 160 | – | – | |
| | 5 | 10 | 125 | – | – | ns |
| | 10 | 10 | 80 | – | – | |
| Typical Total Power Dissipation f = 2 MHz | 5 | 5 | – | 4 | – | |
| Idle "00" at M(0000), $C_L$ = 50 pF f = 4 MHz | 10 | 10 | – | 60 | – | mW |
| Effective Input Capacitance, $C_{IN}$ Any Input | | | – | 5 | – | pF |
| Effective 3-State Terminal Capacitance DATA BUS | | | – | 7.5 | – | pF |

NOTE: Negative set-up indicates the addresses can change after the falling edge of TPA, as shown below:




92CS-29644

## ARCHITECTURE

The COSMAC block diagram is shown in Fig. 14. The principal feature of this system is a register array (R) consisting of sixteen 16-bit scratchpad registers. Individual registers in the array (R) are designated (selected) by a 4-bit binary code from one of the 4-bit registers labeled N, P, and X. The contents of any register can be directed to any one of the following three paths:

1. the external memory (multiplexed, higher-order byte first, on to 8 memory address lines);
2. the D register (either of the two bytes can be gated to D);
3. the increment/decrement circuit where it is increased or decreased by one and stored back in the selected 16-bit register.

The three paths, depending on the nature of the instruction, may operate independently or in various combinations in the same machine cycle.

With two exceptions, COSMAC instructions consist of two 8-clock-pulse machine cycles. The first cycle is the fetch cycle, and the second—and third, if necessary—are execute cycles. During the fetch cycle the four bits in the P designator select one of the 16 registers R(P) as the current program counter. The selected register R(P) contains the address of the memory location from which the instruction is to be fetched. When the instruction is read out from the memory, the higher-order 4 bits of the instruction byte are loaded into the I register and the lower-order 4 bits into the N register. The content of the program counter is automatically incremented by one so that R(P) is now "pointing" to the next byte in the memory.

The X designator selects one of the 16 registers R(X) to "point" to the memory for an operand (or data) in certain ALU or I/O operations.

The N designator can perform the following five functions depending on the type of instruction fetched:

1. designate one of the 16 registers in R to be acted upon during register operations;
2. indicate to the I/O devices a command code or device-selection code for peripherals;
3. indicate the specific operation to be executed during the ALU instructions, types of tests to be performed during the Branch instructions, or the specific operation required in a class of miscellaneous instructions (70-73 and 78-7B);
4. indicate the value to be loaded into P to designate a new register to be used as the program counter R(P);

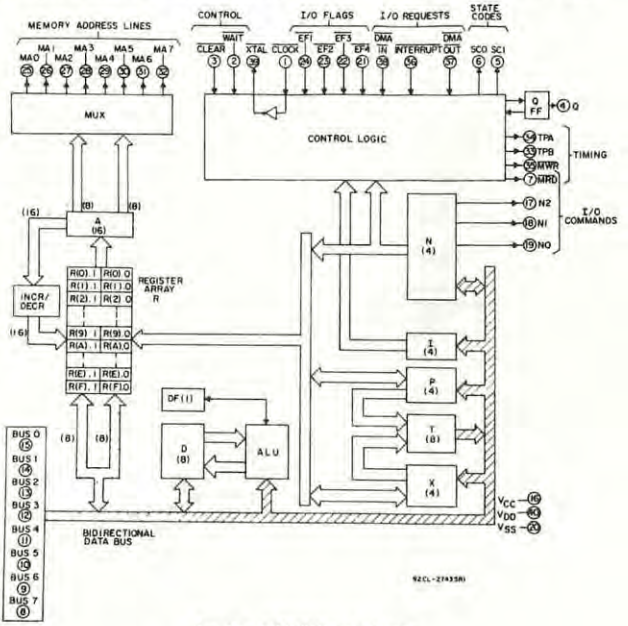



*Fig. 14 – CDP1802 block diagram.*

5. indicate the value to be loaded into X to designate a new register to be used as data pointer R(X).

The registers in R can be assigned by a programmer in three different ways: as program counters, as data pointers, or as scratchpad locations (data registers) to hold two bytes of data.

**Program Counters**

Any register can be the main program counter; the address of the selected register is held in the P designator. Other registers in R can be used as subroutine program counters. By a single instruction the contents of the P register can be changed to effect a "call" to a subroutine. When interrupts are being serviced, register R(1) is used as the program counter for the user's interrupt servicing routine. After reset, and during a DMA operation, R (0) is used as the program counter. At all other times the register designated as program counter is at the discretion of the user.

**Data Pointers**

The registers in R may be used as data pointers to indicate a location in memory. The register designated by X (i.e., R(X)) points to memory for the following instructions (see Table I):

1. ALU operations F1-F5, F7, 74, 75, 77;
2. output instructions 61 through 67;
3. input instructions 69 through 6F;
4. certain miscellaneous instructions—70-73, 78.60, F0.

The register designated by N (i.e., R(N)) points to memory for the "load D from memory" instructions 0N and 4N and the "Store D" instruction 5N. The register designated by P (i.e., the program counter) is used as the data pointer for ALU instructions F8-FD, FF, 7C, 7D, 7F. During these instruction executions, the operation is referred to as "data immediate".

Another important use of R as a data pointer supports the built-in Direct-Memory-Access (DMA) function. When a DMA-In or DMA-Out request is received, one machine cycle is "stolen". This operation occurs at the end of the execute machine cycle in the current instruction. Register R(0) is always used as the data pointer during the DMA operation. The data is read from (DMA-Out) or written into (DMA-In) the memory location pointed to by the R(0) register. At the end of the transfer, R(0) is incremented by one so that the processor is ready to act upon the next DMA byte transfer request. This feature in the COSMAC architecture saves a substantial amount of logic when fast exchanges of blocks of data are required, such as with magnetic discs or during CRT-display-refresh cycles.

A program load facility, using the DMA-In channel, is provided to enable users to load programs into the memory. This facility provides a simple, one-step means for initially entering programs into the microprocessor system and eliminates the requirement for specialized "bootstrap" ROM's.

**Data Registers**

When registers in R are used to store bytes of data, four instructions are provided which allow D to receive from or write into either the higher-order- or lower-order-byte portions of the register designated by N. By this mechanism (together with loading by data immediate) program pointer and data pointer designations are initialized. Also, this technique allows scratchpad registers in R to be used to hold general data. By employing increment or decrement instructions, such registers may be used as loop counters.

**The Q Flip Flop**

An internal flip flop, Q, can be set or reset by instruction and can be sensed by conditional branch instructions. The output of Q is also available as a microprocessor output.

**Interrupt Servicing**

Register R(1) is always used as the program counter whenever interrupt servicing is initiated. When an interrupt request comes in and the interrupt is allowed by the program (again, nothing takes place until the completion of the current instruction) the contents of the X and P registers are stored in the temporary register T, and X and P are set to new values; hex digit 2 in X and hex digit 1 in P. Interrupt enable is automatically deactivated to inhibit further interruptions. The user's interrupt routine is now in control; the contents of T may be saved by means of a single instruction (78) in the memory location pointed to by R (X). At the conclusion of the interrupt, the user's routine may restore the pre-interrupted value of X and P with a single instruction (70 or 71). The interrupt-enable flip-flop can be activated to permit further interrupts or can be disabled to prevent them.

**COSMAC Register Summary**

| | | |
|---|---|---|
| D | 8 Bits | Data Register (Accumulator) |
| DF | 1 Bit | Data Flag (ALU Carry) |
| R | 16 Bits | 1 of 16 Scratchpad Registers |
| P | 4 Bits | Designates which register is Program Counter |
| X | 4 Bits | Designates which register is Data Pointer |

| | | |
|---|---|---|
| N | 4 Bits | Holds Low-Order Instr. Digit |
| I | 4 Bits | Holds High-Order Instr. Digit |
| T | 8 Bits | Holds old X, P after Interrupt (X is high byte) |
| IE | 1 Bit | Interrupt Enable |
| Q | 1 Bit | Output Flip Flop |

## INSTRUCTION SET

The COSMAC instruction summary is given in Table I. Hexadecimal notation is used to refer to the 4-bit binary codes.

In all registers bits are numbered from the least significant bit (LSB) to the most significant bit (MSB) starting with 0.

R(W): Register designated by W, where W=N or X, or P

R(W).0: Lower-order byte of R(W)
R(W).1: Higher-order byte of R(W)
N0 = Least significant Bit of N Register

Operation Notation
M(R(N)) →D; R(N) + 1

This notation means: The memory byte pointed to by R(N) is loaded into D, and R(N) is incremented by 1.

**TABLE I – INSTRUCTION SUMMARY**

(For Notes, see page 13)

| INSTRUCTION | MNEMONIC | OP CODE | OPERATION |
|---|---|---|---|
| MEMORY REFERENCE | | | |
| LOAD VIA N | LDN | 0N | M(R(N))→D; FOR N NOT 0 |
| LOAD ADVANCE | LDA | 4N | M(R(N))→D; R(N) +1 |
| LOAD VIA X | LDX | F0 | M(R(X))→D |
| LOAD VIA X AND ADVANCE | LDXA | 72 | M(R(X))→D; R(X) +1 |
| LOAD IMMEDIATE | LDI | F8 | M(R(P))→D; R(P) +1 |
| STORE VIA N | STR | 5N | D→M(R(N)) |
| STORE VIA X AND DECREMENT | STXD | 73 | D→M(R(X)); R(X) −1 |
| REGISTER OPERATIONS | | | |
| INCREMENT REG N | INC | 1N | R(N) +1 |
| DECREMENT REG N | DEC | 2N | R(N) −1 |
| INCREMENT REG X | IRX | 60 | R(X) +1 |
| GET LOW REG N | GLO | 8N | R(N).0→D |
| PUT LOW REG N | PLO | AN | D→R(N).0 |
| GET HIGH REG N | GHI | 9N | R(N).1→D |
| PUT HIGH REG N | PHI | BN | D→R(N).1 |
| LOGIC OPERATIONS♦♦ | | | |
| OR | OR | F1 | M(R(X)) OR D→D |
| OR IMMEDIATE | ORI | F9 | M(R(P)) OR D→D; R(P) +1 |
| EXCLUSIVE OR | XOR | F3 | M(R(X)) XOR D→D |
| EXCLUSIVE OR IMMEDIATE | XRI | FB | M(R(P)) XOR D→D; R(P) +1 |
| AND | AND | F2 | M(R(X)) AND D→D |
| AND IMMEDIATE | ANI | FA | M(R(P)) AND D→D; R(P) +1 |
| SHIFT RIGHT | SHR | F6 | SHIFT D RIGHT, LSB(D)→DF, 0→MSB(D) |
| SHIFT RIGHT WITH CARRY<br>RING SHIFT RIGHT | SHRC<br>RSHR | 76♦ | SHIFT D RIGHT, LSB(D)→DF, DF→MSB(D) |
| SHIFT LEFT | SHL | FE | SHIFT D LEFT, MSB(D)→DF, 0→LSB(D) |
| SHIFT LEFT WITH CARRY<br>RING SHIFT LEFT | SHLC<br>RSHL | 7E♦ | SHIFT D LEFT, MSB(D)→DF, DF→LSB(D) |

♦NOTE THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED.

♦♦NOTE THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF.

AFTER AN ADD INSTRUCTION:
DF = 1 DENOTES A CARRY HAS OCCURRED
DF = 0 DENOTES A CARRY HAS NOT OCCURRED

AFTER A SUBTRACT INSTRUCTION:
DF = 1 DENOTES NO BORROW. D IS A TRUE POSITIVE NUMBER
DF = 0 DENOTES A BORROW. D IS TWO'S COMPLEMENT.
THE SYNTAX "-(NOT DF)" DENOTES THE SUBTRACTION OF THE BORROW

**TABLE I – INSTRUCTION SUMMARY (CONT'D)**

| INSTRUCTION | MNEMONIC | OP CODE | OPERATION |
|---|---|---|---|
| ARITHMETIC OPERATIONS♦♦ | | | |
| ADD | ADD | F4 | M(R(X)) +D→DF, D |
| ADD IMMEDIATE | ADI | FC | M(R(P)) +D→DF, D; R(P) +1 |
| ADD WITH CARRY | ADC | 74 | M(R(X)) +D +DF→DF, D |
| ADD WITH CARRY, IMMEDIATE | ADCI | 7C | M(R(P)) +D +DF→DF, D R(P) +1 |
| SUBTRACT D | SD | F5 | M(R(X))–D→DF, D |
| SUBTRACT D IMMEDIATE | SDI | FD | M(R(P))–D→DF, D; R(P) +1 |
| SUBTRACT D WITH BORROW | SDB | 75 | M(R(X))–D–(NOT DF)→DF, D |
| SUBTRACT D WITH BORROW, IMMEDIATE | SDBI | 7D | M(R(P))–D–(NOT DF)→DF, D; R(P) +1 |
| SUBTRACT MEMORY | SM | F7 | D–M(R(X))→DF, D |
| SUBTRACT MEMORY IMMEDIATE | SMI | FF | D–M(R(P))→DF, D; R(P) +1 |
| SUBTRACT MEMORY WITH BORROW | SMB | 77 | D–M(R(X))–(NOT DF)→DF, D |
| SUBTRACT MEMORY WITH BORROW, IMMEDIATE | SMBI | 7F | D–M(R(P))–(NOT DF)→DF, D R(P) +1 |
| BRANCH INSTRUCTIONS–SHORT BRANCH | | | |
| SHORT BRANCH | BR | 30 | M(R(P))→R(P).0 |
| NO SHORT BRANCH (SEE SKP) | NBR | 38♦ | R(P) +1 |
| SHORT BRANCH IF D=0 | BZ | 32 | IF D=0, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF D NOT 0 | BNZ | 3A | IF D NOT 0, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF DF=1 | BDF | 33♦ | IF DF=1, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF POS OR ZERO | BPZ | | |
| SHORT BRANCH IF EQUAL OR GREATER | BGE | | |
| SHORT BRANCH IF DF=0 | BNF | 3B♦ | IF DF=0, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF MINUS | BM | | |
| SHORT BRANCH IF LESS | BL | | |
| SHORT BRANCH IF Q=1 | BQ | 31 | IF Q=1, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF Q=0 | BNQ | 39 | IF Q=0, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF EF1=1 (1 = $V_{SS}$) | B1 | 34 | IF EF1=1, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF EF1=0 (0 = $V_{CC}$) | BN1 | 3C | IF EF1=0, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF EF2=1 (1 = $V_{SS}$) | B2 | 35 | IF EF2=1, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF EF2=0 (0 = $V_{CC}$) | BN2 | 3D | IF EF2=0, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF EF3=1 (1 = $V_{SS}$) | B3 | 36 | IF EF3=1, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF EF3=0 (0 = $V_{CC}$) | BN3 | 3E | IF EF3=0, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF EF4=1 (1 = $V_{SS}$) | B4 | 37 | IF EF4=1, M(R(P))→R(P).0 ELSE R(P) +1 |
| SHORT BRANCH IF EF4=0 (0 = $V_{CC}$) | BN4 | 3F | IF EF4=0, M(R(P))→R(P).0 ELSE R(P) +1 |

♦NOTE: THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED.

♦♦NOTE: THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF.
AFTER AN ADD INSTRUCTION:
DF = 1 DENOTES A CARRY HAS OCCURRED
DF = 0 DENOTES A CARRY HAS NOT OCCURRED
AFTER A SUBTRACT INSTRUCTION:
DF = 1 DENOTES NO BORROW. D IS A TRUE POSITIVE NUMBER
DF = 0 DENOTES A BORROW. D IS TWO'S COMPLEMENT
THE SYNTAX "–(NOT DF)" DENOTES THE SUBTRACTION OF THE BORROW

TABLE I – INSTRUCTION SUMMARY (CONT'D)

| INSTRUCTION | MNEMONIC | OP CODE | OPERATION |
|---|---|---|---|
| BRANCH INSTRUCTIONS–LONG BRANCH | | | |
| LONG BRANCH | LBR | C0 | M(R(P))→R(P).1<br>M(R(P) +1)→R(P).0 |
| NO LONG BRANCH<br>(SEE LSKP) | NLBR | C8♦ | R(P) +2 |
| LONG BRANCH IF D=0 | LBZ | C2 | IF D=0, M(R(P))→R(P).1<br>M(R(P) +1)→R(P).0<br>ELSE R(P) +2 |
| LONG BRANCH IF D NOT 0 | LBNZ | CA | IF D NOT 0, M(R(P))→R(P).1<br>M(R(P) +1)→R(P).0<br>ELSE R(P) +2 |
| LONG BRANCH IF DF=1 | LBDF | C3 | IF DF=1, M(R(P))→R(P).1<br>M(R(P) +1)→R(P).0<br>ELSE R(P) +2 |
| LONG BRANCH IF DF=0 | LBNF | CB | IF DF=0, M(R(P))→R(P).1<br>M(R(P) +1)→R(P).0<br>ELSE R(P) +2 |
| LONG BRANCH IF Q=1 | LBQ | C1 | IF Q=1, M(R(P))→R(P).1<br>M(R(P) +1)→R(P).0<br>ELSE R(P) +2 |
| LONG BRANCH IF Q=0 | LBNQ | C9 | IF Q=0, M(R(P))→R(P).1<br>M(R(P) +1)→R(P).0<br>ELSE R(P) +2 |
| SKIP INSTRUCTIONS | | | |
| SHORT SKIP<br>(SEE NBR) | SKP | 38♦ | R(P) +1 |
| LONG SKIP<br>(SEE NLBR) | LSKP | C8♦ | R(P) +2 |
| LONG SKIP IF D=0 | LSZ | CE | IF D=0, R(P) +2<br>ELSE CONTINUE |
| LONG SKIP IF D NOT 0 | LSNZ | C6 | IF D NOT 0, R(P) +2<br>ELSE CONTINUE |
| LONG SKIP IF DF=1 | LSDF | CF | IF DF=1, R(P) +2<br>ELSE CONTINUE |
| LONG SKIP IF DF=0 | LSNF | C7 | IF DF=0, R(P) +2<br>ELSE CONTINUE |
| LONG SKIP IF Q=1 | LSQ | CD | IF Q=1, R(P) +2<br>ELSE CONTINUE |
| LONG SKIP IF Q=0 | LSNQ | C5 | IF Q=0, R(P) +2<br>ELSE CONTINUE |
| LONG SKIP IF IE=1 | LSIE | CC | IF IE=1, R(P) +2<br>ELSE CONTINUE |
| CONTROL INSTRUCTIONS | | | |
| IDLE | IDL | 00# | WAIT FOR DMA OR<br>INTERRUPT; M(R(0))→BUS |
| NO OPERATION | NOP | C4 | CONTINUE |
| SET P | SEP | DN | N→P |
| SET X | SEX | EN | N→X |
| SET Q | SEQ | 7B | 1→Q |
| RESET Q | REQ | 7A | 0→Q |
| SAVE | SAV | 78 | T→M(R(X)) |
| PUSH X,P TO STACK | MARK | 79 | (X,P)→T; (X,P)→M(R(2))<br>THEN P→X; R(2)–1 |
| RETURN | RET | 70 | M(R(X))→(X,P); R(X) +1<br>1→IE |
| DISABLE | DIS | 71 | M(R(X))→(X,P); R(X) +1<br>0→IE |

#An idle instruction initiates a repeating S1 cycle. The processor will continue to idle until an I/O request ($\overline{\text{INTERRUPT}}$, $\overline{\text{DMA-IN}}$, or $\overline{\text{DMA-OUT}}$) is activated. When the request is acknowledged, the IDLE cycle is terminated and the I/O request is serviced, and then normal operation is resumed.

♦NOTE: THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED.

**TABLE I – INSTRUCTION SUMMARY (CONT'D)**

| INSTRUCTION | MNEMONIC | OP CODE | OPERATION |
|---|---|---|---|
| INPUT–OUTPUT BYTE TRANSFER | | | |
| OUTPUT 1 | OUT 1 | 61 | M(R(X))→BUS; R(X) +1; N LINES = 1 |
| OUTPUT 2 | OUT 2 | 62 | M(R(X))→BUS; R(X) +1; N LINES = 2 |
| OUTPUT 3 | OUT 3 | 63 | M(R(X))→BUS; R(X) +1; N LINES = 3 |
| OUTPUT 4 | OUT 4 | 64 | M(R(X))→BUS; R(X) +1; N LINES = 4 |
| OUTPUT 5 | OUT 5 | 65 | M(R(X))→BUS; R(X) +1; N LINES = 5 |
| OUTPUT 6 | OUT 6 | 66 | M(R(X))→BUS; R(X) +1; N LINES = 6 |
| OUTPUT 7 | OUT 7 | 67 | M(R(X))→BUS; R(X) +1; N LINES = 7 |
| INPUT 1 | INP 1 | 69 | BUS→M(R(X)); BUS→D; N LINES = 1 |
| INPUT 2 | INP 2 | 6A | BUS→M(R(X)); BUS→D; N LINES = 2 |
| INPUT 3 | INP 3 | 6B | BUS→M(R(X)); BUS→D; N LINES = 3 |
| INPUT 4 | INP 4 | 6C | BUS→M(R(X)); BUS→D; N LINES = 4 |
| INPUT 5 | INP 5 | 6D | BUS→M(R(X)); BUS→D; N LINES = 5 |
| INPUT 6 | INP 6 | 6E | BUS→M(R(X)); BUS→D; N LINES = 6 |
| INPUT 7 | INP 7 | 6F | BUS→M(R(X)); BUS→D; N LINES = 7 |

1. Long-Branch, Long-Skip and No Op instructions are the only instructions that require three cycles to complete (1 fetch + 2 execute).

   Long-Branch instructions are three bytes long. The first byte specifies the condition to be tested; and the second and third byte, the branching address.

   The long-branch instructions can:
   a) Branch unconditionally
   b) Test for D=0 or D≠0
   c) Test for DF=0 or DF=1
   d) Test for Q=0 or Q=1
   e) effect an unconditional no branch

   If the tested condition is met, then branching takes place; the branching address bytes are loaded in the high-and-low-order bytes of the current program counter, respectively. This operation effects a branch to any memory location.

   If the tested condition is not met, the branching address bytes are skipped over, and the next instruction in sequence is fetched and executed. This operation is taken for the case of unconditional no branch (NLBR).

2. The short-branch instructions are two bytes long. The first byte specifies the condition to be tested, and the second specifies the branching address.

   The short-branch instructions can:
   a) Branch unconditionally
   b) Test for D=0 or D≠0
   c) Test for DF=0 or DF=1
   d) Test for Q=0 or Q=1
   e) Test the status (1 or 0) of the four EF flags
   f) Effect an unconditional no branch

   If the tested condition is met, then branching takes place; the branching address byte is loaded into the low-order byte position of the current program counter. This effects a branch with the current 256-byte page of the memory, i.e., the page which holds the branching address. If the tested condition is not met, the branching address byte is skipped over, and the next instruction in sequence is fetched and executed. This same action is taken in the case of unconditional no branch (NBR)

3. The skip instructions are one byte long. There is one Unconditional Short-Skip (SKP) and eight Long-Skip instructions.

   The Unconditional Short-Skip instruction takes 2 cycles to complete (1 fetch + 1 execute). Its action is to skip over the byte following it. Then the next instruction in sequence is fetched and executed. This SKP instruction is identical to the unconditional no-branch instruction (NBR) except that the skipped-over byte is not considered part of the program.

   The Long-Skip instructions take three cycles to complete (1 fetch + 2 execute).

   They can:
   a) Skip unconditionally
   b) Test for D=0 or D≠0
   c) Test for DF=0 or DF=1
   d) Test for Q=0 or Q=1
   e) Test for IE=1

   If the tested condition is met, then Long Skip takes place; the current program counter is incremented twice. Thus two bytes are skipped over and the next instruction in sequence is fetched and executed. If the tested condition is not met, then no action is taken. Execution is continued by fetching the next instruction in sequence.

## SIGNAL DESCRIPTIONS

**BUS 0 to BUS 7 (Data Bus)**

8-bit directional DATA BUS lines. These lines are used for transferring data between the memory, the microprocessor, and I/O devices.

**N0 to N2 (I/O Lines)**

Activated by an I/O instruction to signal the I/O control logic of a data transfer between memory and I/O interface. These lines can be used to issue command codes or device selection codes to the I/O devices (independently or combined with the memory byte on the data bus when an I/O instruction is being executed). The N bits are low at all times except when an I/O instruction is being executed. During this time their state is the same as the corresponding bits in the N register.

The direction of data flow is defined in the I/O instruction by bit N3 (internally) and is indicated by the level of the $\overline{MRD}$ signal.

$\overline{MRD} = V_{CC}$: Data from I/O to CPU and Memory

$\overline{MRD} = V_{SS}$: Data from Memory to I/O

**$\overline{EF1}$ to $\overline{EF4}$ (4 Flags)**

These inputs enable the I/O controllers to transfer status information to the processor. The levels can be tested by the conditional branch instructions. They can be used in conjunction with the INTERRUPT request line to establish interrupt priorities. These flags can also be used by I/O devices to "call the attention" of the processor, in which case the program must routinely test the status of these flag(s). The flag(s) are sampled at the beginning of every S1 cycle.

**$\overline{INTERRUPT}$, $\overline{DMA\text{-}IN}$, $\overline{DMA\text{-}OUT}$ (3 I/O Requests)**

These inputs are sampled by the CDP1802 during the interval between the leading edge of TPB and the leading edge of TPA.

**Interrupt Action:** X and P are stored in T after executing current instruction; designator X is set to 2; designator P is set to 1; interrupt enable is reset to 0 (inhibit); and instruction execution is resumed. The interrupt action requires one machine cycle (S3).

**DMA Action:** Finish executing current instruction; R(0) points to memory area for data transfer; data is loaded into or read out of memory; and increment R(0).

**Note:** In the event of concurrent DMA and INTERRUPT requests, DMA-IN has priority followed by DMA-OUT and then INTERRUPT.

**SC0, SC1, (2 State Code Lines)**

These outputs indicate that the CPU is: 1 fetching an instruction, or 2) executing an instruction, or 3) processing a DMA request, or 4) acknowledging an interrupt request. The levels of state code are tabulated below. All states are valid at TPA. H = $V_{CC}$, L = $V_{SS}$.

| State Type | State Code Lines | |
|---|---|---|
| | SC1 | SC0 |
| S0 (Fetch) | L | L |
| S1 (Execute) | L | H |
| S2 (DMA) | H | L |
| S3 (Interrupt) | H | H |

**TPA, TPB (2 Timing Pulses)**

Positive pulses that occur once in each machine cycle (TPB follows TPA). They are used by I/O controllers to interpret codes and to time interaction with the data bus. The trailing edge of TPA is used by the memory system to latch the higher-order byte of the 16-bit memory address. TPA is suppressed in IDLE when the CPU is in the load mode.

**MA0 to MA7 (8 Memory Address Lines)**

The higher-order byte of a 16-bit COSMAC memory address appears on the memory address lines MA0-7 first. Those bits required by the memory system can be strobed into external address latches by timing pulse TPA. The low-order byte of the 16-bit address appears on the address lines after the termination of TPA. Latching of all 8 higher-order address bits would permit a memory system of 64K bytes.

$\overline{MWR}$ (Write Pulse)

A negative pulse appearing in a memory-write cycle, after the address lines have stabilized.

$\overline{MRD}$ (Read Level)

A low level on $\overline{MRD}$ indicates a memory read cycle. It can be used to control three-state outputs from the addressed memory which may have a common data input and output bus. If a memory does not have a three-state high-impedance output, $\overline{MRD}$ is useful for driving memory/bus separator gates. It is also used to indicate the direction of data transfer during an I/O instruction. For additional information see Table I.

Q

Single bit output from the CPU which can be set or reset under program control. During SEQ or REQ instruction execution, Q is set or reset between the trailing edge of TPA and the leading edge of TPB.

CLOCK

Input for externally generated single-phase clock. A typical clock frequency is 6.4 MHz at $V_{CC} = V_{DD} = 10$ volts. The clock is counted down internally to 8 clock pulses per machine cycle.

$\overline{XTAL}$

Connection to be used with clock input terminal, for an external crystal, if the on-chip oscillator is utilized. The crystal is connected between terminals 1 and 39 (CLOCK and $\overline{XTAL}$) in parallel with a resistance (10 megohms typ.). Frequency trimming capacitors may be required at terminals 1 and 39. For additional information see ICAN-6565.

$\overline{WAIT}$, $\overline{CLEAR}$ (2 Control Lines)

Provide four control modes as listed in the following truth table:

| $\overline{CLEAR}$ | $\overline{WAIT}$ | MODE |
|---|---|---|
| L | L | Load |
| L | H | Reset |
| H | L | Pause |
| H | H | Run |

The function of the modes are defined as follows:

**Load**

Holds the CPU in the IDLE execution state and allows an I/O device to load the memory without the need for a "bootstrap" loader. It modifies the IDLE condition so that DMA-IN operation does not force execution of the next instruction.

**Reset**

Registers I, N, Q are reset, IE is set and 0's ($V_{SS}$) are placed on the data bus. TPA and TPB are suppressed while reset is held and the CPU is placed in S1. The first machine cycle after termination of reset is an initialization cycle which requires 9 clock pulses. During this cycle the CPU remains in S1 and registers X, P, and R(0) are reset. Interrupt and DMA servicing are suppressed during the initialization cycle. The next cycle is an S0, S1, or an S2 but never an S3. With the use of a 71 instruction followed by 00 at memory locations 0000 and 0001, this feature may be used to reset IE, so as to preclude interrupts until ready for them. Power-up reset can be realized by connecting a buffered RC network to $\overline{CLEAR}$. For additional information see ICAN-6581.

**Pause**

Stops the internal CPU timing generator on the first negative high-to-low transition of the input clock. The oscillator continues to operate, but subsequent clock transitions are ignored.

**Run**

May be initiated from the Pause or Reset mode functions. If initiated from Pause, the CPU resumes operation on the first negative high-to-low transition of the input clock. When initiated from the Reset operation, the first machine cycle following Reset is always the initialization cycle. The initialization cycle is then followed by a DMA (S2) cycle or fetch (S0) from location 0000 in memory.

$V_{DD}$, $V_{SS}$, $V_{CC}$ (Power Levels)

The internal voltage supply $V_{DD}$ is isolated from the Input/Output voltage supply $V_{CC}$ so that the processor may operate at maximum speed while interfacing with various external circuit technologies, including $T^2L$ at 5 volts. $V_{CC}$ must be less than or equal to $V_{DD}$. All outputs swing from $V_{SS}$ to $V_{CC}$. The recommended input voltage swing is $V_{SS}$ to $V_{CC}$.

## RUN-MODE STATE TRANSITIONS

The CDP1802 and CDP1802C CPU state transitions when in the RUN, RESET, and LOAD modes are shown in Fig. 15. Each machine cycle requires the same period of time, 8 clock pulses, except the initialization cycle, which requires 9 clock pulses. The execution of an instruction requires either two or three machine cycles, S0 followed by a single S1 cycle or two S1 cycles. S2 is the response to a DMA request and S3 is the interrupt response. Table II shows the conditions on Data Bus and Memory-Address lines during all machine states.

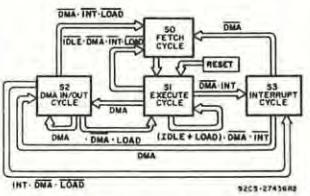
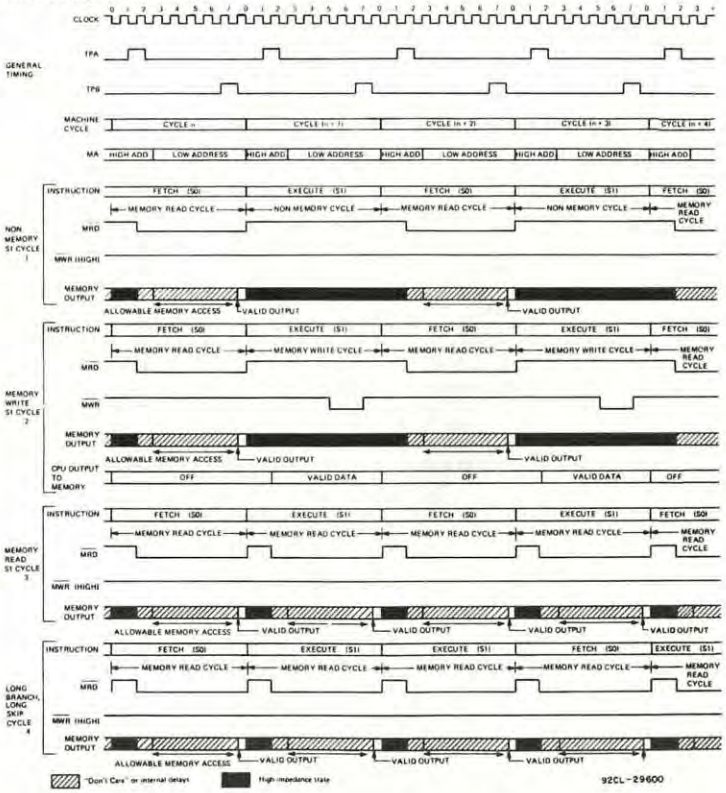



*Fig. 15 – CDP1802 microprocessor state transitions (Run Mode).*



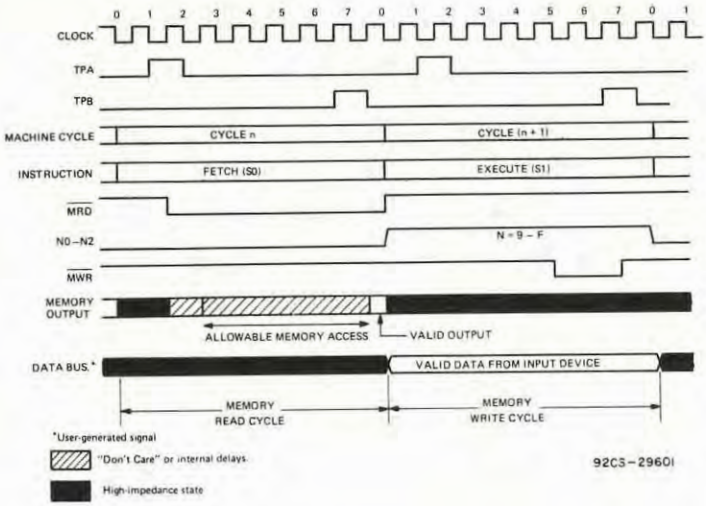



*Fig. 17 – Timing diagram for machine cycle type No. 5.*

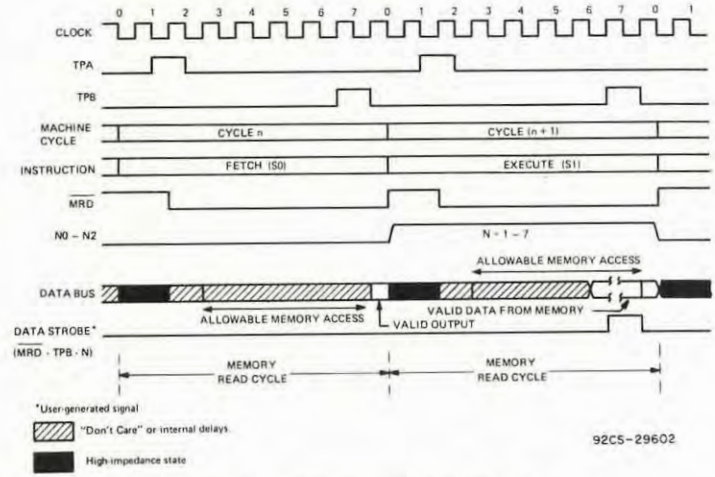



*Fig. 18 – Timing diagram for machine cycle type No. 6.*

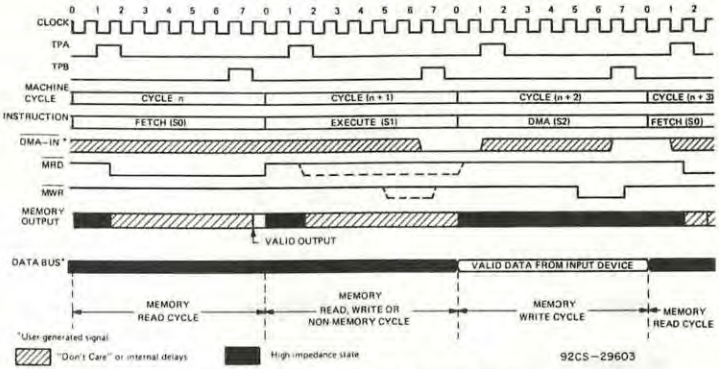



*Fig. 19 – Timing diagram for machine cyle type No. 7.*

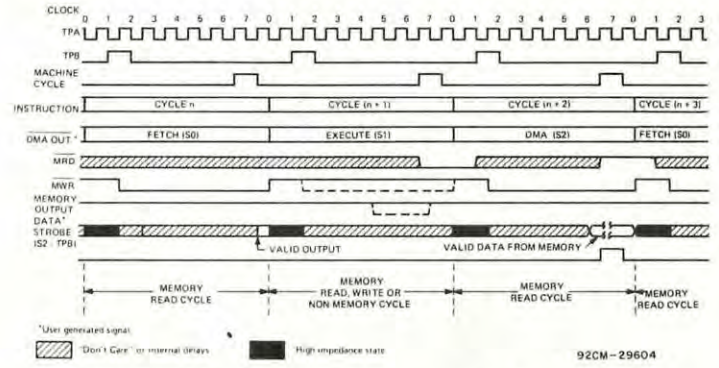



*Fig. 20 – Timing diagram for machine cyle type No. 8.*

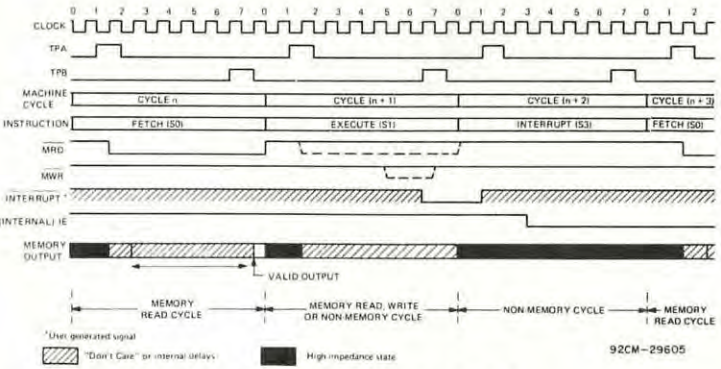



*Fig. 21 – Timing diagram for machine cycle type No. 9.*

**TABLE II. CONDITIONS ON DATA BUS AND MEMORY ADDRESS LINES DURING ALL MACHINE STATES**

| STATE | I | N | MNEMONIC | INSTRUCTION | OPERATION | DATA BUS | MEMORY ADDRESS | $\overline{MRD}$ | NOTES[G] |
|---|---|---|---|---|---|---|---|---|---|
| S1 | RESET | | | | JAM: I,N,Q,X,P = 0 IE = 1 | 0 | R (0) UNDEFINED | 1 | A |
| | FIRST CYCLE AFTER RESET NOT PROGRAMMER ACCESSIBLE | | | | INITIALIZE | 0 | R (0) UNDEFINED | 1 | B |
| S0 | FETCH | | | | M(R(P))→I,N R(P)+1 | M(R(P)) | R(P) | 0 | C |
| S1 (Execute) | 0 | 0 | IDL | IDLE | [Load = 0 (Program Idle)] | M (R(0)) | R (0) | 0 | D,3 |
| | | | | | [Load = 1 (Load Mode)] | M(R(0)) | PREVIOUS ADDRESS | 0 | E,3 |
| | | N≠0 | LDN | LOAD D VIA N | M(R(N))→D | M(R(N)) | R(N) | 0 | 3 |
| | 1 | N | INC | INCREMENT | R(N)+1 | FLOAT | R(N) | 1 | 1 |
| | 2 | N | DEC | DECREMENT | R(N)−1 | FLOAT | R(N) | 1 | 1 |
| | 3 | N | – | SHORT BRANCH | [BRANCH NOT TAKEN] | M(R(P)) | R(P) | 0 | 3 |
| | | | | | [BRANCH TAKEN] | M(R(P)) | R(P) | 0 | |
| | 4 | N | LDA | LOAD ADVANCE | M(R(N))→D R(N)+1 | M(R(N)) | R(N) | 0 | 3 |
| | 5 | N | STR | STORE VIA N | D→M(R(N)) | D | R(N) | 1 | 3 |
| | 6 | 0 | IRX | INC REG X | R(X)+1 | M(R(X)) | R(X) | 0 | 3 |
| | | N=1–7 | OUT N | OUTPUT | M(R(X))→BUS R(X)+1 | M(R(X)) | R(X) | 0 | 6 |
| | | N=9–F | INP N | INPUT | BUS→M(R(X)), D | I/O DEVICE | R(X) | 1 | 5 |
| | 7 | 0 | RET | RETURN | M(R(X))→(X,P) R(X)+1; 1→IE | M(R(X)) | R(X) | 0 | 3 |
| | | 1 | DIS | DISABLE | M(R(X))→(X,P) R(X)+1; 0→IE | M(R(X)) | R(X) | 0 | 3 |
| | | 2 | LDXA | LOAD VIA X AND ADVANCE | M(R(X))→D P(X)−1 | M(R(X)) | R(X) | 0 | 3 |
| | | 3 | STXD | STORE VIA X AND DECREMENT | D→M(R(X)) R(X)−1 | D | R(X) | 1 | 2 |
| | | 4,5,7 | – | | ALU OPERATION | M(R(X)) | R(X) | 0 | 3 |
| | | 6 | – | | ALU OPERATION | FLOAT | R(X) | 1 | 1 |
| | | 8 | SAV | SAVE | T→M(R(X)) | T | R(X) | 1 | 2 |
| | | 9 | MARK | MARK | (X,P)→T, M(R(2)) P→X; R(2)−1 | T | R(2) | 1 | 2 |
| | | A | REQ | RESET Q | Q = 0 | FLOAT | R(P) | 1 | 1 |
| | | B | SEQ | SET Q | Q = 1 | FLOAT | R(P) | 1 | 1 |
| | | C,D,F | | | ALU OPERATION IMMEDIATE | M(R(P)) | R(P) | 0 | 3 |
| | | E | | | ALU OPERATION | FLOAT | R(X) | 1 | 1 |
| | 8 | N | GLO | GET LOW | R(N) .0→D | R(N) .0 | R(N) | 1 | 1 |
| | 9 | N | GHI | GET HIGH | R(N) .1→D | R(N) .1 | R(N) | 1 | 1 |
| | A | N | PLO | PUT LOW | D→R(N) .0 | D | R(N) | 1 | 1 |
| | B | N | PHI | PUT HIGH | D→R(N) .1 | D | R(N) | 1 | 1 |
| | C | 0,1,2 3,8,9 A,B | | LONG BRANCH | [BRANCH NOT TAKEN] | M(R(P)) | R(P) | 0 | 4 |
| | | | | | [BRANCH TAKEN] | M(R(P)) | R(P) | 0 | 4 |
| | | 5,6,7 C,D,E F | | LONG SKIP | [SKIP NOT TAKEN] | M(R(P)) | R(P) | 0 | 4 |
| | | | | | [SKIP TAKEN] | M(R(P)) | R(P) | 0 | 4 |
| | | 4 | NOP | NO OPERATION | NO OPERATION | M(R(P)) | R(P) | 0 | 4 |
| | D | N | SEP | SET P | N→P | N N | R(N) | 1 | 1 |
| | E | N | SEX | SET X | N→X | N N | R(N) | 1 | 1 |
| | F | 0 | LDX | LOAD VIA X | M(R(X))→D | M(R(X)) | R(X) | 0 | 3 |
| | | 1,2,3 4,5,7 | | | ALU OPERATION | M(R(X)) | R(X) | 0 | 3 |
| | | 6 | SHR | SHIFT RIGHT | SHIFT D RIGHT LSB(D)→DF 0→MSB(D) | FLOAT | R(X) | 1 | 1 |
| | | 8 | LDI | LOAD IMMEDIATE | M(R(P))→D R(P)+1 | M(R(P)) | R(P) | 0 | 3 |
| | | 9,A,B C,D,F | | | ALU OPERATION IMMEDIATE | M(R(P)) | R(P) | 0 | 3 |
| | | E | SHL | SHIFT LEFT | ALU OPERATION | FLOAT | R(P) | 1 | 1 |
| S2 | IN REQUEST | | | DMA IN | BUS→M(R(0)) | I/O DEVICE | R (0) | 1 | F,7 |
| | OUT REQUEST | | | DMA OUT | M(R(0))→BUS | M(R(0)) | R (0) | 0 | F,8 |
| S3 | INTERRUPT | | | | X,P→T, 0→IE 2→X, 1→P | FLOAT | R(N) | 1 | 9 |

NOTES:

A. IE = 1; TPA, TPB suppressed, state = S1

B. BUS = 0 for entire cycle

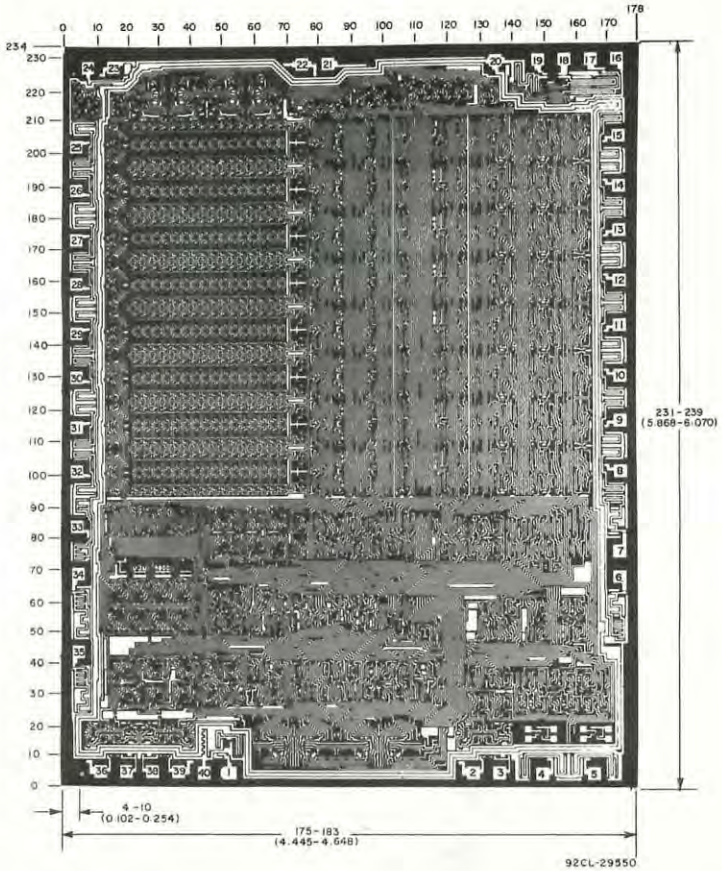
C. Next state always S1

D. Wait for DMA or INTERRUPT

E. Suppress TPA, wait for DMA

F. IN REQUEST has priority over OUT REQUEST

G. Numbers refer to machine cycles types – refer to timing diagrams, Figs. 16 through 20.




*Dimensions in parentheses are in millimeters and are derived from the basic inch dimensions as indicated. Grid graduations are in mils ($10^{-3}$ inch).*

*The photographs and dimensions of each COS/MOS chip represent a chip when it is part of the wafer. When the wafer is cut into chips, the cleavage angles are 57° instead of 90° with respect to the face of the chip. Therefore, the isolated chip is actually 7 mils (0.17 mm) larger in both dimensions.*

*Dimensions and pad layout for CDP1802*

## OPERATING AND HANDLING CONSIDERATIONS

1. **Handling**
   All inputs and outputs of RCA COS/MOS devices have a network for electrostatic protection during handling. Recommended handling practices for COS/MOS devices are described in ICAN-6525, "Guide to Better Handling and Operation of CMOS Integrated Circuits."
2. **Operating**

**Operating Voltage**

During operation near the maximum supply voltage limit, care should be taken to avoid or suppress power supply turn-on and turn-off transients, power supply ripple, or ground noise; any of these conditions must not cause $V_{DD}-V_{SS}$ to exceed the absolute maximum rating.

**Input Signals**

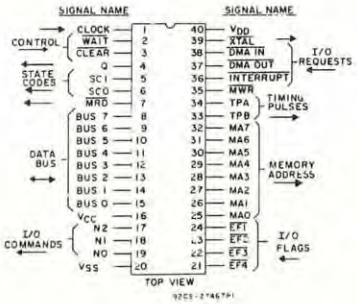
To prevent damage to the input protection circuit, input signals should never be greater than $V_{CC}$ nor less than $V_{SS}$. Input currents must not exceed 10 mA even when the power supply is off.

**Unused Inputs**

A connection must be provided at every input terminal. All unused input terminals must be connected to either $V_{CC}$ or $V_{SS}$, whichever is appropriate.

**Output Short Circuits**

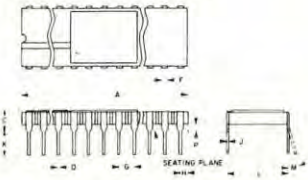
Shorting of outputs to $V_{DD}$, $V_{CC}$, or $V_{SS}$ may damage COS/MOS devices by exceeding the maximum device dissipation.




When incorporating RCA Solid State Devices in equipment, it is recommended that the designer refer to "Operating Considerations for RCA Solid State Devices", Form No. 1CE-402, available on request from RCA Solid State Division, Box 3200, Somerville, N. J. 08876.

## DIMENSIONAL OUTLINE

**CDP1802D, CDP1802CD**
**40-Lead Dual-In-Line Ceramic**

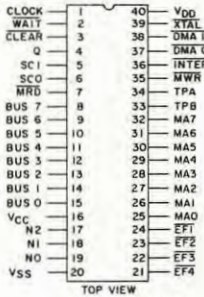



| DIM. | MILLIMETERS | | INCHES | |
|---|---|---|---|---|
| | MIN. | MAX. | MIN. | MAX. |
| A | 50.30 | 51.30 | 1.980 | 2.020 |
| C | 2.42 | 3.93 | 0.095 | 0.155 |
| D | 0.43 | 0.56 | 0.017 | 0.023 |
| F | 1.27 REF. | | 0.050 REF. | |
| G | 2.54 BSC | | 0.100 BSC | |
| H | 0.76 | 1.78 | 0.030 | 0.070 |
| J | 0.20 | 0.30 | 0.008 | 0.012 |
| K | 3.18 | 4.45 | 0.125 | 0.175 |
| L | 14.74 | 15.74 | 0.580 | 0.620 |
| M | – | 7° | – | 7° |
| P | 0.64 | 1.27 | 0.025 | 0.050 |
| N | 40 | | 40 | |

NOTES
1. Leads within 0.13 mm (0.005) radius of true position at maximum material condition.
2. Dimension "L" to center of leads when formed parallel.
3. When this device is supplied solder dipped, the maximum lead thickness (narrow portion) will not exceed 0.013 in. (0.33 mm).

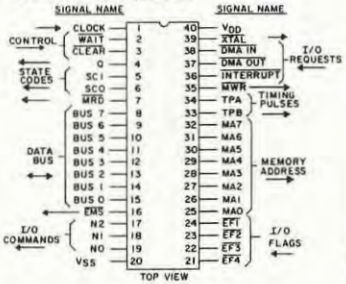
# Terminal Assignment Diagrams

## CDP1802 COSMAC Microprocessor



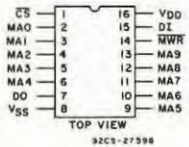

92CS-29554

## CDP1804 COSMAC Microcomputer



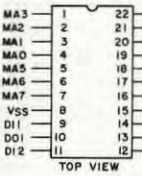

92CS-31059

## CDP1821 1024 x 1 RAM



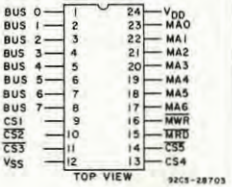

92CS-27596

## CDP1822 256 x 4 RAM



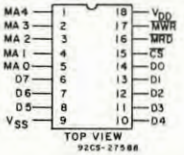

92CS-27592RI

## CDP1823 128 x 8 RAM



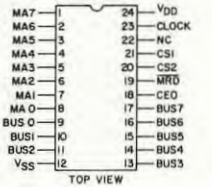

92CS-28703

## CDP1824 32 x 8 RAM



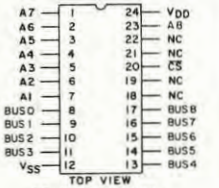

92CS-27588

## CDP1831 512 x 8 ROM



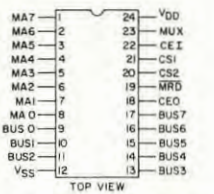

NC = NO CONNECTION 92CS-27584

## CDP1832 512 x 8 ROM



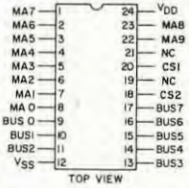

NC = NO CONNECTION
92CS-27579RI

## CDP1833 1024 x 8 ROM




NC = NO CONNECTION 92CS-28889

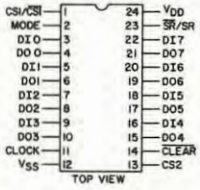
## CDP1834
## 1024 x 8 ROM




NC = NO CONNECTION

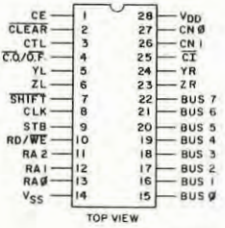
92CS-28727

## CDP1852
## Byte I/O



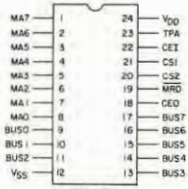

92CS-27572

## CDP1855
## 8-Bit Programmable
## Multiply/Divide Unit



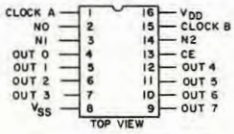

92CS-29965R2

## CDP1835
## 2048 X 8 ROM




92CS-32376

## CDP1853
## N-Bit Decoder




92CS-28726

## CDP1856
## Bus Buffer (Memory)
## Separator


92CS-28097

## CDP1851
## Programmable
## I/O Interface


92CS-31926

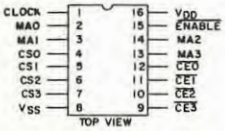
## CDP1854A
## UART
## Mode O


92CS-28455R1

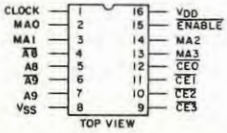
## CDP1857
## I/O Bus Buffer


92CS-28097

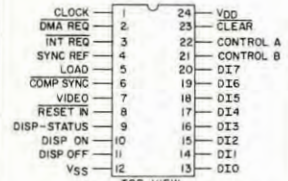
## CDP1858
## 4-Bit Latch



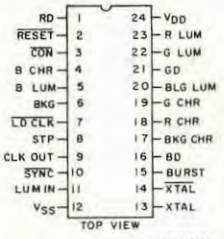

## CDP1859
## 4-Bit Latch



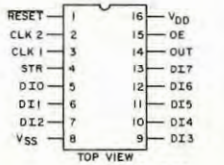

## CDP1861
## Video Display Controller




## CDP1862
## Color Generator Controller

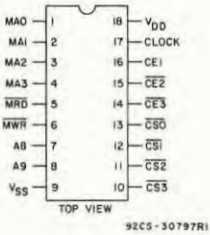



## CDP1863
## Programmable Frequency Generator

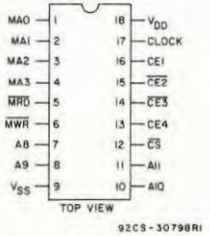



## CDP1864
## PAL-Compatible TV Interface

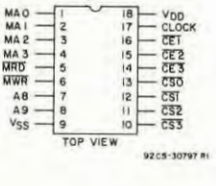

## CDP1866
## 4-Bit Latch and Decoder Memory Interface




## CDP1867
## 4-Bit Latch and Decoder Memory Interface

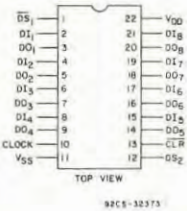



## CDP1868
## 4-Bit Latch and Decoder Memory Interface

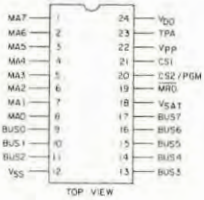


## CDP1869
## Address and Sound Generator


## CDP1872
## Hi-Speed 8-Bit Address Latch

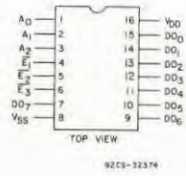



## CDP18U43
## 1K x 8 UV EPROM

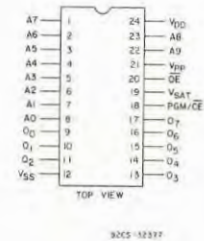



## CDP1870
## Color Video Generator


## CDP1873
## Hi-Speed 1 of 8 Decoder



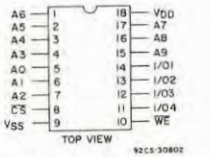

## CDP27C58
## 1K x 8 UV EROM




## CDP1871
## Keyboard Encoder


## CDP18U42CD
## 256 x 8 UV EPROM


## MWS5114
## 1024 x 4 RAM



RCA CDP1802 CODING FORM

NAME : ..... PROGRAM : ..... DATE : ..... PAGE : .....

| ADDRESS | D1 D2 | LN | LABEL : | MNEMONIC | OPERAND | COMMENTS |
|---|---|---|---|---|---|---|
| | | 0 | | | | |
| | | 1 | | | | |
| | | 2 | | | | |
| | | 3 | | | | |
| | | 4 | | | | |
| | | 5 | | | | |
| | | 6 | | | | |
| | | 7 | | | | |
| | | 8 | | | | |
| | | 9 | | | | |
| | | A | | | | |
| | | B | | | | |
| | | C | | | | |
| | | D | | | | |
| | | E | | | | |
| | | F | | | | |

# APPENDIX D

## SELECTED RCA LITERATURE

| | |
|---|---|
| CDL-820G | Linear IC's for Industrial Application |
| CMB-250A | COSMAC Microboard Computer Systems |
| CMS-272 | COS/MOS Integrated Circuits Manual |
| COS-278G | COS/MOS Integrated Circuits Product Guide |
| CPI-279 | Understanding CMOS |
| MPG-180C | COSMAC Microprocessor Product Guide |
| MPM-201B | COSMAC Microprocessor User Manual |
| MPM-202A | COSMAC Timesharing Manual |
| MPM-206A | Fixed-Point Arithmetic Subroutine Manual |
| MPM-207 | Floating-Point Arithmetic Subroutine Manual |
| MPM-212 | COSMAC Microterminal Manual |
| MPM-216A | COSMAC Development Systems Manual |
| MPM-217A | COSMAC Floppy Disk Manual |
| MPM-218A | COSMAC Micromonitor Manual |
| MPM-222A | COSMAC PROM Programmer Manual |
| MPM-223A | COSMAC Macro-Assembler Manual |
| MPM-224 | Evaluation Kit and EK/Assembler-Editor Manual |
| MPM-231A | Micromonitor Operating System (MOPS) Manual |
| MPM-232 | Operator Manual for COSMAC DOS Development System (CDSIII) CDP18S007 |
| MPM-233 | Hardware Reference Manual for COSMAC (CDOS) Development System, CDP18S007 |
| MPM-920A | Microprocessor Instruction Summary |
| SSD-220B | Power Devices DATABOOK |
| SSD-240A | Linear Integrated Circuits DATABOOK |
| SSD-250A | COS/MOS Integrated Circuits DATABOOK |
| SSD-260 | COS/MOS Memory, Microprocessor and Support Systems DATABOOK |

APPENDIX E

## LIST OF RCA SALES OFFICES - EUROPE

BELGIUM

RCA S.A.
Mercure Center
rue de la Fusée 100

1130 BRUXELLES

TEL : 02/720.89.80
TWX : 61566

FRANCE

RCA S.A.
rue Fessart 32

92100 BOULOGNE
France

TEL : 01/603.87.87
TWX : 200144

ITALY

RCA SpA
Piazza San Marco 1

20 122 MILANO
Italy

TEL : 02/65.97.048-051
TWX : 310637

SWEDEN

RCA International Ltd
Box 3047
Hagalundsgatan 8

17103 SOLNA 3
Sweden

TEL : 08/83.42.25
TWX : 11485

UK

RCA Ltd
Lincoln Way,
Windmill Road,
SUNBURY-ON-THAMES

MIDDLESEX TW16 7HW,
UK

TEL : 093.27.85511
TWX : 24246

WEST GERMANY

RCA GmbH
Pfinstrosenstrasse 29
8000 MÜNCHEN 70
W. Germany

TEL : 08971/43047-049
TWX : 05-29051

RCA GmbH
Justus-von-Liebig-Ring 10
2085 QUICKBORN
W. Germany

TEL : 04106/2001
TWX : 02-11582

RCA GmbH
Zeppelinstrasse 35
7302 OSTFILDERN 4
(KENMAT)
W. Germany

TEL : 071145/4001-04
TWX : 07-23838

APPENDIX F

## APPOINTED RCA DISTRIBUTORS - EUROPE

### AUSTRIA

BACHER ELEKTRONISCHE GERÄTE GmbH
Rotenmühlgasse 26
A - 1120 VIENNA

TEL : 0222/83.56.460
TWX : 131532

### BELGIUM

INELCO (BELGIUM) S.A.
Avenue des Croix de Guerre 94
1120 BRUXELLES

TEL : 02/216.01.60
TWX : 25441

### DENMARK

TAGE OLSEN A/S
Ballerup Byvej 222
P.O. Box 225
DK - 2750 BALLERUP

TEL : 02/65.81.11
TWX : 35293

### FINLAND

TELERCAS OY
P.O. Box 2
SF - 01511 VANTAA 51

TEL : 90/821.655
TWX : 12-1111

### FRANCE

ALMEX S.A.
rue de l'Aubépine 48
F - 92160 ANTONY

TEL : 01/666.21.12
TWX : 250067

RADIO EQUIPEMENTS ANTARES S.A.
rue Ernest Cognacq 9
F - 92301 LEVALLOIS-PERRET

TEL : 01/758.11.11
TWX : 620630

TEKELEC AIRTRONIC S.A.
Cité des Bruyères
rue Carle Vernet
F - 92310 SEVRES
TEL : 01/534.75.35
TWX : 204552

WEST GERMANY

ALFRED NEYE ENATECHNIK GmbH
Schillerstrasse 14
2085 QUICKBORN
TEL : 04106/6121
TWX : 02-13590

GUSTAV BECK KG
Eltersdorfer Strasse 7
8500 NURNBERG 15
TEL : 0911/34961-66
TWX : 06-22334

ELKOSE GmbH
Bahnhofstrasse 44
7141 MÖGLINGEN
TEL : 07141/4871
TWX : 07-264472

RTG E. Springorum GmbH&Co.
Bronnerstrasse 7
4600 DORTMUND 1
TEL : 0231/54951
TWX : 08-22534

SASCO GmbH
Hermann-Oberth-Strasse 16
8011 PUTZBRUNN bei MÜNCHEN
TEL : 089/46111
TWX : 05-29829

SPOERLE ELECTRONIC KG
Otto-Hahn-Strasse 13
6072 DREIEICH bei FRANKFURT
TEL : 06103/3041
TWX : 04-17972

GREECE

SEMICON Co
31, Vassileos Georgiou B' Street
ATHENS 516
TEL : 734.353
TWX : 219492

HOLLAND

INELCO Nederland BV
Turfstekerstraat 63
N - 1431 GD AALSMEER
TEL : 02977/2.88.55
TWX : 14693

VEKANO BV
Postbus 6115
N - 5600 HC EINDHOVEN
TEL : 40/81.09.75
TWX : 51804

ICELAND

GEORG AMUNDASON
P.O.Box 698
REYKJAVIK - Iceland
TEL : 81180
TWX 2108

ITALY

ELEDRA 3S SpA
Viale Elvezia 18
I - 20154 MILANO
TEL : 02/349751
TWX : 332.332

IDAC Elettronica SpA
Via Turazza 32
I - 35100 PADOVA
TEL : 049/66.02.22
TWX : 430353

LASI Elettronica SpA
Viale Lombardia 6
I - 20092 CINISELLO BALSAMO (MI)
TEL : 02/61.20.441-5
TWX : 331612

SILVERSTAR Ltd
Via dei Gracchi 20
I - 20146 MILANO
TEL : 02/49.96
TWX : 332.189

NORWAY.

NATIONAL ELEKTRO A/S — TEL : 02/22.19.00
Ulvenveien 75 — TWX : 11265
OKERN - OSLO 5

PORTUGAL

TELECTRA SARL — TEL : 68.60.72-75
Rua Rodrigo do Fonseca, 103 — TWX : 12598
LISBON 1 -

SOUTH AFRICA

ALLIED ELECTRIC (PTY) Ltd — TEL : 011/892.1001
Components Division — TWX : 87823
P.O.Box 6090, DUNSWART
1508

SPAIN

ELECTRICA COMERCIAL COLOMINAS S.A. — TEL : 03/243.20.07
Division Novolectric — TWX : 51433
Valencia 109-111
BARCELONA 11

SISTECO S.A. — TEL : 03/321.73.47/92
Corcega 167 — 03/322.42.05/52
BARCELONA 36 — TWX : 51990

SWEDEN

FERNER Electronics AB
Snörmakarvägen 35
P.O.Box 125
161 26 BROMMA
STOCKHOLM

TEL : 08/80.25.40
TWX : 10312

LAGERCRANTZ ELEKTRONIK AB
Kanalvägen 5
P.O.Box 48
19401 UPPLANDS VASBY

TEL : 0760-861.20
TWX : 11275

SWITZERLAND

BAERLOCHER AG
P.O.Box 485
CH - 8021 ZURICH

TEL : 01/42.99.00
TWX : 53118

TURKEY

TEKNIM COMPANY Ltd
Riza Sah Pehlevi Caddesi 7
KAVAKLIDERE ANKARA

TEL : 27.58.00
TWX : 42155

UK

CRELLON ELECTRONICS Ltd.
380 Bath Road
Slough
Berks, SL1 6JE

TEL : Burnham 06286/4434
TWX : 84571

VSI ELECTRONICS Ltd.
Roydonbury Industrial Park
Horsecroft Road
Harlow
Essex, CM19 5BY

TEL : Harlow 0279/29666
TWX : 81387

I.T.T. ELECTRONIC SERVICES
Edinburgh Way
Harlow
Essex, CM20 2DE

TEL : Harlow 0279/26777
TWX : 81525

JERMYN DISTRIBUTION
Vestry Industrial Estate
Sevenoaks
Kent

TEL : Sevenoaks 0732/50144
TWX : 95142

MACRO MARKETING Ltd
396 Bath Road
Slough
Berks

TEL : Burnham 06286/4422
TWX : 847945

SEMICOMPS NORTHERN Ltd
East Bowmont Street
Kelso
Roxburghshire
Scotland

TEL : Kelso 05732/2366
TWX : 72692

YUGOSLAVIA

AVTOTEHNA
P.O.Box 593
Titova 36-X1
LJUBLJANA 61000

TEL : 317.044
TWX : 31223

EGYPT

SAKRCO ENTERPRISES
P.O.Box 1133
37 Kasr el Nil Street
Apt. 5
CAIRO - Egypt

TEL : 744440
TWX : 93146 SKRCO UN