1100,-

I N D A T A   N.V.
FRANS SMOLDERSSTRAAT   18

1940   ST. STEVENS WOLUWE

=-=-=-=-=-=-=-=-=-=-=-=-=-

tel. 09-32 (0)2 - 721 2090 (INDATA)
tel. 09-32 (0)2 - 762 2818 (hr. Halu
Indata
tbv. DOS V1.Ø

TEST MANUAL

DESK COMPUTER

=-=-=-=-=-=-=-=-=-=-=-=-=-=

COPYRIGHT

DISCLAIMER

INDATA N.V. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, INDATA N.V. reserves the right to revise this Manual and to make changes from time to time in the content hereof without obligation of INDATA N.V. to notify any person of such revision or changes.

First Printing : January 1983

SUMMARY

=-=-=-=-=-=-=-=

INTRODUCTION

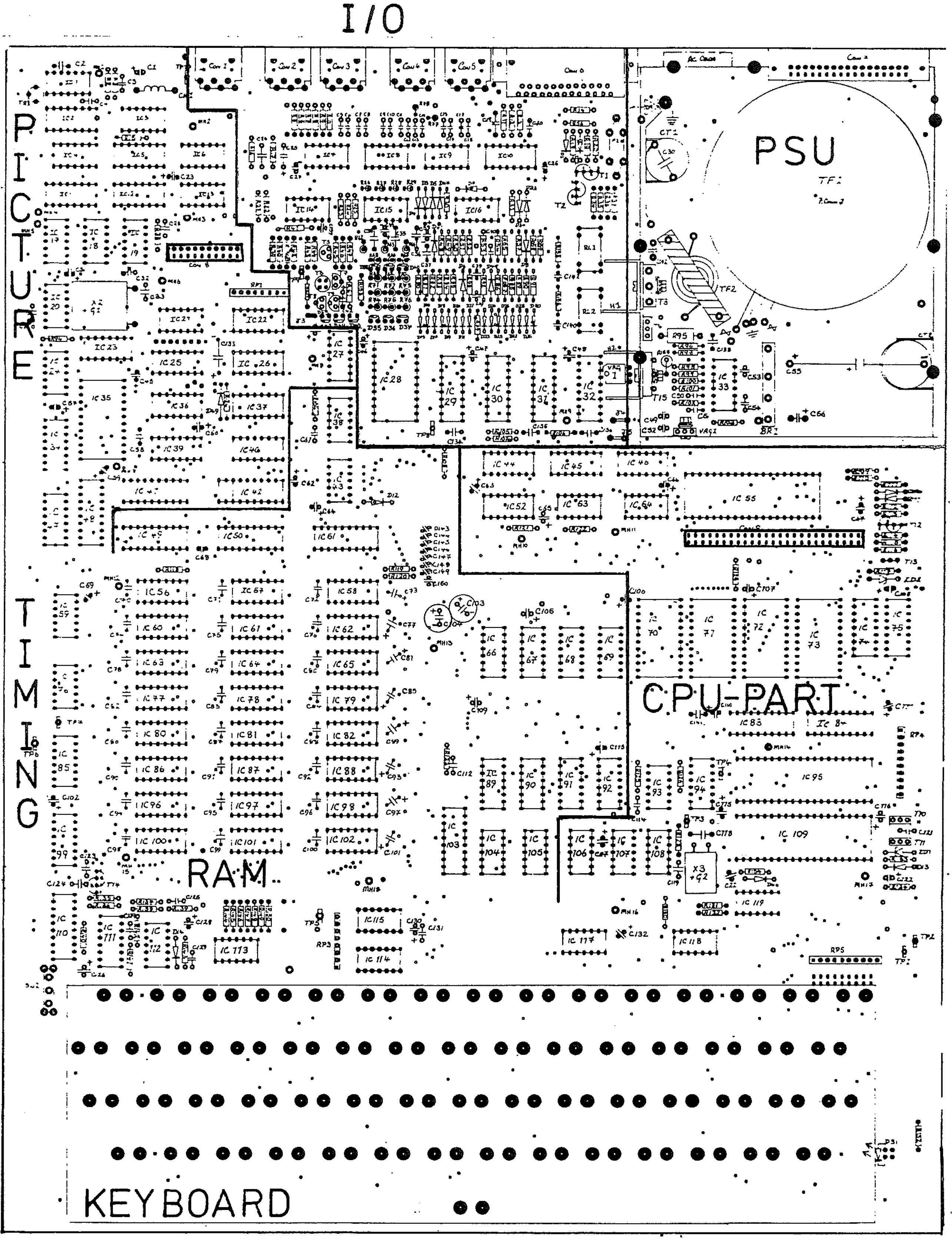
The INDATA DC (DESK COMPUTER) is designed to provide the maximum capability that can economically be provided to an individual. The design is realised such that programs are loaded from a low cost audio cassette or a floppy drive. The result of program execution are output to the user via a PAL or RGB standard television receiver. The Graphical Sound Generation also outputs two tracks of separated sound for left and right stereo combinations, and the sound channel of the television. Fig. 1 is a logical block diagram of the Desk Computer.

The Desk Computer is also equiped with circuitry required to connect two game paddles, two cassette recorder interfaces with motor control, RS232-interface (printer or VDU connection) and a DCE-BUS interface (24 bit parallel I/O Port.)

The Desk Computer is housed in a attractive cream-coloured plastic case, light and yet robust. Behind the keyboard, which has a block metal surround, is a useful well-excellent for holding cassette tapes, pens or pencils. The back plane is also black metal, matching the keyboard. The top case may be removed by popping four plastic plugs.

Inside everything appears neatly laid-out (Fig. 2).
The Desk Computer is partitioned into 6 segments :

- The Power Supply
- Processor Part
- RAM Memory
- Timing
- I/O Section
- PAL or RGB colour card


I/O
PICTURE
PSU
TIMING
RAM
CPU-PART
KEYBOARD
IC 28
IC 29
IC 30
IC 31
IC 32
IC 33
IC 55
IC 95
IC 109
TF1
TF2
X3
CT1

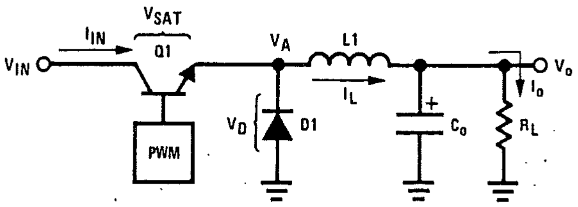
CHAPTER I. THE POWER SUPPLY

The switch of the Power Supply is a red plastic switch which lights when the power is on. A small green led on the board lights also and is a thoughtfull touch since is not always possible to see the power switch. On the left side of the power switch is the AC main connector (always set on 220V).

After the trafo we have an AC voltage of 17V (on both coils) and after the bridge rectifier (BR1) 22V DC (pos. and neg.). These DC voltages are stabilised by C55 and C56.

The +12V and the -5V are obtained by two voltage regulators (7812 and 7905) with on the +12 regulator a current driver (TIP 34B). So the output current of these voltages can be : +12V:4A and -5V:1A). The +5V is not obtained with a usual +5V regulator but with a switching regulator circuit.

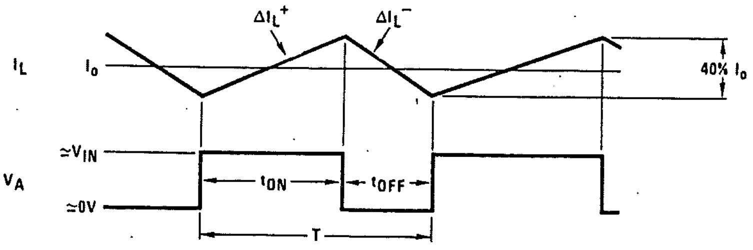
The basic circuit of a step-down switching regulator circuit is shown in Fig. 3



Basic Step-Down Switching Regulator

The circuit works as follows :

Q1 is used as a switch which has ON and OFF times controlled by the pulse width modulator. When Q1 is ON, power is drawn from Vin and supplied to the load through L1; VA is at approximatively Vin. D1 is reverse biased and Co is charging. When Q1 turns OFF the inductor L1 will force VA negative to keep the current flowing in it. D1 will start conducting and the load current will flow through D1 and L1. The voltage at VA is smoothed by the L1, Co filter giving a clean DC output. The current flowing through L1 is equal to the nominal DC load current plus someΔ IL which is due to the changing voltage across it.



The DC convertor without losses or the story of the switching Power Supply

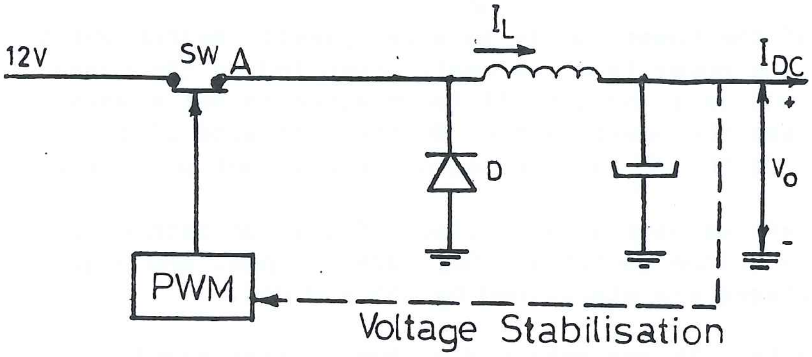
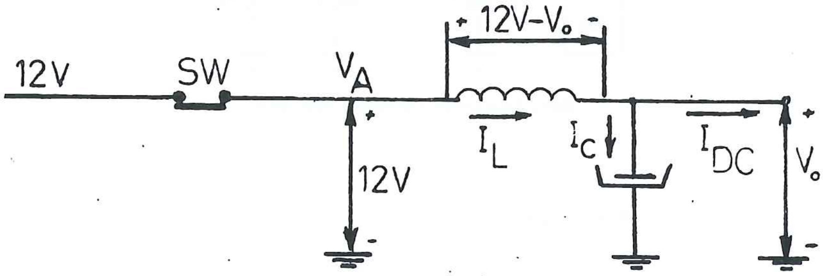


fig 5

The Switch SW has ON and OFF times controlled by the pulse with modulator. The frequency of the PWM is cte but the ON and OFF time can be changed by the error-and comparator-circuit. To understand the working of the circuit we assume that Vo cannot change in such a short period (because of the great LC time cto). Idc is the load current. The time, SW is ON, is t1 and the time, SW is OFF, is t2. The sum of both is the total period T.

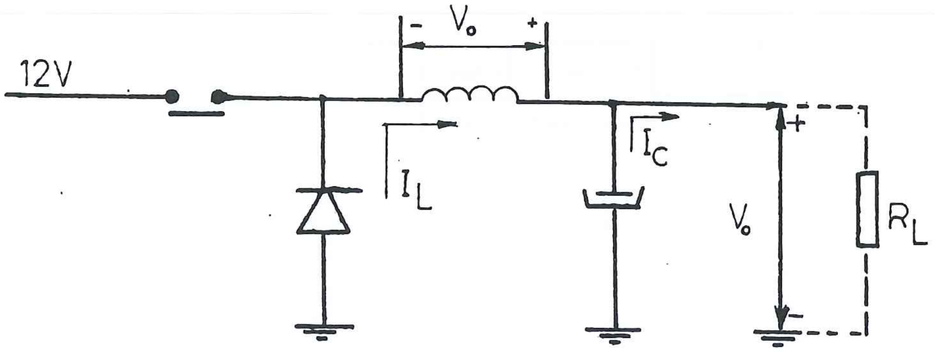
TIME T1



When SW is ON, Power is drawn from Vin=12V and supplied to the load through L1. VA is equal to Vin. D1 is reverse biasd and Co is charging. The voltage over the coil is constant (12V-Vo) and so the current through L will be an increasing sawtooth (di/dt =-eL/L)

The DC part of IL flows through the load while the AC part charge the capacitor. The current IL also builds a magnetic field in the coil so it will take a certain energy.

TIME T2



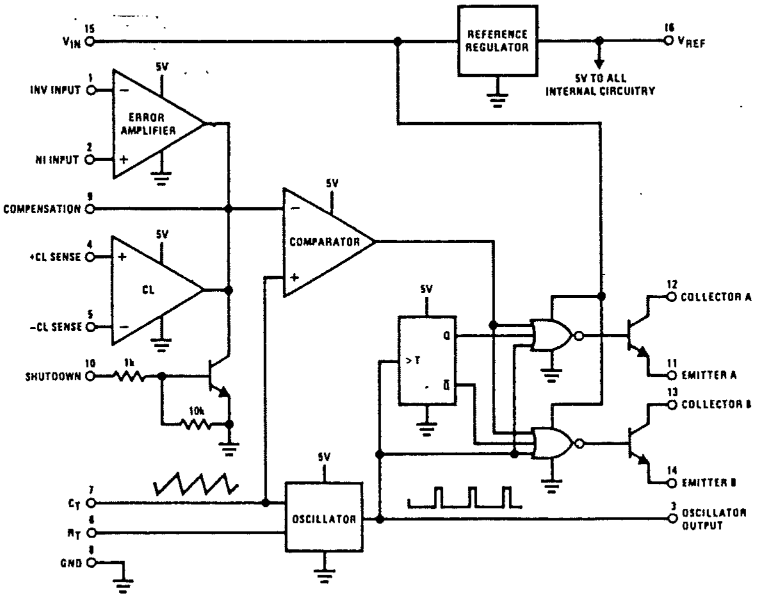
The energy that was built up during T1 will now be used to try to keep the same current in the circuit (current in a coil can't change immediately) so there will be voltage of self induction that has the opposite polarity of the voltage during t1 ($e_L = -L \frac{di}{dt}$) . So the current will again be a savetooth but this time decreasing.

What is the value of Vo?

T is very small and LC has a very big time cte it is obvious that Vo=Vc is the average value of the voltage on A

$$Vo = \frac{12V \times t1}{t1 + t2} = \frac{12V \times t1}{T} = 12V \times \text{duty cycle}$$

So when you change the duty cycle you change the output voltage. The error amplifier measures Vo. When Vo is too high then the duty cycle will be decreased by the comparator so Vo will be decreasing until Vo is too low and the opposite happens. All the control circuity (error amplifier, oscillator, PWM) is installed in the LM 3524. See block diagram Fig. 8.



Internal voltage regulator

The LM3524 has on chip a 5V,50mA short circuit protected voltage regulator. This voltage regulator provides a supply for all internal circuity of the device and can be used as an external reference.

Oscillator

The LM3524 provides a stable on-board oscillator. Its frequency is set by an external resistor Rt and Ct. The oscillator's output provides the signals for triggering an internal Flip Flop, which directs the PWM information to the outputs, and a blanking pulse to turn of both outputs during transitions to ensure that cross conduction doesn't occur. The width of the blanking puls, or dead time is controlled by the value of Ct.

Error amplifier

The error amplifier is a differential input, transconductance amplifier. The amplifier's inputs have a common-mode input range of 1,8V-3,4V. The on board regulator is biasing the inputs in this range.

Output stage

To maintain more power off the +5V (the LM324 gives a max-current of 100 mA) the switch is made of 2 transistors (T3 and T4) which can be used for currents till 4A.

The currents drawn from the Power Supply by the computer (with PAL Color Card) are :

- +12V : 300mA
- + 5V : 2,1A
- - 5V : 50 mA

CHAPTER II. PROCESSOR PART

A) INTRODUCTION

The DESK COMPUTER processor section is designed around the 8080A Microprocessor. The design is based upon the popular and economical DCE microcomputer architecture. The microcomputer section consists of the microprocessor and timing circuity; the ROM and Static RAM memory. Interrupt control and Internal Timer Logic.

This chapter will introduce certain basic computer concepts. It provides background information and definitions which will be useful in later chapters of this manual. Those already familiar with computers may skip this material at their option.

A typical digital computer consists of :

a) A central processor unit (CPU)
b) A memory
c) Input/output (I/O) ports

The memory serves as a place to store **Instructions,** the coded pieces of information that direct the activities of the CPU, and **Data,** the coded pieces of information that are processed by the CPU. A group of logically related instructions stored in memory is referred to as a **Program.** The CPU "reads" each instruction from memory in a logically determined sequence, and uses it to initiate processing actions. If the program sequence is coherent and logical, processing the program will produce intelligible and useful results.

The memory is also used to store the data to be manipulated, as well as the instructions that direct that manipulation. The program must be organized such that the CPU does not read a non-instruction word when it expects to see an instruction. The CPU can rapidly access any data stored in memory; but often the memory is not large enough to store the entire data bank required for a particular application. The problem can be resolved by providing the computer with one or more **Input Ports.** The CPU can address these ports and input the data contained there. The addition of input ports enables the computer to receive information from external equipment (such as cassette recorder or floppy disk) at high rates of speed and in large volumes.

A computer also requires one or more **Output Ports** that permit the CPU to communicate the result of its processing to the outside world. The output may go to a TV, for use by a human operator, to a peripheral device that produces "hard-copy" such as a line-printer, to a peripheral storage device, such as floppy disk unit, or the output may constitute process control signals that direct the operations of another system, such as the DCE Cards. Like input ports, output ports are addressable. The input and output ports together permit the processor to communicate with the outside world.

The CPU unifies the system. It controls the functions performed by the other components. The CPU must be able to fetch instructions from memory, decode their binary contents and execute them. It must also be able to reference memory and I/O ports as necessary in the execution of instructions. In addition, the CPU should be able to recognize and respond to certain external control signals, such as INTERRUPTand WAIT requests.

A typical central processor unit (CPU) consists of the following interconnected functional units :

* Registers
* Arithmetic/Logic Unit (ALU)
* Control Circuitry

Registers are temporary storage units within the CPU. Some registers, such as the program counter and instruction register, have dedicated uses. Other registers, such as the accumulator, are for more general purpose use.

ACCUMULATOR

The accumulator usually stores one of the operands to be manipulated by the ALU. A typical instruction might direct the ALU to add the contents of some other register to the contents of the accumulator and store the result in the accumulator itself. In general, the accumulator is both a source (operand) and a destination (result) register.

Often a CPU will include a number of additional general purpose registers that can be used to store operands or intermediate data. The availability of general purpose registers eliminates the need to "shuffle" intermediate results back and forth between memory and the accumulator, thus improving processing speed and efficiency.

PROGRAM COUNTER (JUMPS, SUBROUTINES AND THE STACK)

The instructions that make up a program are stored in the system's memory. The central processor references the contents of memory, in order to determine what action is appropriate. This means that the processor must know which location contains the next instruction.

Each of the locations in memory is numbered, to distinguish it from all other locations in memory. The number which identifies a memory location is called its **Address**.

The processor maintains a counter which contains the address of the next program instruction. This register is called the **Program Counter**. The processor updates the program counter by adding "1" to the counter each time it fetches an instruction, so that the program counter is always current (pointing to the next instruction).

The programmer therefore stores his instructions in numerically adjacent addresses, so that the lower addresses contain the first instructions to be executed and the higher addresses contain later instructions. The only time the programmer may violate this sequential rule is when an instruction in one section of memory is a **Jump** instruction to another section of memory.

A jump instruction contains the address of the instruction which is to follow it. The next instruction may be stored in any memory location, as long as the programmed jump specifies the correct address. During the execution of a jump instruction, the processor replaces the contents of its program counter with the address embodied in the Jump. Thus, the logical continuity of the program is maintained.

A special kind of program jump occurs when the stored program **"Calls"** a subroutine. In this kind of jump, the processor is required to "remember" the contents of the program counter at the time that the jump occurs. This enables the processor to resume execution of the main program when it is finished with the last instruction of the subroutine.

A **Subroutine** is a program within a program. Usually it is a general-purpose set of instructions that must be executed repeatedly in the course of a main program. Routines which calculate the square, the sine, or the logarithm of a program variable are good examples of functions often written as subroutines. Other examples might be programs designed for inputting or outputting data to a particular peripheral device.

The processor has a special way of handling subroutines, in order to insure an orderly return to the main program. When the processor receives a Call instruction, it increments the Program Counter and stores the counter's contents in a reserved memeory area known as the **Stack.** The stack thus saves the address of the instruction to be execute'd after the subroutine is completed. Then the processor loads the address specified in the Call into its Program Counter. The next instruction fetched will therefore be the first step of the subroutine.

The last instruction in any subroutine is a **Return.** Such an instruction need specify no address. When the processor fetches a Return instruction, it simply replaces the current contents of the Program Counter with the address on the top of the stack. This causes the processor to resume execution of the calling program at the point immediately following the original Call Instruction.

Subroutines are often **Nested;** that is, one subroutine will sometimes call a second subroutine. The second may call a third, and so on. This is perfectly acceptable, as long as the processor has enough capacity to store the necessary return addresses, and the logical provision for doing so. In other words, the maximum depth of nesting is determined by the depth of the stack itself. If the stack has space for storing three return addresses, then three levels of subroutines may be accomodated.

Processors have different ways of maintaining stacks. Some have facilities for the storage of retunr addresses built into the processor itself. Other processors use a reserved area of external memory as the stack and simply maintain a **Pointer** register which contains the address of the most recent stack entry. The two static mosRAM'S (IC74x75 ; sheet 1) are used as stack memory in the Desk Computer. IC74 is the RAM for the low nibble and IC75 is the high nibble RAM). So these RAM's are selected together (as their memory space is only 256x4 bit) to form a 8bit memory. The external stack allows virtually unlimited subroutine nesting. In addition, if the processor provides instructions that cause the contents of the accumulator and other general purpose registers to be "pushed" onto the stack or "popped" off the stack via the address stored in the stack pointer, multi-level interrupt processing (described later in this chapter) is possible.. The status of the processor (i.e. the contents of all the registers) can be saved in the stack when an interrupt is accepted and then restored after the interrupt has been serviced. This ability to save the processor's status at any given time is possible even if an interrupt service routine, itself, is interrupted.

INSTRUCTION REGISTER AND DECODER

Every computer has a **Word Length** that is characteristic of that machine. A computer's word length is usually determined by the size of its internal storage elements and interconnecting paths (referred to as **Busses**); for example, a computer whose registers and busses can store and transfer 8 bits of information has a characteristic word length of 8-bits and is referred to as an 8-bit parallel processor. An eight-bit parallel processor generally finds it most efficient to deal with eight-bit binary fields, and the memory associated whith such a processor is therefore organized to store eight bits in each addressable memory location. Data and instructions are stored in memory as eight-bit binary numbers, or as numbers that are integral multiples of eight bits : 16 bits, 24 bits, and so on. This characteristic eight-bit field is often referred to as a **Byte.**

Each operation that the processor can perform is identified by a unique byte of data known as an **Instruction Code** or **Operation Code.** An eight-bit word used as an instruction code can distinguish between 256 alternative actions, more than adequate for most processors.

The processor fetches an instruction in two distinct operations. First, the processor transmits the address in its Program Counter to the memory . Then the memory returns the addressed byte to the processor. The CPU stores this instruction byte in a register known as the **Instruction Register,** and uses it to direct activities during the remainder of the instruction execution.

The mechanism by which the processor translates an instruction code into specific processing actions requires more elaboration than we can here afford. The concept, however, should be intuitively clear to any logic designer. The eight bits stored in the instruction register can be decoded and used to selectively activate one of a number of output lines, in this case up to 256 lines. Each line represents a set of activities associated with execution of a particular instruction code. The enabled line can be combined with selected timing pulses, to develop electrical signals that can then be used to initiate specific actions. This translation of code into action is performed by the **Instruction Decoder** and by the associated control circuitry.

An eight-bit instruction code is often sufficient to specify a particular processing action. There are times, however, when execution of the instruction requires more information than eight bits can convey.

One example of this is when the instruction references a memory location. The basic instruction code identifies the operation to be performed, but cannot specify the object address as well. In a case like this, a two-or three-byte instruction must be used. Successive instruction bytes are stored in sequentially adjacent memory locations , and the processor performs two or three fetches in succession to obtain the full instruction. The first byte retrieved from memory is placed in the processor's instruction register, and subsequent bytes are placed in temporary storage; the processor then proceeds with the execution phase. Such an instruction is referred to as **Variable Length.**

ADDRESS REGISTER(S)

A CPU may use a register or register-pair to hold the address of a memory location that is to be accessed for data. If the address register is **Programmable**, (i.e. if there are instructions that allow the programmer to alter the contents of the register) the program can "build" an address in the address register prior to executing a **Memory Reference** instruction (i.eW. an instruction that reads data from memory, writes data to memory or operates on data stored in memory).

ARITHMETIC/LOGIC UNIT (ALU)

All processors contain an arithmetic/logic unit, which is often referred to simply as the **ALU.** The ALU as its name implies, is that portion of the CPU hardware which performs the arithmetic and logical operations on the binary data.

The ALU must contain an **Adder** which is capable of combining the contents of two registers in accordance with the logic of binary arithmetic. This provision permits the processor to perform arithmetic manipulations on the data it obtains from memory and from its other inputs.

Using only the basic adder a capable programmer can write routines which will subtract, multiply and divide, giving the machine complete arithmetic capabilities. In practice, however, most ALU's provide other built-in functions, including hardware subtraction, boolean logic operations, and shift capabilities.

The ALU contains **Flag Bits** which specify certain conditions that arise in the course of arithmetic and logical manipulations. Flags typically include **Carry, Zero, Sign,** and **Parity.** It is possible to program jumps which are conditionally dependent on the status of one or more flags. Thus, for example, the program may be designed to jump to a special routine if the carry bit is set following an addition instruction.

CONTROL CIRCUITRY

The control circuitry is the primary functional unit within a CPU. Using clock inputs, the control circuitry maintains the proper sequence of events required for any processing task. After an instruction is fetched and decoded, the control circuitry issues the appropriate signals (to units both internal and external to the CPU) for initiating the proper processing action. Often the control circuitry will be capable of responding to external signals, such as an interrupt or wait request. An **Interrupt** request will cause the control circuitry to temporarily interrupt main program execution, jump to a special routine to service the interrupting device, then automatically return to the main program. A **Wait** request is often issued by a memory or I/O element that operates slower than the CPU (for example the 9511 Math Chip Processor). The control circuitry will idle the CPU until the memory or I/O port is ready with the data.

COMPUTER OPERATIONS

There are certain operations that are basic to almost any computer. A sound undestanding of these basic operations is a necessary prerequisite to examining the specific operations of a particular computer.

TIMING :

The activities of the central processor are cyclical. The processor fetches an instruction, performs the operations required, fetches the next instruction, and so on. This orderly sequence of events requires precise timing, and the CPU therefore requires a free running oscillator clock which furnishes the reference for all processor actions. The combined fetch and execution of a single instruction is refered to as an **Instruction Cycle.** The portion of a cycle identified with a clearly defined activity is called a **State.** And the interval between pulses of the timing oscillator is referred to as a **CLock Period.** As a general rule, one or more clock periods are necessary for the completion of a state, and there are several states in a cycle.

INSTRUCTION FETCH

The first state(s) of any instruction cycle will be dedicated to fetching the next instruction. The CPU issues a read signal and the contents of the program counter are sent to memory, which responds by returning the next instruction word. The first byte of the instruction is placed in the instruction register. If the instruction consist of more than one byte, additional states are required to fetch each byte of the instruction. When the entire instruction is present in the CPU, the program counter is incremented (in preparation for the next instruction fetch) and the instruction is decoded. The operation specified in the instruction will be executed in the remaining states of the instruction cycle. The instruction may call for a memory read or write, an input or output and/or an internal CPU operation, such as a register-to-register transfer or an add-registers operation.

MEMORY READ

An instruction **fetch** is merely a special memory read operation that brings the instruction to the CPU's instruction register. The instruction fetched may then call for data to be read from memory into the CPU. The CPU again issues a read signal and sends the proper memory address; memory responds by returning the requested word. The data received is placed in the accumulator or one of the other general purpose registers (not the instruction register).

MEMORY WRITE

A memory write operation is similar to a read except for the direction of data flow. The CPU issues a write signal, send the proper memory address, then sends the data word to be written into the addressed memory location.

WAIT (MEMORY SYNCHRONIZATION)

As previously stated, the activities of the processor are timed by a master clock oscillator. The clock period determines the timing of all processing activity.

The speed of the processing cycle, however, is limited by the memory's **Access Time** Once the processor has sent a read address to memory, it cannot proceed until the memory has had time to respond. Most memories are capable of responding much faster than the processing cycle requires. A few, however, cannot supply the addressed byte within the minimum time established by the processor's clock.

Therefore a processor should contain a synchronization provision, which permits the memory to request a **Wait State**. When the memory receives a read or write enable signal, it places a request signal on the processor's **READY** line, causing the CPU to idle temporarily. After the memory has had time to respond, it frees the processor's **READY** line and the instruction cycle proceeds.

INPUT/OUTPUT

Input and Output operations are similar to memory read and write operations with the exception that a peripheral I/O device is addressed instead of a memory location. The CPU issues the appropriate input or output control signal, sends the proper device address and either receives the data being input or sends the data to be output.

Data can be input/output in either parallel or serial form. All data within a digital computer is represented in binary coded form. A binary data word consists of a group of bits; each bit is either a one or a zero. **Parallel I/O** consists of transferring all bits in the word at the same time, one bit per line (8255). **Serial I/O** consists of transferring one bit at a time on a single line (5501). Naturally serial I/O is much slower, but it requires considerably less hardware than does parallel I/O.

INTERRUPTS

Interrupt provisions are included on many central processors, as a means of improving the processor's efficiency. Consider the case of a computer that is processing a large volume of data, portions of which are to be output to a printer. The CPU can output a byte of data within a single machine cycle but it may take the printer the equivalent of many machine cycles to actually print the character specified by the data byte. The CPU could then remain idle waiting until the printer can accept the next data byte. If an interrupt capability is implemented on the computer, the CPU can output a data byte then return to data processing. When the printer is ready to accept the next data byte, it can request an interrupt. When the CPU acknowledges the interrupt, it suspends main program execution and automatically branches to a routine that will output the next data byte. After the byte is output, the CPU continues with main program execution. Note that this is, in principle, quite similar to a subroutine call, except that the jump is initiated externally rather than by the program.

More complex interrrupt structures are possible, in which several interrupting devices share the same processor but have different priority levels. Interruptive processing is an important feature that enables maximum untilization of a processor's capacity for high system throughput.

HOLD (Used for DCE-BUS CONNECTION;PIN 49; SHEET 1)

Another important feature that improves the throughput of a processor is the **Hold**. The hold provision enables **Direct Memory Access (DMA)** operations.

In ordinary input and output operations, the processor itself supervises the entire data transfer. Information to be placed in memory is transferred from the input device to the processor, and then from the processor to the designated memory location. In similar fashion, information that goes from memory to output devices goes by way of the processor.

Some peripheral devices, however, are capable of transferring information to and from memory much faster than the processor itself can accomplish the transfer. If any appreciable quantity of data must be transferred to or from such a device, then **system throughput** will be increased by having the device accomplish the transfer directly. The processor must temporarily suspend its operation during such a transferr, to prevent conflicts that would arise if processor and peripheral device attempted to access memory simultaneously. It is for this reason that a **hold** provision is included on some processors.

B) THE 8080 CENTRAL PROCESSOR UNIT

The 8080 is a complete 8-bit parallel, central processor unit (CPU) for use in general purpose digital computer systems. It is fabricated on a single LSI chip using Intel's n-channel silicon gate MOS process. The 8080 transfers data and internal state information via an 8-bit, bidirectional 3-state Data Bus (Do-D7). Memory and peripheral device addresses are transmitted over a separate 16-bit 3-state Address Bus (Ao-A15). Six timing and control outputs (SYNC,DBIN,WAIT,WR,HLDA and INTE) emanate from the 8080, while four control inputs (READY,HOLD,INT and RESET), four power inputs (+12V, +5V, -5V, and GND) and two clock inputs (Ø1 and Ø2) are accepted by the 8080

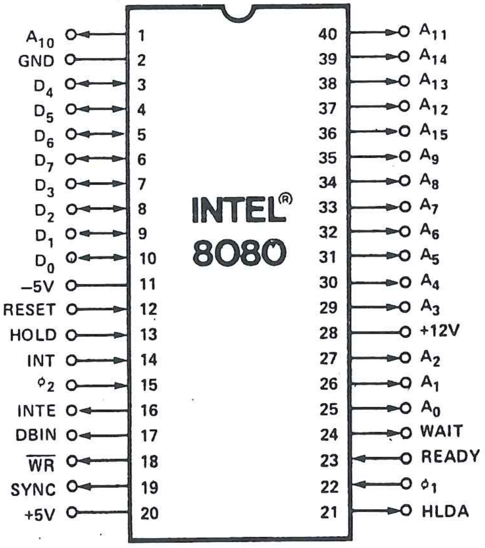


Fig. 2.1

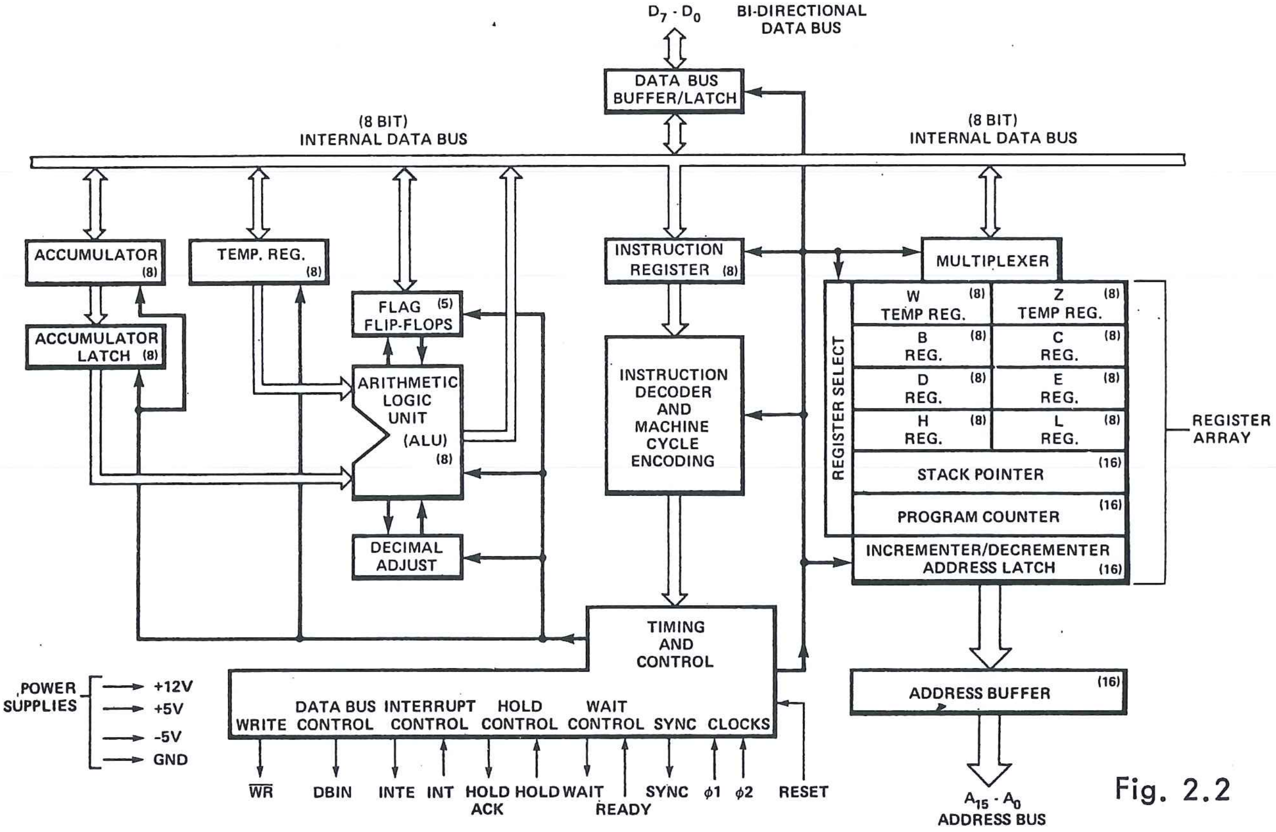


Fig. 2.2

ARCHITECTURE OF THE 8080 CPU

The 8080 CPU consists of the following functional units :

* Register array and address logic
* Arithmetic and logic unit (ALU)
* Instruction register and control section
* Bi-directional, 3-state data bus buffer

Figure 2.2 illustrates the functional blocks within the 8080 CPU.

REGISTERS

The register section consists of a static RAM array organized into six 16-bit registers :

* Program counter (PC)
* Stack pointer (SP)
* Six 8-bit general purpose registers arranged in pairs, referred to as B,C; D,E; and H,L
* A temporary register pair called W,Z

The program counter maintains the memory address of the current program instruction and is incremented automatically during every instruction fetch. The stack pointer maintains the address of the next available stack location in memory. The stack pointer can be initialized to use any portion of read-write memory as a stack. The stack pointer is decremented when data is "pushed" onto the stack and incremented when data is "popped" off the stack (i.e. the stack grows "downward").

The six general purpose registers can be used either as single registers (8-bit) or as register (16-bit). The temporary register pair, W,Z, is not program addressable and is only used for the internal execution of instructions.

Eight-bit data bytes can be transferred between the internal bus and the register array via the register-select multiplexer. Sixteen-bit transfers can proceed between the register array and the address latch or the incrementer/decrementer circuit. The address latch receives data from any of the three register pairs and drives the 16 address output buffers (Ao-A15), as well as the incrementer/decrementer circuit. The incrementer/decrementer circuit receives data from the address latch and sends it to the register array. The 16-bit data can be incremented or decremented or simply transferred between registers.

ARITHMETIC AND LOGIC UNIT (ALU)

The ALU contains the following registers :

* An 8-bit accumulator
* An 8-bit temporary accumulator (ACT)
* A 5-bit flag register : zero, carry, sign, parity and auxiliary carry
* An 8-bit temporary register (TMP)

Arithmetic, logical and rotate operations are performed in the ALU. The ALU is fed by the temporary register (TMP) and the temporary accumulator (ACT) and carry flip-flop. The result of the operation can be transferred to the internal bus or to the accumulator; the ALU also feeds the flag register.

The temporary register (TMP) receives information from the internal bus and can send all or portions of it to the ALU, the flag register and the internal bus.

The accumulator (ACC) can be loaded from the ALU and the internal bus and can transfer data to the temporary accumulator (ACT) and the internal bus. The contents of the accumulator (ACC) and the auxiliary carry flip-flop can be tested for decimal correction during the execution of the DAA instruction.

INSTRUCTION REGISTER AND CONTROL

During an instruction fetch, the first byte of an instruction (containing the OP code) is transferred from the internal bus to the 8-bit instruction register.

The contents of the instruction register are, in turn, available to the instruction decoder. The output of the decoder, combined with various timing signals, provides the control signals for the register array, ALU and data buffer blocks. In addition, the outputs from the instruction decoder and external control signals feed the timing and state control section which generates the state and cycle timing signals.

DATA BUS BUFFER

The 8-bit bidirectional 3-state buffer is used to isolate the CPU's internal bus from the external data bus. (D0 through D7). In the output mode, the internal bus content is loaded into a 8-bit latch that, in turn, drives the data bus output buffers. The output buffers are switched off during input or non-transfer operations.

During the input mode, data from the external data bus is transferred to the internal bus. The internal bus is precharged at the beginning of each internal state, except for the transfer state (T3-described later in this chapter).

THE PROCESSOR CYCLE

An instruction cycle is defined as the time required to fetch and execute an instruction. During the fetch, a selected instruction (one, two or three bytes) is extracted from memory and deposited in the CPU's instruction register. During the execution phase, the instruction is decoded and transleted into specific processing activities.

Every instruction cycle consists of one, two, three, four or five machine cycles. A **machine cycle** is required each time the CPU accesses memory or an I/O port. The fetch portion of an instruction cycle requires one machine cycle for each byte to be fetched. The duration of the execution portion of the instruction cycle depends on the kind of instruction that has been fetched. Some instructions do not require any machine cycles other than those necessary to fetch the instruction; other instructions, however, require additional machine cycles to write or read data to/from memory or I/O devices. The DAD instruction is an exception in that it requires two additional machine cycles to complete an internal register-pair add.

Each machine cycle consists of three, four or five states. A state is the smallest unit of processing activity and is defined as the interval bewteen two successive positive-going transitions of the Ø1 driven clock pulse. The 8080 is driven by a two-phase clock oscillator. All processing activities are referred to the period of this clock. The two non-overlapping clock pulses, labeled Ø1 and Ø2, are furnished by the 8224.

The 8224 (IC 94) is a single chip Clock Generator/Driver for the 8080 CPU. It contains a crystal-controlled oscillator, a "divide by nine" counter, two high-level drivers and several auxiliary logic functions.

The oscillator circuit derives its basic operating frequency from an external, series resonant, fundamental mode crystal. Two inputs are povided for the crystal connections (XTAL1,XTAL2).

The selection of the external crystal frequency depends mainly on the speed at which the 8080A is to be run at. Basically, the oscillator operates at 9 times the desired procecssor speed.

A simple formula to guide the crystal selection is :

$$\text{Crystal frequency} = 1 \text{ times } 9$$
$$\overline{\text{tCY}}$$

(500ns tCY)
2mHz times 9 = 18mHz*

The output of the oscillator is buffered and brought out on OSC (pin 12) so that other system timing signals can be derived from this stable, crystal-controlled source.

* When using crystals above 10mHz a small amount of frequency "trimming" may be necessary to produce the exact desired frequency. The addition of a small selected capacitance (10pF)(C118) in series with the crystal will accomplish this function.

The Clock Generator consists of a synchronous "divide by nine" counter and the associated decode gating to create the waveforms of the two 8080A clocks and auxiliary timing signals.

The waveforms generated by the decode gating follow a simple 2-5-2 digital pattern. See Fig. 2-4. The clocks generated; phase 1 and phase 2, can best be thought of as consisting of "units" based on the oscillator frequency. Assume that one "unit" equals the period of the oscillator frequency. By multiplying the number of "units" that are contained in a pulse width or delay, times the period of the oscillator frequency, the approximate time in nanoseconds can be derived.

The outputs of the clock generator are connected to two high level drivers for direct interface to the 8080A CPU. A TTL level phase 2 is also brought out Ø2 (TTL) for external timing purposes.

Several other signals are also generated internally so that optimum timing of the auxiliary flip-flops and status strobe (STSTB) is achieved.

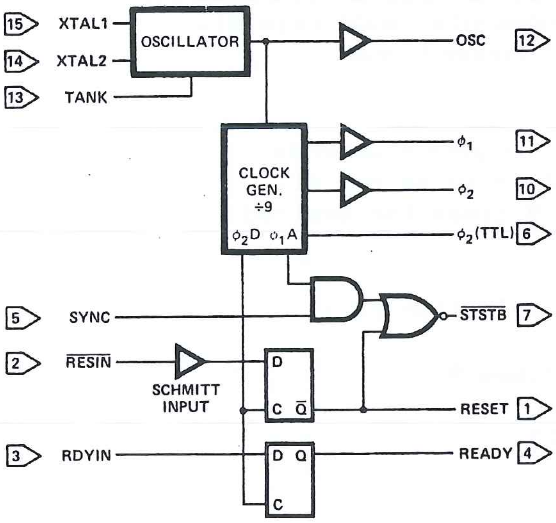


Fig. 2.3

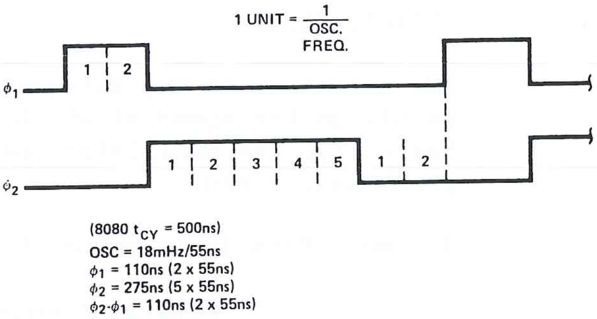

1 UNIT = $\frac{1}{\text{OSC. FREQ.}}$
ϕ_1
ϕ_2
1
2
1
2
3
4
5
1
2
(8080 t_{CY} = 500ns)
OSC = 18mHz/55ns
ϕ_1 = 110ns (2 x 55ns)
ϕ_2 = 275ns (5 x 55ns)
ϕ_2-ϕ_1 = 110ns (2 x 55ns)

Fig. 2.4

At the beginning of each machine cycle the 8080A CPU issues status information on its data bus. This information tells what type of action will take place during that machine cycle. By bringing in the SYNC signal from the CPU, and gating it with an internal timing signal, (Ø1A) an active low strobe can be derived that occurs at the start of each machine cycle at the earliest possible moment that status data is stable on the bus.

The power-on Reset also generates STSTB but of course for a longer period of time.

A common function in 8080A Microcomputer systems is the generation of an automatic system reset and start-up upon initial power-on. The 8224 has a built in feature to accomplish this feature.

THE POWER ON RESET CIRCUIT

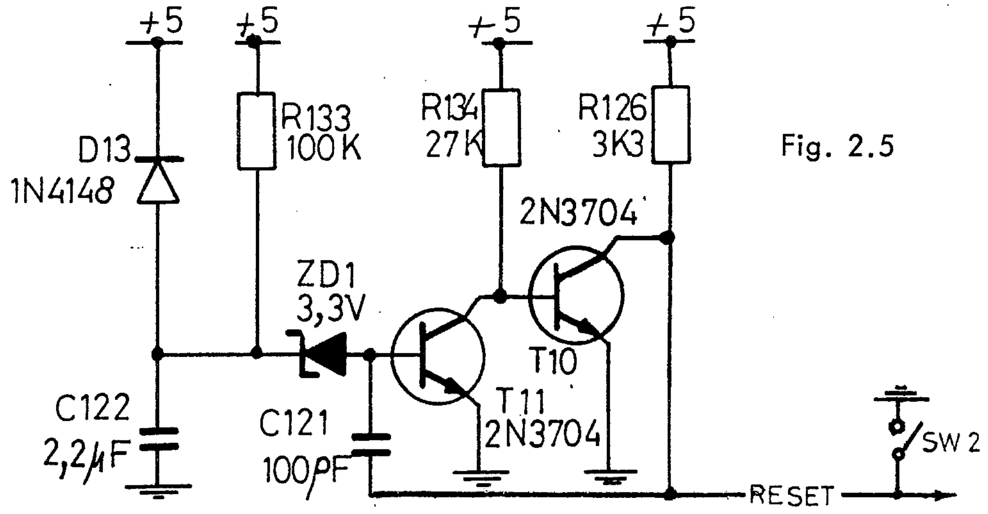
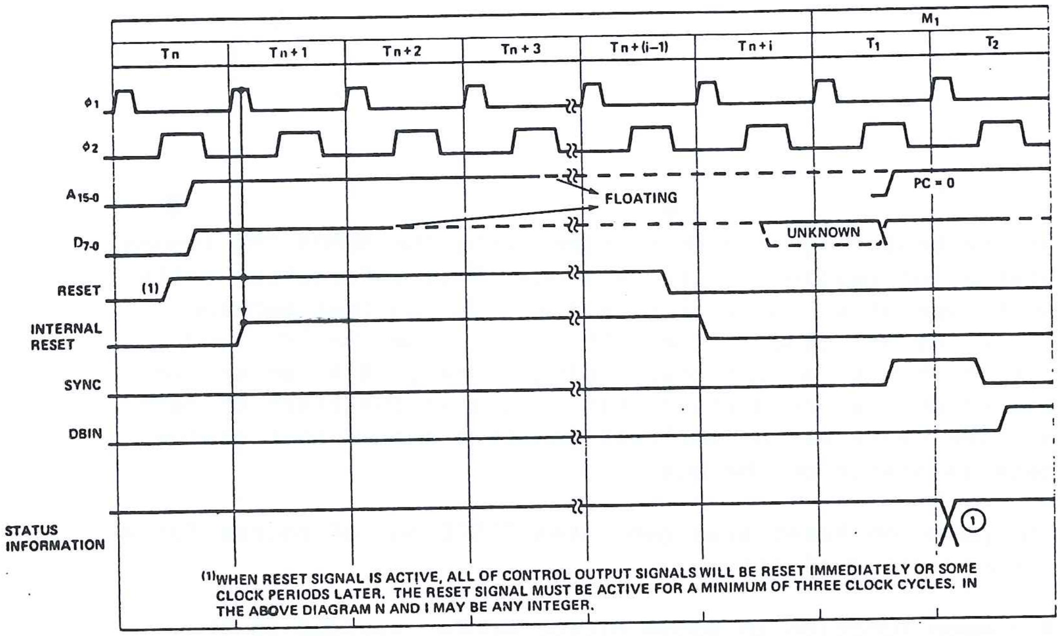


Fig. 2.5

When power is supplied initially to the 8080 the processor begins operating immediately. The contents of its program counter, stack pointer, and the other working registers are naturally subject to random factors and cannot be specified. For this reason, it will be necessary to begin the Power-up sequence with Reset. An external reset signal of three clock period duration (minimum) restore the processor's internal program counter to zero Program execution thus begins with memory location zero, following a reset. Note, however, that the reset has no effect on status flags, or on any of the processor's working registers (accumulator, registers, or stack pointer). The contents of these registers remain indeterminate until initialized explicitely by the Program.



(1)WHEN RESET SIGNAL IS ACTIVE, ALL OF CONTROL OUTPUT SIGNALS WILL BE RESET IMMEDIATELY OR SOME CLOCK PERIODS LATER. THE RESET SIGNAL MUST BE ACTIVE FOR A MINIMUM OF THREE CLOCK CYCLES. IN THE ABOVE DIAGRAM N AND I MAY BE ANY INTEGER.

NOTE: (N) Refer to Status Word Chart

Fig. 2.6

<u>**The circuit**</u> (Fig. 2.5)

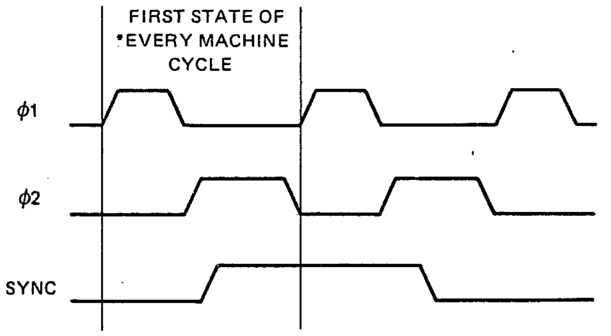
When Power is applied, the voltage of C122 can't change immediately. At the beginning is the voltage zero volts because C122 is discharged via D13 and the small resistance between the +5V and ground.

T11 is OFF so T10 is ON and the RESIN (PIN2) off the 8224 is low. The 8080 gets its reset from PIN1 of the 8224. When the charge of C122 (via R133:100K) gets above 3,3V then T11 turns ON; T10 runs OFF and the RESET off the CPU becomes low. The capacitor (C145) takes care that there is no bounce of the circuit during Power on.

The READY input to the 8080A CPU has certain timing specifications such as "set-up and hold" thus, an external synchronizing flip-flop is required. The 8224 has this feature built-in. The RDYIN input presents the asynchronous "wait request" to the "D" type flip-flop. By clocking the flip-flop with Ø2D, a synchronized READY signal at the correct input level, can be connected directly to the 8080A.

The reason for requiring an external flip-flop to synchronize the "wait request" rather than internally in the 8080 CPU is that due to the relatively long delays of MOS logic such an implementation would "rob" the designer of about 200ns during the time his logic is determining if a "wait" is necessary. An external bipolar circuit built into the clock generator eliminates most of this delay and has no effect on component count.

The Ø1 clock pulse divides each machine cycle into states. Timing logic within the 8080 uses the clock inputs to produce a SYNC pulse, which identifies the beginning of every machine cycle. The SYNC pulse is triggered by the low-to-high transition of Ø2, as shown in Fig. 2.7



*SYNC DOES NOT OCCUR IN THE SECOND AND THIRD MACHINE CYCLES OF A DAD INSTRUCTION SINCE THESE MACHINE CYCLES ARE USED FOR AN INTERNAL REGISTER-PAIR ADD.

There are three exceptions to the defined duration of a state. They are the WAIT state, the hold (HLDA) state and the halt (HLTA) state, described later in this chapter. Because the WAIT, the HLDA, and the HLTA states depend upon external events, they are by their nature of indeterminate length. Even these exceptional states, however, must be synchronized with the pulses of the driving clock. Thus, the duration of all states are integral multiples of the clock period.

To summarize then, each **clock period** marks a **state**, three to five states constitute a machine cycle; and one to five **machine cycles** comprise an **instruction cycle.** A full instruction cycle requires anywhere from four to eighteen states for its completion, depending on the kind of instruction involved.

MACHINE CYCLE IDENTIFICATION

With the exception of the DAD instruction, there is just one consideration that determines how many machine cycles are required in any given instruction cycle; the number of times that the processor must reference a memory address or an addressable peripheral device, in order to fetch and execute the instruction. Like many processors, the 8080 is so constructed that it can transmit only one address per machine cycle. Thus, if the fetch and execution of an instruction requires two memory references, then the instruction cycle associated with that instruction consists of two machine cycles. If five such references are called for, then the instruction cycle contains five machine cycles.

Every instruction cycle has at least one reference to memory, during which the instruction is fetched. An instruction cycle must always have a fetch, even if the execution of the instruction requires no further references to memory. The first machine cycle in every instruction cycle is therefore a FETCH. Beyond that, there are no fast rules. It depends on the kind of instruction that is fetched.

Consider some examples. The add-register (ADD r) instruction is an instruction that requires only a single machine cycle (FETCH) for its completion. In this one-byte instruction, the contents of one of the CPU's six general purpose registers is added to the existing contents of the accumulator. Since all the information necessary to execute the command is contained in the eight bits of the instruction code, only one memory reference is necessary. Three states are used to extract the instruction from memory, and one additional state is used to accomplish the desired addition. The entire instruction cycle thus requires only one machine cycle that consists of four states, or four periods of the external clock.

Suppose now, however, that we wish to add the contents of a specific memory location to the existing contents of the accumulator (ADD M). Although this is quite similar in principe to the example just cited, several additional steps will be used. An extra machine cycle will be used, in order to address the desired memory location.

The actual sequence is as follows. First the processor extracts from memory the one-byte instruction word addressed by its program counter. This takes three states. The eight-bit instruction word obtained during the FETCH machine cycle is deposited in the CPU's instruction register and used to direct activities during the remainder of the instruction cycle. Next, the processor sends out, as an address, the contents of its H and L registers. The eight-bit data word returned during this MEMORY READ machine cycle is placed in a temporary register inside the 8080 CPU. By now three more clock periods (states) have elapsed. In the seventh and final state, the contents of the temporary register are added to those of the accumulator. Two machine cycles, consisting of seven states in all, complete the "ADD M" instruction cycle.

At the opposite extreme is the save H and L registers (SHLD) instruction, which requires five machine cycles. During an "SHLD" instruction cycle, the contents of the processor's H and L registers are deposited in two sequentially adjacent memory lcations, the destination is indicated by two address bytes which are stored in the two memory locations immediately following the operation code byte. The following sequence of events occurs :

1) A FETCH machine cycle, consisting of four states. During the first three states of this machine cycle, the processor fetches the instruction indicated by its program counter. The program counter is then incremented. The fourth state is used for internal instruction decoding.

2) A MEMORY READ machine cycle, consisting of three states. During this machine cycle, the byte indicated by the program counter is read from memory and placed in the processor's Z register. The program counter is incremented again.

3) Another MEMORY READ machine cycle, consisting of three states, in which the byte indicated by the processor's program counter is read from memory and placed in the W register. The program counter is incremented, in anticipation of the next instruction fetch.

4) A MEMORY WRITE machine cycle, of three states, in which the contents of the L register are transferred to the memory location pointed to by the present contents of the W and Z registers. The state following the transfer is used to increment the W,Z register pair so that it indicates the next memory location to receive data.

5) A MEMORY WRITE machine cycle, of three states, in which the contents of the H register are transferred to the new memory location pointed to by the W,Z register pair.

In summary, the "SHLD" instruction cycle contains five machine cycles and takes 16 states to execute.

Most instructions fall somewhere between the extremes typified by the "ADD r" and the "SHLD" instructions. The input (INP) and the output (OUT) instructions, for example, require three machine cycles :

* a FETCH to obtain the instruction
* a MEMORY READ to obtain the address of the object peripheral
* an INPUT or an OUTPUT machine cycle, to complete the transfer

While no one instruction cycle will consist of more then five machine cycles, the following ten different types of machine cycles may occur within an instruction cycle :

(1) FETCH (M1)
(2) MEMORY READ
(3) MEMORY WRITE
(4) STACK READ
(5) STACK WRITE
(6) INPUT
(7) OUTPUT
(8) INTERRUPT
(9) HALT
(10) HALT . INTERRUPT

The machine cycles that actually do occur in a particular instruction cycle depend upon the kind of instruction with the overriding stipulation that the first machine cycle in any instruction cycle is always a FETCH.

The processor identifies the machine cycle in progress by transmitting an eight-bit status word during the first state of every machine cycle. Updated statuts information is presented on the 8080's data lines (D0-D7) during the SYNC interval. This data should be saved in latches, and used to develop control signals for external circuitry. Table 1 shows how the positive-true status information is distributed on the processor's data bus.

Status signals are provided principally for the control of external circuitry. Simplicity of interface, rather than machine cycle identification, dictates the logical definition of individual status bits. You will therefore observe that certain processor machine cycles are uniquely identified by a single status bit, but that others are not. The M1 status bit (D6) for example, unambiguously identifies a FETCH machine cycle. A STACK READ, on the other hand, is indicated by the coincidence of STACK and MEMR signals. Machine cycle identification data is also valuable in the test and de-bugging phases of system development. Table 1 lists the status bit outputs for each type of machine cycle.

STATUS INFORMATION DEFINITION

Symbols	Data Bus Bit	Definition
INTA	Do	Acknowledge signal for INTERRUPT request. Signal should be used to gate a restart instruction onto the data bus when DBIN is active.
WO	D1	Indicates that the operation in the current machine cycle will be a WRITE memory or OUTPUT function (WO = 0). Otherwise a READ memory or INPUT operation will be executed.
STACK	D2	Indicates that the address bus holds the pushdown stack address from the Stack Pointer.
HLTA	D3	Acknowledge signal for HALT instruction.
OUT	D4	Indicates that the address bus contains the address of an output device and the data bus will contain the output data when WR is active.
M1	D5	Provides a signal to indicate that the CPU is in the fetch cycle for the first byte of an instruction.
INP	D6	Indicates that the address bus contains the address of an input device and the input data should be placed on the data bus when DBIN is active.
MEMR	D7	Designates that the data bus will be used for memory read data.

STATUS WORD CHART

TYPE OF MACHINE CYCLE

DATA BUS BIT	STATUS INFORMATION	INSTRUCTION FETCH	MEMORY READ	MEMORY WRITE	STACK READ	STACK WRITE	INPUT READ	OUTPUT WRITE	INTERRUPT ACKNOWLEDGE	HALT ACKNOWLEDGE	INTERRUPT ACKNOWLEDGE WHILE HALT
		①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
D_0	INTA	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{\text{WO}}$	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0

Ⓝ STATUS WORD

Table 1. 8080 Status Bit Definitions

STATE TRANSITION SEQUENCE

Every machine cycle within an instruction cycle consists of three to five active states (referred to as T1,T2,T3,T4,T5 or TW). The actual number of states depends upon the instruction being executed, and on the particular machine cycle within the greater instruction cycle. The state transition diagram in Figure 2.10 shows how the 8080 proceeds from state to state in the course of a machine cycle. The diagram also shows how the READY, HOLD and INTERRUPT lines are sampled during the machine cycle, and how the conditions on these lines may modify the basic transition sequence. In the present discussion, we are concerned only with the basic sequence and with the READY function. The HOLD and INTERRUPT functions will be discussed later.

The 8080 CPU does not directly indicate its internal state by transmitting a "state control" output during each state; instead, the 8080 supplies direct control output (INTE, HLDA, DBIN, WR and WAIT) for use by external circuitry.

Recall that the 8080 passes through at least three states in every machine cycle , with each state defined by successive low-to-high transitions of the Ø1 clock. Figure shows the timing relationships in a typical FETCH machine cycle. Events that occur in each state are referenced to transitions of the Ø1 and Ø2 clock pulses.

The SYNC signal identifies the first state (T1) in every machine cycle. As shown in Figure 2.11, the SYNC signal is related to the leading edge of the Ø2 clock. There is a delay (tDC) between the low-to-high transition of Ø2 and the positive-going edge of the SYNC pulse. There also is a corresponding delay (also tDC) between the next Ø2 pulse and the falling edge of the SYNCH signal. Status information is displayed on D0-D7 during the same Ø2 to Ø2 interval. Switching of the status signals is likewise controlled by Ø2.

The rising edge of Ø2 during T1 also leads the processor's address lines (A0-A15). These lines become stable within a brief delay (tDA) of the Ø2 clocking pulse and they remain stable until the first Ø2 pulse after state T3. This gives the processor ample time to read the data returned from memory.

Once the processor has sent an address to memory, there is an opportunity for the memory to request a WAIT. This it does by pulling the processor's READY line low, prior to the "Ready set-up" interval (tRS) which occurs during the Ø2 pulse within state T2 or TW. As long as the READY line remains low, the processor will idle, giving the memory time to respond to the addressed data request. Refer to Figure 2.11.

The processor responds to a wait request by entering an alternative state (TW) at the end of T2, rather than proceeding directly to the T3 state. Entry into the TW state

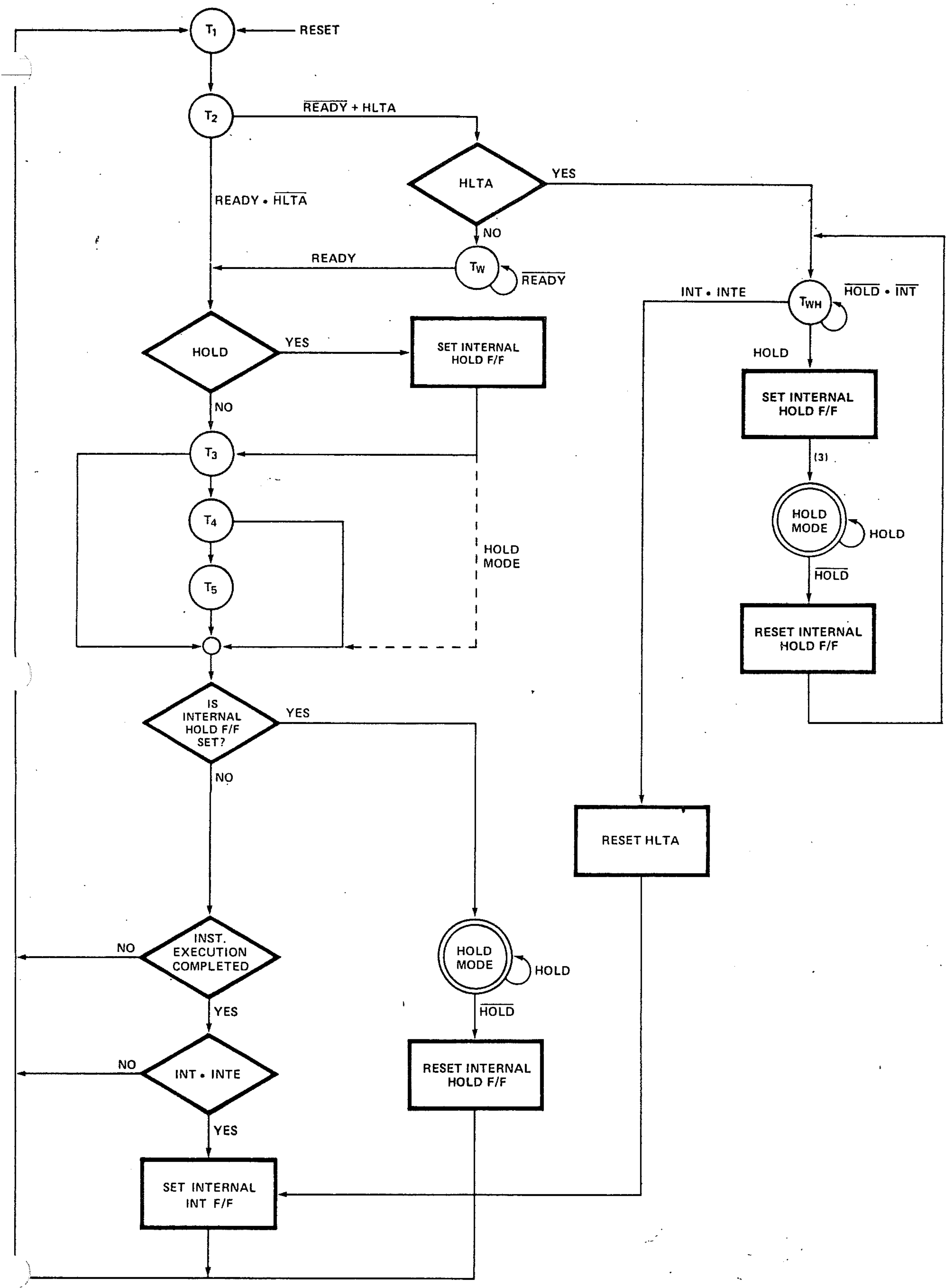
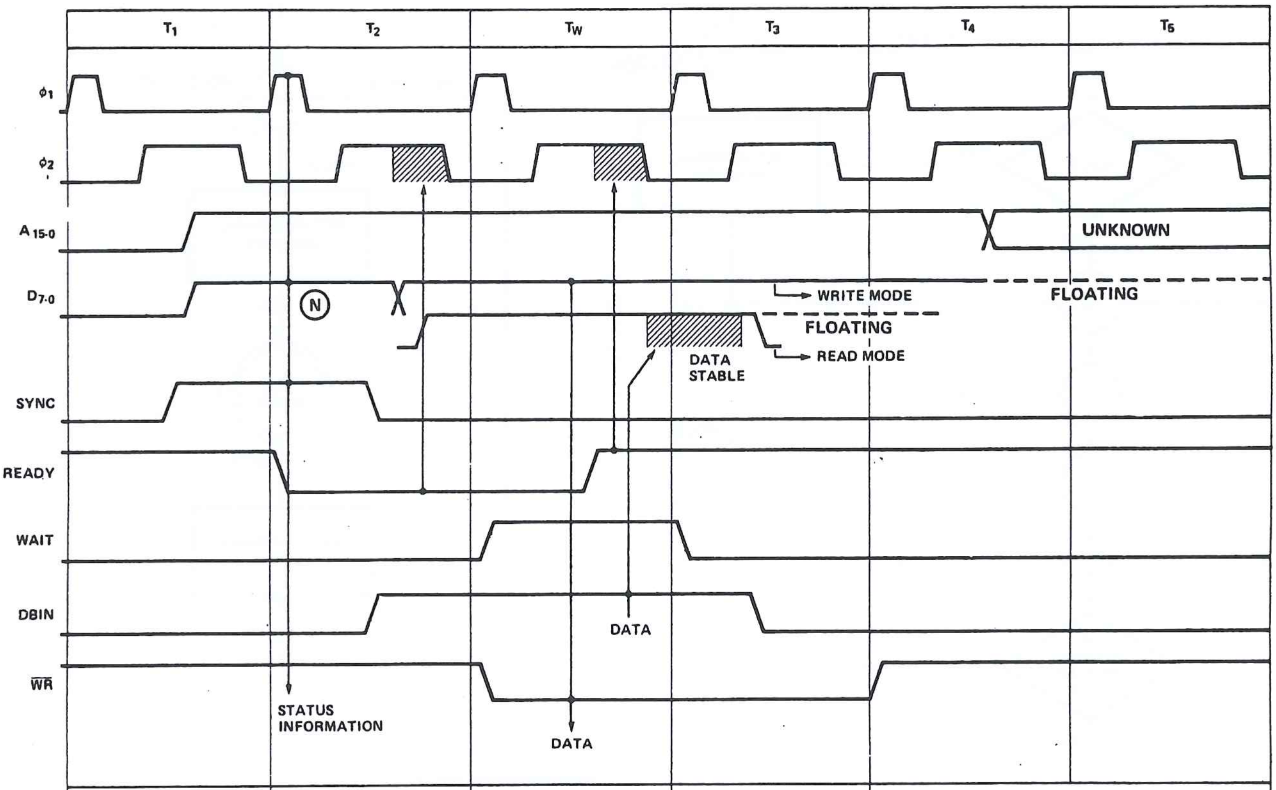

T1
RESET
T2
READY + HLTA
HLTA
YES
NO
READY • HLTA
READY
TW
READY
TWH
INT • INTE
HOLD • INT
HOLD
YES
SET INTERNAL HOLD F/F
HOLD
SET INTERNAL HOLD F/F
(3)
NO
T3
T4
T5
HOLD MODE
HOLD MODE
HOLD
HOLD
RESET INTERNAL HOLD F/F
IS INTERNAL HOLD F/F SET?
YES
NO
RESET HLTA
NO
INST. EXECUTION COMPLETED
YES
HOLD MODE
HOLD
HOLD
RESET INTERNAL HOLD F/F
NO
INT • INTE
YES
SET INTERNAL INT F/F

fig 2.10

is indicated by a WAIT signal from the processor, acknowledging the memory's request. A low-to-high transition on the WAIT line is triggered by the rising edge of the Ø1 clock and occurs within a brief delay (tDC) of the actual entry into the TW state.



$A_{15\text{-}0}$ MEMORY ADDRESS OR I/O DEVICE NUMBER $D_{7\text{-}0}$ STATUS INFORMATION INTA OUT HLTA $\overline{WO}$ MEMR M_1 INP STACK	SAMPLE READY HOLD AND HALT	OPTIONAL	FETCH DATA OR INSTRUCTION OR WRITE DATA	OPTIONAL
		HALT OR MEMORY ACCESS TIME ADJUST		INSTRUCTION EXECUTION IF REQUIRED

NOTE: Ⓝ Refer to Status Word Chart

fig. 2.11

When the 9511 is selected it needs a certain time (4xØ2) to send the asked information outside. The chip select signal off the 9511 is connected with ready on the positieve edge of the $\overline{\text{STSTB}}$ so the CPU will respond to that with some WAIT cycli (Fig. 2.8)

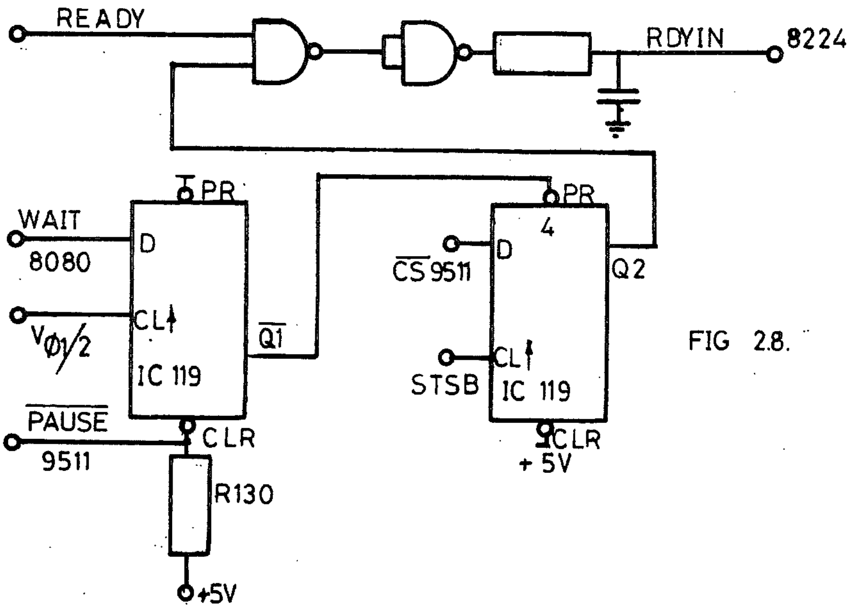
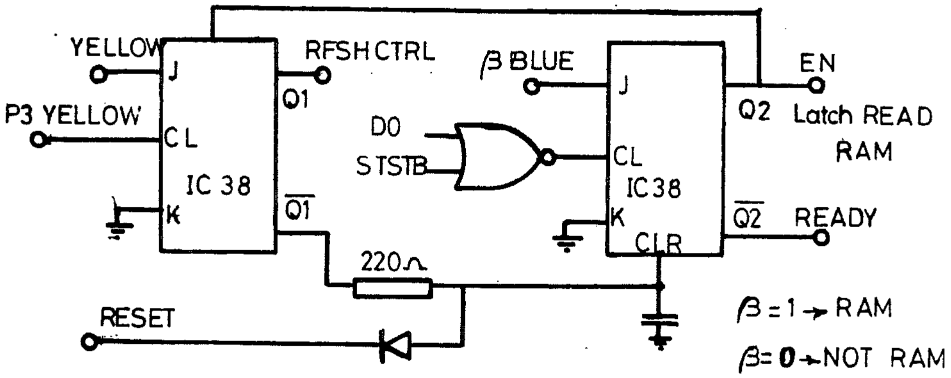


FIG 2.8.

The Pause is low so Q1=PR2 and CLR2 are high. Q2 stays low. When the Pause gets back high again then on the pos. edge of Ø1 will $\overline{\text{Q1}}$ get zero (because WAIT was high).
As result is PR2=Ø and Q2=1 and the processor can start to take the information off the 9511 inside and work further in its program.

The ready line is also used when the CPU reads the dynamic RAM memory. As clock we use the $\overline{\text{STSTB}}$ but only when there is no interrupt (Fig. 2.9)

FIG 2.9.



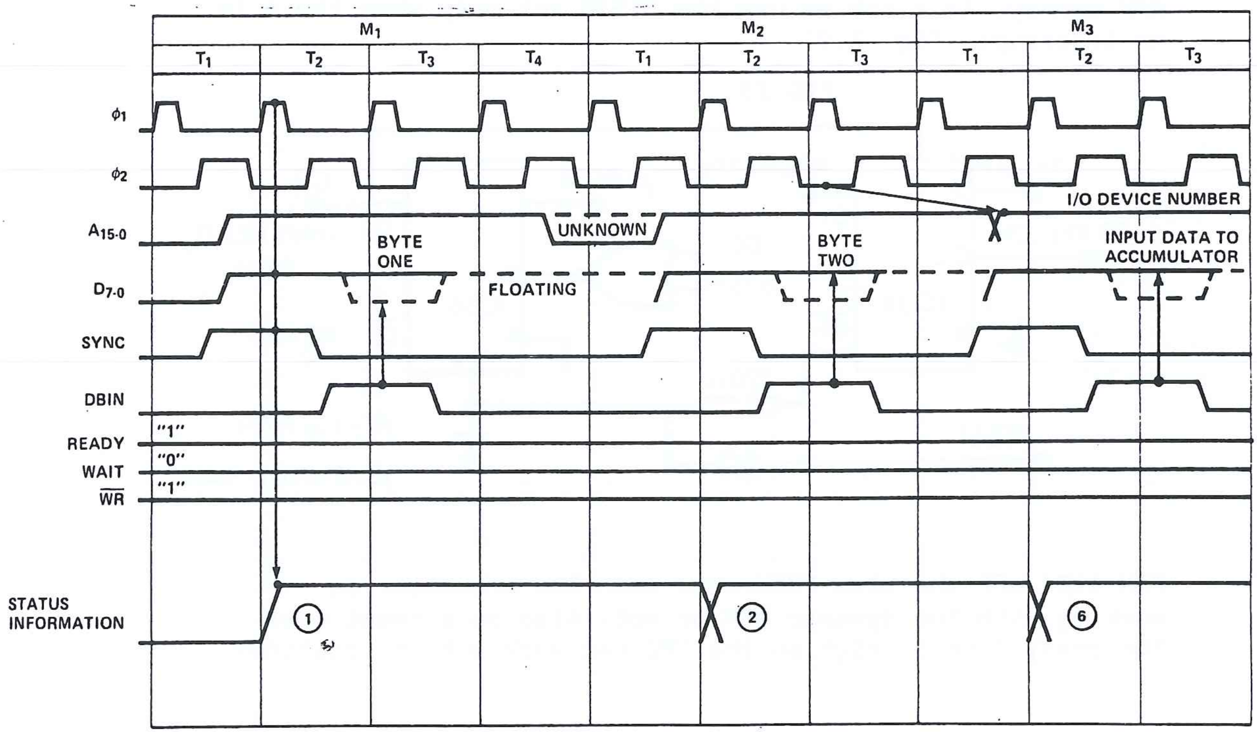
PIN 1(β) off the bleu PROM says when the processor is working with the dynamic RAM or not. Also on a reset must the ready line be high so the CPU can work out the routines for initialisation.

The refreshing of the dynamic RAM is build up with external hardware so this must also be synchronised with the ready-line of the CPU.

A wait period may be of indefinite duration. The processor remains in the waiting condition until its READY line again goes high. A READY indication **must** precede the falling edge of the Ø2 clock by a specified interval (tRS) in order to guarantee an exit from the TW state. The cycle may then proceed, beginning with the rising edge of the next Ø1 clock. A WAIT interval will therefore consist of an integral number of TW states and will always be a multiple of the clock period.

The events that take place during the T3 state are determined by the kind of machine cycle in progress. In a FETCH machine cycle, the processor interprets the data on its data bus as an instruction. During a MEMORY READ or a STACK READ, data on this bus is interpreted as a data word. The processor outputs data on this bus during a MEMORY WRITE machine cycle. Durting I/O operations, the processor may either transmit or receive data, depending on whether an OUTPUT or on INPUT operation is involved.

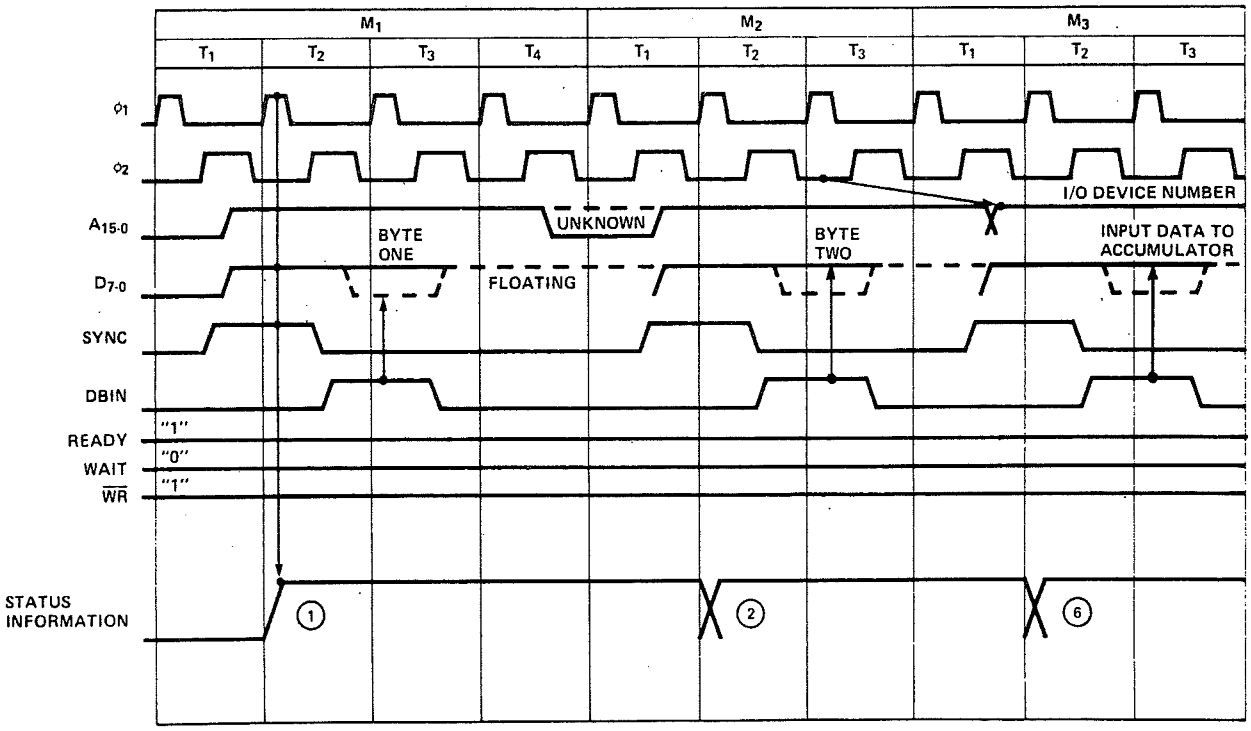
Fig. 2.12 illustrates the timing that is characteristic of a data input operation. As shown, the low-to-high transition of Ø2 during T2 clears status information from the processor's data lines, preparing these lines for the receipt of incoming data. The data presented to the processor must have stabilized prior to both the "Ø1-data set-up" interval (tDS1) that precedes the falling edge of the Ø1 pulse defining state T3, and the "Ø2-data set-up" interval (tDS2), that precedes the rising edge in state T3. This same data must remain stable during the "data hold" interval (tDH) that occurs following the rising edge of the Ø2 pulse. Data placed on these lines by memory or by other external devices will be sampled during T3.



NOTE: Ⓝ Refer to Status Word Chart

FIG. 2.12

During the input of data to the processor, the 8080 generates a DBIN signal which should be used externally to enable the transfer. Machine cycles in which DBIN is available include : FETCH, MEMORY READ, STACK READ, and INTERRUPT. DBIN is initiated by the rising edge of Ø2 during state T2 and terminated by the corresponding edge of Ø2 during T3. Any TW phases intervening between T2 and T3 will therefore extend DBIN by one or more clock periods.



NOTE: (N) Refer to Status Word Chart

Figure 2.13 shows the timing of a machine cycle in which the processor outputs data. Output data may be destined either for memory of for peripherals. The rising edge of Ø2 within state T2 clears status information from the CPU's data lines, and loads in the data which is to be output to external devices. This substitution takes place within the "data output delay" interval (tDD) following the Ø2 clock's leading edge. Data on the bus remains stable throughout the remainder of the machine cycle, until replaced by updated status information in the subsequent T1 state. Observe that a READY signal is necessary for completion of an OUTPUT machine cycle. Unless such an indication is present, the processor enters the TW state, following the T2 state. Data on the output lines remains stable in the interim, and the processing cycle will not proceed until the READY line again goes high.

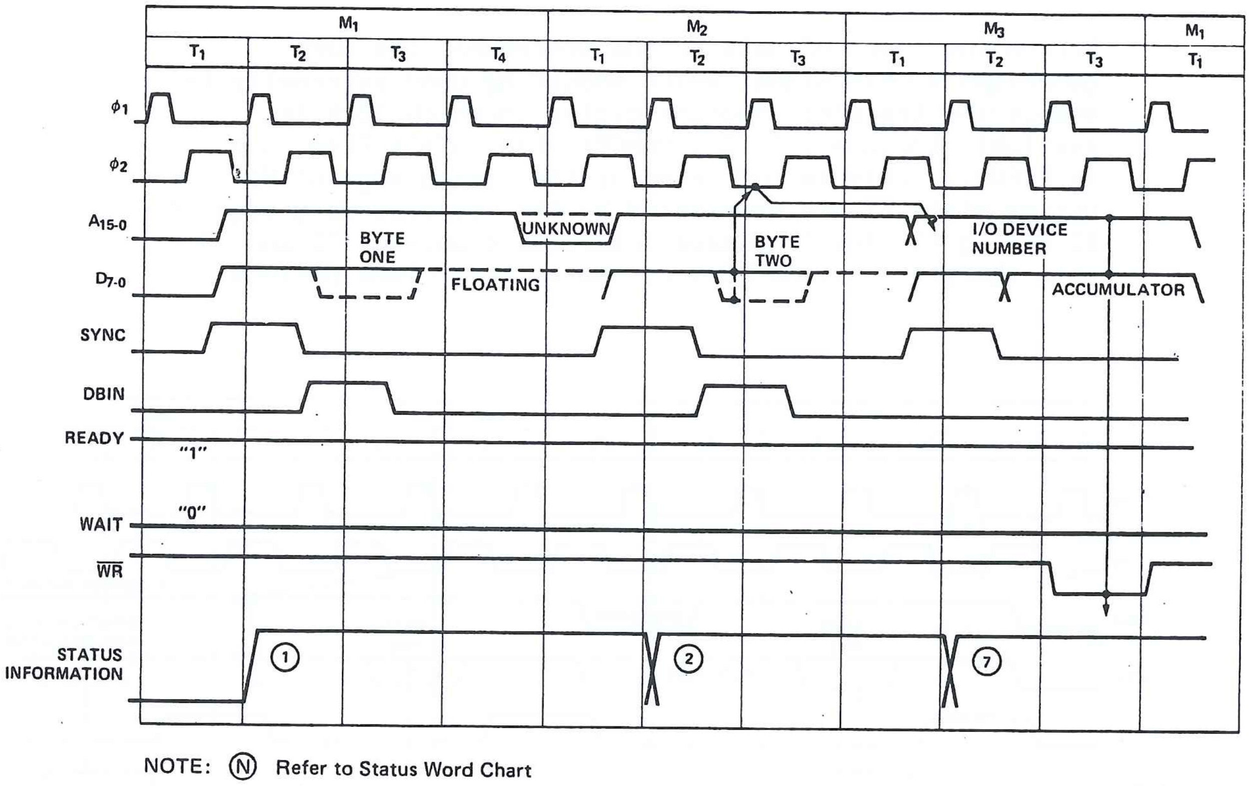


Fig. 2.13

The 8080 CPU generates a WR output for the synchronization of external transfers, during those machine cycles in which the processor outputs data. These include MEMORY WRITE, STACK WRITE, and OUTPUT. The negative-going leading edge of WR is referenced to the rising edge of the first Ø1 clock pulse following T2, and occurs within a brief delay (TDC) of that event. WR remains low until re-triggered by the leading edge of Ø1 during the state following T3. Note that any TW state intervening between T2 and T3 of the output machine cuycle will necessarily extend WR, in much the same way that DBIN is affected during data input operations.

All processor machine cycles consist of at least three states : T1, T2 and T3 as just described. If the processor has to wait for a response from the peripherals or memory with which it is communicating, then the machine cycle may also contain one or more TW states. During the three basic states, data is transferred to or from the processor.

After the T3 state, however, it becomes difficult to generalize. T4 and T5 states are available, if the execution of a particular instruction requires them. But not all machine cycles make use of these states. It depend upon the kind of instruction being executed, and on the particular machine cycle within the instruction cycle. The processor will terminate any machine cylce as soon as its processing activities are completed, rather than proceeding through the T4 and T5 states every time. Thus the 8080 may exit a machine cycle following the T3, the T4, or the T5 state and proceed directly ,to the T1 state of the next machine cycle.

STATE	ASSOCIATED ACTIVITIES
T_1	A memory address or I/O device number is placed on the Address Bus ($A_{15\text{-}0}$); status information is placed on Data Bus ($D_{7\text{-}0}$).
T_2	The CPU samples the READY and HOLD inputs and checks for halt instruction.
TW (optional)	Processor enters wait state if READY is low or if HALT instruction has been executed.
T3	An instruction byte (FETCH machine cycle), data byte (MEMORY READ, STACK READ) or interrupt instruction (INTERRUPT machine cycle) is input to the CPU from the Data Bus; or a data byte (MEMORY WRITE, STACK WRITE or OUTPUT machine cycle) is output onto the data bus.
T4 T5 (optional)	States T_4 and T_5 are available if the execution of a particular instruction requires them; if not, the CPU may skip one or both of them. T_4 and T_5 are only used for internal processor operations.

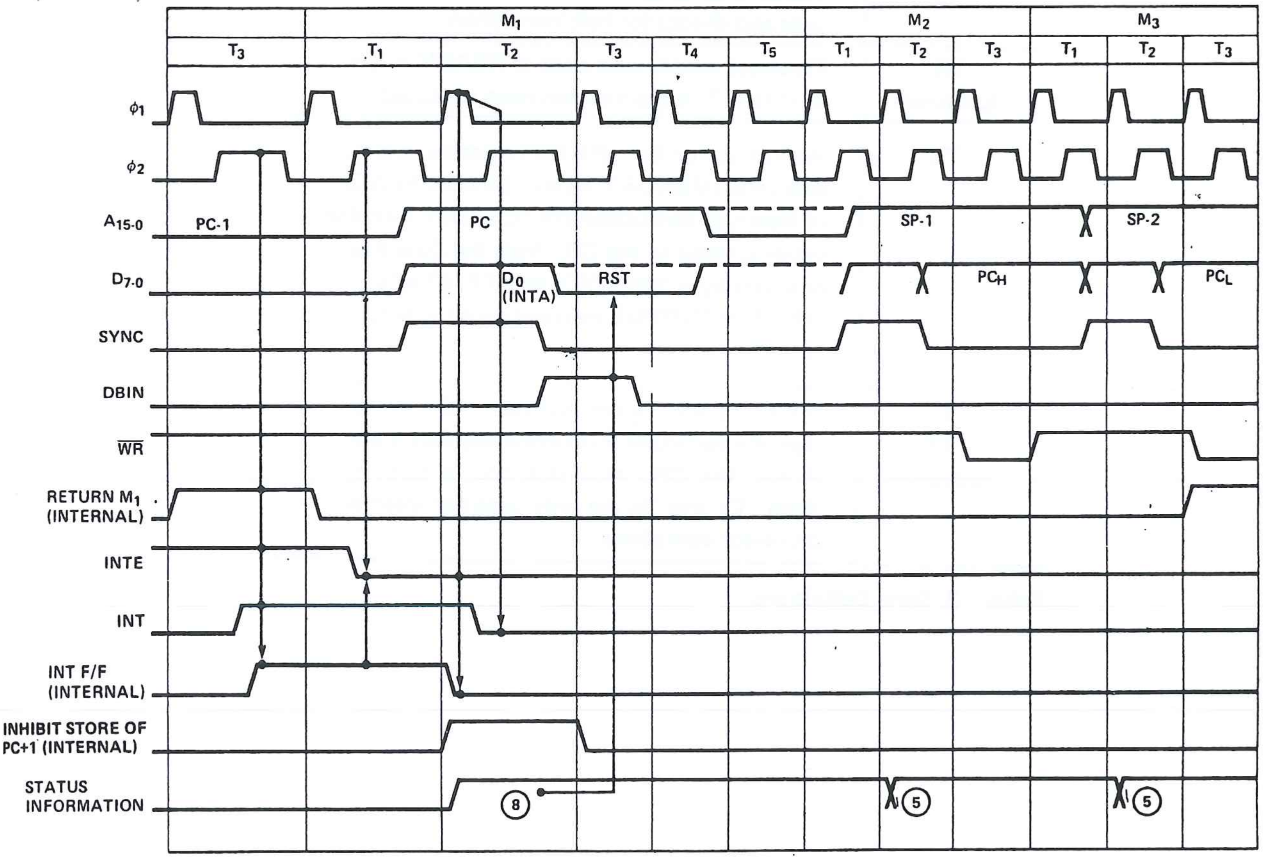
Table 2. State Definitions

INTERRUPT SEQUENCES

The 8080 has the built-in capacity to handle external interrupt requests. A peripheral device can initiate an interrupt simply by driving the processor's interrupt (INT) line high.

The interrupt (INT) input is asynchronous and a request may therefore originate at any time during any instruction cycle. Internal logic re-clocks the external request, so that a proper correspondence with the driving clock is established. As Figure 2.14 shows, an interrupt request (INT) arriving during the time that the interrupt enable line (INTE) is high, acts in coicidence with the Ø2 clock to set the internal interrupt latch. This event takes place during the last state of the instruction cycle in which the request occurs, thus ensuring that any instruction in progress is completed before the interrupt can be procecssed.

The INTERRUPT machine cycle which follows the arrival of an enabled interrupt request resembles an ordinary FETCH machine cycle in most respects. The M1 status bit is



NOTE: Ⓝ Refer to Status Word Chart

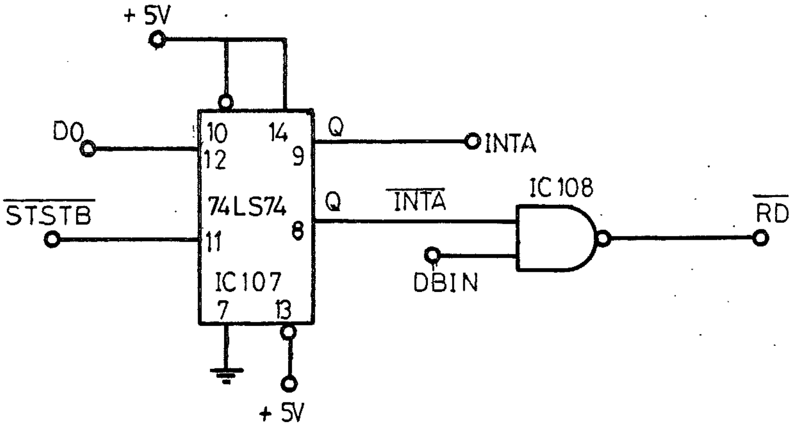
Fig. 2.14

transmitted as usual during the SYNC interval. It is accompanied, however, by an INTA status bit (Do) which acknowledges the external request.

The task of IC107 (sheet 1) is the following .

When an interrupt occurs (from the 5501) then the CPU sends a status bit out via DØ. This bit can be memorized with the status strobe of the 8224.

Fig. 2.15



Port 1 of IC108 takes care that there will be no read of the system during an interrupt acknowledge. Because the system is build up in memory map, the DBIN can be used (inverted) as the read.

The contents of the program counter is latched onto the CPU's address lines during T1 but the counter itself is not incremented during the INTERRUPT machine cycle, as it otherwise would be.

In this way, the pre-interrupt status of the program counter is preserved, so that data in the counter may be restored by the interrupted program after the interrupt request has been processed.

The interrupt cycle is otherwise indistinguishable from an ordinary FETCH machine cyle. The processor itself takes no further special action. It is the responsability of the peripheral logic to see that an eight-bit interrupt instruction is "jammed" onto the processor's data bus during state T3. In a typical system, this means that the data-in bus from memory must be temporarily disconnected from the processor's main data bus, so that the interrupting device can command the main bus without interference.

The 8080's instruction set provides a special one-byte call which facilitates the processing of interrupts (the ordinary program Call takes three bytes). This is the RESTART instruction (RST). A variable three-bit field embedded in the eight-bit field of the RST enables the interrupting device to direct a Call to one of eight fixed memory locations. The decimal addresses of these dedicated locations are : 0,8,16,24,32,40,48 and 56. Any of these addresses may be used to store the first instruction(s) of a routine designed to service the requirements of an interrupting device. Since the (RST) is a call, completion of the instruction also stores the old program counter contents on the STACK.

RESTART ROUTINES IN THE DESK COMPUTER

The 8080 microprocessor in the Desk Computer knows 8 instruction codes that are one-byte CALL instructions : RST 0 through RST 7. In many computer systems, these instructions are used in combination with interrupts.

RST 1, RST 4, RST 5

These 3 restart instructions are used for bank switching. In the memory area E000-EFFF the DAI uses 4 banks of each 4K ROM "in parallel". Via bits 6 and 7 of output port FD06, one of these banks is selected. (see later in this chapter)

Normally, bank 0 is switched on, but via software instructions one of the other banks can be activated. Therefore, the RST 1, RST 4 and RST 5 instruction codes are used. These instructions are followed by one date byte.

When the program counter encounters one of these RST instructions, it goes to the interrupt vector routines in the area 0000-003F. The interrupt vector address from the area 0062-0071 is loaded, and the program counter is set to this address.

The routines which are found on the vector addresses prepare the selection of the required ROM bank :

RST 1 : ROM bank 3 (encode - utility)
RST 4 : ROM bank 1 (math. package)
RST 5 : ROM bank 2 (screen package)

Via the general ROM bank switching routine on address C6CWF the selected ROM bank is activated.

The data byte after the RST instruction indicates which address in the particular ROM bank has to be jumped to. It is an offset to the startaddress E000.

Example : RST 5, data 18 : ROM bank 2, address E018.
There a jump to the screen mode changing routine can be found.

When switching to another ROM bank, the previous selection is saved in memory. On return from the switched bank, the old bank select is restored again.

THE OTHER RESTART INSTRUCTIONS

All other RST instructions are used on interrupt base. The interrupts are generated by the timers in the 5501 Timer and Interrupt controller (see Chapter V).

RST 7 - CLOCK INTERRUPT

The 20 ms page blanking signal for the TV is used as clock signal. Each time this clock interrupt is present, the program counter is set on 0038. Via the interrupt vector routine, the program counter is set on address D9A9.

The RST 7 routine on this address enables only stack interrupts and checks the contents of timer 01BE/F. Each time when a RST 7 interrupt is present, this counter is decremented. As long as it is not zero, nothing happens.

When this timer is zero, then on each RST 7 interrupt the clock timer 01C0 is decremented. Again, nothing happens when it is not zero.

But when the clock timer is also zero, a RST 5, data 12 routine is activated.

This routine **flashes the cursor** according the information in the pointers 0074-0077 (see memory map). After changing the contents of the screen location pointed by the cursor, the old interrupt mask is restored and the program returns from interrupt to its normal sequence.

RST 6 - KEYBOARD INTERRUPT SERVICE

Each time an interrupt from timer 4 is present, the program counter is set to D578 via the interrupt vector routine address 0030.

The RST 6 routine reloads timer 4 and enables only clock and stack interrupts.

The keyboard counter 01C1 is decremented on each RST 6 interrupt. When the result is not zero, the routine is aborted. Else, the keyboard counter is reloaded and a **keyboard scan is performed** (the GETC routine).

On exit, the original interrupt mask is restored again.

RST 3 - SOUND INTERRUPT

On an interrupt from timer 3, the interrupt vector routine on address 0018 load D755 into the program counter.

The RST 3 routine enables clock and sound interrupts only. Timer 3 is reloaded and ROM bank 1 is selected.

Now the program continues on address EE6E in bank 1, which is the Sound program. On exit, the old ROM bank and the old interrupt mask is restored again.

RST 2 - STACK INTERRUPT

When **Stack Overflow** occurs, an RST 2 interrupt is the result. Via address 0010 in the interrupt vector routine area, the program counter is loaded with D9E2.

The RST 2 routine resets the stackpointer on F900. The running of inputs and the encoding of stored lines is disabled. The input is returned to the keyboard and the timers for sound and keyboard interrupts are reloaded.

Then the error messages "STACK OVERFLOW" is printed.

RST 0 - UTILITY

The RST 0 interrupt is used only by the Utility Program. On this interrupt, the program counter is set on 0000.

The vector address required in this interrupt vector routine, is only present after a Z2 or a Z3 command in utility. Then location 0062/63 is loaded with EB5D the startaddress of the RST 0 routine in ROM bank 3.

The RST 0 interrupt is caused by timer 0; it is used in the **LOOK routine** in utility.

On a RST 0 interrupt, all CPU registers are saved in the RAM area 0053-005E. Then the program continues on a address which is given by the LOOK routine and indicates the next instruction to be performed.

The program checks this instruction. If it is a CALL or a RST instruction, then the next address is saved too.

Then a check is performed to see if the next instruction address is within the frame given by the LOOK window. When the result is positive, the contents of all registers, including stack pointer, flags and program counter, is displayed on the screen.

On exit, the timer 0 is reloaded, the interrupt mask set and - among other instructions - the CPU registers are restored again.

Because the program runs now under RST 0 interrupts, it runs much slower than in normal runtime.

HOLD SEQUENCES

The 8080A CPU contains provisions for Direct Memory Access (DMA) operations. By applying a HOLD to the appropriate control pin on the processor, an external device can cause the CPU to suspend its normal operations and relinquish control of the address and data busses. The processor responds to a request of this kind by floating its address to other devices sharing the busses. At the same time, the processor acknowledges the HOLD by placing a high on its HLDA outpin pin. During an acknowledged HOLD, the address and data busses are under control of the peripheral which originated the request, enabling it to conduct memory transfers without processor intervention.

Like the interrupt, the HOLD input is synchronized internally. A HOLD signal must be stable prior to the "HOLD SET-UP" interval (tHS) that precedes the rising edge of Ø2.

Figures 2.16 and 2.17 illustrate the timing involved in HOLD operations. Note the delay between the asynchronous HOLD REQUEST and the re-clocked HOLD. As shown in the diagram, a coincidence of the READY, the HOLD, and the Ø2 clocks sets the internal hold latch. Setting the latch enables the subsequent rising edge of the Ø1 clock pulse to trigger the HLDA output.

Acknowledgment of the HOLD REQUEST precedes slightly the actual floating of the processor's address and data lines; The processor acknowledges a HOLD at the beginning of T3 if a read or an input machine cycle is in progress (see Fig. 2. 16). Otherwise, acknowledgement is deferred until the beginning of the state following T3 (see Fig.2.17). In both cases, however,, the HLDA goes high within a specified delay (tDC) of the rising edge of the selected Ø1 clock pulse. Address and data lines are floated within a brief delay after the rising edge of the next Ø2 clock pulse. This relationship is also shown in the diagrams.

To all outward appearances, the processor has suspended its operations once the address and data busses are floated. Internally, however, certain functions may continue. If a HOLD REQUEST is acknowledged at T3, and if the processor is in the middle of a machine cycle which requires four or more states to complete, the CPU proceeds through T4 and T5 before coming to a rest. Not until the end of the machine cycle is reached will processing activities cease. Internal

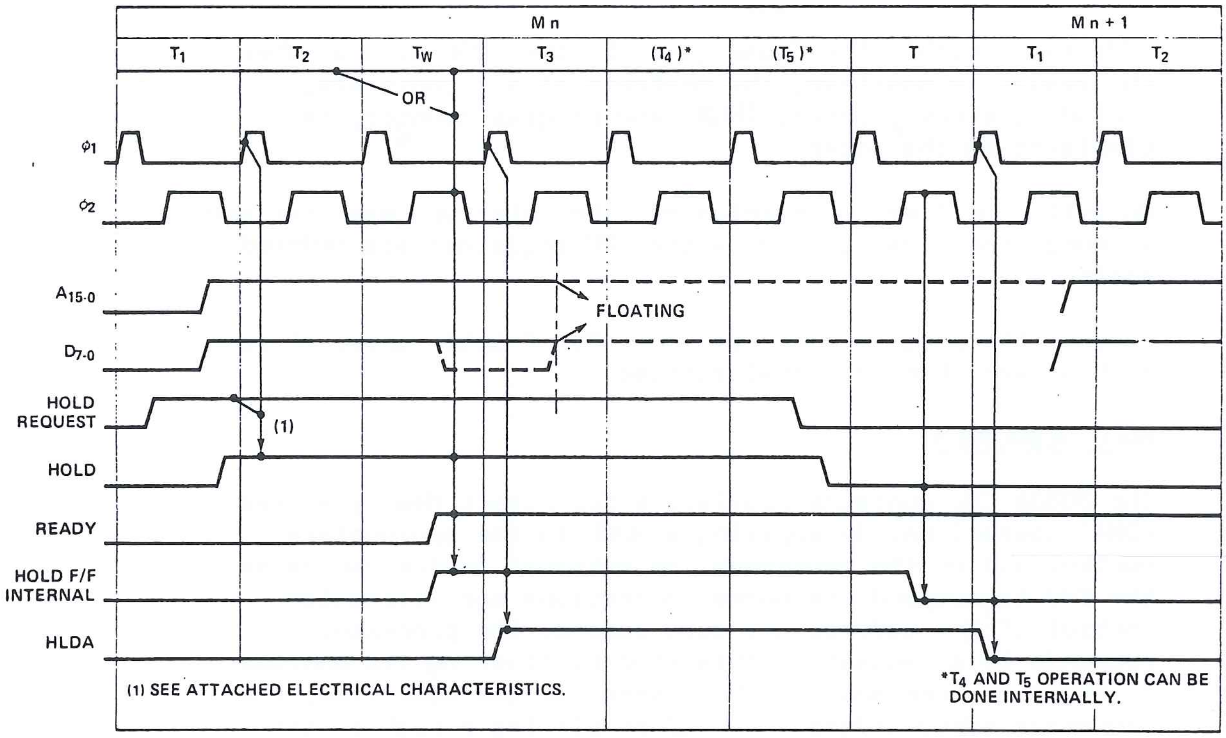


Fig. 2.16

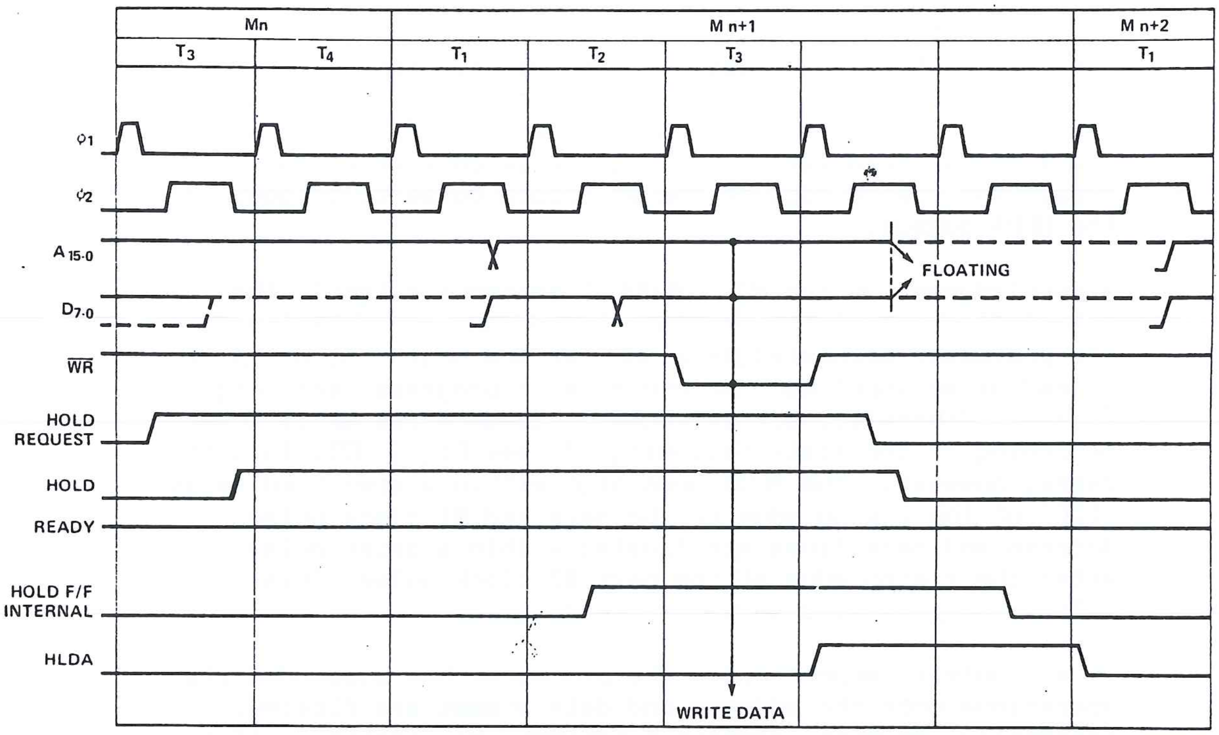


Fig. 2.17

processing is thus permitted to overlap the external DMA transfer, improving both the efficiency and the speed of the entire system.

The processor exits the holding state through a sequence similar to that by which it entered. A HOLD REQUEST is terminated asynchronously when the external device has completed its data transfer. The HLDA output returns to a low level following the leading edge of the next Ø1 clock pulse. Normal processing resumes with the machine cycle following the last cycle that was executed.

HALT SEQUENCES

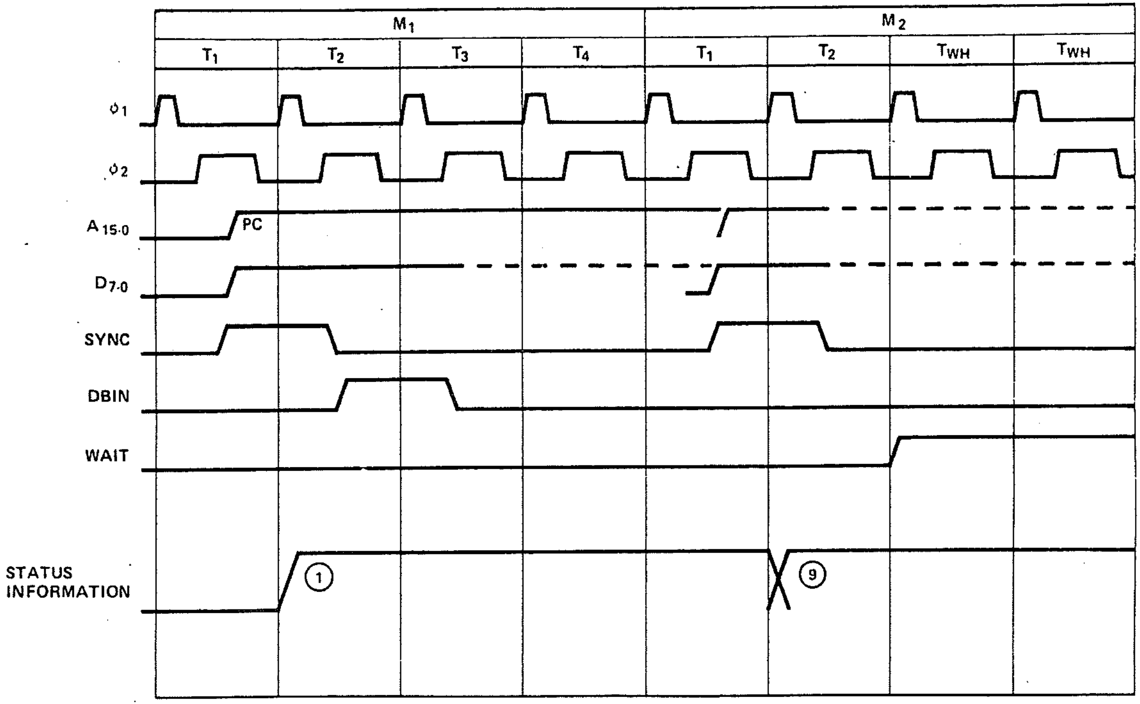
When a halt instruction (HLT) is executed, the CPU enters the halt state (TWH) after state T2 of the next machine cycle, as shown in Fig. 2.18 . There are only three ways in which the 8080 can exit the halt state :

* A high on the RESET line will always reset the 8080 to state T1; RESET also clears the program counter.

* A HOLD input will cause the 8080 to enter the hold state, as previously described. When the HOLD line goes low, the 8080 re-enters the halt state on the rising edge of the next Ø1 clock pulse.

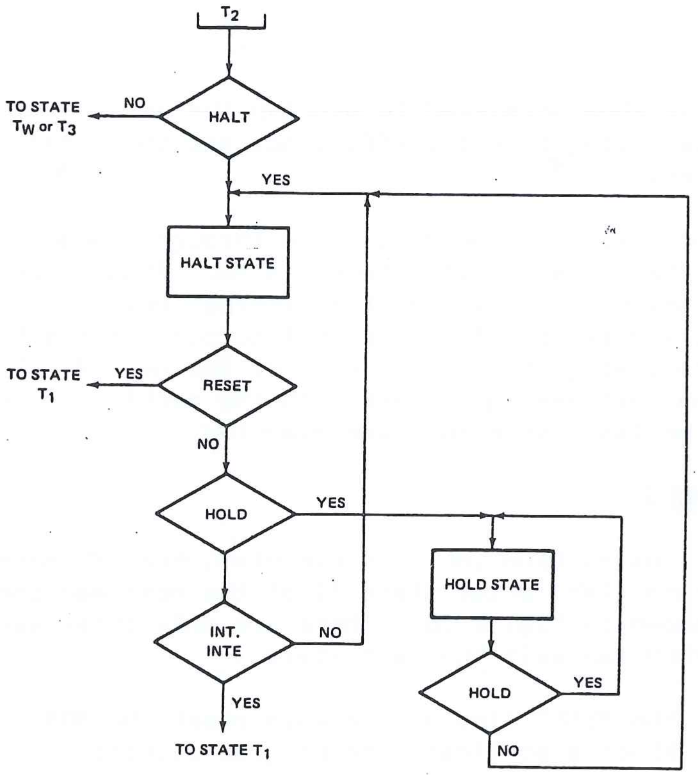
* An interrupt (i.e. INT goes high while INTE is enabled) will cause the 8080 to exit the Halt state and enter state T1 on the rising edge of the next Ø1 clock pulse. NOTE : The interrupt enable (INTE) flag **must** be set when the halt state is entered; otherwise, the 8080 will only be able to exit via a RESET signal.

Figure 2.19 illustrates halt sequencing in flow chart form



NOTE: (N) Refer to Status Word Chart

Fig. 2.18



T_2
HALT
NO
TO STATE T_W or T_3
YES
HALT STATE
RESET
YES
TO STATE T_1
NO
HOLD
YES
HOLD STATE
INT. INTE
NO
YES
TO STATE T_1
HOLD
YES
NO

Fig. 2.19

INSTRUCTION FUNCTION HEX

MOVE GROUP

INSTRUCTION	FUNCTION	reg	A	B	C	D	E	H	L	M
MOV A, reg	(A)←(reg)		7F	78	79	7A	7B	7C	7D	7E
MOV B, reg	(B)←(reg)		47	40	41	42	43	44	45	46
MOV C, reg	(C)←(reg)		4F	48	49	4A	4B	4C	4D	4E
MOV D, reg	(D)←(reg)		57	50	51	52	53	54	55	56
MOV E, reg	(E)←(reg)		5F	58	59	5A	5B	5C	5D	5E
MOV H, reg	(H)←(reg)		67	60	61	62	63	64	65	66
MOV L, reg	(L)←(reg)		6F	68	69	6A	6B	6C	6D	6E
MOV M, reg	(M)←(reg)		77	70	71	72	73	74	75	--

ACCUMULATOR GROUP

INSTRUCTION	FUNCTION		A	B	C	D	E	H	L	M
ADD reg	(A)←(A)+(reg)	*	87	80	81	82	83	84	85	86
ADC reg	(A)←(A)+(reg)+(CY)	*	8F	88	89	8A	8B	8C	8D	8E
SUB reg	(A)←(A)-(reg)	*	97	90	91	92	93	94	95	96
SBB reg	(A)←(A)-(reg)-(CY)	*	9F	98	98	9A	9B	9C	9D	9E
ANA reg	(A)←(A)∧(reg)	*	A7	A0	A1	A2	A3	A4	A5	A6
XRA reg	(A)←(A)⊻(reg)	*	AF	A8	A9	AA	AB	AC	AD	AE
ORA reg	(A)←(A)∨(reg)	*	B7	B0	B1	B2	B3	B4	B5	B6
CMP reg	(A)-(reg)	*	BF	B8	B9	BA	BB	BC	BD	BE

INCREMENT/DECREMENT REGISTER

INSTRUCTION	FUNCTION		A	B	C	D	E	H	L	M
INR reg	(reg)←(reg)+1	**	3C	04	0C	14	1C	24	2C	34
DCR reg	(reg)←(reg)-1	**	3D	05	0D	15	1D	25	2D	35

REGISTER PAIR GROUP

INSTRUCTION	FUNCTION	rp	B	D	H	SP	PSW	
INX rp	(rp)←(rp)+1		03	13	23	33	--	
DCX rp	(rp)←(rp)-1		0B	1B	2B	3B	--	
LDAX rp	(A)←((rp))		0A	1A	--	--	--	
STAX rp	((rp))←(A)		02	12	--	--	--	
DAD rp	(H,L)←(H,L)+(rp) ***		09	19	29	39	--	
PUSH rp	((SP)-1)←(rh),((SP)-2)←(rl), (SP)←(SP)-2		C5	D5	E5	--	F5	
POP rp	(rl)←((SP)),(rh)←((SP)+1), (SP)←(SP)+2		C1	D1	E1	--	F1	*

DIRECT ADDRESS GROUP

INSTRUCTION	FUNCTION	HEX
LDA addr	(A)←(addr)	3A al ah
STA addr	(addr)←(A)	32 al ah
LHLD addr	(L)←(addr),(H)←(addr+1)	2A al ah
SHLD addr	(addr)←(L),(addr+1)←(H)	22 al ah

IMMEDIATE GROUP

INSTRUCTION	FUNCTION		HEX
MVI A, data	(A)←data		3E dd
MVI B, data	(B)←data		06 dd
MVI C, data	(C)←data		0E dd
MVI D, data	(D)←data		16 dd
MVI E, data	(E)←data		1E dd
MVI H, data	(H)←data		26 dd
MVI L, data	(L)←data		2E dd
MVI M, data	(M)←data		36 dd
ADI data	(A)←(A)+data	*	C6 dd
ACI data	(A)←(A)+data+(CY)	*	CE dd
SUI data	(A)←(A)-data	*	D6 dd
SBI data	(A)←(A)-data-(CY)	*	DE dd
ANI data	(A)←(A)∧data	*	E6 dd
XRI data	(A)←(A)⊻data	*	EE dd
ORI data	(A)←(A)∨data	*	F6 dd
CPI data	(A)-data	*	FE dd
LXI B, addr	(B)←ah,(C)←al		01 al ah
LXI D, addr	(D)←ah,(E)←al		11 al ah
LXI H, addr	(H)←ah,(L)←al		21 al ah
LXI SP,addr	$(SP_h)\leftarrow ah,(SP_l)\leftarrow al$		31 al ah

INSTRUCTION FUNCTION HEX

JUMP GROUP

INSTRUCTION	FUNCTION	HEX
JMP addr	(PC)←addr	C3 al ah
JNZ addr	If Z=0, (PC)←addr	C2 al ah
JZ addr	If Z=1, (PC)←addr	CA al ah
JNC addr	If CY=0,(PC)←addr	D2 al ah
JC addr	If CY=1,(PC)←addr	DA al ah
JPO addr	If P=0, (PC)←addr	E2 al ah
JPE addr	If P=1, (PC)←addr	EA al ah
JP addr	If S=0, (PC)←addr	F2 al ah
JM addr	If S=1, (PC)←addr	FA al ah
PCHL	$(PC_h)\leftarrow(H),\ (PC_l)\leftarrow(L)$	E9

CALL GROUP

INSTRUCTION	FUNCTION	HEX
CALL addr	(TOS)←(PC),(PC)←addr	CD al ah
CNZ addr	If Z=0, (TOS)←(PC),(PC)←addr	C4 al ah
CZ addr	If Z=1, (TOS)←(PC),(PC)←addr	CC al ah
CNC addr	If CY=0,(TOS)←(PC),(PC)←addr	D4 al ah
CC addr	If CY=1,(TOS)←(PC),(PC)←addr	DC al ah
CPO addr	If P=0, (TOS)←(PC),(PC)←addr	E4 al ah
CPE addr	If P=1, (TOS)←(PC),(PC)←addr	EC al ah
CP addr	If S=0, (TOS)←(PC),(PC)←addr	F4 al ah
CM addr	If S=1, (TOS)←(PC),(PC)←addr	FC al ah

N.B. (TOS)←(PC) designates the following:-
$((SP)-1)\leftarrow(PC_h),((SP)-2)\leftarrow(PC_l),(SP)\leftarrow(SP)-2$

RETURN GROUP

INSTRUCTION	FUNCTION	HEX
RET	(PC)←(TOS)	C9
RNZ	If Z=0, (PC)←(TOS)	C0
RZ	If Z=1, (PC)←(TOS)	C8
RNC	If CY=0,(PC)←(TOS)	D0
RC	If CY=1,(PC)←(TOS)	D8
RPO	If P=0, (PC)←(TOS)	E0
RPE	If P=1, (PC)←(TOS)	E8
RP	If S=0, (PC)←(TOS)	F0
RM	If S=1, (PC)←(TOS)	F8

N.B. (PC)←(TOS) designates the following:-
$(PC_l)\leftarrow((SP)),(PC_h)\leftarrow((SP)+1),(SP)\leftarrow(SP)+2$

RESTART GROUP

INSTRUCTION	FUNCTION	HEX
RST 0	$(TOS)\leftarrow(PC),(PC)\leftarrow 0_{16}$	C7
RST 1	$(TOS)\leftarrow(PC),(PC)\leftarrow 8_{16}$	CF
RST 2	$(TOS)\leftarrow(PC),(PC)\leftarrow 10_{16}$	D7
RST 3	$(TOS)\leftarrow(PC),(PC)\leftarrow 18_{16}$	DF
RST 4	$(TOS)\leftarrow(PC),(PC)\leftarrow 20_{16}$	E7
RST 5	$(TOS)\leftarrow(PC),(PC)\leftarrow 28_{16}$	EF
RST 6	$(TOS)\leftarrow(PC),(PC)\leftarrow 30_{16}$	F7
RST 7	$(TOS)\leftarrow(PC),(PC)\leftarrow 38_{16}$	FF

ROTATE/CONTROL/SPECIAL GROUP

INSTRUCTION	FUNCTION		HEX
RLC	$(A_{n+1})\leftarrow(A_n),(A_0)\leftarrow(A_7),(CY)\leftarrow(A_7)$	***	07
RRC	$(A_n)\leftarrow(A_{n+1}),(A_7)\leftarrow(A_0),(CY)\leftarrow(A_0)$	***	0F
RAL	$(A_{n+1})\leftarrow(A_n),(A_0)\leftarrow(CY),(CY)\leftarrow(A_7)$	***	17
RAR	$(A_n)\leftarrow(A_{n+1}),(A_7)\leftarrow(CY),(CY)\leftarrow(A_0)$	***	1F
NOP	No operation		00
HLT	Processor stopped until interrupt or reset		76
DI	Interrupts disabled		F3
EI	Interrupts enabled after next instruction		FB
XTHL	(L)↔((SP)),(H)↔((SP)+1)		E3
SPHL	$(SP_h)\leftarrow(H),(SP_l)\leftarrow(L)$		F9
XCHG	(H)↔(D),(L)↔(E)		EB
DAA	Decimal adjust accumulator	*	27
CMA	$(A)\leftarrow(\overline{A})$		2F
STC	(CY)←1	***	37
CMC	$(CY)\leftarrow(\overline{CY})$	***	3F
OUT port	Not used in DCE Systems		D3 port
IN port			DB port

INSTRUCTION FUNCTION HEX

TICC GROUP

INSTRUCTION	FUNCTION	DCE 1/2	DCE-X
STXMT	(Transmit buffer)←(A)	32 06 98	32 16 FF
LDRCV	(A)←(Receive buffer)	3A 00 98	3A 10 FF
STOUT	(output port)←(A)	32 07 98	32 17 FF
LDIN	(A)←(input port)	3A 01 98	3A 11 FF
STTIM 1	(Timer 1)←(A)	32 09 98	32 19 FF
STTIM 2	(Timer 2)←(A)	32 0A 98	32 1A FF
STTIM 3	(Timer 3)←(A)	32 0B 98	32 1B FF
STTIM 4	(Timer 4)←(A)	32 0C 98	32 1C FF
STTIM 5	(Timer 5)←(A)	32 0D 98	32 1D FF
LDSTA	(A)←(TICC status reg.)	3A 03 98	3A 13 FF
STTCM	(TICC Command reg.)←(A)	32 04 98	32 14 FF
STCRR	(Rate register)←(A)	32 05 98	32 15 FF
LDIPR	(A)←(Interrupt pending reg.)	3A 02 98	3A 12 FF
STIMR	(Interrupt Mask reg.)←(A)	32 08 98	32 18 FF

GIC GROUP

INSTRUCTION	FUNCTION	HEX
GICC am,bm	(GICC Cmd reg.)←cd	3E cd + 32 03 1C or 32 03 FF
BCLR n	(P2Bn)←0	3E cc + 32 03 1C or 32 03 FF
BSET n	(P2Bn)←1	3E cs + 32 03 1C or 32 03 FF
LDGI m	(A)←(Port m)	3A 0m 1C or 3A 0m FF
STGI m	(Port m)←(A)	32 0m 1C or 32 0m FF
LDGIS 0	(A)←(Port 0)	3A 00 5C or 3A 08 FF
STGIS 0	(Port 0←(A)	32 00 5C or 32 08 FF

N.B. cd = $80_{16} + (8_{10} \times am)_{16} + bm.$

eg. for am = 3, bm = 3; cd = $80_{16} + (8_{10} \times 3)_{16} + 3$
$= 80_{16} + 18_{16} + 3 = 9B$

co = 2 x n
cs = (2 x n) + 1
m = 0, 1 or 2
(reg)/(rp) = contents of reg. or reg. pair.
((rp)) = contents of memory location whose address is held in reg. pair.
M = memory location whose address is held in reg. pair HL.
dd = 2 digit hex. data.
addr = 4 digit hex. address or data.
ah = high order address byte.
al = low order address byte.
* = All flags affected.
** = All flags except CY affected.
*** = Only CY flags affected.

For modification of flags, and execution times for each instruction, refer to section 5.5 of DCE Systems Designers Handbook.

00	NOP	
01	LXI	B,addr
02	STAX	B
03	INX	B
04	INR	B
05	DCR	B
06	MVI	B,dd
07	RLC	
08	---	
09	DAD	B
0A	LDAX	B
0B	DCX	B
0C	INR	C
0D	DCR	C
0E	MVI	C,dd
0F	RRC	
10	---	
11	LXI	D,addr
12	STAX	D
13	INX	D
14	INR	D
15	DCR	D
16	MVI	D,dd
17	RAL	
18	---	
19	DAD	D
1A	LDAX	D
1B	DCX	D
1C	INR	E
1D	DCR	E
1E	MVI	E,dd
1F	RAR	
20	---	
21	LXI	H,addr
22	SHLD	addr
23	INX	H
24	INR	H
25	DCR	H
26	MVI	H,dd
27	DAA	
28	---	
29	DAD	H
2A	LHLD	addr
2B	DCX	H
2C	INR	L
2D	DCR	L
2E	MVI	L,dd
2F	CMA	
30	---	
31	LXI	SP,addr
32	STA	addr
33	INX	SP
34	INR	M
35	DCR	M
36	MVI	M,dd
37	STC	
38	---	
39	DAD	SP
3A	LDA	addr
3B	DCX	SP
3C	INR	A
3D	DCR	A
3E	MVI	A,dd
3F	CMC	
40	MOV	B,B
41	MOV	B,C
42	MOV	B,D
43	MOV	B,E
44	MOV	B,H
45	MOV	B,L
46	MOV	B,M
47	MOV	B,A
48	MOV	C,B
49	MOV	C,C
4A	MOV	C,D
4B	MOV	C,E
4C	MOV	C,H
4D	MOV	C,L
4E	MOV	C,M
4F	MOV	C,A
50	MOV	D,B
51	MOV	D,C
52	MOV	D,D
53	MOV	D,E
54	MOV	D,H
55	MOV	D,L
56	MOV	D,M
57	MOV	D,A
58	MOV	E,B
59	MOV	E,C
5A	MOV	E,D
5B	MOV	E,E
5C	MOV	E,H
5D	MOV	E,L
5E	MOV	E,M
5F	MOV	E,A
60	MOV	H,B
61	MOV	H,C
62	MOV	H,D
63	MOV	H,E
64	MOV	H,H
65	MOV	H,L
66	MOV	H,M
67	MOV	H,A
68	MOV	L,B
69	MOV	L,C
6A	MOV	L,D
6B	MOV	L,E
6C	MOV	L,H
6D	MOV	L,L
6E	MOV	L,M
6F	MOV	L,A
70	MOV	M,B
71	MOV	M,C
72	MOV	M,D
73	MOV	M,E
74	MOV	M,H
75	MOV	M,L
76	HLT	
77	MOV	M,A
78	MOV	A,B
79	MOV	A,C
7A	MOV	A,D
7B	MOV	A,E
7C	MOV	A,H
7D	MOV	A,L
7E	MOV	A,M
7F	MOV	A,A
80	ADD	B
81	ADD	C
82	ADD	D
83	ADD	E
84	ADD	H
85	ADD	L
86	ADD	M
87	ADD	A
88	ADC	B
89	ADC	C
8A	ADC	D
8B	ADC	E
8C	ADC	H
8D	ADC	L
8E	ADC	M
8F	ADC	A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
96	SUB	M
97	SUB	A
98	SBB	B
99	SBB	C
9A	SBB	D
9B	SBB	E
9C	SBB	H
9D	SBB	L
9E	SBB	M
9F	SBB	A
A0	ANA	B
A1	ANA	C
A2	ANA	D
A3	ANA	E
A4	ANA	H
A5	ANA	L
A6	ANA	M
A7	ANA	A
A8	XRA	B
A9	XRA	C
AA	XRA	D
AB	XRA	E
AC	XRA	H
AD	XRA	L
AE	XRA	M
AF	XRA	A
B0	ORA	B
B1	ORA	C
B2	ORA	D
B3	ORA	E
B4	ORA	H
B5	ORA	L
B6	ORA	M
B7	ORA	A
B8	CMP	B
B9	CMP	C
BA	CMP	D
BB	CMP	E
BC	CMP	H
BD	CMP	L
BE	CMP	M
BF	CMP	A
C0	RNZ	
C1	POP	B
C2	JNZ	addr
C3	JMP	addr
C4	CNZ	addr
C5	PUSH	B
C6	ADI	dd
C7	RST	0
C8	RZ	
C9	RET	
CA	JZ	
CB	---	
CC	CZ	addr
CD	CALL	addr
CE	ACI	addr
CF	RST	1
D0	RNC	
D1	POP	D
D2	JNC	addr
D3	OUT	dd
D4	CNC	addr
D5	PUSH	D
D6	SUI	dd
D7	RST	2
D8	RC	
D9	---	
DA	JC	addr
DB	IN	dd
DC	CC	addr
DD	---	
DE	SBI	dd
DF	RST	3
E0	RPO	
E1	POP	H
E2	JPO	addr
E3	XTHL	
E4	CPO	addr
E5	PUSH	H
E6	ANI	dd
E7	RST	4
E8	RPE	
E9	PCHL	
EA	JPE	addr
EB	XCHG	
EC	CPE	addr
ED	---	
EE	XRI	dd
EF	RST	5
F0	RP	
F1	POP	PSW
F2	JP	addr
F3	DI	
F4	CP	addr
F5	PUSH	PSW
F6	ORI	dd
F7	RST	6
F8	RM	
F9	SPHL	
FA	JM	addr
FB	EI	
FC	CM	addr
FD	---	
FE	CPI	dd
FF	RST	7

ASCII - HEX - ASCII CONVERSION TABLE

	MSD	0	1	2	3	4	5	6	7
LSD		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENG	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	VS	/	?	O	←	o	DEL

C) MEMORY MAP HARDWARE

a) GENERAL THEORY

As in any computer based system, the 8080 CPU must be able to communicate with devices or structures that exist outside its normal memory array. Devices like keyboards, paper tape, floppy disks, printers, displays and other control structures are used to input information into the 8080 CPU and display or store the results of the computational activity.

The basic operation of the I/O structure can best be vieuwed as an array of single byte memory locations that can be Read from or WRitten into. The 8080 CPU has special instructions devoted to managing such transfers (IN, OUT). These instructions generally isolate memory and I/O arrays so that memory address space is not effected by the I/O structure and the general concept is that of a simple transfer to or from the Accumulator with an addressed "PORT". Another method of I/O architecture is to treat the I/O structure as part of the Memory array. This is generally referred to as "Memory Mapped I/O" and provides the designer with a powerful new "instruction set" devoted to I/O manipulation.

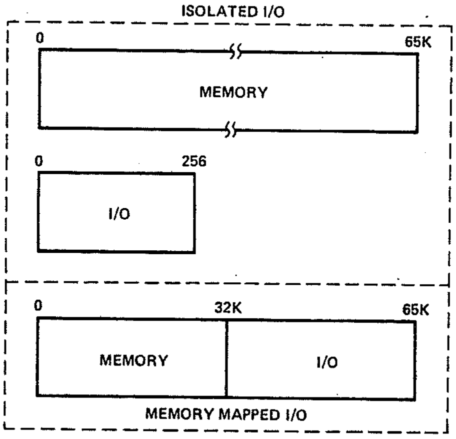


Figure 2.20 Memory I/O Mapping.

ISOLATED I/O

In Fig. 2.21 the system control signals, previously detailed in this chapter, are shown. This type of I/O architecture separates the memory address space from the I/O address space and uses a conceptually simple transfer to or from Accumulator technique. Such an architecture is easy to understand because I/O communicates only with the Accumulator using the In or OUT instructions. Also because of the isolation of memory and I/O, the full address space (65K) is uneffected by I/O addressing.

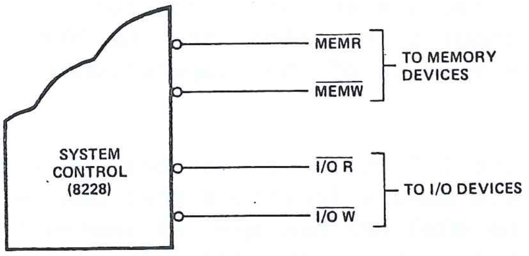


Figure 2.21 Isolated I/O

MEMORY MAPPED I/O

By assigning an area of memory address space as I/O a powerful architecture can be developed that can manipulate I/O using the same instructions that are used to manipulate memory locations. Thus, a "new" instruction set is created that is devoted to I/O handling.

I/O devices are still considered addressed "ports" but instead of the Accumulator as the only transfer medium any of the internal registers can be used. All instructions that could be used to operate on memory locations can be used in I/O.

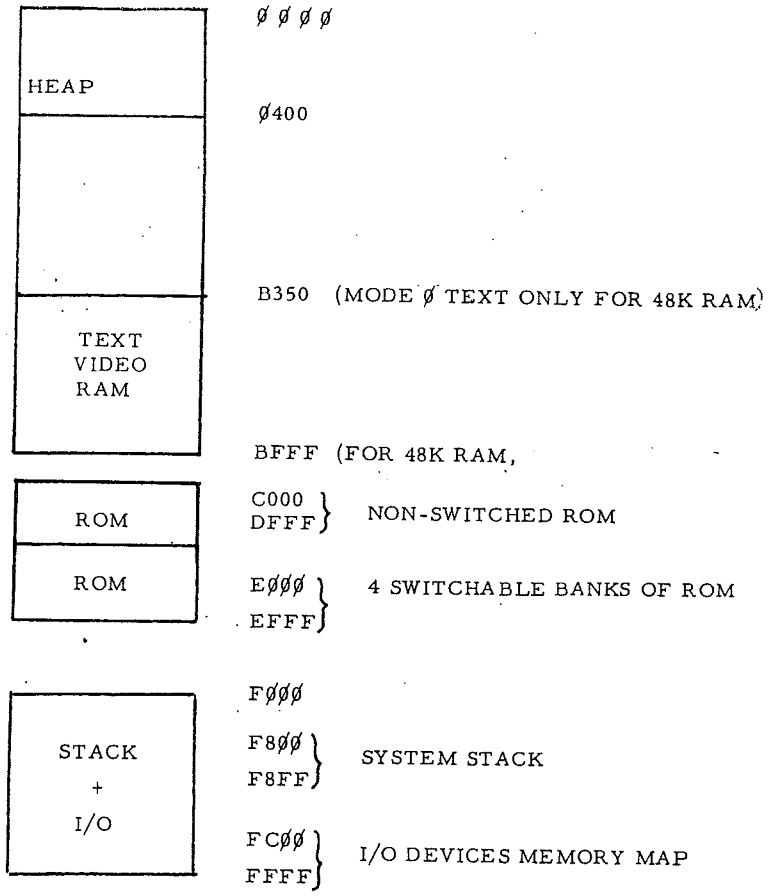
Examples :

MOVr, M	(Input Port to any Register)
MOV M, r	(Output any Register to Port)
MVI M	(Output immediate data to Port)
LDA	(Input to ACC)
STA	(Output from ACC to Port)
LHLD	(16 Bit input)
SHLD	(16 Bit Output)
ADD M	(Add Port to ACC)
ANA M	("AND" Port with ACC)

It is easy to see that from the list of possible "new" instructions that this type of I/O architecture could have a drastic effect on increased system throughput. It is conceptually more difficult to understand than Isolated I/O and it does limit memory address space, but Memory Mapped I/O can mean a significant increase in overall speed and at the same time reducing required program memory area.

b) MEMORY MAP

The bleu PROM (sheet 2) is the one that selects the biggest Parts in the computer. That is : input/output; upper or lower Prom; reading from dynamic memory : bank A/B or C; and the use of different dynamic memory chips (4027 or 4116).

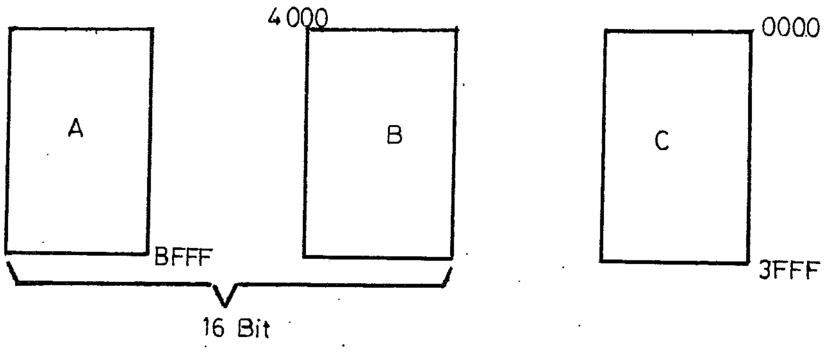


DYNAMIC RAM

RAM is build up in 3 blocks of 16K.

bank A (IC 56-100) } Picture (even and odd adresses)
bank B (IC 57-100) }

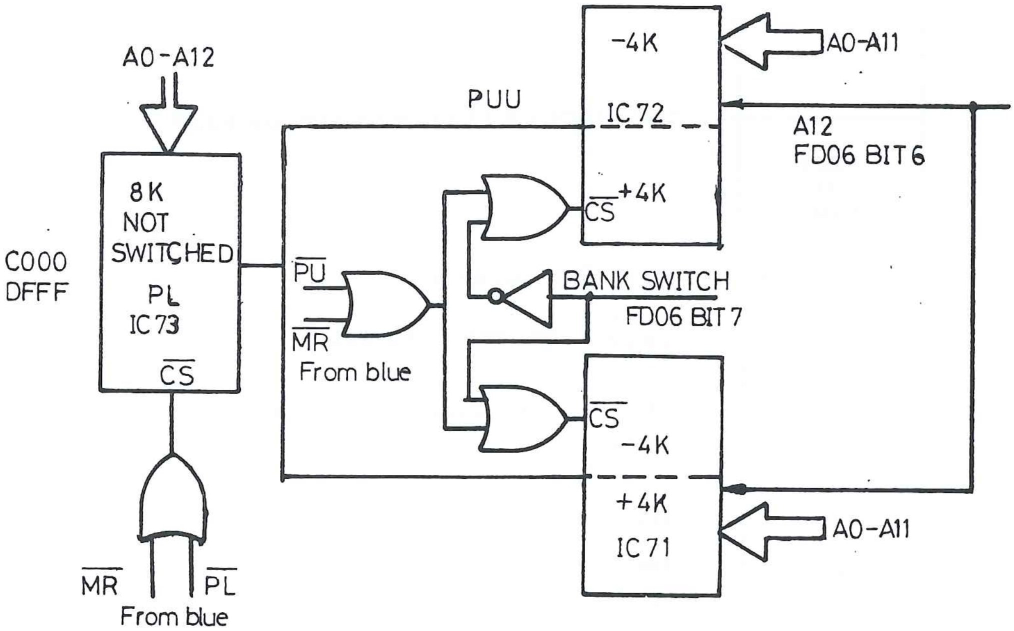
bank C (IC 58-102) Programs + heap



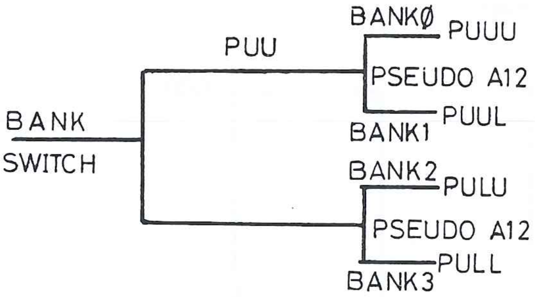
for the processor B : even address byte 4000
A : odd address bye 4001

SELECTION ROM

The firmware of the computer is stored in 24K ROM (the memory space of the ROM is 8Kx8). The address range (000-EFFF) is only 12K so bank switching is used to enable the use of 24K ROM in this address range.



On sheet 5 we find the three ROM's (IC 71,72,73). The upper PROM's (IC 71 and 72) are selected via the blue PROM (same as the lower PROM) but also with the bank switch bit (via latch address FDØ6 bit 7) so IC 72 is the upper upper ROM (PUu) and IC 71 is the upper lower ROM (PUL) but it is only one of the two ROM's which is selected. The pseudo A12 (via latch address FDØ6 bit 6) split IC71 and 72 ROM's in 2 banks of 4K each. So we use only 4K in the memory map (E000 - EFFF)



Bank 0 = 0E000-0EFFF
Bank I = 1E000-1EFFF
Bank II = 2E000-2EFFF
Bank III = 3E000-3EFFF

The switching of these banks is done under software control via the RST commands to the CPU

The complete memory map is given below

MEMORY MAP

INTERRUPT VECTOR ROUTINES: 0000 - 003F

```
0000-07          Interrupt vector routine 0:
                    Used by Utility (LOOK).
0008-0F          Interrupt vector routine 1:
                    Used by Utility and encoding Basic.
                    RST 1 + data: Switch to ROM-bank 3.
0010-17          Interrupt vector routine 2:
                    Used by stack interrupt.
0018-1F          Interrupt vector routine 3:
                    Used by sound interrupt.
0020-27          Interrupt vector routine 4:
                    Used for math. routines.
                    RST 4 + data: Switch to ROM-bank 1.
0028-2F          Interrupt vector routine 5:
                    Used for screen handling routines.
                    RST 5 + data: Switch to ROM-bank 2.
0030-37          Interrupt vector routine 6:
                    Used for keyboard service routines.
0038-3F          Interrupt vector routine 7:
                    Used to flash the cursor.

Interrupt vector    00  NOP
routines:           E5  PUSH H
                    2A  LHLD:
                    ..  ) vector address location
                    ..  ) see (#0062-#0071).
                    E9  PCHL
                    00  NOP
                    00  NOP
```

UTILITY WORK AREA: 0040 - 0061

```
0040     !POROM:  ) Memory of last outputs to output ports.
         !POR1M:  ) Duplicate of (#FD06).
          POROM:  )
0041/42   RSWK1:  Save (psw) during ROM bank switching
0043/44   RSWK2:  Save (H,L) during ROM bank switching
0045/46           Spare.
```

Used by LOOK:

```
0047                Store EI/DI instructions after using LOOK
                    the first time (No clear occurs).
0048/49             High address trace window.
004A/4B             Low address trace window.
004C                Store EI/DI when LOOK is used.
004D-4F             Store current instruction of traced
                    program.
0050                Flag for Look initialisation:
                       #FF: init. Look, else: #00.
0051/52   IADR:     I: Address current instruction.
0053      AFSAV:    A: Contents A after execution of I.
0054                F: Idem status flags.
0055      BCSAV:    B: Idem B register.
0056                C: Idem C register.
0057      DESAV:    D: Idem D register.
0058                E: Idem E register.
0059      HLSAV:    H: Idem H register.
005A                L: Idem L register.
005B/5C   SPSAV:    S: Idem stackpointer.
005D/5E   PCSAV:    P: Address next instruction to be
                       executed.
005F      TICIM:    M: Current interrupt mask.
                       Duplicate of (#FFF8)
0060                T: Value TICC control word.
                       (#FC after Z2).
0061                G: Value GIC control word.
                       (#1B after Z2).
```

INTERRUPT VECTOR ADDRESSES: 0062 - 0071

```
0062/63   I0USA:    Vector address RST 0: set by UT (Z2): 3#EB5D.
0064/65   I1USA:    Vector address RST 1: utility/encode:  #C70E.
0066/67   I2USA:    Vector address RST 2: stack interrupt: #D9E2.
0068/69   I3USA:    Vector address RST 3: sound interrupt: #D755.
006A/6B   I4USA:    Vector address RST 4: math. restart:   #C6C0.
006C/6D   I5USA:    Vector address RST 5: screen restart:  #C6FD.
006E/6F   I6USA:    Vector address RST 6: keyb. int. serv: #D578.
0070/71   I7USA:    Vector address RST 7: clock interrupt: #D9A9.
                    By changing the vector addresses, other
                    interrupt routines can be used.
```

SCREEN VARIABLES: 0072 - 00CF

Character mode variables:

```
0072/73   CURSOR:   Cursor position address.
0074      CURTY:    Cursor type:
                       #00: cursor flashes in colour.
                       #01: cursor alternates between actual
                            character and contents #0075.
0075      CURIN:    Cursor information:
                       If type = 0: Mask which is EXOR'ed with
                                    the colour byte for that
                                    character to flash it.
                       If type = 1: Cursor alterates between
                                    actual character and this
                                    information.
```

```
0076/77  CURSV:   Contents screen RAM location indicated by
                  the cursor:
                     #0076 contains the colour byte,
                     #0077 contains the data.
0078/79  LNSTR:   Address line mode byte of currently used
                  line of the screen RAM.
007A     LNEND:   Lsbyte of end of cursor line.
                  Used to check if end of line is reached.
007B     LCONT:   Number of extended lines.
007C     COLMT:   ) colours for colour    #80+X3
007D              ) registers COLORT      #90+X2
007E              )                       #A0+X1
007F              )                       #B0+X0
```

Variables set to describe the current state of the screen:

```
0080/81  SCREEN: Points to first byte of screen RAM (#BFFF).
0082/83  SCTOP:  Points after header (#BFEF).
0084/85  FFB:    First free byte in this mode.
0086/87  GRR:    Points to top of rolled area. Contains the
                 line mode byte of the line where split mode starts.
0088/89  GRE:    Points after end of graphics area.
008A/8B  CHS:    Points to start of character area.
008C/8D  GAE:    Unsplit: End archive area.
         CHE:    Split: After end of character area.
008E/8F  SCE:    End of screen (after trailer).
0090/91  GTE:    End area used splitting mode.
0092/93  GAS:    Unsplit: start archive area.
         GTS:    Split: start temporary save area.
0094/95  GRC:    Number of blobs horizontally in mode.
0096     GRL:    Number of lines of graphics in mode.
0097     GAL:    Number saved lines of graphics.
0098     GXB:    Number of bytes/line this mode.
0099/9A  GREQ:   Previous end of graphics.
009B/9C  CHSO:   Previous start characters:
                 Was split: previous mode byte of 1st text line.
                 Was graphics: Previous last COLORT-byte.
009D     SMODE:  Current screen mode (updated after mode
                 changed):
                    #00 mode 1         #08 mode 5
                    #01 mode 1A        #09 mode 5A
                    #02 mode 2         #0A mode 6
                    #03 mode 2A        #0B mode 6A
                    #04 mode 3
                    #05 mode 3A        #10 during init.
                    #06 mode 4
                    #07 mode 4A        #FF mode 0
                       bits 4-7 are ignored;
                       #0C,0D,0E,0F are inhibited.
```

Graphics mode variables (From #00A2-#00B5 also used by the EDIT mode):

```
009E     COLMG:  ) colours for colour #80+X3
009F             ) registers COLORG   #90+X2
00A0             )                    #A0+X1
00A1             )                    #B0+X0
00A2     SCVR:
00A3-AA  SCXBUF: Buffer used to hold contents of an 8 bit
                 field during 16 colour updates.
00AB     SBGOU:  Flags when colour is being carried out to
                 next field.
00AC     SBGOC:  Colour being carried out.
00AD-B4  COLS:   Buffer for impossible requests.
```

!Edit variables:

00A2/A3	!EBUFR:	Address start EDIT buffer.
00A4/A5	!EBUFN:	Address end of text in EDIT buffer.
00A6/A7	!EBUFS:	End available space in EDIT buffer.
00A8	!EWINX:	Offset of left side of window.
00A9/AA	!EWINY:	Offset of top of window from start buffer.
00AB	!ECURX:	X-offset of cursor in document (current cursor position in text line).
00AC/AD	!ECURY:	Y-offset of cursor in document (count of current cursor line).
00AE/AF	!CURPT:	Pointer to cursor position in buffer.
00B0/B1	!CURLS:	Pointer to line mode byte of cursor line on screen.
00B2/B3	!CURLB:	Pointer to line mode byte of cursor line in buffer.
00B4/B5	!TABTP:	Address tab position table.

Line drawing variables:

00B5/B6	DELTA:	Amount to add into count.
00B7/B8	RT:	Count.
00B9/BA	COR:	Adjustments for long sectors.
00BB/BC	SECT:	Lower of 2 possible sector lengths.
00BD	SECTC:	Number of sectors.
00BE	TRIM:	Amount to trim off last sector.
00BF	DIRN1:	Set if Y-direction is negative.
00C0	DIRN2:	Set if swap X,Y directions.
00C1	ANIM:	Set if animate in 4 colour mode.
00C2/C3	FCOLR:	Details of colour required.
00C4/C5	ASMKRM:	Address memory management routine (#CA01). Checks available RAM space.
00C6/C7	AESTOP:	Address emergency stop routine (#CA25). Return-routine for 'Out of space for mode'.
00C8-CF		Spare. (print spooler table pointers)

MATH. WORKING AREA: 00D0 - 00FF
================================

00D0/D1		Pointer to table with error routines (#C7F2).
00D2/D3		Pointer to input routine (#DDE0).
00D4	MVECA:	Math. chip flag: offset of start HW/SW vector; (offset for RST 4 restart routines): #00 No math. chip. #7B math. chip present.
00D5-D8	FPAC:	Arithmetic accumulator.
00D9-FF:		Used as scratch pad memory for math. package. Used in single and double byte configurations.
00DE	EXPDF:	
00DF-E2		Also used for data save during stack operations.
00E3	XN:	Length output string in #00E4-F0.
00E4		Sign output string.
00E4-F0		Used for output conversion.
00E7-EA		
00EB-EE		
00EF-F2	FTWRK:	
00F1		Digit count in output conversions.

BASIC VARIABLES: 0100 - 02EB

```
          User state:

          Following are saved by soft break: (SFRAME = SYSTOP - SYSBOT)

0100/01   SYSBOT: ) Start of current line. Points to first
          CURRNT: ) byte of line number.
0102/03   BRKPT:  Start of current command.
0104/05   LOPVAR: Points to current loop variable. Points to
                  position of variable in symbol table.
                     #00 if no running loop.
0106      LSTPF:  Flag for integer/fpt loop and
                  implicit/explicit step.
                     bit 0: 0 = implicit step.
                            1 = expicit step.
                     bit 7: 0 = FPT loop variable.
                            1 = INT loop variable.
0107-0A   LSTEP:  Step value if explicit.
010B-0E   LCOUNT: Loop iteration count.
010F-10   LOPPT:  Pointer to start address loop.
0111/12   LOPLN:  Pointer to start loop line.
0113/14   STKGOS: Stack level at last GOSUB.
                     #00 if no active call.
0115      SYSTOP: )
         (STRFL:  ) Trace/step flag together)
          TRAFL:  ) Trace flag (#FF when set).
0116      STEPF:  Step flag (#FF when set).
0117      RDIPF:  Flag set while running input (set: #FF).
0118      RUNF:   Flag set while running program.
                  (Previous 2 bytes must be consecutive)

          Runtime scratch area:

0119/1A   GSNWK:  Scratch area for GOSUB/NEXT (2 bytes).
                  Points to destination address last GOSUB.
          LISW1:  Startaddress of listed area.
          COLWK:  Scratch area for SCOLG, SCOLT (4 bytes).
                  Contains last selected COLORT/COLORG values.
011B/1C   LISW2:  End address listed area.

          Save area for restart on error:

011D/1E   ERSSP:  Stack pointer.
011F-21
0122      ERSFL:  Set if encoding a stored line (set: #01).

          Data/read variables:

0123      DATAC:  Offset of next character to encode.
0124/25   DATAP:  Pointer to current data line.
         !DATAQ:  Pointer after current data line.
0126      CONFL:  Set if there is a suspended program (set: #01).
0127/28   STACK:  Current base stack level.

          Scratch location for expression/function evaluation.

0129-2C   WORKE:  Scratch area. Contains also the argument A of
                  the last software random RND(A).

          Random number kernel:

012D-30   RNUM:   Random number kernel.
         !RNDLY:  Random number delay count (1 byte).
```

Output switching:

0131 OTSW: #00 output to screen + RS232.
#01 output to screen only.
#02 output to edit buffer.
#03 output via DOUTC.

Encoding input source switching:

0132/33 EFEPT: Encoding input pointer. Points to start-address of Basic-line just being encoded.

0134 EFECT: Encoded input count. Counts length of line.

0135 EFSW: Encoded input switching:
#00 Input from keyboard/DINC.
#01 Input from string.
#02 From edit buffer to program area.

Variables used during expression encoding (could overlap with runtime variables):

0136 TYPE: Type of latest expression or item:
#00 FPT
#01 INT
#02 STR

0137 RGTOP: Latest priority operator:

#00	no operation	#6A	IOR
#38	AND	#6C	IXOR
#39	OR	#8D	SHL
#50	>=	#8E	SHR
#51	>	#A0	+
#52	<>	#A1	-
#53	<=	#C2	/
#54	<	#C3	*
#55	=	#CF	MOD
#69	IAND	#E4	^

0138 OLDOP: Old priority operator.

0139/3A HOPPT: Pointer to place in encoded input buffer for next operator.

013B/3C RGTPT: Pointer to place in encoded input buffer of operand latest operator.

Mask to select cassette 1 or 2:

013D CASSL: #10 Cassette 1 activated.
#20 Cassette 2 activated.

Encoded input buffer:

013E-BD EBUF: 128 bytes buffer. Also used by utility.

Interrupt handler variables:

01BE/BF TIMER: Timer location. Also used in WAIT TIME.

01C0 CTIMR: Cursor clock. Used for cursor flashing.
CTIMV: #0F: Flash time in 20 ms units.
When #00, cursor flashes.

01C1 KBXCT: Extend keyboard scan time counter. When #00, keyboard scan will be performed.
KBXCK: #02: Keyboard scan time (16 ms units). Also used by RAND routine.

Sound control block storage:

```
01C2-CF:           Sound control block 0.
                      SCBL: Length of a sound block (14 bytes).
01C2      SCB0:    Duration count of present volume.
01C3/C4            Pointer to present envelope volume/duration
                   in envelope table.
01C5/C6            Pointer to start envelope table.
01C7               Sound-volume *8.
01C8               Volume, calculated from sound-volume and
                   present envelope volume.
01C9               Counter for tremolo.
01CA               Final volume, calculated from volume and
                   tremolo fluctuations.
01CB               Glissando flag:
                      #00  Endperiod reached.
                      #02  Endperiod not reached.
01CC/CD            Present period.
01CE/CF            Final period (glissando).
01D0-DB   SCB1:    Sound control block 1 (see SCB0).
01DE-EB   SCB2:    Sound control block 2 (idem).
01EC-F4   NCB:     Noise control block.
                      NCBL: Length of noise block (9 bytes).
                   The noise control block is identical to the sound
                   control block, but without period-values and
                   tremolo.
```

Envelope storage:

```
01F5-     ENVST:   Envelope storage (128 bytes).
  -0274               ENVLL:  #40: Number of bytes/envelope
                      NUMENV: #02: Number of envelopes.
                   Two envelope tables of each 64 bytes:
                      #01F5-#0234 and #0235-#0274.
0275-     IMPTAB:  Implicit type table.
  -28E                #0275  A    #027C  H    #0283  O    #028A  V
                      #0276  B    #027D  I    #0284  P    #028B  W
                      #0277  C    #027E  J    #0285  Q    #028C  X
                      #0278  D    #027F  K    #0286  R    #028D  Y
                      #0279  E    #0280  L    #0287  S    #028E  Z
                      #027A  F    #0281  M    #0288  T
                      #027B  G    #0282  N    #0289  U
028F      IMPTYP:  Default number type. Selected by IMP command.
                      #00  FPT
                      #10  INT
                      #20  STR
0290      REQTYP:  Required number type for present operation.
                      #00  FPT
                      #10  INT
                      #20  STR
                      #40  Array without arguments
```

Spare variable space:

```
0291/92   DATAQ:   Pointer to begin current data line.
0293      RNDLY:
0294      POROM:   Duplicate of (#FD04).
0295      POR1M:   Duplicate of (#FD05).
0296      INSW:    Input switching:
                      If #00, input from keyboard.
                      If <>#00, input from DINC.
0297-9A            Spare.
```

Heap/text buffer/symbol table pointers:

```
029B/9C  HEAP:    Start address of HEAP.
029D/9E  HSIZE:   Size of HEAP.
                     HSIZD: #100: Default size.
029F/A0  TXTBGN:  Start address of text buffer.
02A1/A2  TXTUSE:  End text buffer and.
         STBBGN:  Start symbol table.
02A3/A4  STBUSE:  End of symbol table.
02A5/A6  SCRBOT:  Bottom screen RAM area (48K):
                     mode 0:        #B350
                     mode 1/2(A):   #B7A0
                     mode 3/4(A):   #A65C
                     mode 5/6(A):   #63B8
```

Keyboard variables + constants:

```
02A7/A8  KBTPT:   Pointer to table with ASCII-codes.
02A9-B0  MAP1:    Latest scan of keys (key-codes).
                  (row 0 in #02A9, row 7 in #02B0)
   02AF  RPLOC:   Byte containing REPT key.
                     RPMSK: #20: Rept key bit.
                     BRSEL: #40: Column select mask for BREAK.
                     BRMSK: #40: Break key bit.
   02B0  SHLOC:   Byte containing SHIFT.
                     SHMSK: #40: Shift key bit.
02B1-B8  MAP2:    Previous scanning of keyboard.
02B9     KNSCAN:  Set to scan for BREAK only. When (#02B9)
                  is #FF: scan for BREAK only.
02BA-BD  KLIND:   4 byte circular buffer to store the ASCII
                  values for keys pressed.
                     KBLEN/KEYL: #04: length rollover buffer.
02BE/BF  KLIIN:   Next position for input to KLIND.
02C0/C1  KLIOU:   Next position for output from KLIND.
02C2     RPCNT:   Count for REPT. #01 when REPT is not
                  pressed. Else it is used as timer for the
                  repeat function.
02C3     SHLK:    Set to #FF when CTRL is pressed to
                  invert SHIFT. Else #00. Used to
                  calculate the offset for the ASCII code
                  table.
02C4     KBRFL:   Break flag. #FF indicates BREAK pressed
                  (Only if suspended program). When BREAK is pressed,
                  #02C4 counts from 00 to #0F before stopping
                  the program.
```

Data/cassette switching vectors:

Copy of ROM (#D7A4 - #D7CA) for cassette and RS232.
Can be loaded with other I/O vectors.

```
02C5-EB  IOVEC:  02C5  WOPEN:   C3 B8 D2   JMP: D2B8
                 02C8  WBLK:    C3 F1 D2   JMP: D2F1
                 02CB  WCLOSE:  C3 27 D4   JMP: D427
                 02CE  ROPEN:   C3 25 D3   JMP: D325
                 02D1  RBLK:    C3 40 D3   JMP: D340
                 02D4  RCLOSE:  C3 45 D4   JMP: D445
                 02D7  MBLK:    C3 A2 D3   JMP: D3A2
                 02DA  RESET:   C9 00 00   RET
                 02DD  DOUTC:   C9 00 00   RET
                 02E0  DINC:    C3 B4 DD   JMP: DDB4
                 02E3           C9 00 00   RET
                 02E6  TAPSL:   24 24      Tape speed leader.
                 02E8  TAPSD:   24 3C      Tape speed data.
                 02EA  TAPST:   24 18      Tape speed trailer.
```

HEAP, PROGRAM AREA, SCREEN RAM: 02EC - BFFF

```
02EC-   (RAM:      HEAP (Strings + arrays)  - See (#029B/9C).
 -BFFF  (VAREND:   Program (compiled Basic) - See (#029F/A0).
        (VARLAST:  Symbol table             - See (#02A1/A2).
                   Not used RAM             - See (#02A3/A4).
                   Screen RAM               - See (#02A5/A6).
```

ROM AND CPU AREA: C000 - F8FF

```
C000-              24K ROM:
  -EFFF               #C000-#DFFF: 8K non-switched ROM.
         VECA:        #E000-#EFFF: 4 banks of each 4K ROM.
                                   (switchable).


F000-              Can be used for ROM extension (reading only).
  -F7FF            Is already completely used by Memocom MDCR-D.


F800-              Microcomputer stack.
  -F8FF            Incl. vector for MDS jump instructions.
                      #F800   SRBOT    Bottom of stack RAM.
                      #F900   STTOP    Top of stack RAM.
```

I/O DEVICE ADDRESSES: F900 - FFFF

```
F900-              Spare I/O device addresses.
  -FAFF            (Not wired on pC board).
```

MATH. CHIP AMD 9511: FB00 - FBFF

```
FB00     MTHAD:   ) Data math.chip.
         MDATA:   )
FB02     MCOMD:   ) Command + status.
         MSTATUS:)
```

AMD9511 operator and status bytes:

ODADD:	#2C	Int addition	OFADD:	#10	Fpt addition
ODSUB:	#2D	Int subtract	OFSUB:	#11	Fpt subtract
ODMUL:	#2E	Int multiply	OFMUL:	#12	Fpt multiply
ODDIV:	#2F	Int division	OFDIV:	#13	Fpt division
OSQRT:	#01	Square root	OFIXD:	#1E	Fix
OSIN:	#02	Sine	OFLTD:	#1C	Float
OCOS:	#03	Cosine	OCHSD:	#34	Change sign int
OTAN:	#04	Tangent	OCHSF:	#15	Change sign fpt
OASIN:	#05	Arc sine	OPTOD:	#37	Push int/fpt
OACOS:	#06	Arc cosine	OPOPD:	#38	Pop int/fpt
OATAN:	#07	Arc tangent			
OLOG:	#08	Log base 10			
OLN:	#09	Log base e	MBUSY:	#80	Busy status bit
OEXP:	#0A	Expotential	MERRB:	#1E	All error bits
OPWR:	#0B	X^Y	MZERO:	#20	Top of stack

PROGRAMMABLE INTERVAL TIMER 8253: FC00 - FCFF

Used for sound generator. 3 independent 16 bits down counters with programmable counter modes.

```
FC00/01  SNDAD:   )
         SNDO:    ) Counter 0 (oscillator channel 0).
        !PDLCH:   Used as counter for paddle operations.
FC02/03  SND1:    Counter 1 (oscillator channel 1).
FC04/05  SND2:    Counter 2 (oscillator channel 2).
                  (16 bit data; LSB first)
FC06     SNDC:    Command 8253. To be loaded prior to freq.
                  selection with resp. #36, #76 and #B6.
                  Command word format:
                  bit 0    : 0   binairy counter 16 digits.
                             1   BCD counter (4 decades).
                      3,2,1: 000 mode 0: Int. on end count.
                             001 mode 1: Progr. one shot.
                             x10 mode 2: Rate generator.
                             x11 mode 3: Sq.wave rate gen.
                             100 mode 4: SW trig. strobe.
                             101 mode 5: HW trig. strobe.
                      5,4  : 00  Counter latch operation.
                             01  Read/load MSB only.
                             10  Read/load LSB only.
                             11  Read/load LSB first, then
                                 MSB.
                      7,6  : 00  Select counter 0.
                             01  Select counter 1.
                             10  Select counter 2.
                             11  Illegal.
                  Several control words:
                     COFIX: #00  Fix count on channel 0.
                     COMO:  #30  Chan.0, mode 0, 2 byte op.
                     COM1:  #32  Chan.0, mode 1, 2 byte op.
                     COM3:  #36  Chan.0, mode 3, 2 byte op.
                     C1M3:  #76  Chan.1, mode 3, 2 byte op.
                     C2M3:  #B6  Chan.2, mode 3, 2 byte op.
```

DISCRETE I/O DEVICE ADDRESSES: FD00 - FDFF

```
FD00     PORI:    IN (1)  bit 0: -
                              1: -
                              2: PIPGE: Page signal
                              3: PIDTR: Serial output ready
                              4: PIBU1: Button on paddle 1
                                        (1 = closed)
                              5: PIBU2: Button on paddle 2
                                        (1 = closed)
                              6: PIRPI: Random data
                              7: PICAI: Cassette input data

FD01     PDLST:   OUT (3) Single pulse used to trigger
                          paddle timer circuit.
```

```
FD04      PORO:    OUT (2) bit 0 - 3: volume osc. channel 0
                               4 - 7: volume osc. channel 1
FD05      POR1:    OUT (2) bit 0 - 3: volume osc. channel 2
                               4 - 7: volume random noise
                                      generator.
FD06      PORO:    OUT (3) bit 0: POCAS: Cassette data output
                             1,2: PDLMSK: Paddle select
                               3: POPNA:  Paddle enable
                               4: POCM1:  Cassette 1 motor
                                          control.(0 = run)
                               5: POCM2:  Cassette 2 motor
                                          control.(0 = run)
                             7,6:         ROM bank switching:
                                             00  bank 0
                                             01  bank 1
                                             10  bank 2
                                             11  bank 3
```

PROGR. PERIPHERAL INTERFACE 8255 : FE00 - FEFF

Used for DCE-bus (GIC Controller).

```
FE00      GIC:     (1) I/O port A
FE01               (1) I/O port B
FE02               (1) I/O port C
FE03               (6) Command word 8255:
```

Contr.	PA	PCH	PCL	PB	(mode 0)
#80	out	out	out	out	RWMOP
#81	out	out	in	out	
#82	out	out	out	in	
#83	out	out	in	in	
#88	out	in	out	out	
#89	out	in	in	out	
#8A	out	in	out	in	
#8B	out	in	in	in	
#90	in	out	out	out	RWMIP
#91	in	out	in	out	
#92	in	out	out	in	
#93	in	out	in	in	
#98	in	in	out	out	
#99	in	in	in	out	
#9A	in	in	out	in	
#9B	in	in	in	in	

TICC: TIMER + INTERRUPT CONTROLLER 5501: FF00-FFFF

```
FFF0      (4) Serial input buffer. Contains the last
              character received on the RS232 interface.
FFF1      (4) Keyboard input port. Bottom 7 bits are data
              input from the keyboard. Bit 7 is the IN7
              line from the DCE-bus and is attached to
              the page-blanking signal for the TV. Every
              20 ms. an impulse is present.
FFF2      (5) Interrupt address register:
              bits 5,4,3: Number of pending
                          interrupt.
                   7,6  : )
                   2,1,0: ) always '1'
FFF3      (4) Status register:
              bit 0: Frame error. Set by a BREAK on the
                     RS232 input.
                  1: Overrun error. Set if a character
                     has been received but not taken by
                     the CPU.
                  2: Serial input. Set if no data is
                     received.
                  3: Receive buffer loaded. Set if a
                     character has been received.
                  4: Transmit buffer empty. Set if RS232
                     output is ready to accept another
                     character.
                  5: Interrupt pending. Set if one or more
                     of the enabled interrupts has
                     occured.
                  6: Full bit detected. Set if the first
                     data bit of an incoming character
                     has been detected.
                  7: Start bit detected. Set if the start
                     bit of an incoming character has been
                     detected.
FFF4      (5) Command register:
              bit 0: TICC reset.
                  1: Send Break. If set, the serial output
                     is high impedance.
                  2: Interrupt 7 select. A '1' selects IN7
                     of the DCE-bus, a '0' selects Timer 5.
                  3: Interrupt acknowledge enable.
                     A '1' enables TICC to accept a INTA
                     signal from the CPU.
                  4 - 7: Always 0.
FFF5      (6) Communications rate register:
              bit 0:  110 baud
                  1:  150 baud
                  2:  300 baud
                  3: 1200 baud
                  4: 2400 baud
                  5: 4800 baud
                  6: 9600 baud
                  7: 1 - one stop bit
                     0 - two stop bits
```

FFF6	(6)	Serial output buffer. Write byte to this location to send it on the RS232 output. Use only when #FFF3-bit 4 is high.
FFF7	(7)	Keyboard output port. Data output to scan keyboard.
FFF8	(5)	Interrupt mask register: bit 0: timer 1 has expired (UTIM). 1: timer 2 has expired. 2: External interrupt (STKIM). 3: Timer 3 has expired (SNDIM). 4: Serial receiver loaded. 5: Serial transmitter empty. 6: Timer 4 has expired (KBIM). 7: Timer 5 has expired or IN7 (CLKIM). (react only on low-high transition)
FFF9	(5)	UTIAD: Timer 1 address (UT).
FFFA	(5)	Timer 2 address.
FFFB	(5)	SNDIAD: Timer 3 address (sound).
FFFC	(5)	KBIAD: Timer 4 address (keyboard).
FFFD	(5)	Timer 5 address.
FFFE		not used.
FFFF		not used.

NOTES:
(1) Read and write allowed.
(2) Reading allowed. Writing too, but may be overwritten by BASIC program.
(3) No writing allowed.
(4) Reading allowed, writing not.
(5) Should not be accessed.
(6) Writing allowed, reading not.
(7) Reading not allowed, writing is harmless but useless; keyboard scanner will overwrite it.

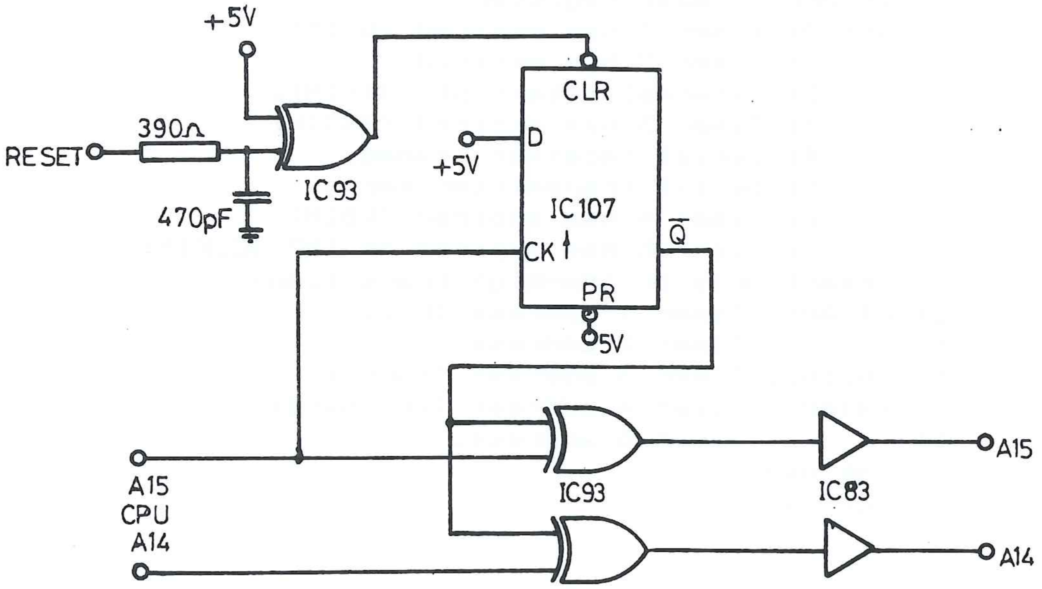
REMARKS:

ADDRESSES FB00 - FFFF:

The 2 highest bytes of the address are used for the chip select signal CS of the peripheral equipment 8253, 8255, 5501 etc. The lowest byte is used to address the several registers of the peripheral. The 2nd LSB does not have any value. So addresses in this range can be read as FBx0 - FFxF, in which x is a don't care.

STARTING ON ADDRESS CØØØ

When resetting the computer, the program counter in the CPU starts from address ØØØØ. We can change the start address outside the processor by putting some external hardware on the highest address lines.



When resetting is the RESET pin 1⇒CLR IC107 is Ø⇒Q̄ is 1 and A14 and A15 are zero. So A14ʹ and A15ʹ are high and the system starts from address CØØØ.

	A14 and A15		A14ʹ and A15ʹ	
after RESET	0	0	1	1
	0	1	0	1
	1	0	1	0
	1	1	1	1

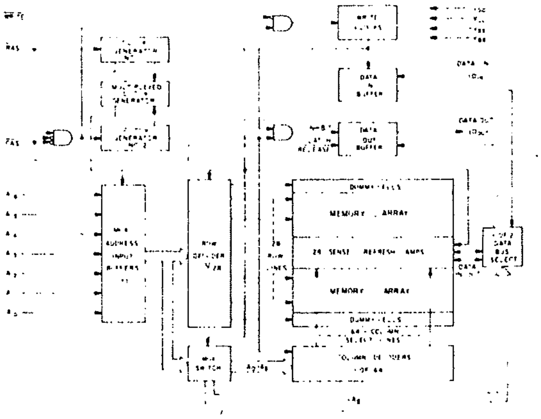
CHAPTER III   DYNAMIC RAM MEMORY

The concept of dynamic memory storage can be seen as follow

You can store a digital "0" or "1" by a low or high voltage storing on a capacitor in a 3-transistor-cell. However, this can cause a problem since the charge will eventually leak of any capacitor. If data is to be retained for longer than the self discharge time of a cell storage capacitor, typically two milliseconds, the data must be sensed before it is lost and then restored to its original voltage level. The operation of restoring the cell voltages to good levels is called a refresh operation. This simultaneously occurs in all cells of the externally addressed row of the memory matrix. To refresh the entire memory array, it is necessary to perform a refresh cycle to each of the 128 rows of the memory array at least once every two milliseconds.

MK 4116 FUNCTIONAL DIAGRAM

Figure 1.



A memory chip is physically arranged as a two dimentional array of cells. The address inputs are used for row and for column selection so their must be a multiplexing of the addresses (multiplexers are IC66;67;68;69 : 4 to 1 multiplexers). The multiplex part requires two timing signals. The first signal RAS initiates a cycle and strobes in the address and the second signal CAS strobes in the column address.

Although address multiplexing provides some very substantial system benefits, it complicated system timing. It requires that both row and column addresses get into the chip in a short time using the same address pins. This establishes a rather tight timing window during which the individual events must occur. The sequence of events required to address the chip is as follows :

1) establish row adresses
2) bring $\overline{RAS}$ low
3) maintain row addresses valid for some minimum hold time
4) establish column adresses

5) bring $\overline{\text{CAS}}$ low
6) hold column addresses valid for some minimum time.

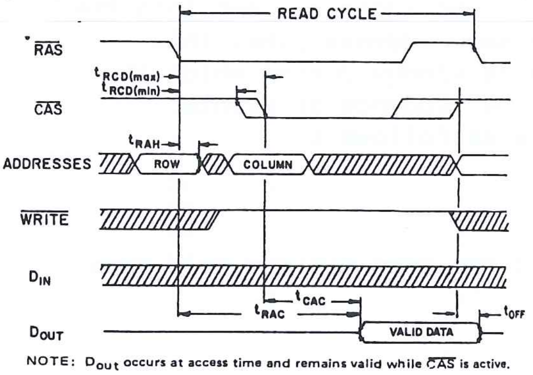
To achieve specified access time from $\overline{\text{RAS}}$, it is necessary to bring $\overline{\text{CAS}}$ low within some specified maximum delay after RAS.

The 14 addresses of a 16K memory are strobed into the memory chip in two groups of 7. When an address becomes available for a memory operation, the row address must first be presented to the chip address pins. As soon as the row address inputs are valid, the first of two timing signals to the chip initiates a cycle. This signal strobes or latches the row address into the chip and is appropriately called **ROW ADDRESS STROBE** or **$\overline{\text{RAS}}$**. With no further commands to the chip, the latched addresses are converted to MOS voltage levels, decoded, and the selected row is enabled. Data is thereby destructively read from each cell in the selected row by dumping its charge onto its respective column sense line. A sense aplifier for,each column detects the change in voltage level on the column line resulting from this deposited charge, and amplifies this signal. The amplified signals from the sense aplifiers are then impressed back onto the column sense lines, returning the cells to their original voltages. A cell whose voltage had decayed is restored to its original voltage in the process. At this time the sense amplifiers contain the same data or information contained in the selected row, and the destructively-read cells in the row are restored (refreshed) to their proper voltage.

The **Column Address Strobe ($\overline{\text{CAS}}$)** on the other hand, controls column selection circuitry and the transfer of data from the selected sense amplifier to the output circuitry. After $\overline{\text{RAS}}$ strobes the row address information from the multiplexed address input pins, $\overline{\text{CAS}}$ strobes the column address from the same pins. When $\overline{\text{CAS}}$ goes active (low) the column address is strobed or latched into the circuit. This addess is then decoded to select the proper column. Data from the selected sense amplifier is then transferred to the output buffer, completing read access.

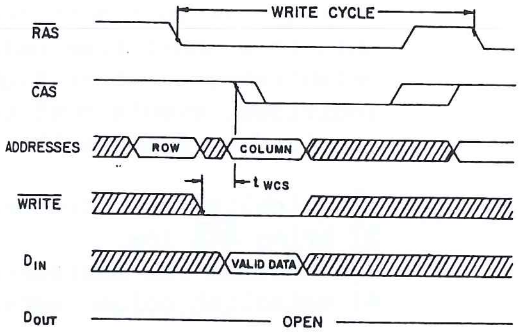
During a write operation, the same sequence of events occurs as in a read cycle, with identically the same timing as in a read cycle except that the write enable signal, $\overline{\text{WRITE}}$, is brought active (low). This causes the data at the data input to be strobed into the chip, buffered, and written into the selected sense amplifier and, thereby, into the selected cell.

READ CYCLE



NOTE: D_{out} occurs at access time and remains valid while $\overline{\text{CAS}}$ is active.

WRITE CYCLE



NOTE: If, WRITE command occurs before $\overline{\text{CAS}}$, then D_{out} remains high impedance.

Refresh of the dynamic cell matrix is accomplished by performing a memory cycle at each of the 128 row addresses within each 2 millisecond time interval. Although any normal memory cycle will perform the refresh operation, this function is most easily accomplished with "$\overline{\text{RAS}}$-only" cycles.

If refresh takes place after a read cycle it may be required that the read data be held while refresh takes place. The 4116 requires that $\overline{\text{CAS}}$ is held low to maintain the output data which means that no cycle may start while the data is being held. The only way to accomplish this is by adding data latches (IC49;50;51).

In the memory systems it is difficult to guarantee that hte normal order of events will cause all the rows within a memory to be accessed within the specified refresh interval. For this reason, the memory system has special circuitry that will cause extra memeory cycles in an ordered manner such that all rows of memory devices are accessed within the 2 ms interval.

The refresh cycles are periodically generated for the refresh but they are introduced at a time when the memory is not being accessed (when RAM ENABLE is high no refresh control out of IC112 pin 13 : sheet 2) so the CPU is not affected by refresh.

Every time the RAM has been accessed (by the 8080 or the timing circuit) the refresh counter (IC 104 sheet 3) will add 1 to its address count and comes available when $\overline{\text{RFSH}}$ is low (RAM ENABLE is the opposite of $\overline{\text{RFSH}}$: see sheet 2 IC 43). When S8 is active (low) the CPU has the possibility to access RAM because the read and write signals of the 3 RAM banks are programmed in the green PROM (IC44 sheet 2).

Pin 14 : $\overline{\text{RD}}$

Pin 12 : $\overline{\text{WR}}$ (when S8 is low)

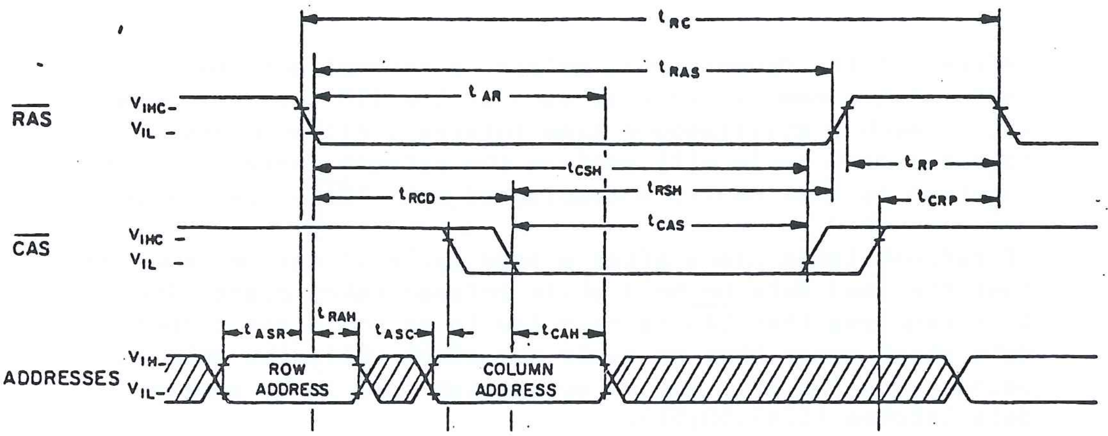
Pin 2-5 : $\overline{\text{RD}}$ & $\overline{\text{WR}}$ of the RAM

Pin 10 :)
) Bank A/B or C
Pin 11 :)

Pin 13 : AØ is used to work with 4027 through Pin 1

When reading the RAM the ready line goes low. For this time the EN (bottom left sheet 2) will be high to enable the output latches (IC49; 50; 51) during CPU READ. The read pulses (actif low) are connected to the output control of these latches. When the ready line goes high again there will be a clear for IC38 (sheet 2; FF1) and the refresh control is enabled again.

The timing of $\overline{RAS}$ and $\overline{CAS}$ signals is controlled with IC 111; 112; 113 (sheet 2)



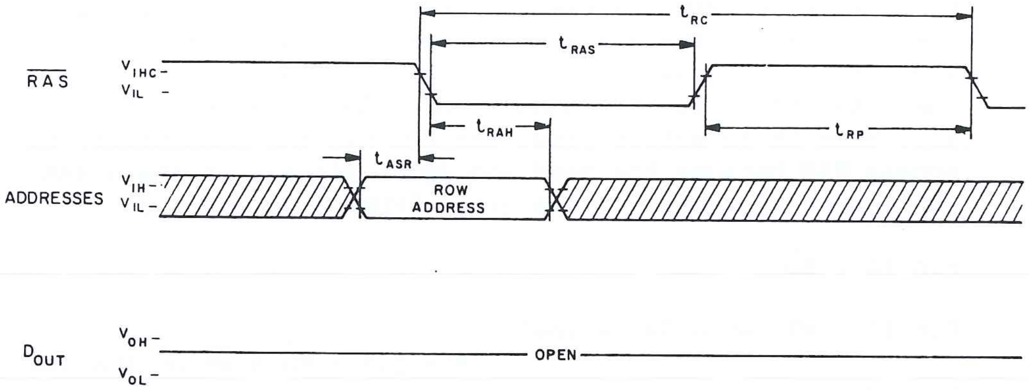
tRAS : RAS pulse width : 250 ns (IC 111 : 74123)
tCAS : CAS pulse width : 200 ns
tRCD : 90 ns

The CAS signal for the RAM's is a delayed pulse from the column address switching signal (via IC112).

IC113 is used for the RAS only refresh. During the refresh we need only a RAS signal because RAM is refreshed ROW by ROW and not bit by bit.

"$\overline{RAS}$–ONLY" REFRESH CYCLE

NOTE: $\overline{CAS}$ = V_{IHC}, $\overline{WRITE}$ = Don't Care



As result we can say that 3 circuits need the dynamic RAM

1) The CPU for reading and writing of data
2) The refresh circuit with a "RAS only" cycle
3) The timing circuit for reading data out of RAM to build up the picture.

The maestro which is conducting this Timing for CPU; refresh and picture access, is the Yellow Prom.

The layout (grid system) and the decoupling of the dynamic RAM is done with much care to keep the larg current transients and the noise (due to high switching speed) as low as possible.

The resistors (47Ω) in the $\overline{\text{CAS}}$, $\overline{\text{RAS}}$ and data lines are used as a prevention for negative spikes on the inputs of the 4116. Negative spikes more than 1V could mean the end of some dynamic RAM chips.

CHAPTER IV

VIDEO TIMING

1) PROGRAMMABLE GRAPHICS GENERATOR

The programmable video graphics + character system makes use of a scheme of variable length data to give efficient use of memory when creating pictures. A few definitions are necessary before further examination of the scheme.

A "SCAN" is :

One traverse of the screen by the electron beam drawing the picture.(there are 625 in a European television picture).

A "LINE" is :

A number of scans all of which are controlled by the same information in the RAM.

A "MODE" is:

One of the different ways information may be displayed on the screen. For instance, in "character mode" bytes in memory are shown as characters on the screen, in "4 colour graphics" mode, bytes describe the colour of blobs on the screen.

A "BLOB" is :

The smallest area on the screen whose color can be set (The Physical size of a blob is different in different screen modes).

A "FIELD" is :

A set of 8 blobs whose colour is controlled by a pair of bytes from memory.

The picture is defined by a number of lines, one after another down the screen. Each line is independent of all others and may be in any of the possible modes.

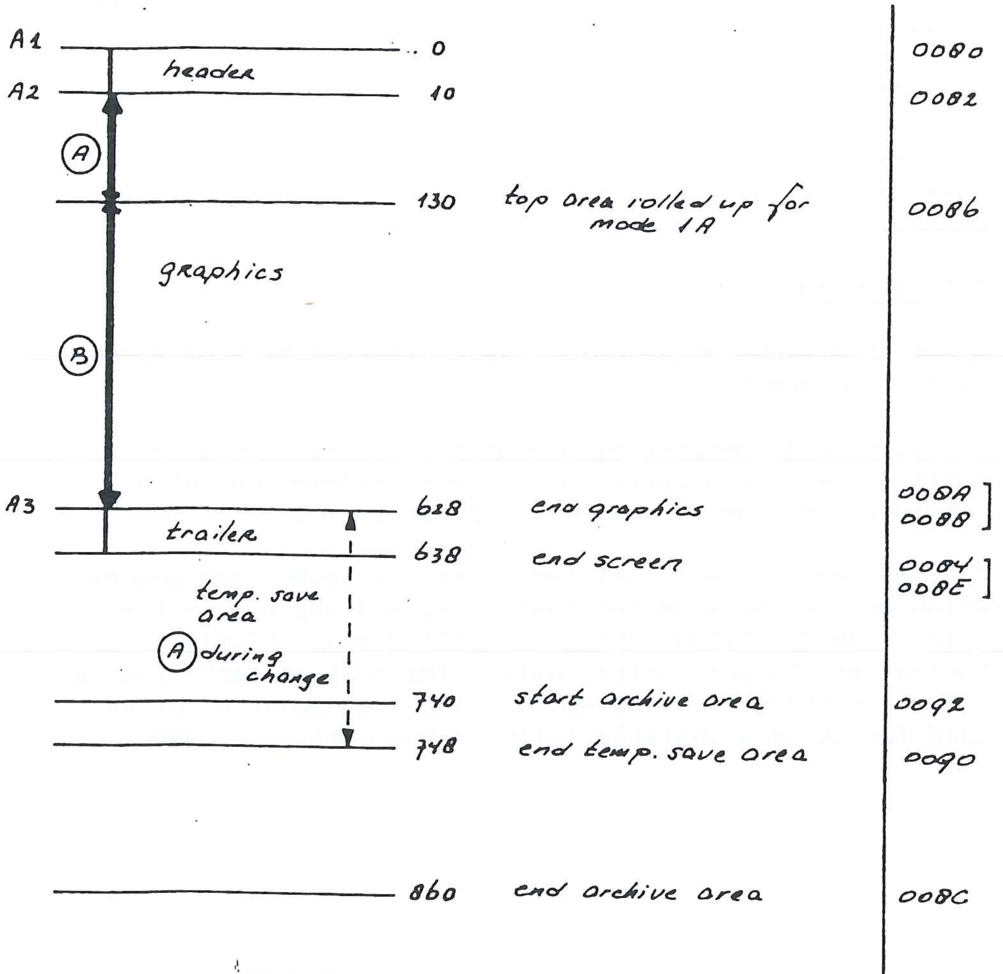
At the start of each line two bytes are taken from memory which define the mode for that line, and may update the colour RAM two bytes. These are called respectively the Control and Colour Control bytes. The rest of each line is colour or character information, and the number of bytes used for it is a characteristic of the particular mode.

Mode	Graphics size	Text size	Colours
0	-	24X60	any 2 of 16
1	72 X 65	-	16
1A	72 X 65	4 X 60	16
2	72 X 65	-	any 4 of 16
2A	72 X 65	4 X 60	any 4 of 16
3	160 X 130	-	16
3A	160 X 130	4 X 60	16
4	160 X 130	-	any 4 of 16
4A	160 X 130	4 X 60	any 4 of 16
5	336 X 256	-	16
5A	336 X 256	4 X 60	16
6	336 X 256	-	any 4 of 16
6A	336 X 256	4 X 60	any 4 of 16

In an "all-graphic mode" the screen consists of :

A1. Header
A2 Screen Colour Data
A3 Trailer

The header and trailer areas are blanking lines at the beginning and the end of the screen.



Examples MODE 1

In "Split-Graphics" modes, the screen organisation is as follows :

B1	Header
B2	Picture area corresponding to the bottom part of A2
B3	Middle
B4	Character area
B5	Trailer
B6	Archive area, containing the upper part of A2, which is not displayed in B2

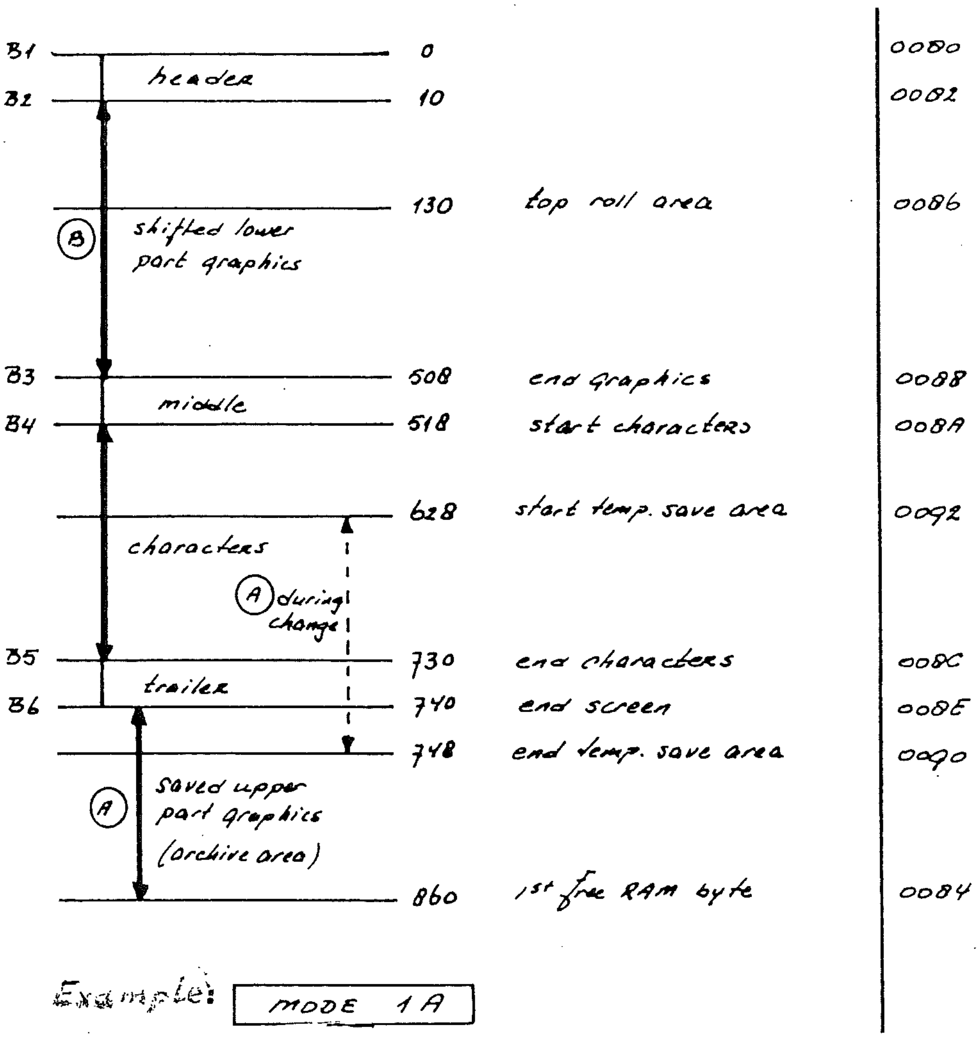
When changing from split-mode to all-graphic-mode and visa-versa, the archive area is temporarily moved to the temporary save area, which starts at A3.

When changing from all-graphics to split :

Top of screen into temporary save area
Bottom part graphics shifted upwards
Top part graphics from temporary save area into archive area.
Set up character screen

When changing from split to all-graphics :

Archive area into temporary save area
Bottom part graphics shifted downwards
Temporary save area into top of screen



The numbers at the right side of the lines area offsets from the address of the top of the screen (//BFFF for 48K). These values are stored in RAM pointers (addresses in separate columns) during the set-up of a particular screen mode.

For each mode, these constants are retrieved from ROM. The annexed table gives all the constants as they can be found in ROM-bank 2 on the addresses 2E030 - 2E0C2. Vectors to these constant tables are located on addresses 2E59A - 2E5A5.

These constant tables are moved into the screen variable (see memory map) when the appropriate mode is entered.

SCREEN CONSTANTS GRAPHIC MODES
==============================

RAM	POINTER	MODE 0	MODE 1/2	MODE 1A/2A	MODE 3/4	MODE 3A/4A	MODE 5/6	MODE 5A/6A
0080/81	1st byte screen RAM	0	0	0	0	0	0	0
0082/83	Points after header	10	10	10	10	10	10	10
0084/85	1st free RAM byte	CB0	638	860	177C	19A4	5A20	5C48
0086/87	Top rolled area	0	130	130	460	460	F88	F88
0088/89	End graphics area	0	628	508	176C	131C	5A10	4A98
008A/8B	Start character area	10	628	518	176C	132C	5A10	4AA8
008C/8D	End character area End archive area	CA0	860	730	19A4	1534	5C48	4CC0
008E/8F	End screen	CB0	638	740	177C	1554	5A20	4CD0
0090/91	End area used changing mode	0	748	748	1BBC	1BBC	6988	6988
0092/93	Start archive area Start temp.save area	0	740	628	1554	176C	4CD0	5A10
0094/95	Number of hor. blobs	0	48	48	A0	A0	150	150
0096	Number of graph. lines	0	41	41	82	82	0	0
0097	Number of saved graphic lines	0	0C	0C	18	18	2C	2C
0098	Number bytes/line	0	18	18	2E	2E	5A	5A

All constants given are offset of top of screen address (#BFFF) except the last 4 data blocks (0094-0098)

The screen can operate at a number of different definitions horizontally (e.g. blobs/scan). In the highest definition graphics mode there are 352 visible blobs across the screen. The two lower definitions have respectively 1/2 and 1/4 of this number. There are about 520 scans visible on a "625 line" television, and the screen hardware can only draw (at minimum) 2 scans per line, due to the interlacing. This gives a maximum definition of 260 by 352 which is close to the 3:4 ratio of the screen sides. Thus circles come out round .

Characters are fitted onto this grid by using 8 columns of blobs per character, the dot positions being defined for each character by a ROM. This allows 44 characters per line maximum (or 22/11 in lower definition modes).

A fourth horizontal definition provides for a "high density" character mode with 66 characters/line.

A total of 16 different colours, including white and black can be displayed by the system. Whenever a 4 bit code is used to describe a colour, it selects from this range of possibilities. In some modes (characters + or four colour graphics) a set of 4 of these colours (not necessarily distinct) are loaded into a set of "colour registers". Any 2 bit code describing a colour selects an entry from these registers.

Vertical definition is set by a 4 bit field in the control byte. In graphics modes this simply allows repetition of the information to fill any even number at scans from 2 to 32. In character mode it defines the number of scans occupied by each line of characters; thus the vertical spacing on the screen can be changed to allow anything between an 8 x 7 (the sensible minimum) and 8 x 16 character matrix, giving between 35 and 15 lines of characters on the screen.

The first byte of information for the screen is located at the top of a 32K block of memory. Successive bytes follow at descending addresses. The screen takes memory and displays a picture on the screen accordingly until the whole screen has been filled. It then starts again at the first byte.

At the beginning of the data for each line, two bytes of data represent the lines control word. The control word defines the raster scan depth of the line, the horizontal graphical resolution of the line and selects the display mode of that particular line. Subsequent to this control word a number of data words are stored that represent the colour of pixels, or definition and colour of characters according to the selected display mode.

A) CONTROL WORD FORMAT

MC	RC	LRC	CS	U CM	SCR	S16C	

a.) Mode Byte

LRC : **Line Repeat Count** (bit 0-4)

The line repeat count controls the number of horizontal raster scans for which the same data will be displayed. Since interlace of the TV scan is ignored a minimum of two raster scans correspond to a line repeat count of zero. Thereafter, each additional repeat adds two scans to the line. The maximum programmable depth of any horizontal display segment is thus 32 scans. (European TV sets will show approximatively 520 scans total for a full picture).

RC : **Resolution Control** (bit 4/5)

The resolution control bits allow selection of one of four different horizontal definitions for display of data on the TV screen for each individual line.

Code (Bit 5,Bit 4)	Definition(pixels per screen width
00	88 (Low definition graphics) 24 bytes
01	176 (Medium definition graphics)46byt.
10	352 (High definition graphics) 90 byt.
11	528 (Text with 66 characters per line) 134 bytes (Screendriver uses 60 characters for text) (Could be used for a very high definition graphics mode)

The number of bytes per screen width can be obtained by multiplying the number of characters by two and add two for the mode byte and the colour byte. There is an exception for the characters in low and medium resolution (resp. 26 and 48 byte per line).

MC :**Mode Control** (bit 6/7)

Code	Display Mode
(Bit 7, Bit 6)	
00	Four colour graphics
01	Four colour characters

10	Sixteen colour graphics
11	Sixteen colour characters

b) Colour Type Byte

The Low address control byte is used to store colours into a set of 4 "colour registers" for the four colour mode. Any one of the four colours in the registers can be changed at the beginning of any line of display data. Only the colours in these registers can be displayed in any 4 colour mode. The four colours are freely selectable from the sixteen colours defined in Colour Select Table.

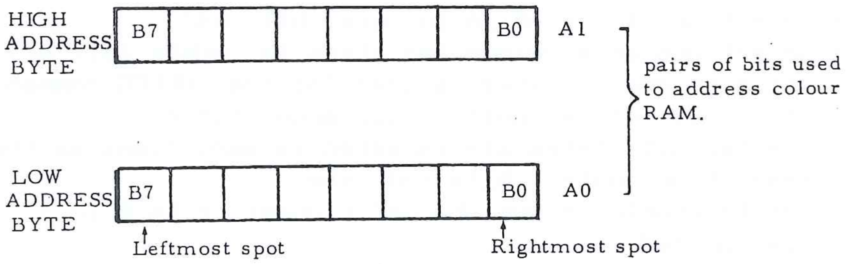
S16C : Selection of one of 16 colours (bit 0-4)
SCR : Select one of 4 colour registers to update (bit 4/5) which colour register is used for the COLORG command.
VCM : if unset, forces "unit colour mode (bit 6) The two data bytes are repeated as many times as the resolution control bits indicate.
CS : Set to enable colour change/if unset, bits Ø to 5 are ignored.

Code	Code
0	Black
1	Dark Blue
2	Purple Red
3	Red
4	Purple Brown
5	Emerand Green
6	Kakhi Brown
7	Mustard Brown
8	Grey
9	Middle Blue
10	Orange
11	Pink
12	Light Blue
13	Light Green
14	Light Yellow
15	White

B) DATA MODE

a) 4 Colour Mode

In this mode only two bits of data are required to define the colour of a pixel. These data bits are obtained in parallel from the upper and lower bytes of each data word using the high order bits first. The 2 bytes in a field are considered as 8 pairs of bits. Each pair sets the colour for one spot.



The 2 bits for each spot select one of the four colours which have been loaded into the colour RAM by previous Colour Control bytes. So on any line 4 colours are available. On the next line any one of these may be changed for another, and so on.

b) Sixteen Colour Mode

This graphics mode is designed to allow multi-colour high definition pictures in half the memory requirement of other systems.

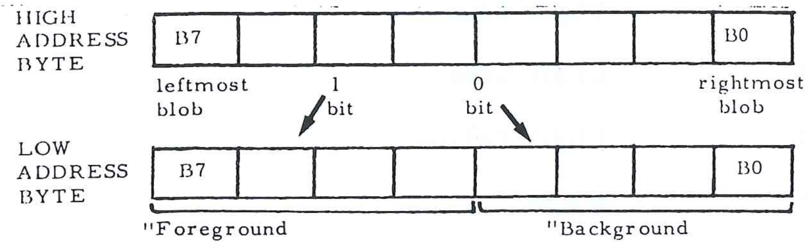
The basic organization is that the low address byte selects two of the sixteen possible colours.

Bits 0 - 3	"Background" colour
Bitsd 4 - 7	"Foreground" colour

The high address byte than defines by each successive bit whether a colour blob should be foreground or background.

NB

The two bytes in the field serve different purposes, one being used to define two available colours for use in the field, and the other to choose one of these for each spot.



The bit for each spot can select either the "foreground" or the "background" colour. However, what these colours are is totally independent of the preceding or following fields. So any line may use any and all of the total 16 colours. The contents of the colour RAM are irrelevant in this mode.

One additional feature is added to eliminate restrictions of the scheme. After each eight bit field of colour the background is extended into a new area, even if a new background is specified, until the new foreground is first used. It is therefore possible to create a required picture by suitable combination of foreground and background.

c) Character Mode

In this mode, characters are generated using a character generator ROM in conjunction with the four colour registers or using any 2 colours for each in the 16 colour character mode.

The usual character matrix is 6 x 9 bits out of a possible 8 x 16. Therefore the line repeat count should be at least eleven, to guarantee full character display plus line spacing.

Four colour characters are produced on the screen in a way similar to the four colour graphic mode, but with the character ASCII data replacing the high address data byte used for four colours. The result is that characters are displayed using colours from the four colour registers. The data from the character generator ROM control the lower address bit and bits from the low-address byte determine the other. This allows characters on a single horizontal display segment to be in one of two colour combinations of character/background, or even with a vertical striped pattern controlled by the low address byte.

However, note that as compared with four colour mode information (but not the low-address byte)is subject to a one character position delay before appearing on the screen.

In character mode the height of the characters is a set number of horizontal scans. The character width is determined by the definition selection in the control byte. A definition of 352 yields 44 characters per line, 528 yields the normal 66 characters per line. Other definitions in applications such as the power-on message. However, this feature is not supported by the resident BASIC.

There is a very important difference between the character and the graphic mode : the place of the colour byte.

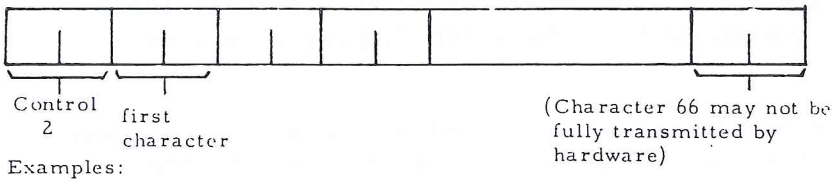
```
graphics   : address colourbyte = address databyte-1
characters : address colourbyte = address databyte-3
```

This means that the colour information for the last character of a line is defined by the colour byte of the next line.

<u>Example Mode Ø :</u>

Line 1 Control byte is located at address BFEF and line 1 Color Control byte address at BFEE. The first character byte of line 1 is located at line 1 Control byte address minus 2, and the character Colour Control byte at line 1 Control byte address minus 3. Each of the 66 positions of the screen is located at line Control byte - (2 * position of character on the line) for the character and at line Coltrol byte - (2 * position of character on the line) for the Colour Control byte of the character.

Remember that there are 66 character positions on the screen but that the first and last three characters are kept blank for the margins. Therefore, the Control byte for the next line is located at Control byte of previous line (i.e. BFEF) less 134 bytes (# 86. So if the Control byte ofline 1 is at BFEF, the Control byte of line 2 will be at # BFEF - # 86 = # BF69).



Examples:

Control Byte Line 1	# BFEF	
Control Byte Line 5	# BFEF - (# 86*5)	= # BDD7
Colour Control Byte Line 5		= # BDD6
Character No.6 on Line 5 # BDD7 - 6*2		= # BDCC
Colour Character 6 of Line 5		= # BDCB

Use the POKE in your program for changing line background, letter colour, or letter, and Utility 3 for checking the location you intend to POKE (when you return to BASIC the colour changes you made in Utility mode aree erased of you enter MODE 1, RETURN, MODE 0.

C) UNIT COLOUR MODE

This mode is available for space saving during uniform scans of the picture. A horizontal band of constant colour (or repeated pattern) can be drawn using only one control word and one data word. The data for this mode should be in high speed format. Using the mode a full screen of data need be no more than 40 bytes of ram.

The hardware for the unit colour mode knows two different ways to read out the picture : Medium/low and high/super resolution. In both ways we use 4 bytes per line : The line mode and the colour type byte + data + colour byte.

In low/medium resolution is the gap between the line mode bytes two bytes, in the high/super is the distance 4 bytes. So in the low/medium resolution is the data and colour byte of the first line also line mode and colour type byte of the second line. In the high/super resolution can data and colour byte be filled in by choice.

D) EXAMPLE

```
 1Ø    Mode Ø : PRINT CHR $(12) : COLORT Ø14ØØ

 2Ø    CURSOR Ø,17 : PRINT"  INDATA       DESK COMPUTER"

 3Ø    POKE # BCCB,#5F:POKE #BC9B,#5F:POKE
#BC9A,#4Ø
```

BCCB : Mode byte line 7: 5F

LRC = 1111 number of horizontal scans for which the same data will be displayed

RC = Ø1 medium definition 23 characters per line

MC = Ø1 four colour characters

Mode byte for next line is on BCCB-48=BC9B
Place the same mode byte on this address + place the colour type byte on next address = 4Ø

BC9A : colour type byte : 4Ø

CS = Ø no colour change enable
VCM = 1 no unit colour mode
Bits Ø → 5 are ignored (no colour change enable)

2) HARDWARE TIMING

The idea of the hardware is as follows :

First a counter is loaded with the highest RAM address and the two instructions are read (line mode-and colour type byte). These two bytes determine the number of bytes till the next instruction (line). Also is the colour mode, character graphical mode, speed control and short line control available. When the counter comes to zero (TOPRAM-line length) the next instruction bytes are read.

4 COLOUR PICTURE GENERATION

A spot on a four colour picture is generated using two data bits (note that one of the four colours could be changed at each control byte so that a "4 colour" picture can use all 16 colours, though this is not supported by software). Each bit is obtained from an eight bit shift register, either IC39 for low order or IC40 for high order. The data for the two shift registers is obtained in parallel from RAM as 16 bits labelled VO-V15.

The two bits pass through IC27 and select one of four colours from the RAM chip IC37. The 4 bit colour code is then latched by IC22 to synchronise information. Information is loaded from memory to the shift registers by the signal VL from IC20. This is a very short pulse timed to overside one clock pulse to the shift registers.

LINE REPEAT CONTROL IC47; IC34

To control vertical definition, and for characters, data from memory may be scanned several times to create a band of graphics or text on the picture.

At the start of each line the line repeat count is latched by IC48. At the end of each line the counter IC34 is incremented. After a number of lines the four bit comparator IC47 generates and EQ output. This allows both memory counts to count in parallel with the result that on the next line different information is obtained from memory. At the same time the EQ signal causes the line counter IC34 to be reloaded with zero ready for the next count at a clock pulse LINEQ.

Signal Pinout

Pin	Signal	Use
IC47 pin 6	EQ	Allow both memory counters Reset line counter (IC34)
IC34 pin 2	LINEQ	Clocks line counter
IC34 pin 1	PAGE	Clears counter at start of a new page. (Otherwise picture jumping can happen)
IC34 pins 14 13,12,11	LINE COUNT	To select the correct horizontal scan for character generation.

CONTROL WORD LATCH IC48; IC37; IC3 (HALF)

At the start of each line control information is read from memory and stored for use during a line scan, or for colour information, during the remainder of the picture scan.

The storage strobe (CTRL BYTE)(Pin 11IC48) is a clock pulse gated by LINEC from IC2, P1 from the phase information, and the spot clock pulse from IC12.

IC48 latches the line repeat count, spot feed control bits, graphic/characters mode selection bit and 16/4 colour selector bit.

IC37 latches one of four colours per 4 colour mode.
IC3 latches the short line control, SLC, for IC11 pin 15.

Signal Pinout

Pin	Signal	Use
IC48 Pin 9,5,12,16	LINE REPEAT COUNT	Vertical definition control
IC48 Pin 6,2	SPOT SPEED CONTROL	Horizontal definition control
IC48 Pin 15	GRAPHIC/CHARACTER	Switches picture data
IC48 Pin 19	16/4 COLOUR	Switches colour mode
IC37 Pin 15,1,2,3	NEW COLOUR INPUT	)
IC32 Pin 13,14	NEW COLOUR ADDRESS	) Define one of four colours
IC37 Pin 12	COLOUR WRITE STROBE	)
IC 3 Pin 12	SLC INPUT	
IC 3 Pin 8	SLC OUTPUT	Controls short line logic

MEMORY ADDRESS COUNTERS IC105, 106, 89, 90, 91, 92

Memory is addressed in two stages to allow for line repetition according to definition, or for character generation.

The first stage IC105, 106 is a slow counter and is active counting only during the last line scan of a particular memory area.

The second stage IC89, 80, 91, 92 is a high speed counter which is loaded at the start of every line and counts throughout the line, except for short lines where the same memory data is read throughout a line scan. On the last line of any scan of a memory area the gating signal LEQ (line count equal) from IC47 allows the counters to count in parallel.

The page signal resets the first stage counters.

SIGNAL PINOUT

Pin	Signal	Use
IC105 2 X 12) IC106 2 x 12)	PAGE	Memory address reset
IC105 Pin 1	CLOCK IN	First stage clock, requires correct gate and P3 to be active. Thus counts each eight BORROW of IC12
IC89,90,91,92 Pin 2	CLOCK IN	Second stage clock, requires correct gate

		and P3 to be active. Thus counts each eight BORROW of IC12
IC89,90,91,92 Pins 14,13,12,11	MEMORY ADDRESS out. MV0-15	

PHASE GENERATE IC85

This IC is a standard binary divide by eight used to generate basic phase signal to drive dynamic ram and to synchronise memory access for the picture. It is driven by the divided spot oscillator SDIV, and by the synchronised line sync. LINEQ.

SIGNAL PINOUT

Pin	Signal	Use
1	LINEQ	Holds IC85 cleared during LINEQ
2	SDIV	Input clock
14,13,12	OUTPUT	Phase signals for memory control

SPOT OSCILLATOR IC1, IC3 (half) IC18 (half)

The basic spot oscillator is a standard TTL oscillator controlled by the LINEQ signal. The circuit oscillatos only during the active part of each line scan. Start and stop should be clean with no spurious pulses which may cause loos of dynamic ram information.

Note carefully the operation of IC3. When LINEQ stops the oscillator, the last oscillator cycle clocks IC3 to povide a reset signal to IC12. The additional delay so introduced after LINEQ is to prevent IC12 from generating spurious output pulses.The IC18 provides a divide by two (IC18=toggleFF) to enable the counters IC4,6 to be sufficient for a full line scan.

SIGNAL PINOUT

Pin	Signal	Use
IC1 Pin 4	LINEQ	Oscillator control
IC1 Pin 11	CLOCK OUTPUT	All spot + control timing

IC3 Pin 5	RESET FOR IC12
IC18 Pin 9	1/2 CLOCK TO LINE CONTROL

SPOT CONTROL SIGNAL GENERATOR IC4,6,11,2

Control signals for internal (not TV) line control are generated by a counter IC4,6 driven from the spot oscillator, and reset by $\overline{\text{LINEQ}}$.

To enable the use of a single decoder IC11, two signals are combined via a gate of IC24. This introduces a non-standard count sequence at the input pins of IC11.

IC11 also uses the **GRAPHIC** signal to modify control output for character/graphic mode and the short lines LINE signal for short line mode (Unit Colour Mode. Minimum memory use).

IC2 latches IC11 output to eliminate spikes, using a convenient signal from IC4 (spot clock/8). During design much care was necessary to ensure that clock strobes from IC12 came correctly within gate signals from IC2, since the gate logic is independant of the count division of IC12.

SIGNAL PINOUT

Pin	Signal	Purpose
IC4 Pin 1	CLOCK DRIVE	
IC2 Pin 15	MEMC	Gates counting of the memory address
IC2 Pin 10	LINEC	A line control gate for latching of control signals
IC2 Pin 7	LORDC	A control to load a new line start memory address
IC2 Pin 2	IBLANK	An internal blank ,to control ICS, therefore the the spot count divider IC12

SPOT SPEED DIVIDER IC12

The basic spot oscillator has a fixed frequency which must be divided to achieve lower definition pictures. The divide count comes from IC5 and is counted down. The combined BORROW and LOAD signals are a very important feature of this circuit as is the reset signal on pin14. The general timing in this area has been optimised to avoid double pulses at the moment that the spot oscillator is at the end of a line. The BORROW output of IC12 drives **all** picture data

IC12 SIGNAL PINOUT

Pin	Signal
15,1,10,9	COUNT INPUT
13	BORROW OUT
11	LOAD IN
14	RESET IN

SPOT SPEED CONTROL IC5

The spot speed control provides a count to IC12 which is used as a spot clock divider to determine the ultimate spot rate for the picture. IC5 is a fuse-link Rom with outputs determined by the speed control signals from RAM picture data, the blank signal (CBLANK) and the LINE signal after synchronisation, LINEQ.

Note that the blank signal is not a true picture blanking and does not directly affect the picture. IC5 outputs a fixed standard count during CBLANK.

IC5 SIGNAL PINOUT

Pin	Signal	Use
10 11	LOW ORDER SPEED) HIGH ORDER SPEED)	Spot speed control
13	LINEQ	Fixes output count
2,1,5,6	OUTPUT COUNT	Used to reset the divider IC12

LINE/SPOT SYNCHRONISATION IC18 (Half)

During normal line activity the spot clock (IC1) drives a counter to generate an 8 stage cycle corresponding to the 8 bits of each byte of information. The stages are called Po to P7.

At the end of each line we must reach the state Po and because the picture controller IC25 is not directly synchronised to the spot oscillator an additional synchronising cricuit is needed. This is provided by one half of IC18.

As soon as Po becomes true after $\overline{\text{LINE}}$ the synchronising FF changes and stops the spot clock. It also resets the spot counter IC4, IC6.

IC18 LINE/SPOT SYNC.

Pin,	Signal	Use
2	LINE INPUT	
3	PO CLOCK IN	
5Q, 6Q	SYNCHRONISED LINE OUT	Stop spot clock. Reset spot counter Lock phase counter to Po

FRAME TIMING IC25

The line and frame timing and standard picture timings are provided by a single logic IC25 (ZNA134J) custom built for the purpose. A jumper is available for European or USA standard but the' software supports only the European form.

IC25 incorporates a Xtal oscillator and this controls all picture timings except dot speed, hence picture width. Dynamic ram timing is directly driven from the picture timing.

IC25 SIGNAL PINOUT

3	$\overline{\text{SYNC}}$	Used only for TV Card
4	$\overline{\text{BLANK}}$	Used only for TV Card
5	$\overline{\text{LINE}}$	Generates synchronised line , reset IC18
8,9	XTAL DRIVE	
13	CATHODE BLANKING	Used only for TV Card
14	$\overline{\text{EVEN FIELD}}$	Used only for TV Card
16	$\overline{\text{PAGE}}$	Resets all picture counters together with LINE

CHAPTER V

INPUT/OUTPUT CIRCUITS

1) KEYBOARD CONTROL + RS232 BUS

KEYBOARD

The Desk Computer contains a software keyboard scan and encoder. This can be used by other programs which may use the standard key encoding tables, or supply their own.

All keys are scanned periodically, and action is taken when a key is noticed to have been newly pressed. Alternatively, if the repeat key is pressed, then periodically all currently pressed down keys are acted on. The repeat speed is fixed.

The actual code for the key is obtained from a table. The "shift" system selects which of two possible tables of use. By setting a flag byte the keyboard handler can be made to scan only for the "**BREAK**" key which obviously takes less time.

The ASCII keyboard is scanned as a matrix of switches. Encoding, debouncing and roll-over are realized via a software routine.

KEYBOARD LAYOUT

The keys are assigned to rows and columns.

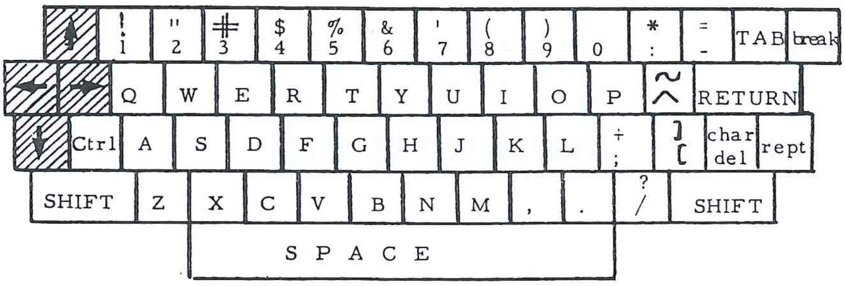
ROWS

Output lines (FF07)

	0	1	2	3	4	5	6
0	0	8	re-turn	H	P	X	↑
1	1	9	A	I	Q	Y	↓
2	2	:	B	J	R	Z	←
3	3	;	C	K	S	[	→
4	4	,	D	L	T	^	Tab
5	5	-	E	M	U	space bar	ctrl
6	6	.	F	N	V	rept	break
7	7	/	G	O	W	char del	shift

COLUMNS

Input lines (FF01)



The hardware control of the keyboard is done by one multifunction input/output circuit : the TMS 5501

The I/O section of the TMS 5501 contains an eight bit parallel input port and a separate eight-bit parallel output port with storage register. Five programmable interval timers provide time intervals from 64 µs to 16.32 ms.

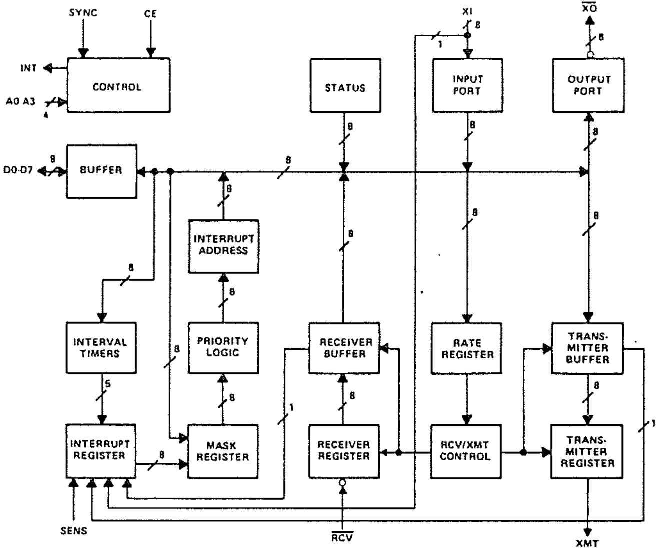
The interrupt systems allows the processor to effectively communicate with the interval timers, external signals, and the communication interface by providing 8080 compatible interrupt logic with masking capability.

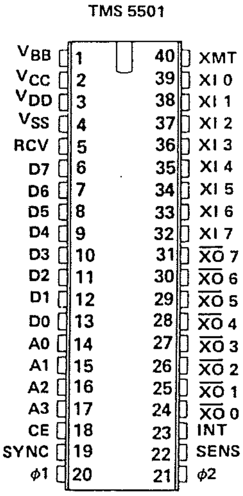
Data transfers between the TMS 5501 and the CPU are carried by the data bus and controlled by the interrupt, chip enable, sync, and address lines. The 8080 uses four of its memory-address lines Ao-A3 to select one of 14 commands to which the TMS 5501 will respond. These commands allow the CPU to :

* read the receiver buffer
* read the input port
* read the interrupt address
* read TMS 5501 status
* issue discrete commands
* load baud rate register
* load the transmitter buffer
* load the output port
* load the mask register
* load and interval timer

The commands are generated by executing memory referencing instructions such as MOV (register to memory) with the memory address being the TMS 5501 command. (for addresses see memory map; interrupt sections 0-7 and addresses FFFØ-FFFD).

The TMS 5501 moves data between the CPU and the keyboard through its internal data bus, input port, and output port. When data is present on the bus that is to be sent to the keyboard, a Load Output Port (LOP) command from the CPU puts the data on the XO pins of the TMS 5501 by latching it in the output port. The data remains in the port until another LOP command is received. When the CPU requires data that is present on the External Input (XI) lines(when you touch a key), it issues a command that gates the data onto the internal data bus of the TMS 5501 and consequently onto the CPU's data bus at the correct time during the CPU cycles.





To start a countdown by any of the five interval timers, the program selects the particular timer by an address to the TMS 5501 and loads the required interval into the timer via the data bus. Loading the timer activates it and it counts down in increments of 64 microseconds. The 8-bit counter provide intervals that vary in duration from 64 to 16,320 microseconds. Much longer intervals can be generated by cascading the timers through software. When a timer reaches zero, it generates an interrupt that typically will be used to point to a subroutine that performs a servicing function as scanning the keyboard. Loading an interval value of zero causes an immediate interrupt. A new value loaded while the interval timer is counting overrides the previous value and the interval timer starts counting down the new interval. When an interval timer reaches zero it remains inactive until a new interval is loaded.

The TMS 5501 provides the system with several interrupt control functions by receiving external interrupt signals, generating interrupt signals, masking out undersired interrupts, establishing the priority of interrupts, and generating RST instructions for the 8080. An external interrupt is received on pin 22, SENS to send stack overflow message interrupt vector routine 2. An additional external interrupt is received on pin 32, XI17. It is connected to the page signal (20 ms) to flash the cursor on the screen , (see also restart routines chapter II). The TMS 5501 generates an interrupt when any of the five interval timers count to zero. Interrupts are also generated when the receiver buffer is loaded and when the transmitter buffer is empty.

The highest priority interrupt passes through to the interrupt address logic, which generates the RST instruction to be read by the CPU. See table 5.1 for relationship of interrupt sources to RST instruction.

The INT signal of the TMS 5501 is tied to the INT input of the 8080. The sequence of events will be :

1) The TMS 5501 receives (or generates) an interrupt signal and readies the appropriate RST instruction.

2) The TMS 5501 INT output, tied to the 8080 INT input, goes high signaling the CPU that an interrupt has occured.

3) If the 8080 is enabled to accept interrupts, it sets the INTA (interrupt acknowledge) status bit high at SYNC time of the next machine cycle.

4) If the TMS 5501 has previously received an interrupt-acknowledge-enable command from the CPU, the RST instruction is transferred to the data bus.

RST INSTRUCTIONS

TABLE 5.1.

DATA BUS BIT								INTERRUPT CAUSED BY		
0	1	2	3	4	5	6	7			
H	H	H	L	L	L	H	H	Interval Timer 1	C7 = RST 0	0
H	H	H	H	L	L	H	H	Interval Timer 2	CF = RST 1	8
H	H	H	L	H	L	H	H	External Sensor	D7 = RST 2	10
H	H	H	H	H	L	H	H	Interval Timer 3	DF = RST 3	18
H	H	H	L	L	H	H	H	Receiver Buffer	E7 = RST 4	20
H	H	H	H	L	H	H	H	Transmitter Buffer	EF = RST 5	28
H	H	H	L	H	H	H	H	Interval Timer 4	F7 = RST 6	30
H	H	H	H	H	H	H	H	Interval Timer 5 or X17	FF = RST 7	38

RS232 BUS

The Desk Computer has an RS232 compatible interface giving a serial input line, serial output line and a status line to halt output (DTR). These are available on a CCITT standard connector at the rear of the machine. The DTR signal allows synchronisation of the output with a printer. If unused, then output will be unimpeded.

The communciations section of TMS 5501 is an asynchronous transmitter and receiver for serial communications and provides the following functions :

* Programmable baud rate - A CPU command ,selects a baud rate of 110,150,300,1200,2400,4800, or 9600 baud.

* Incoming character detection - The receiver detects the start and stop bits of an incoming character and places

the character in the receive buffer (via the circuit around T12).

- * Character transmission - The transmmitter generates start and stop bits for a character received from the CPU and shifts it out via T13.

- * Status and command signals - Via the data bus, the TMS 5501 signals the status of : framing error and overrun error flags; data in the receiver and transmitter buffers; start and data bit detectors; and end-of-transmission (break) signals from external equipment.

The TMS 5501 operates as memory device for the CPU. Functions are initiated via the 8080 address bus and the TMS 5501 address inputs. Address decoding to determine the command function being issued is defined below.

COMMAND ADDRESS DECODING
When CHIP Enable Is High

	A3	A2	A1	A0	COMMAND	FUNCTION
FFF0	L	L	L	L	Read receiver buffer	RBn → Dn
FFF1	L	L	L	H	Read external inputs	XIn → Dn
FFF2	L	L	H	L	Read interrupt address	RST → Dn
FFF3	L	L	H	H	Read TMS 5501 status	(Status) → Dn
FFF4	L	H	L	L	Issue discrete commands	
FFF5	L	H	L	H	Load rate register	
FFF6	L	H	H	L	Load transmitter buffer	Dn → TBn
FFF7	L	H	H	H	Load output port	Dn → XOn
FFF8	H	L	L	L	Load mask register	Dn → MRn
FFF9	H	L	L	H	Load interval timer 1	Dn → Timer 1
FFFA	H	L	H	L	Load interval timer 2	Dn → Timer 2
FFFB	H	L	H	H	Load interval timer 3	Dn → Timer 3
FFFC	H	H	L	L	Load interval timer 4	Dn → Timer 4
FFFD	H	H	L	H	Load interval timer 5	Dn → Timer 5
FFFE	H	H	H	L	No function	
FFFF	H	H	H	H	No function	

RBn	=	Receiver buffer bit n
Dn	=	Data bus I/O terminal n

XIn	=	External input terminal n
RST	=	11 (IA2) (IA1) (IA0) 1 1 1 (see table 5.1)
TBn	=	Transmit buffer bit n
XOn	=	Output register bit n
MRn	=	Mask register bit n

For detailed functions of the 5501 commands see memory management (Chapter I).

2) THE DCE-BUS *

The DCE-BUS is a way to connect the computer with other peripherals e.g. floppy disks, printer with parallel input, the DAI real world cards, or self designed circuitry, etc.

There are two methods to use the DCE-BUS

1) Use it as a usual 3x8 bit parallel I/O port.
   You are free in the use of the I/O port and the software.

2) Use it as the DCE-BUS standard with the real word cards
   or compatible circuitry.

DESCRIPTION OF THE DCE-BUS

The DCE-BUS exists in the Desk Computer off the Intel 8255 (programmable peripheral interface). This IC is also named GIC (general interface control). It has two 8-bit Ports (PØ and P1) and two 4-bit Ports (P2H and P2L). These Ports can be used as in-or output. We do this by sending a control byte to the internal register in the GIC. The 8255 is connected with the CPU through the data addresses control bus.

The addresses of the GIC are :

* FE00 : Port 0
* FE01 : Port 1
* FE02 : Port 2H + 2L
* FE03 : control byte

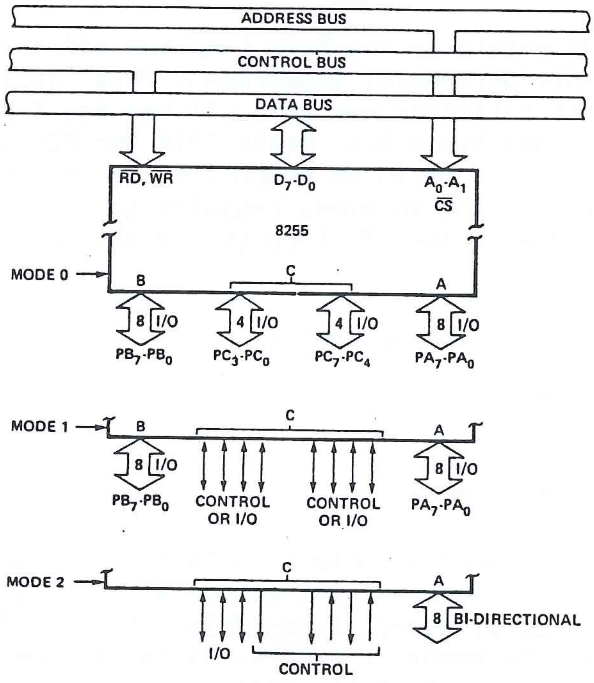
A port, programmed as output, acts the same as a latch, this means when we write certain data into it, the data will stay stable until we write new data into it. This data can also be read by means of a PEEK. As far as now we have only described the mode Ø of the 8255. (Mode 1 provides a mean for transferring I/O data to or from a specified port in conjuction with strobes or "handshaking signals". In mode 1 Port A and B use the lines on Port C to generate or accept these "handshaking" signals. When you change from one mode to another then the output latches are reset. All ports will be set as input port after a hand reset. The list of these control bytes is displayed in the memory management (in mode Ø). For the numbers of the connector see Desk Computer Manual.

Mode 2 of the 8255 provides communication with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bi-directional I/O). Handshaking signals are provided to maintain proper bus flow discipline in a similar manner to mode 1. Interrupt generation and enable/disable functions are also available.

* the 2* 80K floppy controller DOS V1.0 is controlled as a DAI Real World Card.

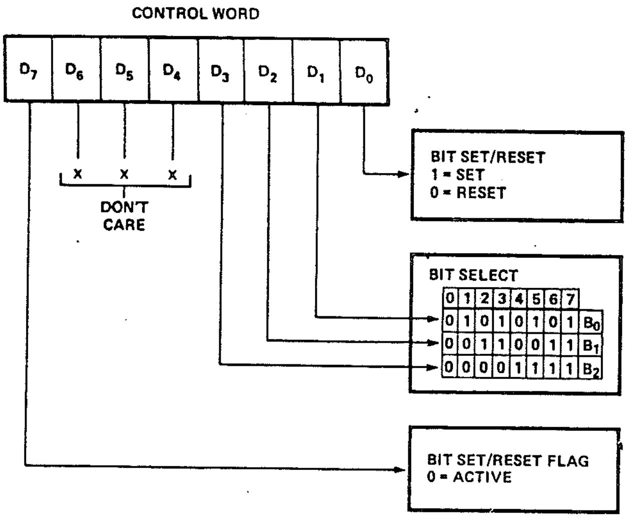
The Input signals (AØ and A1) in conjunction with the RD and WR inputs, control the selection of one of the 3 ports or the control word register. The are connected to the least significant bits of the address Bus (A0 and A1).

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION (READ)
0	0	0	1	0	PORT A ⇒ DATA BUS
0	1	0	1	0	PORT B ⇒ DATA BUS
1	0	0	1	0	PORT C ⇒ DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS ⇒ PORT A
0	1	1	0	0	DATA BUS ⇒ PORT B
1	0	1	0	0	DATA BUS ⇒ PORT C
1	1	1	0	0	DATA BUS ⇒ CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS ⇒ 3-STATE
1	1	0	1	0	ILLEGAL CONDITION

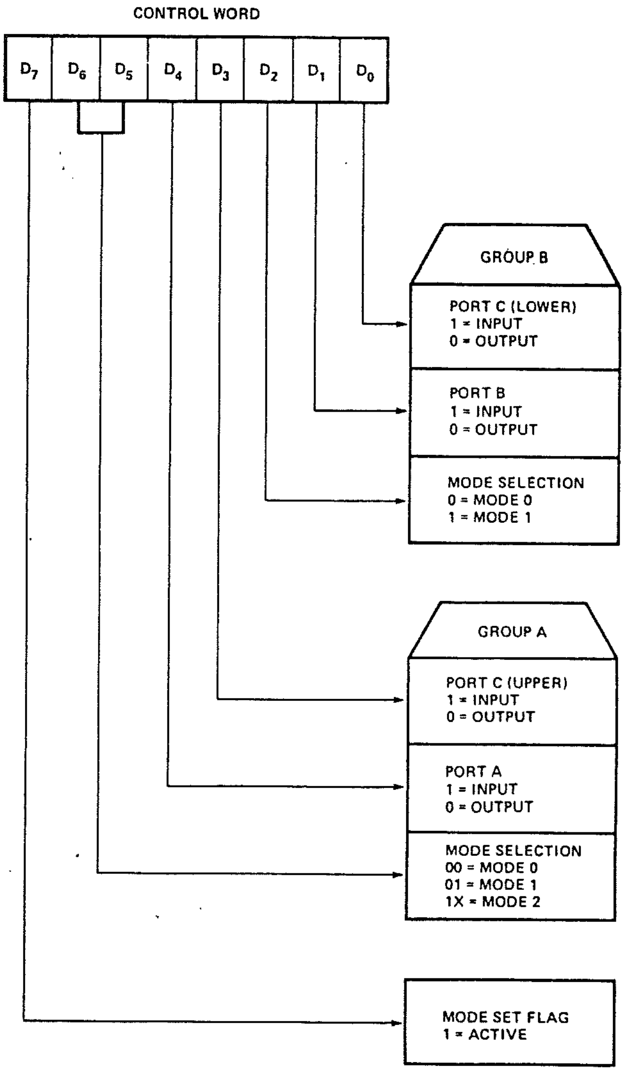


Basic Mode Definitions and Bus Interface

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure.



Bit Set/Reset Format



Mode Definition Format

THE DCE-BUS MODE

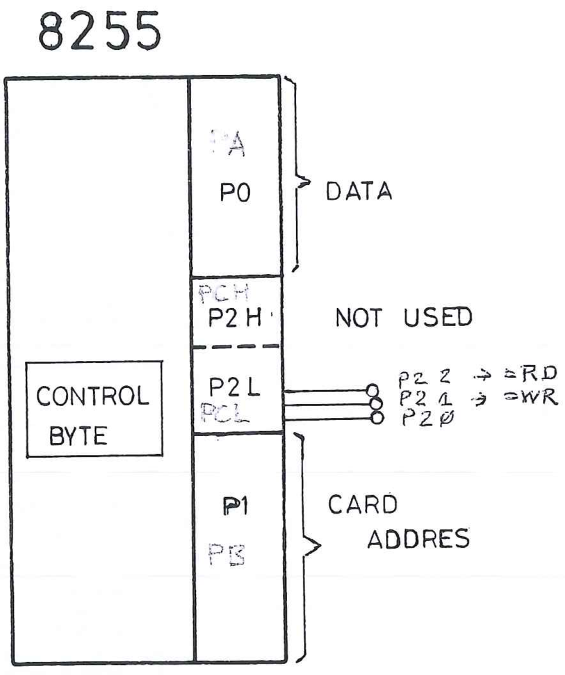
In this mode we use the Ports from the 8255 according to certain rules . The meaning is to build a universal system where a lot of I/O cards are connected simultaneously.

Signals :

DATA : bi-directional channel to and from the cards
RD : Puls (P2 bit 2) to read data from cards
WR : (P2 bit 1) to write data to cards

BUS EXP : Bus expand (P2 bit Ø) used as BUS ON/OFF
1 = OFF Ø = ON
CARD ADDRESS : selection of one card + register selection on card

The basic instructions **OUT** and **INP** are used to communicate with these cards

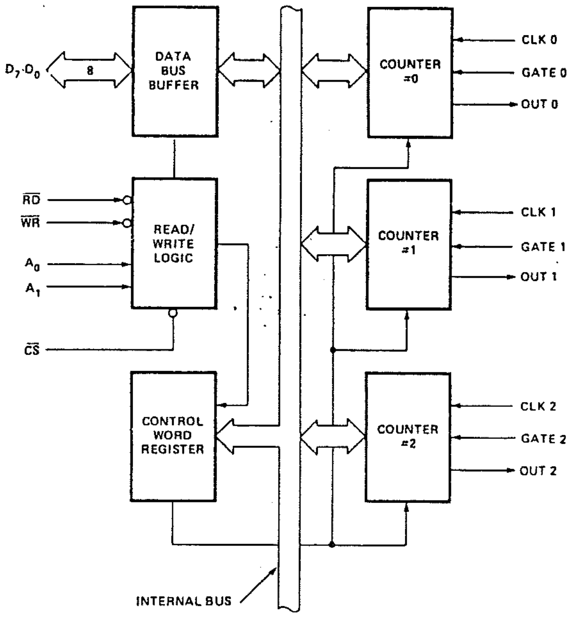


3) SOUND & PADDLE (sheet 7 & 8)

The sound generator of the Desk Computer has considerable flexibility because every frequency is generated by digital oscillators that yield precise results. Additional random noise generation and digital volume controls complete the system.

The Programmable Graphical Sound Generator is realised via three independent programmable oscillators and a random noise generator. Each oscillator is connected as an I/O device to the microprocessor and is programmable to any frequency within the range 33HZ to 0,5MHZ. Obviously the higher frequencies are not interesting for audio work but since the three oscillators are added together before modulation of the audio channgel of the TV interesting effects can be obtained by beating together various possibilities. The programmable oscillators (IC28:8253) are used for sound generation and game paddle interfaces.

The 8553 is organized as three independent 16-bit counters. All modes of operation are softwarre programmable by the 8080. The inputs Ao,A1 are connected to the address lines A1 and A2. Their function is to select one of the three counters to be operated on and to address control word register for mode selection. The information stored in the control word register controls the operational mode of each counter, selection of binary or BCD counting and the loading of each count register. The control word register can only be written into, no read operation of its contents is available.



$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Load Counter No. 0
0	1	0	0	1	Load Counter No. 1
0	1	0	1	0	Load Counter No. 2
0	1	0	1	1	Write Mode Word
0	0	1	0	0	Read Counter No. 0
0	0	1	0	1	Read Counter No. 1
0	0	1	1	0	Read Counter No. 2
0	0	1	1	1	No-Operation 3-State
1	X	X	X	X	Disable 3-State
0	1	1	X	X	No-Operation 3-State

The three functional (counter Ø; 1 and 2) blocks are identical in operation so only a signle Counter will be described. Each Counter consists of a single, 16-bit, pre-settable, DOWN counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of MODES stored in the Control Word Register.

The Counters are fully independent and each can have separate Mode configuration and counting operation, binary or BCD. Also, there are special features in the control word that handle the loading of the count value so that software overhead can be minimized for these functions.

The reading of the contents of each counter is available to the programmer with simple READ operations for event counting applications and special commands and logic are included in the 8253 so that the contents of each counter can be read "on the fly" without having to inhibit the clock input.

All of the modes for each counter are programmed by the systems software by simple memory operations.

The control word format of the 8253 is as follows :

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

Definition of Control Fields

SC-Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL-Read/Load

RL1	RL0	
0	0	Counter Latching operation (see READ/WRITE Procedure Section)
1	0	Read/Load most significant byte only.
0	1	Read/Load least significant byte only.
1	1	Read/Load least significant byte first, then most significant byte.

M-MODE

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

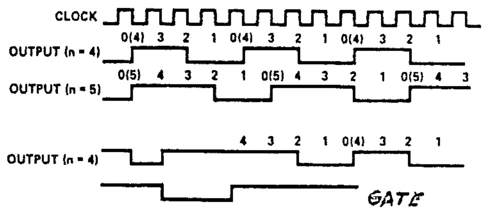
BCD

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

In the initialisation of the sound generator is the Control Word of the 3 timers programmed in Mode 3; a binary counter 16-bits and read/load,least significant byte first; then most significant byte (CW = # 36;# 76; # 136).

So the sound generators are in Mode 3 , this means that the 8253 is programmed as a square wave generator. The 3 gates are high, thus enabling counting and the output will remain high until one half of the count has been completed and go low for the other half of the count. If the counter register is reloaded with a new value during counting, this new value will be reflected immediately after the output transition of the current count.

MODE 3



If the count is odd, the output will be high for (N+1)/2 counts and low for (N-1)/2 counts (see Fig.) The addresses for reading and loading the counters are FCØØ/FCØ2/FCØ4) and the control word can be programmed on address FCØ6.

When the frequencies are programmed (with the sound command) their volume can be changed with an ENVELOPE. This facility is obtained by setting or resetting levels in a resistance network (see sheet 7). These levels can be programmed by making the corresponding bits of IC29 and IC30 (sheet 5/address FDØ4 and FDØ5) high or low.

IC15 (LM324) is used as a buffer (gain=1) and all the sound channels as well as the output of the noise generator are added together before modulation of the audio channel. Channels 1 and 2 and 2 and 3 are added together for left and right stereo output. Noise is also inserted in channels 1 and 3.

A noise generator is included within the sound generation circuitry. It is generated with white noise created in the base/emitter junction of a transistor (T6).

The purpose of this device is to simulate complex sound generation and to provide a time random sequence for random number generation (IC16 - RPI). Random events generated by this circuit provide the basis for information input on IC31 (sheet 5) to generate a true random number.

The Desk Computer is also equiped with circuitry required to connect two game paddles as input devices. Each paddle contains three variable resistors whose positions are read as values and one on-off event (single contact switch address FDØØ, bit 4 and 5).

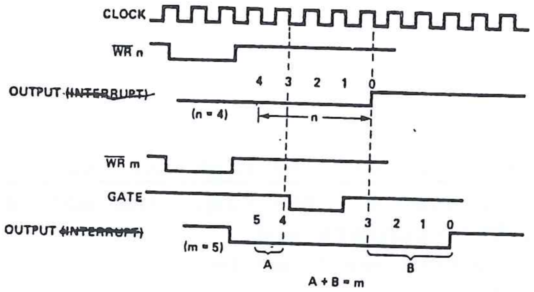
The position of any paddle resistor is found by putting its binary address onto the 3 bits in port FD06 (IC32;sheet 5) and IC10 (multiplexer, sheet 8) will select one paddle.

The channel Ø of the sound generator is programmed in mode Ø (# 3Ø to address FCØ6), this means it is used as an interrupt on terminal count and we can read while counting.

The OUTput will be initially low after the Mode Set operation. After the count is loaded into the selected count register, the OUTput will remain low and the counter will count. When terminal count is reached, the OUTput will go high and remain high until the selected count register is reloaded with the Mode.

The GATE input will enable the counting when high and inhibit counting when low.

MODE 0



The counter Ø is set to FFFF and the read of the positions is triggered by reading location FDØ1

To read the contents of the counter without effecting or disturbing the counting operation the 8253 has special internal logic that can be accessed using simple WR commands to the MODE register. Basically, when you wish to read the contents of a selected counter "on the fly" he loads the MODE register with a special code which latches the present count value into a storage register so that its contents contain an accurate, stable quantity. The programmer then issues a normal read command to the selected counter and the contents of the latched register is available.

MODE Register for Latching Count

AO, A1 = 11

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	X	X	X	X

SC1,SC0 - specify counter to be latched.

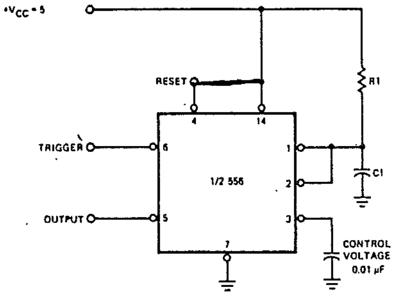
D5,D4 - 00 designates counter latching operation.

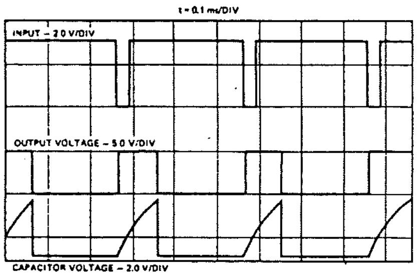
X - don't care.

The enabling of the counters (gate Ø) is connected with the paddle pulses coming from IC7;IC8 or IC9 (sheet 8). These timers (556) are used as one-shot circuits. The external capacitor is initially held discharged by a transistor inside the timer.

When a negative trigger pulse is applied to lead 6, the flip-flop is set, releasing the short circuit across the external capacitor and drives the output HIGH. The voltage across the capacitor increases exponentially with the time constant T = R1C1. When the voltage across the capacitor equals 2/3 VCC, the comparator resets the flip-flop which then discharges the capacitor rapidly and drives the output to its LOW state.

The circuit triggers on a negative-going input signal when the level reaches 1/3 VCC. Once triggered, the circuit remains in this state until the set time has elapsed, even if it is triggered again during this interval.





R1 = 9.1 kΩ, C1 = 0.01 µF, R_L = 1.0 kΩ

When the paddle count is done the value is mapped onto an 8-bit range (0-255) for a result.

4) CASSETTE INTERFACE

The Desk Computer contains the entire logic and interface circuits needed to connect a low cost adudio cassette for the input and output of data and programs.

The Computer input from the cassette should be made via the crystal ear phone outlet or the external speaker outlet. In these cassettes that have no such outputs simply connect the speaker wires to the Computer input.

The data to the cassette is outputted by IC32 (address #FDØ6 bit Ø) and is level adjusted with R11/R13 (output +1V p.p.) CH2 and C19 are placed to make the signal more like a sinus.

The way a program is put on a cassette is as follows :

1) Start cassette motor
2) Save program
3) Stop motor

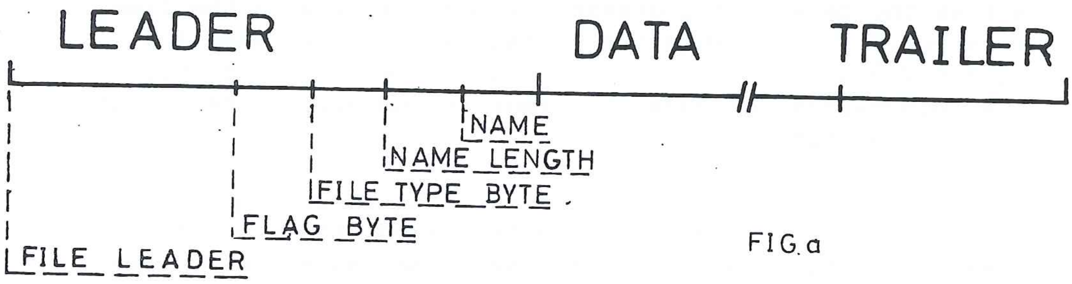


FIG.a

Before and after the data comes a leader and a trailer on the tape (see Fig.a) The whole cassette interface is build up under software control.

The pulse lenght of the leader can be found on address # Ø2E6 (# 2424) so the duty cycle is 0,5.

To end the file leader there is written a "1" bit (see Fig. b) . The write file type byte is Ø (for BASIC programs). The two last items written in the leader are the length of the name and the name itself.

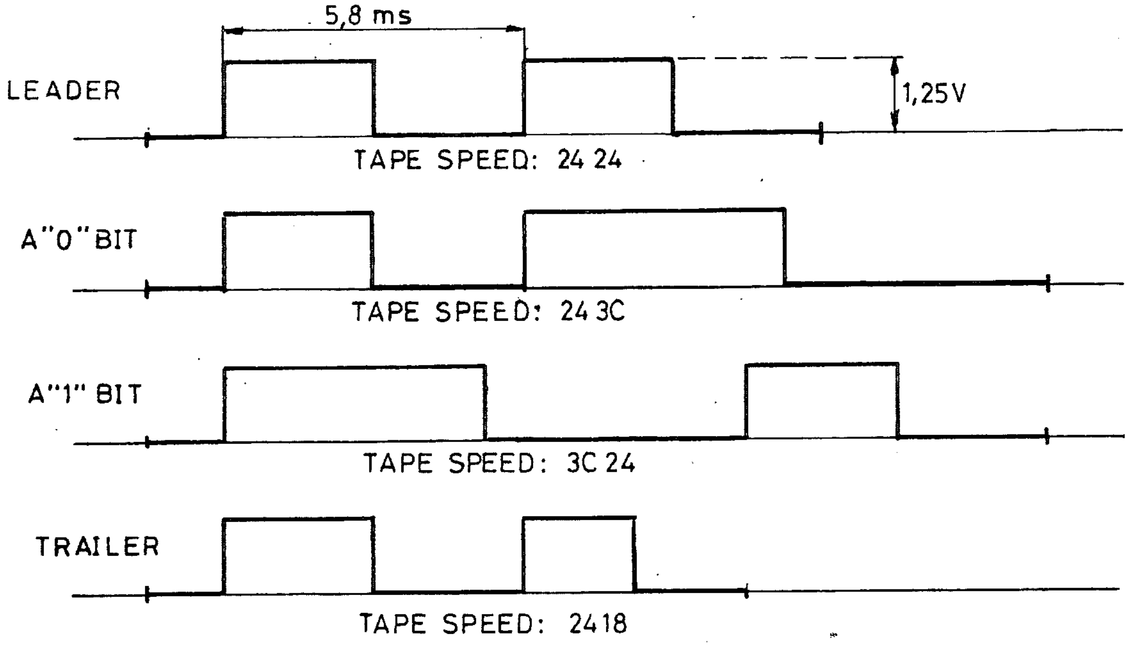
The data consists of the block length a checksum on the length and the block of data itself. The data is closed with the length and the contents of the symbol table and to end the saving there is written a trailer. This consist of a number of zero's written with a different speed.

The input signal from the cassette (address FØØØ bit 7) comes through an OPAMP circuit (IC14) so the signal is level matched to the computer.

The load routine clears the heap and all variables. It evaluates the name of the program and looks for the required file type. When a file has been found, the text buffer and symboltable are loaded and the pointers will be updated. When loading during program run, the program continous with the program just loaded.

So the load routine is build up as follows :

* Switch on cassette motor
* Read header and name
* Load textbuffer
* Load symboltable
* Stop cassette motor
* Enable interrupts

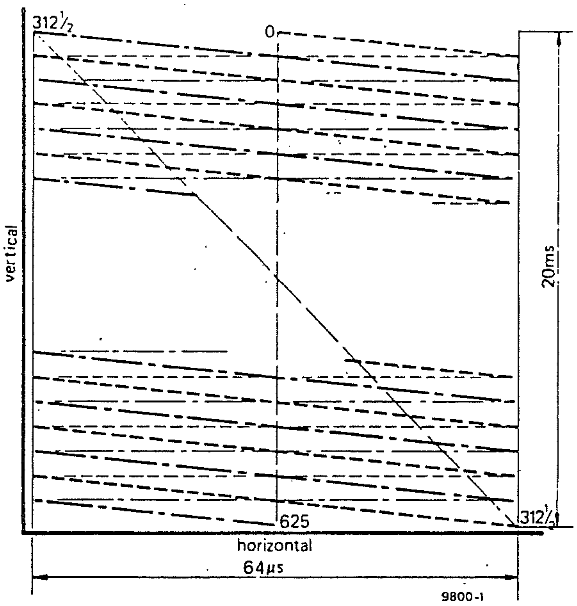


FI G. b

CHAPTER VI   PAL & RGB CARD

The television interface is realised such that a separate adaptor module plugs into the fundamental logic to realise a normal standard PAL or RGB signal.

A picture of the TV screen is defined by number of scans (625 for Europe). They are not written one after another but first one raster (312,5 scans) and then the other half. There are written 25 pictures (of 625 scans) per sec., but the eye has a certain slowness so you can see a steady picture.



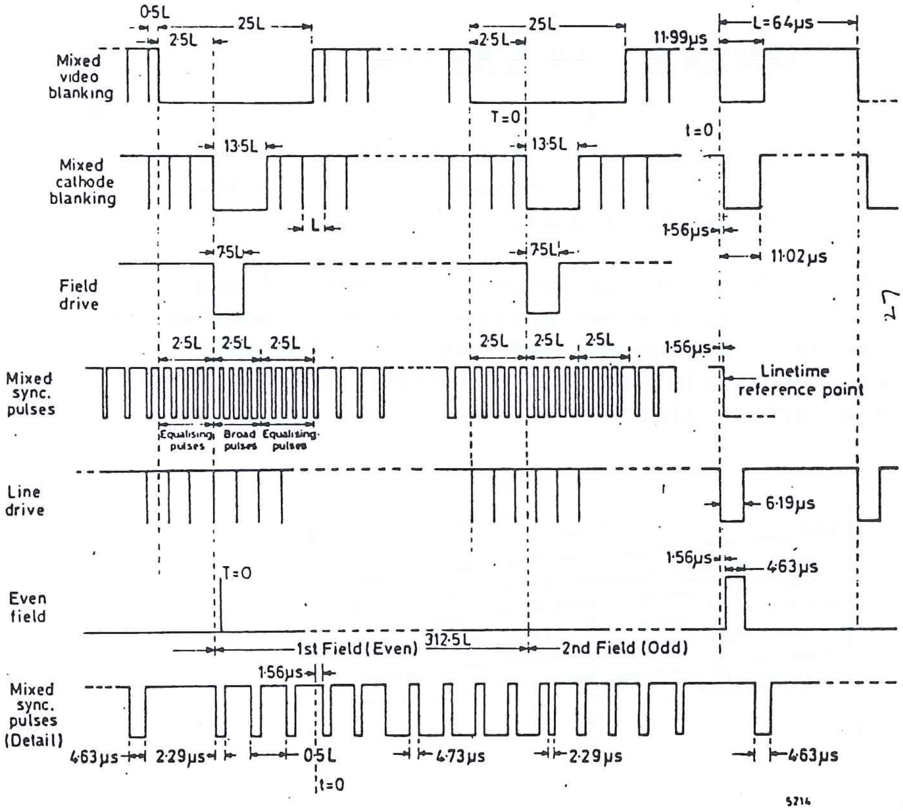
One scan is written in 64µs and the scansynchronisation pulse is 5,8µs. This pulse is necessary to keep the movement of the scan in the TV synchronically with the signal in the computer. It is obvious that the elektronbeam must be blanked when going from right to left of the screen.

There is also a rastersynchronisation pulse which is used every time a picture is written. The equilising pulses are needed to get a good synchronisation of the picture during the writing of the even and od fields.

All the signals are generated by IC25 (ZNA134J;sheet 4) and are necessary to form a picture on the TV set.

OUTPUT WAVEFORMS   IC25

625 line CCIR standard output
Crystal frequency = 2.5625 MHz
Line frequency   = 15.625 kHz, Field frequency = 50 Hz
Line period   = 64 us, Field period = 20 ms



1-7

All the signals, synchronisation, color, information and supply voltages are available at connector 1 on the PC Board.

The Pin description is as follows and the signals are shown in Fig. 1-7.

2	GROUND	1	+12V
4	Mixes cathode blanking	3	-5V
6	-	5	SOUND
8	Colour signal 3	7	Colour signal Ø
10	Colour signal 2	9	Colour Signal 1
12	+5V	11	Clock
14	Page	13	Page
16	Even field	15	Videoblanking (VBL)
18	Sync	17	Line
20	Line	19	VBL

To explain the sheme of the PAL colour card there is made a picture with colour bars on the screen (16 colours from black to white) in mode 3.

Connector 1

10µS/div. 2V/div

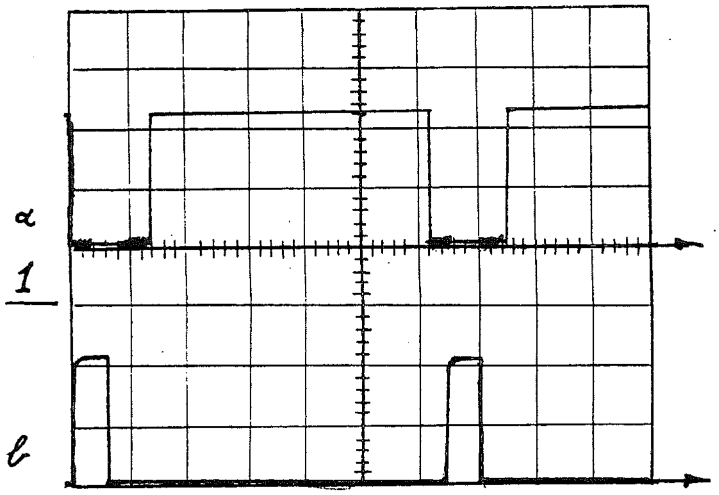


Fig 1

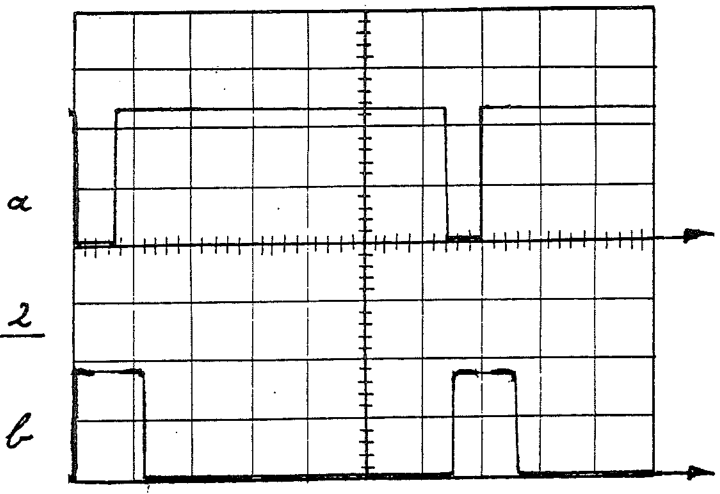


Fig 2

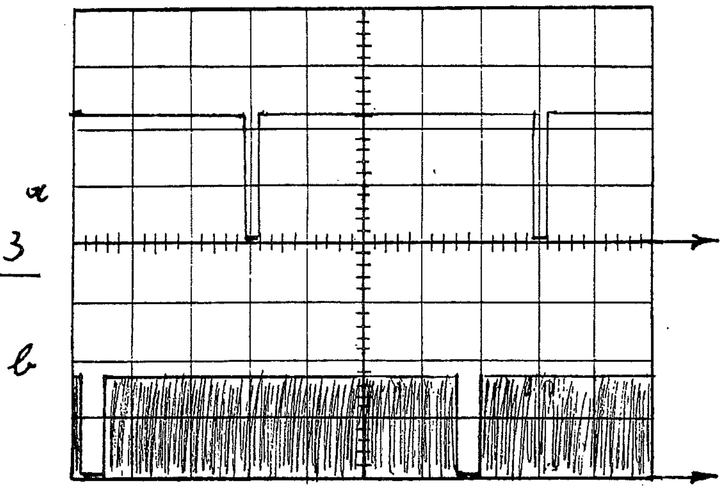


Fig 3

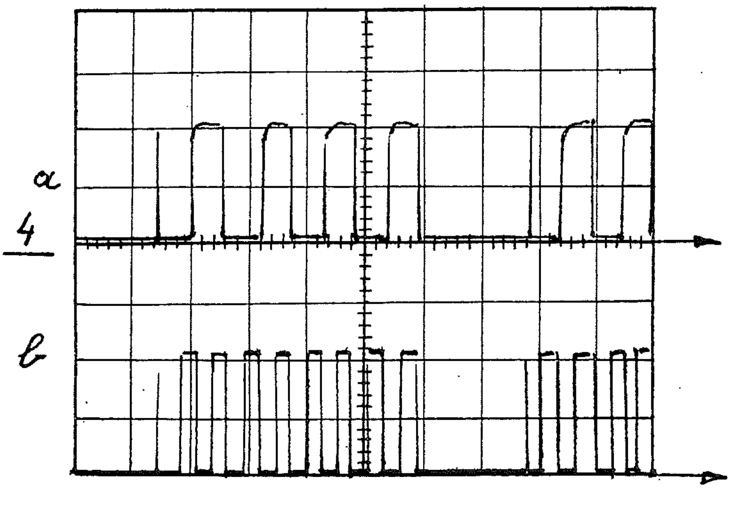


Fig 4

Connector 1

10 µS/div 2V/div

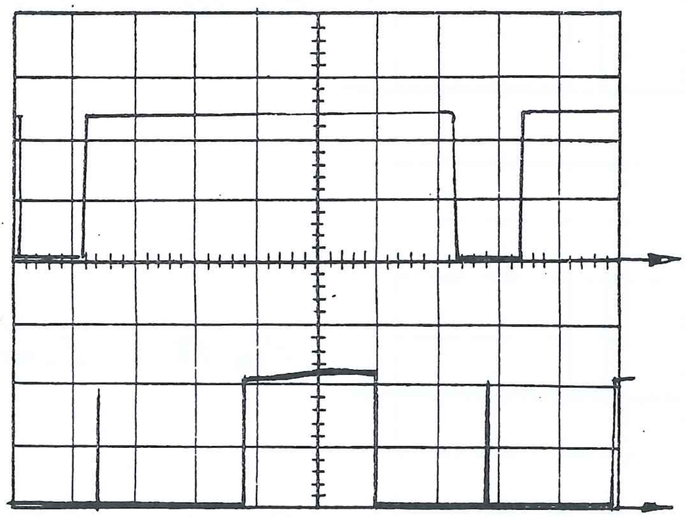


Fig 5

Pin 4 Mixed cathode blanking

Pin 8 KL 3

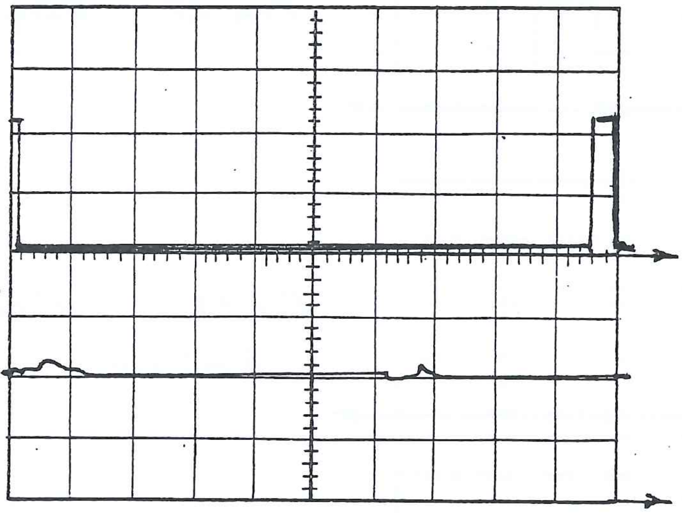


Fig 6

Pin 14 $\overline{\text{Page}}$
2 mS/div

Pin 16 Even Field

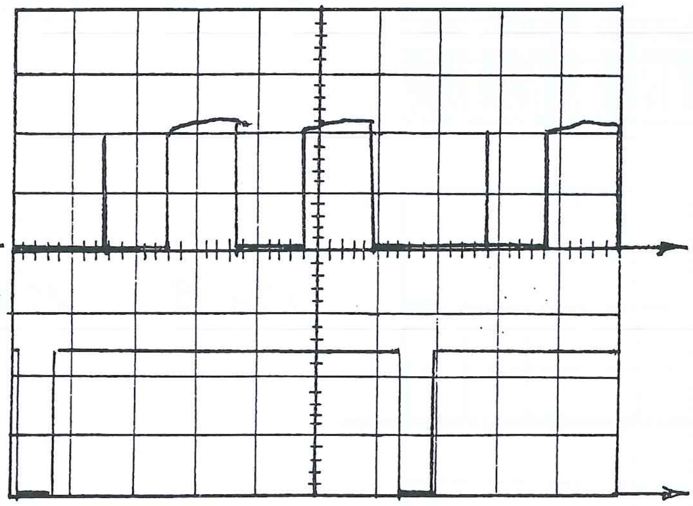


Fig 7

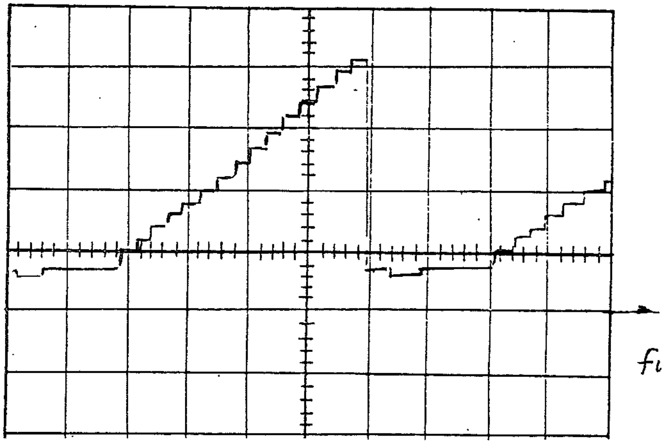
Pin 10 KL2

Pin 18 Sync

A) THE LUMINANCE SIGNAL (Y-signal)

The 4 colour signals are synchronized via IC7 to IC10 (74LS164) with the clock (CKD) and the Video blanking signal (VBL). These IC's are used to have the same delay time as the color signals.

These signals are added via a resisternetwork (RP2; R20 - R23; with a different balance) to form the Y-signal (in this example a step up signal , see Fig. 8).



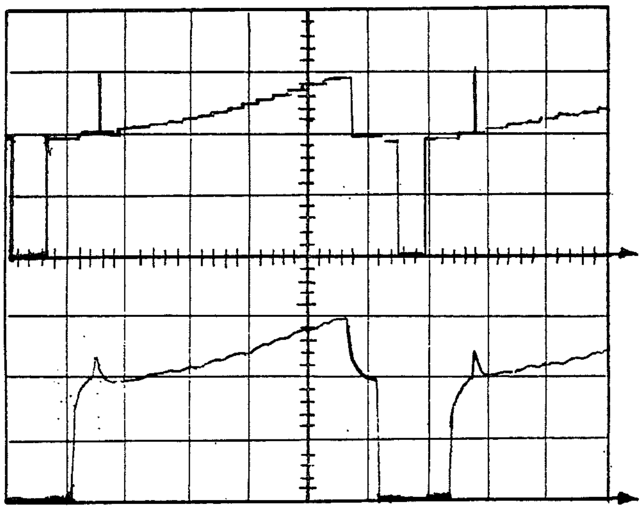
CH3

2V/div

10µs/div

fig.8

The Blanking signal (is also responsible for the front and end porch) and the sync signal (adjustable with PT2) are added together with the Y-signal (Fig.9)



IC2 Pin 3 1V/div.

Fig 9

IC2 Pin 6 1V/div.

The emitter follower (T1) or buffer (impedance match) to IC1 (LM 1889) pin 12 (Fig. 10).

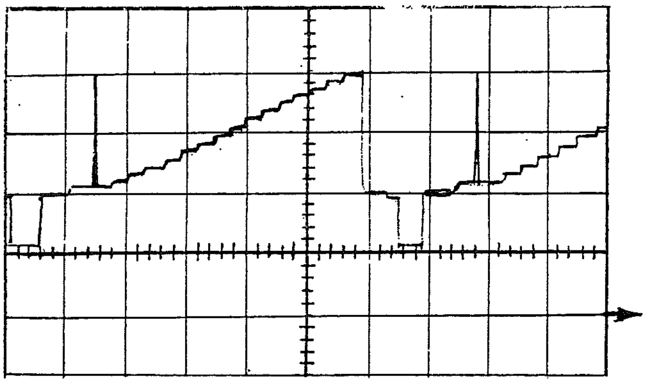


Fig 10

IC1 pin 12

50mV/div.

B) THE COLOR SIGNALS (R-Y and B-Y signals)

The 4 information signals (K0L-K3L) are also connected with IC5 (PROM 74S288) which is selected by the video blanking signal. The 3 state outputs of IC5 are high or low when VBL is actif, because RP1;R36 and R41 are used as pull up or pull down resistors. These signals are shown in Fig. 11; 12 and 13.

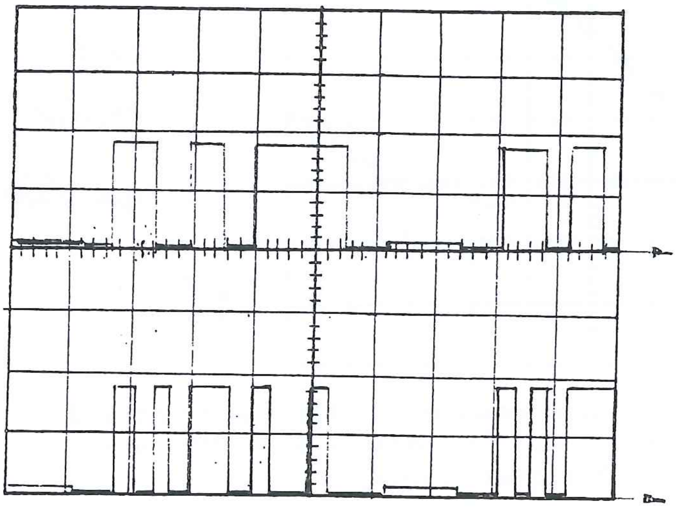


Fig 11

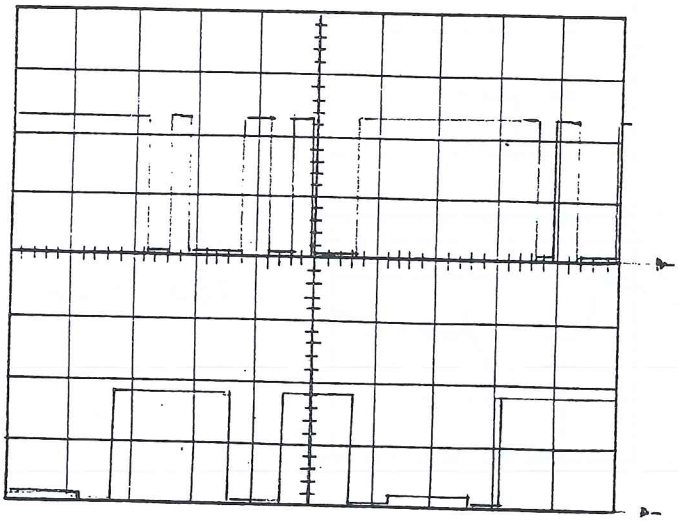


Fig 12

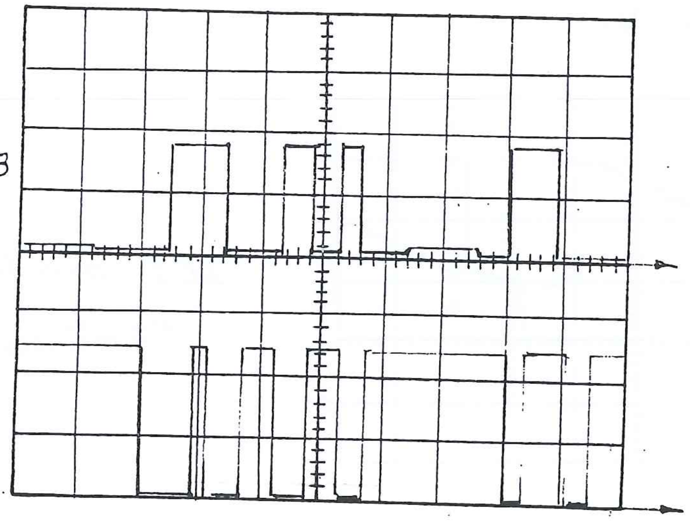
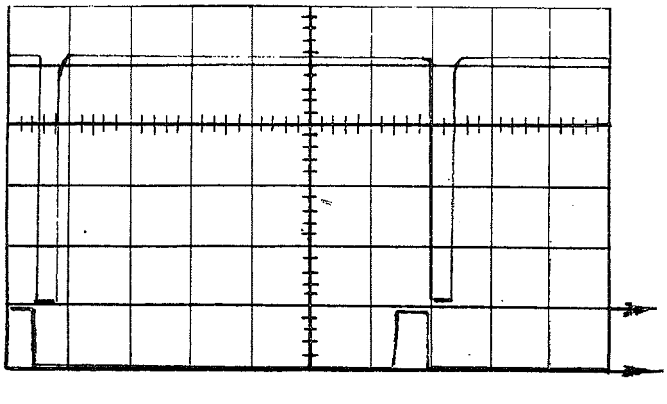


Fig 13



IC2 pin 13
IC6 pin 12
1V/div.

Line Con 1
pin 20

fig. 14a.

With IC3 (74LS374) these signals from the PROM are synchronised with the clock. The R-Y and B-Y signals are formed by the resistornetwork R26-R30; R39 and R40.

The place of the burstsignal is a delay from the sync-signal through two monostabel multivibrators. The signal is replaced in time and in pulswidth so it comes on the back porch of the video signal (Fig. 14).

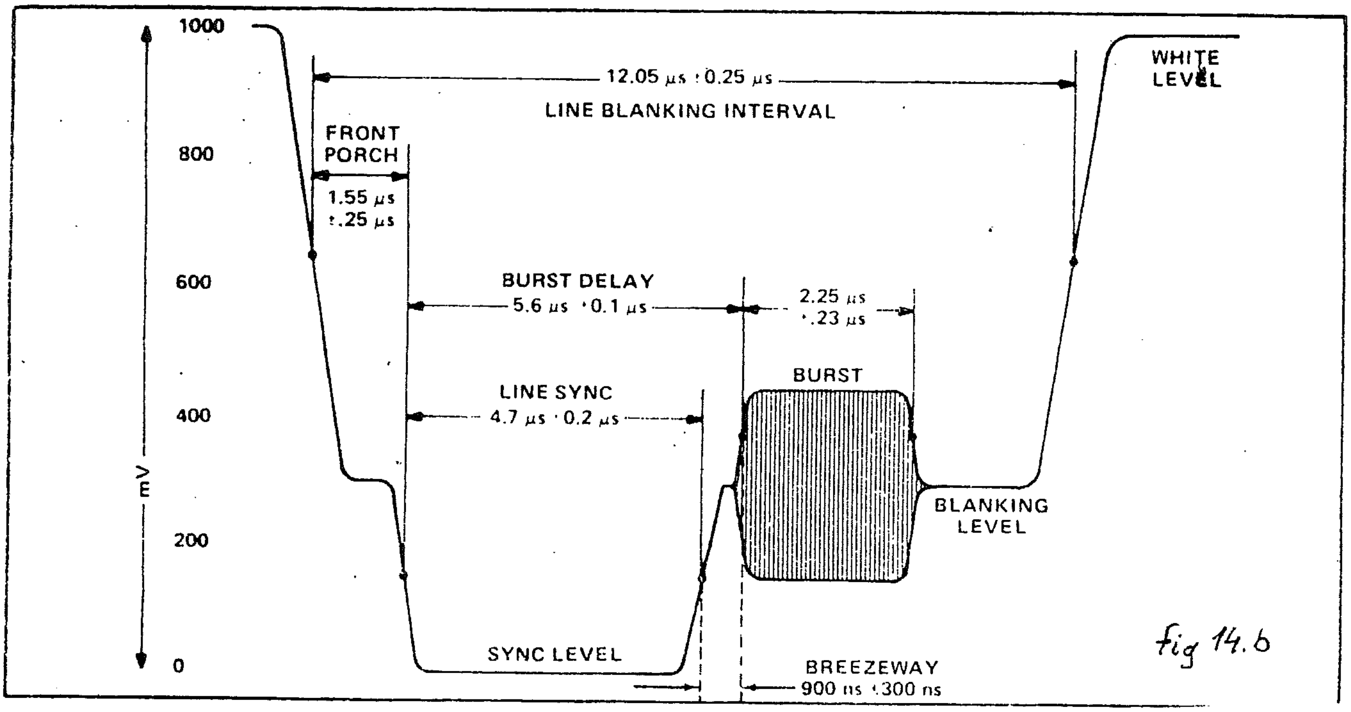
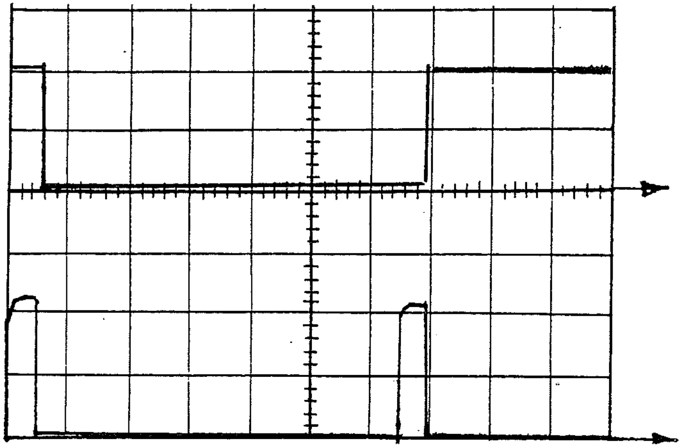


fig 14.b

So the place of the burst is a delayed signal of the sync-puls. The switching of the burst signal in the V-signal is done via a D-FF (IC4 used as toggle-FF- which is connected with the line signal. So IC2 is used as PAL switch (Fiq. 15)



IC2 Pin 12

Line

fig.15

C) THE VIDEO SIGNAL

IC1 (LM1889) takes care to make the video signal. The following signals have to be connected to let the IC work properly : Pin 1; 17 and 18. Clocksignal (4,43 MHZ to form the Ku and the Kv signal). The chrominance signal K can be measured on pin 13 (Fig. 16 and 17).

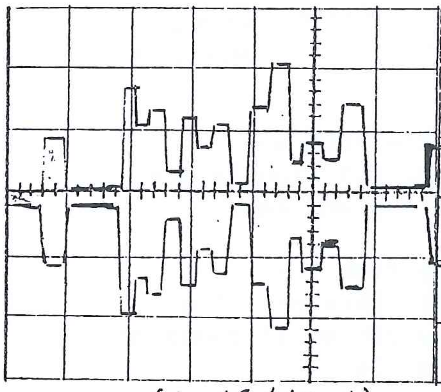


fig 16 (Lijn 1)

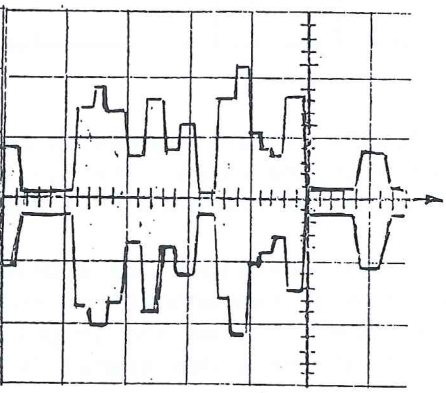
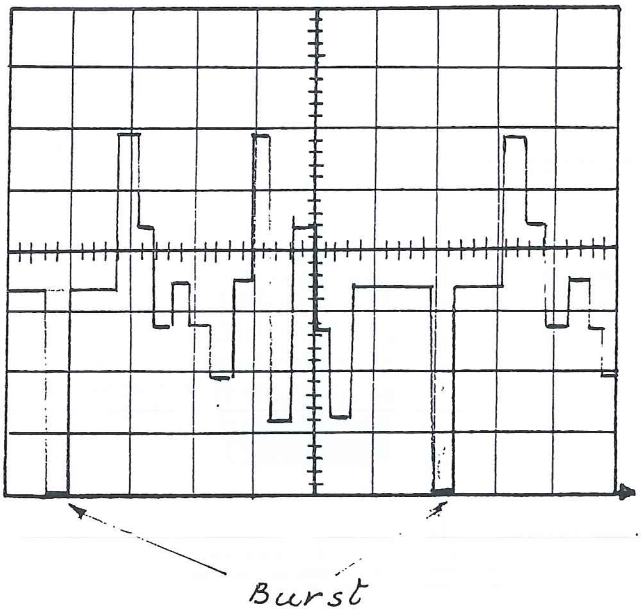


Fig 17 (Lijn 2)

IC 1 Pin 13

Pin 4 : B-Y signal (Fig. 18)

fig18

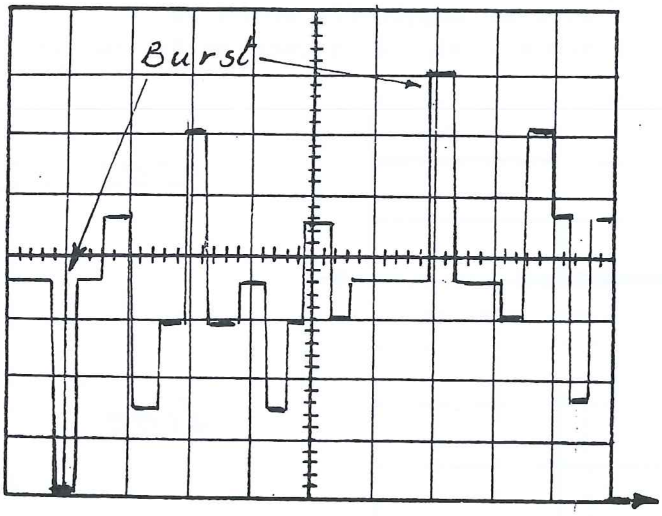


(B-Y) measured on pin 8 IC 2

50 mV/div.

Pin 2 : R-Y signal (Fig. 19)

Fig 19

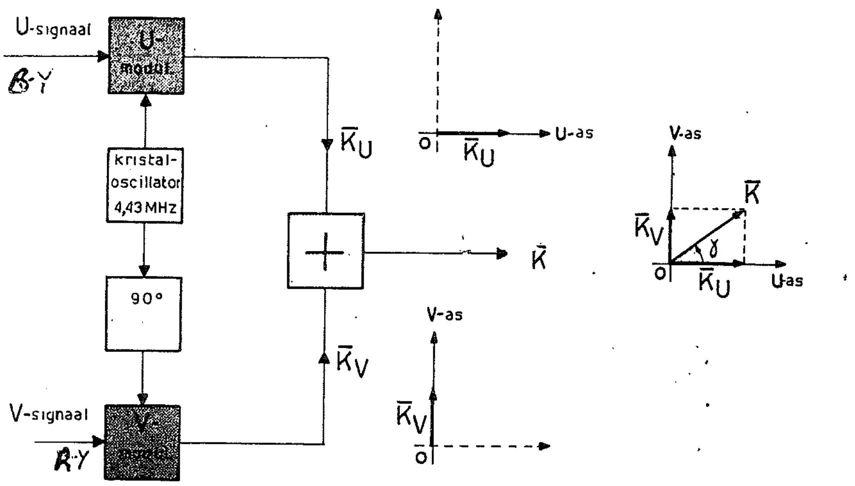


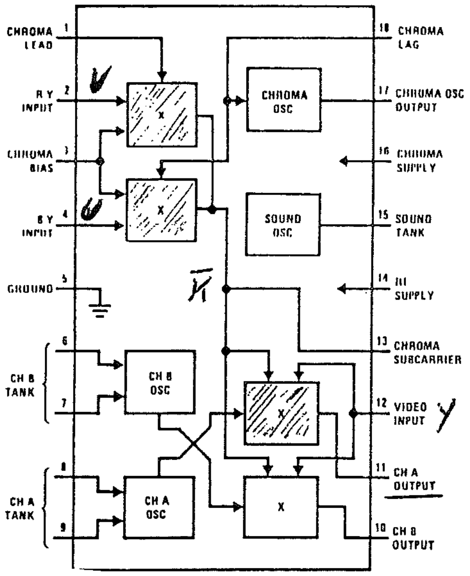
(R-Y) gemeten op pin 11 IC 2

50 mV/div.

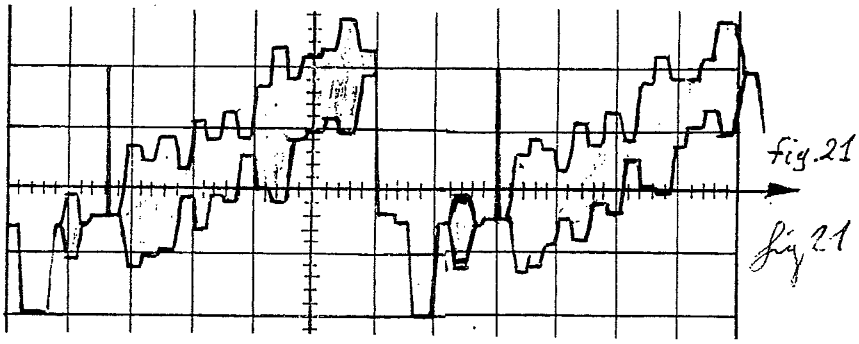
On the PAL card there are only 3 blocks of the LM1889 (Fig. 20) used :.

The 2 U and V modulators and the modulation of the crominance and the luminance signal.

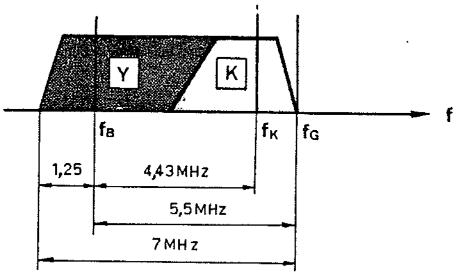




The DC level of the Video signal (which can be measured on pin 11 (Fig. 21) can be adjusted with PT1.



The UHF modulator (channel 36) M1 modulates the signal so the computer can be connected to a normal TV set. M1 is also equiped with a sub-carrier for the sound so the TV signal can be seen (in frequencyspectrum) as follows :



fb carrier of the picture
fH carrier of chroninancesignal
fG carrier of the sound

The RGB card is build up in the same way as the PAL color card and except that we have already the RGB signals after the Resister network. The PROM is programmed in a different way to the PAL PROM so when you draw the sixteen color bars on the screen you will see there is a difference in colours. The transistors T11 ,T2, T3 and T4 are used to match the signals to a normal colour monitor.

CHAPTER VII TESTING THE D.C.

CHECK OUT PROCEDURE

1) Plug out IC73 (511)

- Put on EPROM test card V-2 on X-BUS with UTILITY PROM or stack TEST PROM and RAM cycle PROM

- Connect terminal to the RS232 bus

  * First check stack,if OK program starts automatically after reset
  * Put in UTILITY PROM
  * Memory commands : these commands enable to trace a program while its running. You can also display blocks of memory and insert test programs
  * Register commands : these commands afford the facility to examine and modify the 8080 registers and the vector and intilisation bytes

no UTILITY :

check : Clock IC1 (pin 12)
DØ-D7
Ø1-Ø2 (pin 2Ø, 5501)
Ready high (R128 top)
Reset pin 1 & 2 (8224) and pin 12/8080
Serial outpin 40/5501 must be high and have low going pulses when reset
If not ready see if there is RAM action : check blue,red,green and yellow PROM.

RAM test GC800 (second PROM on test card) test automatically dynamic RAM for reading and writing. When all RAM's OK all zero's on the terminal.

	Bank	Bit	0	1	2	3	4	5	6	7	
	A		00	00	00	00	00	00	00	00	address 0-3FFF
even bank	B		00	00	00	00	00	00	00	00	))address 4000-BFFF
odd bank	C		00	00	00	00	00	00	00	00	)

When RAM Problems : type in Test Program for continuous writing ar reading (in UT)

```
writing : SF800   21   00   3E  55   77   C3   03F8

reading : SF800   21   00   3E  55   7E   C3   03F8
```

2) If all working plug in 511 and start in basic.

When no BASIC : check Page and stack overflow interrupt.
If OK check Bank Switch (Back to UTILITY)

BANK SWITCHING CHECK

Requirements

1) Running Utility
2) ROMS S14 and S12 in place

```
SFD06   FF-00 (if not FF check all data lines,high on RESET held on
DE000   E00F
E000    D6 FF  00  C4  DA  CE  5F  7B  AE  A2  BA  C8  CD  A5  D6  D2


SFD06   FF-40
DE000   E00F
E000    C3 AA  ED  C3  B4  ED  C3  FE  E0  C3  08  E1  C3  12  E1  C3


SFD06   FF-80
DE000   E00F
E000    C3C3   E0  C3  02  E1  C3  37  E2  C3  79  E2  C3  CC  E2  C3


SFD06   FF-C0
DE000   E00F
E000    C3 24  E0  C3  2A  E7  C3  45  E1  C3  74  EA  C3  90  EF  C3
```

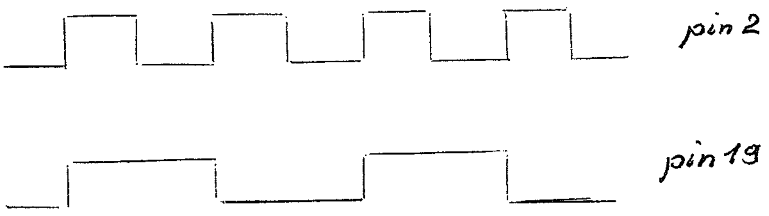
If the above is not correct check bank switching bits, ROM insertion or data or address open circuits.

BANK SWITCHING CHECK WITH SCOPE :-

```
SF800 21  06  FD  36  C0  3A  00  E0  3680  3A  00  E0  36  40  3A

          00  E0  36  00  3A  00  E0  C3  03  F8
```

Then GF800 and check IC32 (pin 2 and 19 for toggle)



if not see Red PROM pin 15.

3) if BASIC OK test Picture Timing

GENERAL TIMING CHECK

Requirements :

1) Running Utility
2) Reasonable dynamic RAM status

1) Sync/scope on $\overline{\text{LINE}}$ at colour card connector
Second probe at pin 1 of IC 105

2) Fill RAM with #00
There is one positive pulse per line of about 220 n sec.

3) Fill RAM with #40
There are two pulses about 2 µ sec apart, followed by 11 pulses about 4 u sec apart, per line

4) Fill RAM with #50
There are 24 pulses about 2 µ sec apart

5) Fill RAM with #60
There are two pulses about 2 µ sec apart, followed by 43 pulses about 1 u sec apart

6) Fill RAM with #70
There are two pulses about 2 µ sec apart, followed by 65 pulses about 1,5 u sec apart

If the above are not correct check through from IC48, IC12, IC5, IC4, IC6, IC11, IC2, IC17, IC19

P.C. PICTURE ADDRESS TEST

Use the following program to check the correct addressing of memory for picture display :

```
F   F800  F87F0        fill ram  with zero

S   F800  21 FF BF     LXI  H, 0BFFFH
          3E 7A        MVI  A, 07AH
          77           MOV  M, A
```

```
              2B            DCX   H
              AF            XRA   A

S   F810      3C            INR   A

S   F84A      C2 10 F8      JNZ   LOOP 1
              C3 03 F8      JMP   LOOP 2
```

To run the program fill dynamic ram with zero. This blanks the screen. On running the above program the screen must evenly fill with Z characters. Any departure indicates an addressing problem.

RAM TIMING

The following program will provide a RAM access pattern synchronised to LINE. Waveforms will be similar to the published photographs.

```
F800        11 00 FF        LXZ     D,0FF00H  Data Pattern

F803        21 F0 03        LXZ     H,03F0 H  Address

F804        72              MOV     M,D

F805        7E              MOV     A,M

F806        73              MOV     M,E

F807        7E              MOV     A,M

F808        72              MOV     M,D

F809        7E              MOV     A,M

F80A        73              MOV     M,E

F80B        7E              MOV     A,M

F80C        00              NOT

F80D        00              NOT

F80E        C3 06  F8       JMP     0F806H
```

To run the above program

```
Z3
F1000-BFFF 0
GF800
```

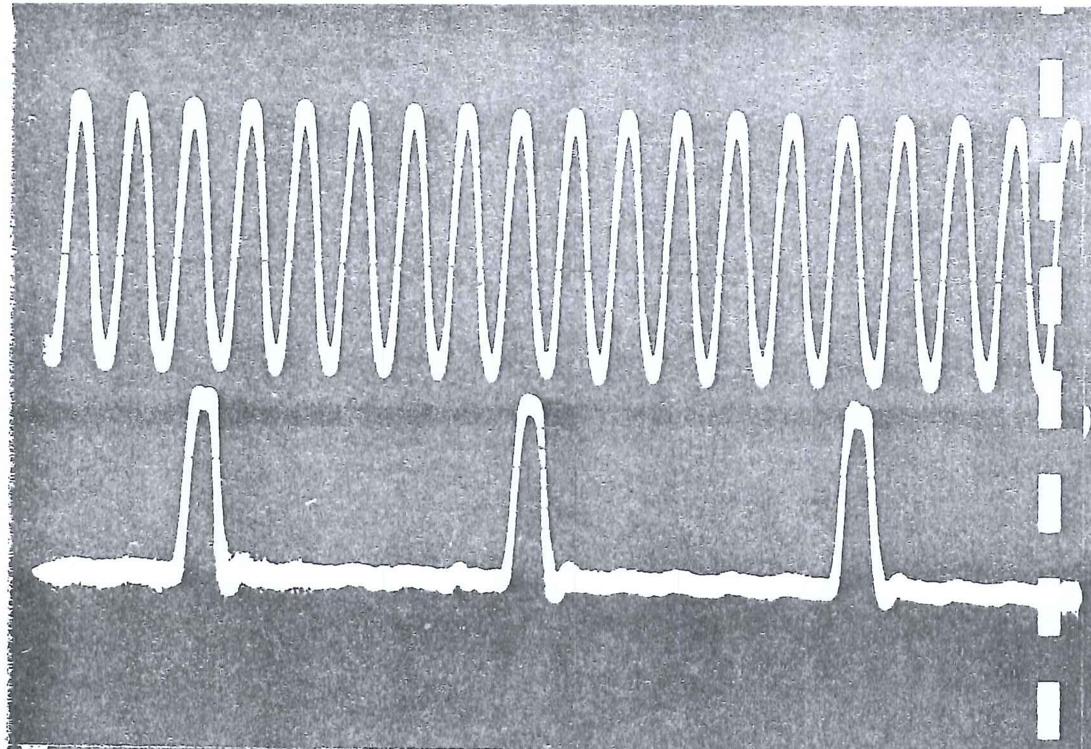
Sync on $\overline{\text{LINE}}$

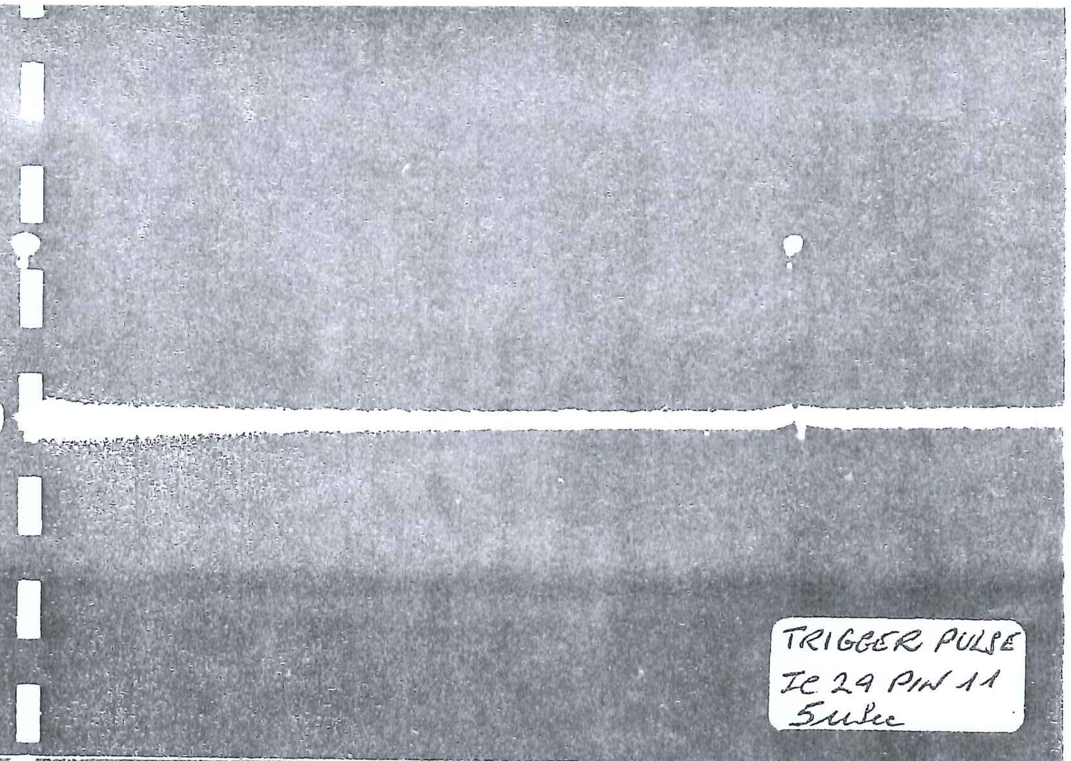
* Waveforms for the DC under RAM test conditions

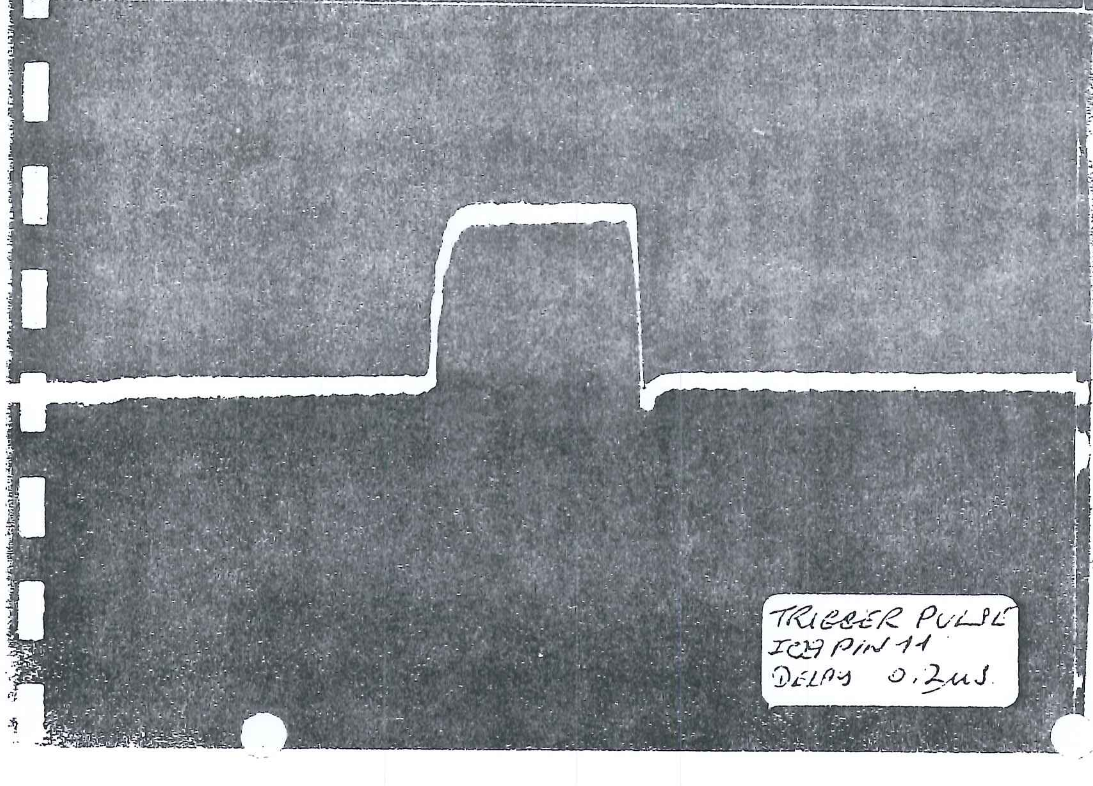
Pin 8 of IC21 to 6ND

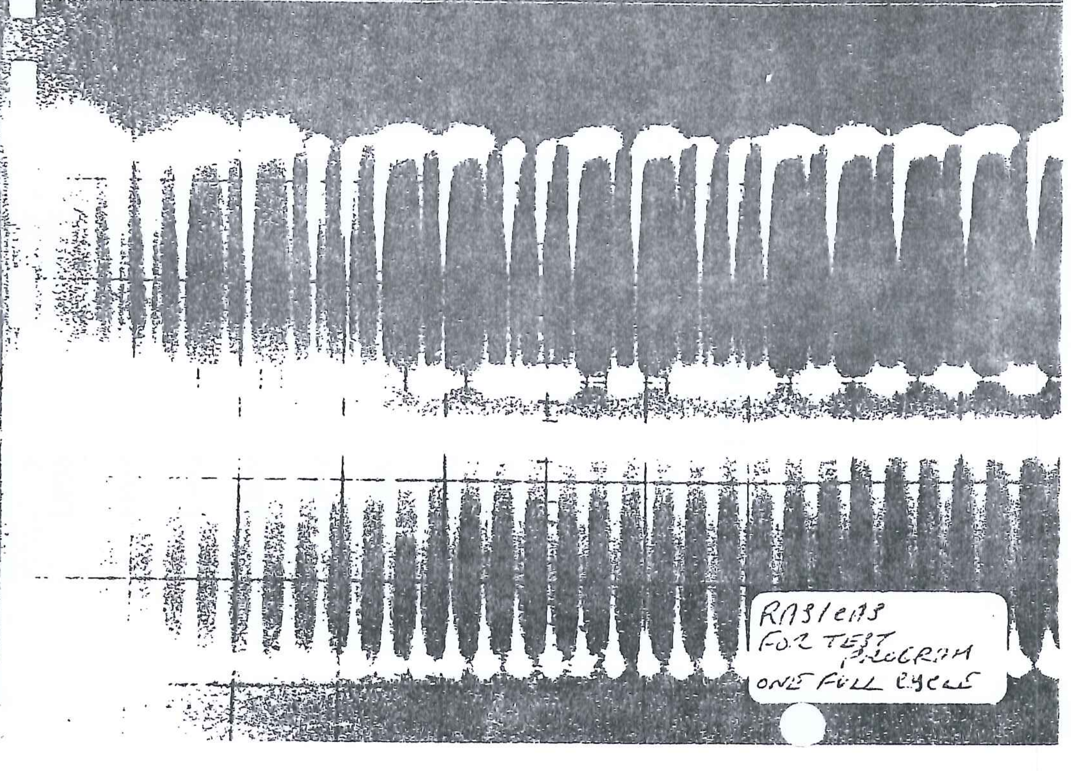
Pin 12 of IC 5 to 6ND

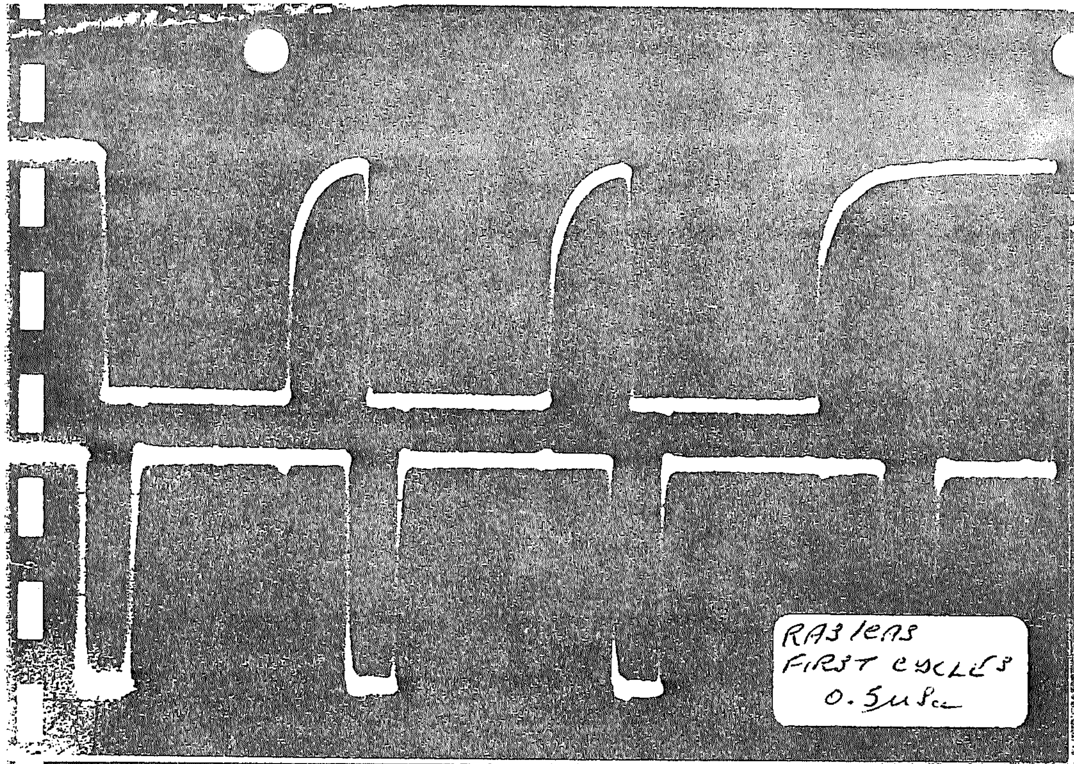
20 MH clock USED for CPU

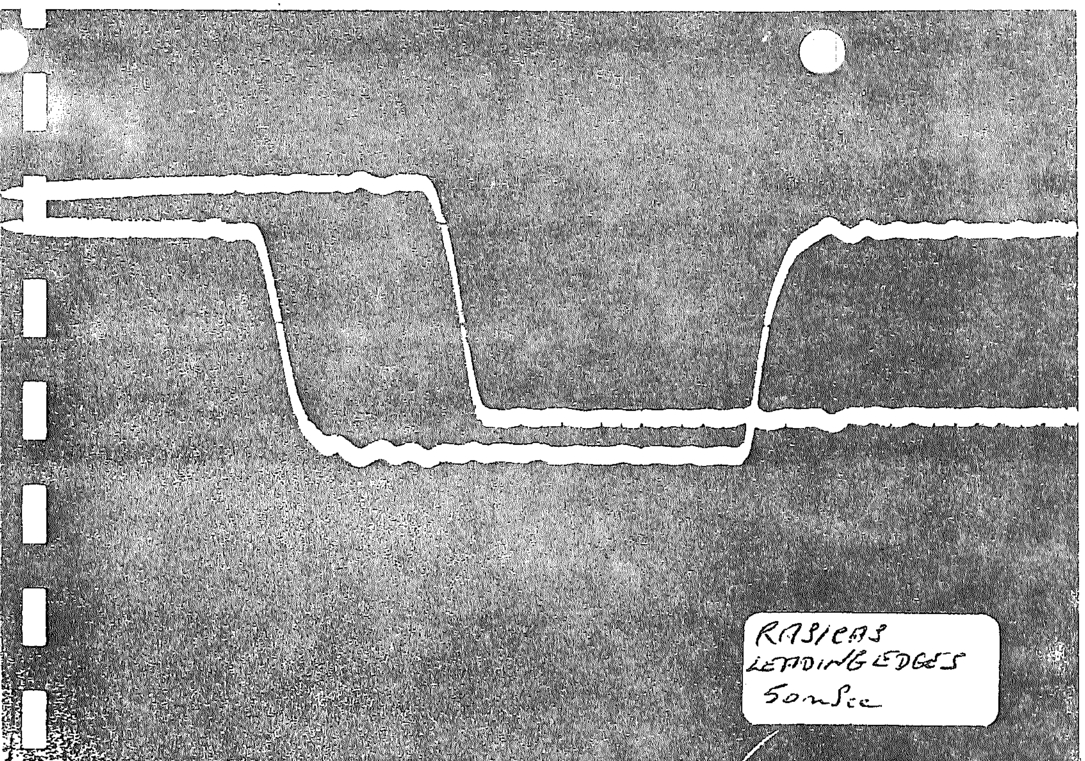


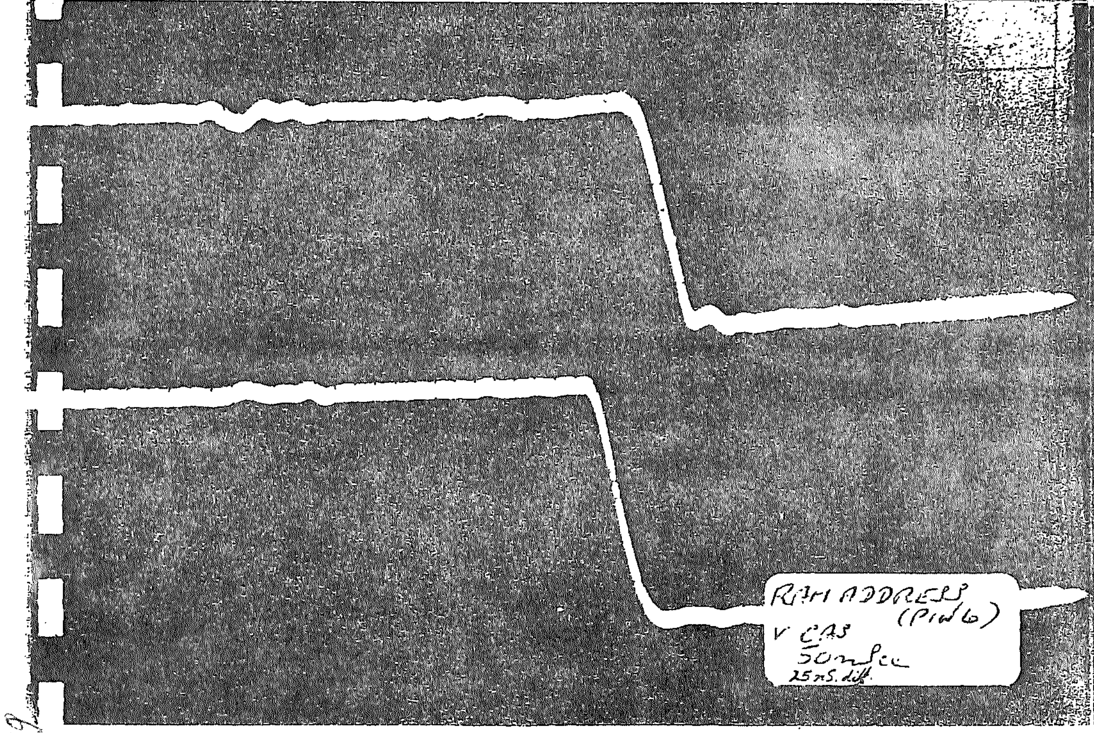

TRIGGER PULSE
IC 29 PIN 11
5uSec

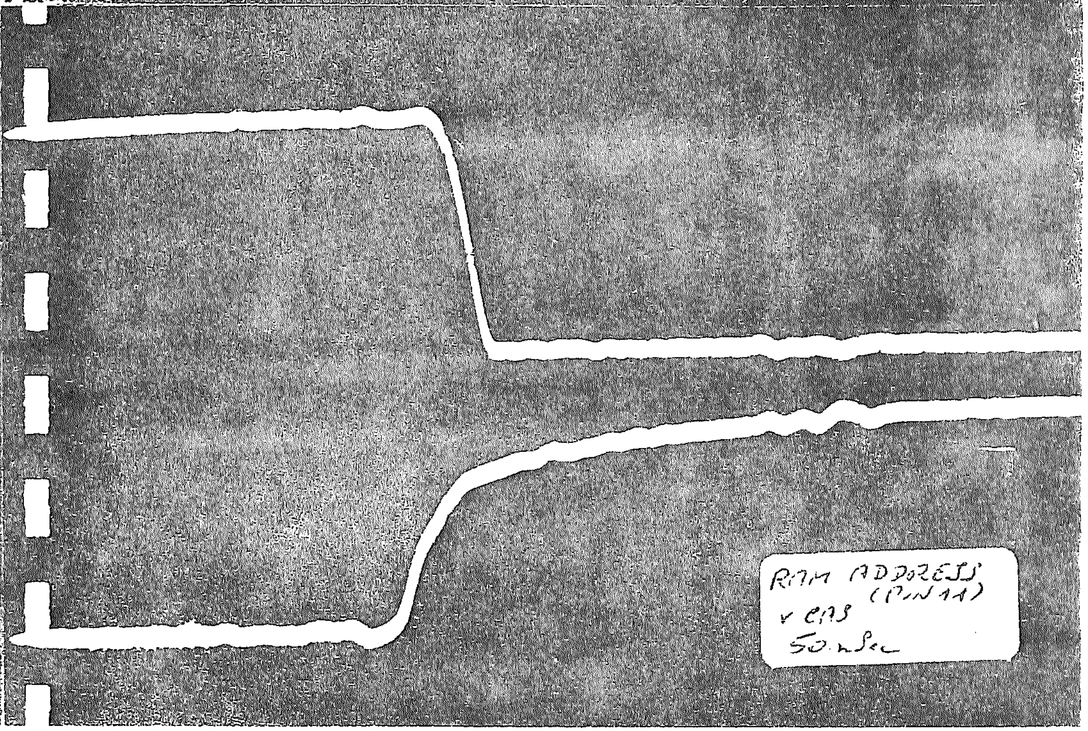

TRIGGER PULSE
IC29 PIN 11
DELAY 0.2uS.

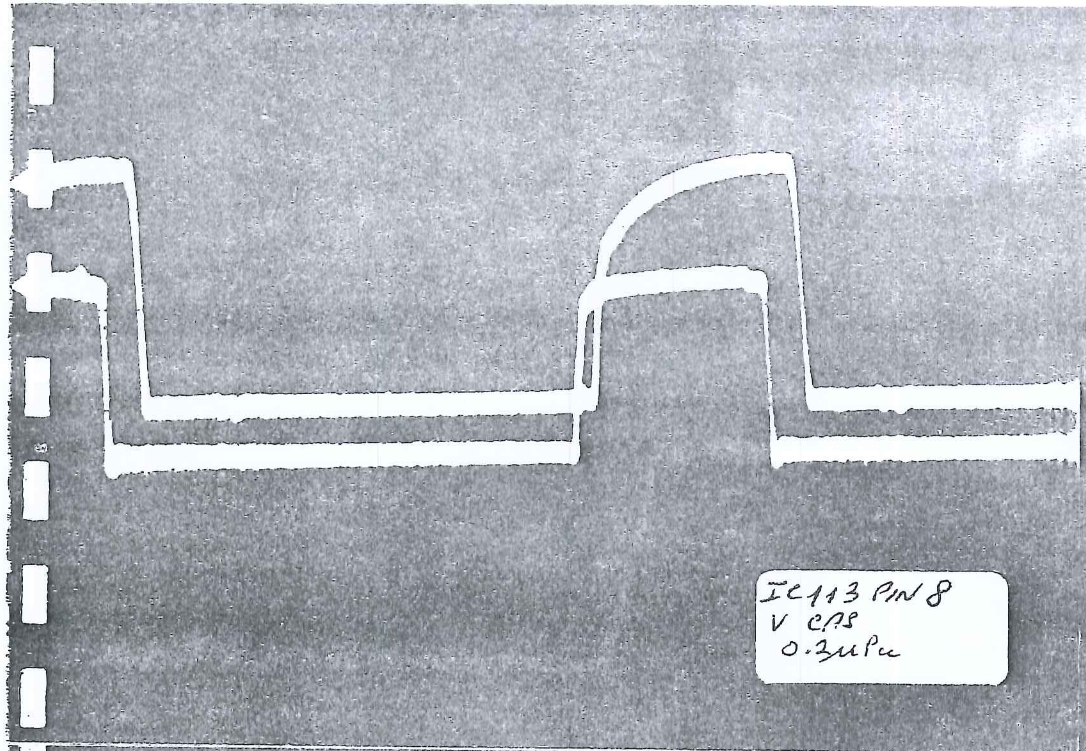

RAS/CAS
FOR TEST PROGRAM
ONE FULL CYCLE


RAS/CAS
FIRST CYCLES
0.5uSec

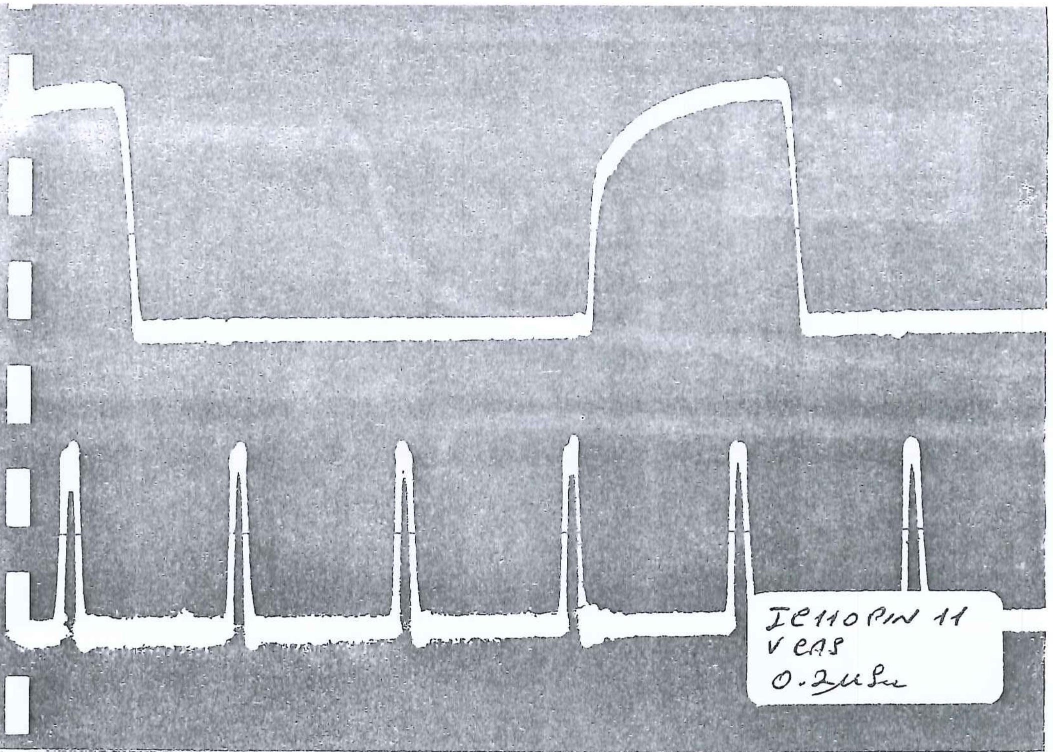

RAS/CAS
LEADING EDGES
50nSec


RAM ADDRESS
(PIN6)
V CAS
50nSec
25nS. diff.

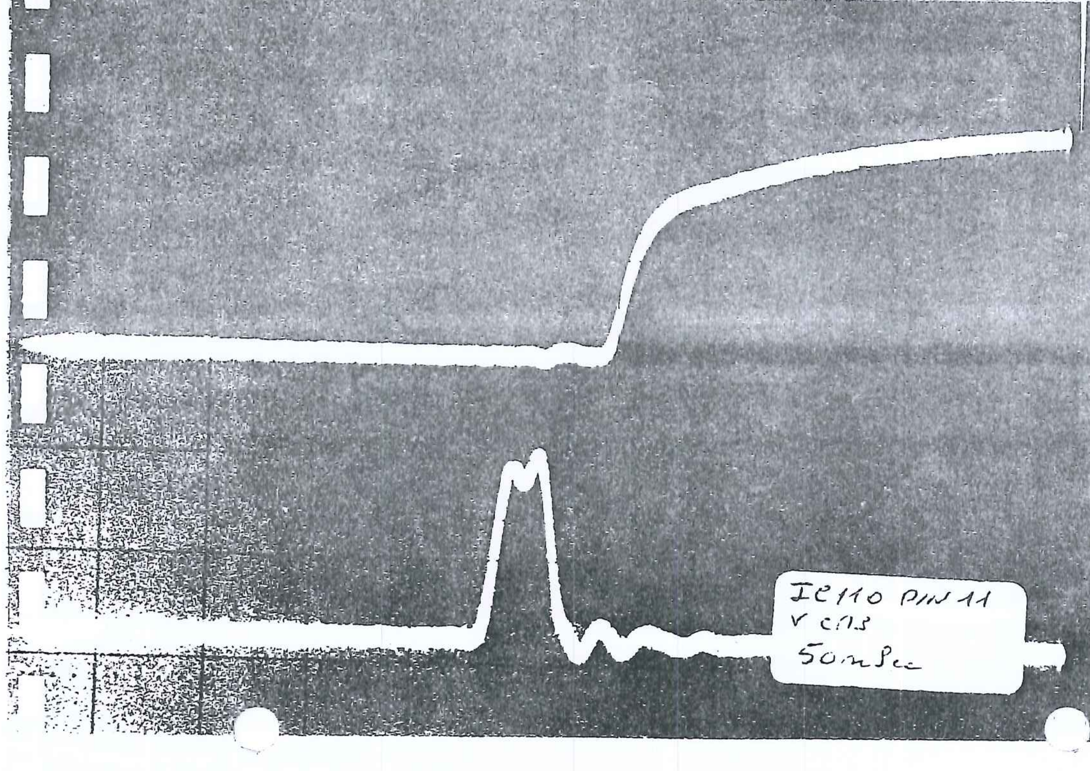

RAM ADDRESS
(PIN11)
V CAS
50nSec



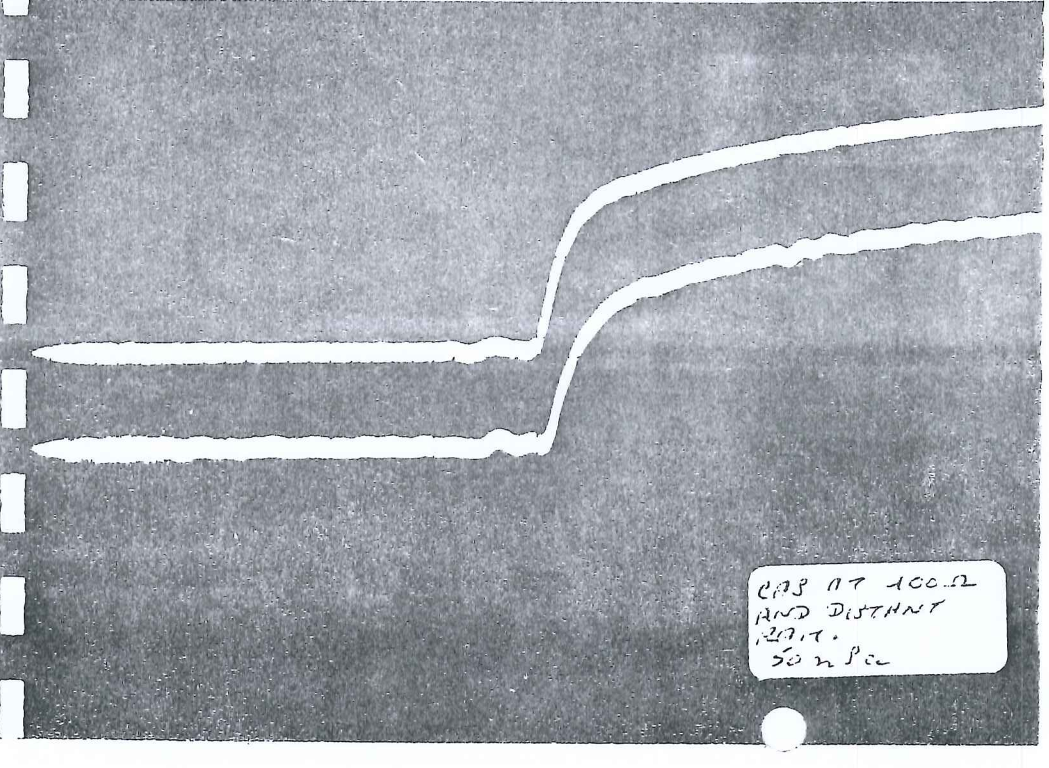
IC113 PIN 8
V CAS
0.2µSec



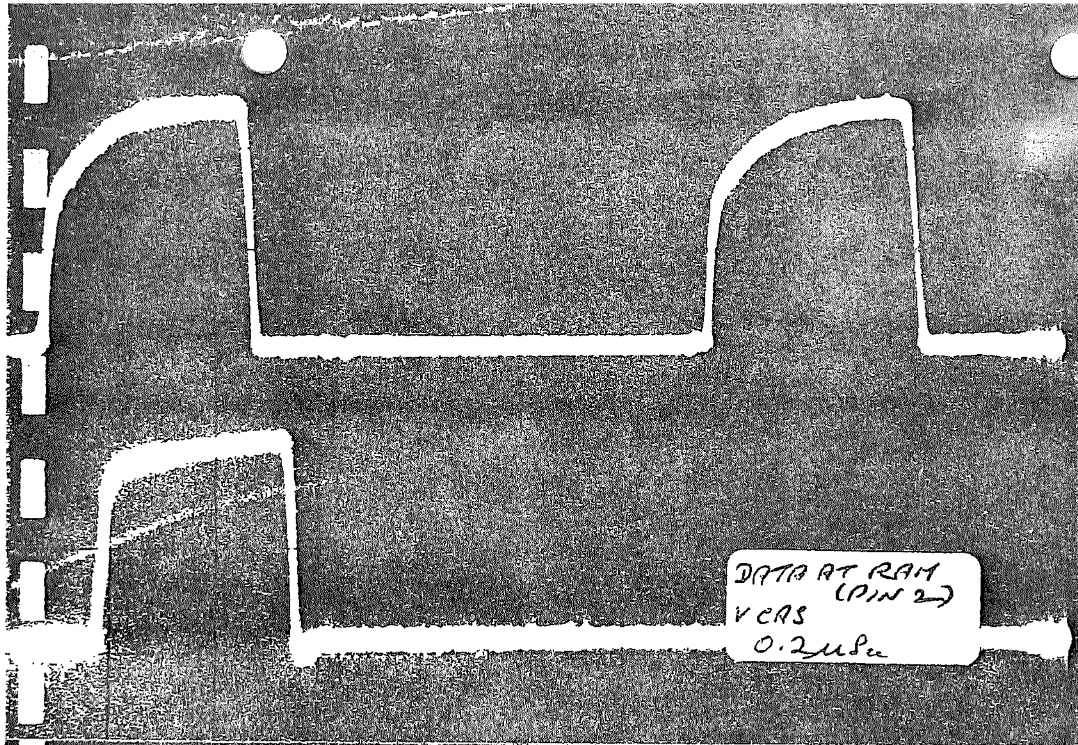
IC110 PIN 11
V CAS
0.2µSec



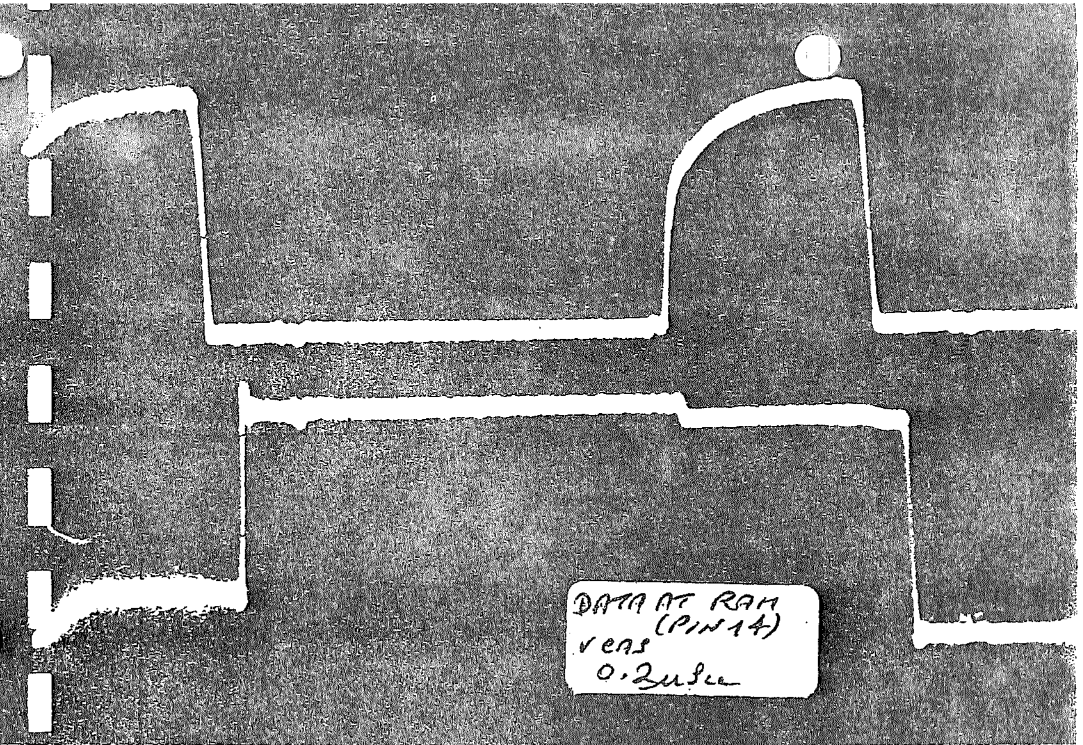
IC110 PIN 11
V CAS
50nSec



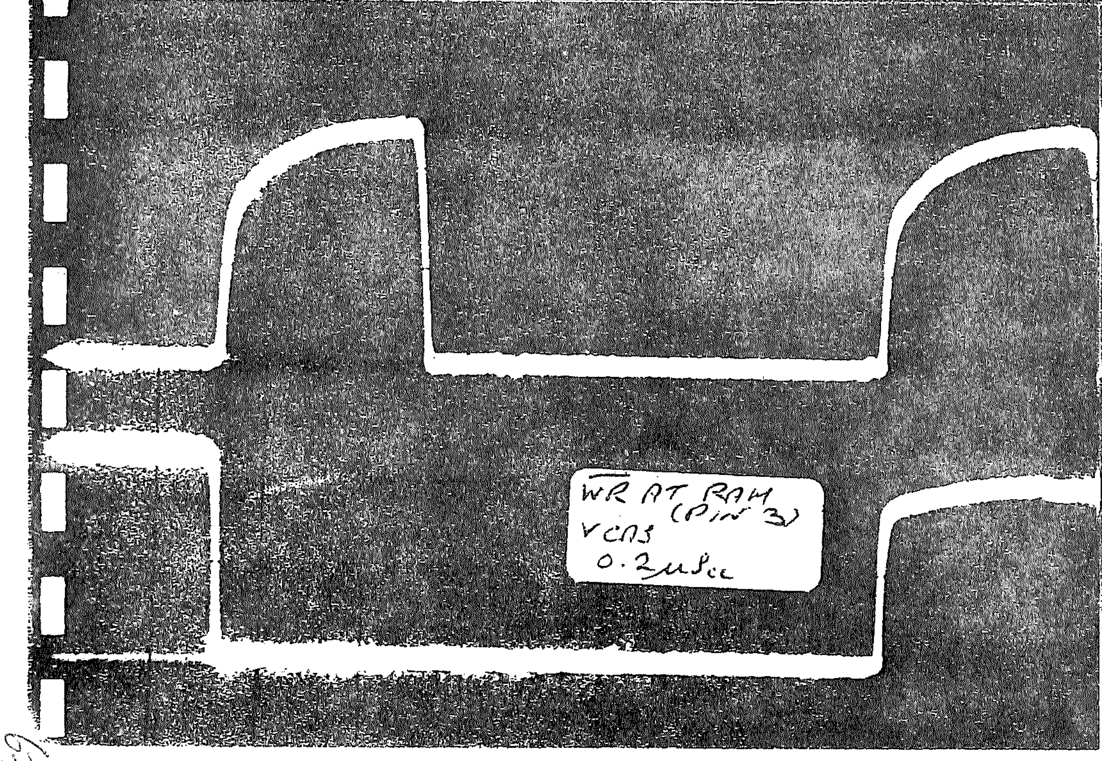
CAS AT 100Ω
AND DISTANT
RAM.
50nSec



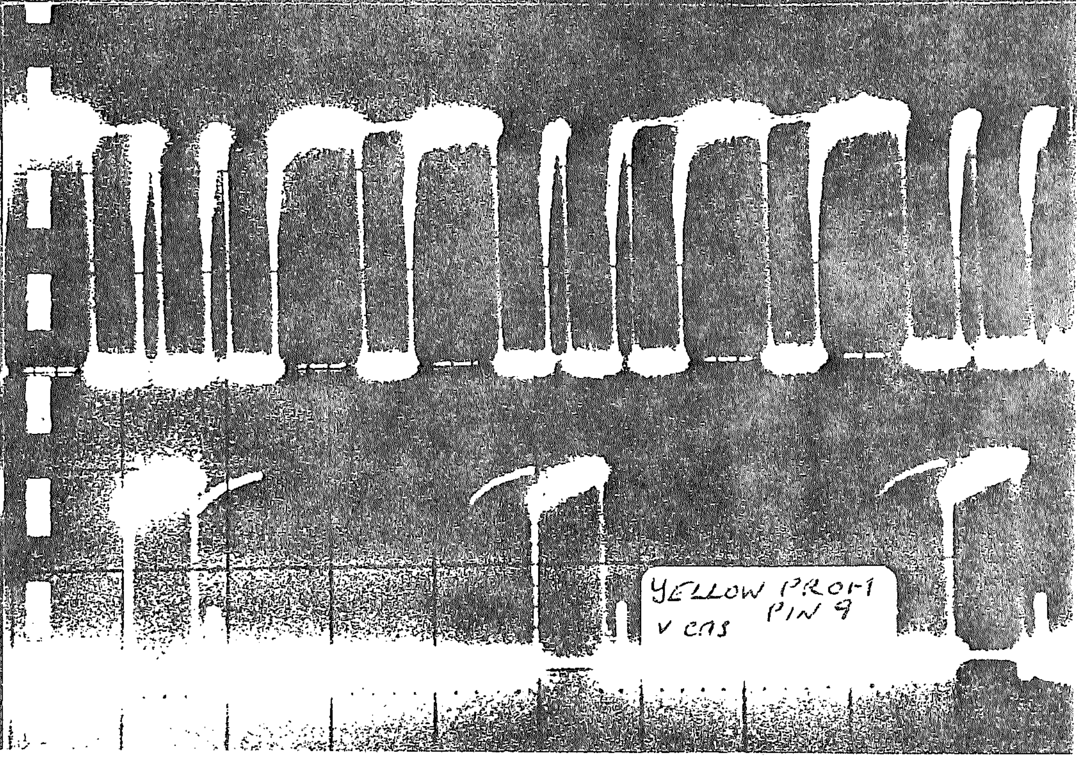
DATA AT RAM
(PIN 2)
V CAS
0.2uSec



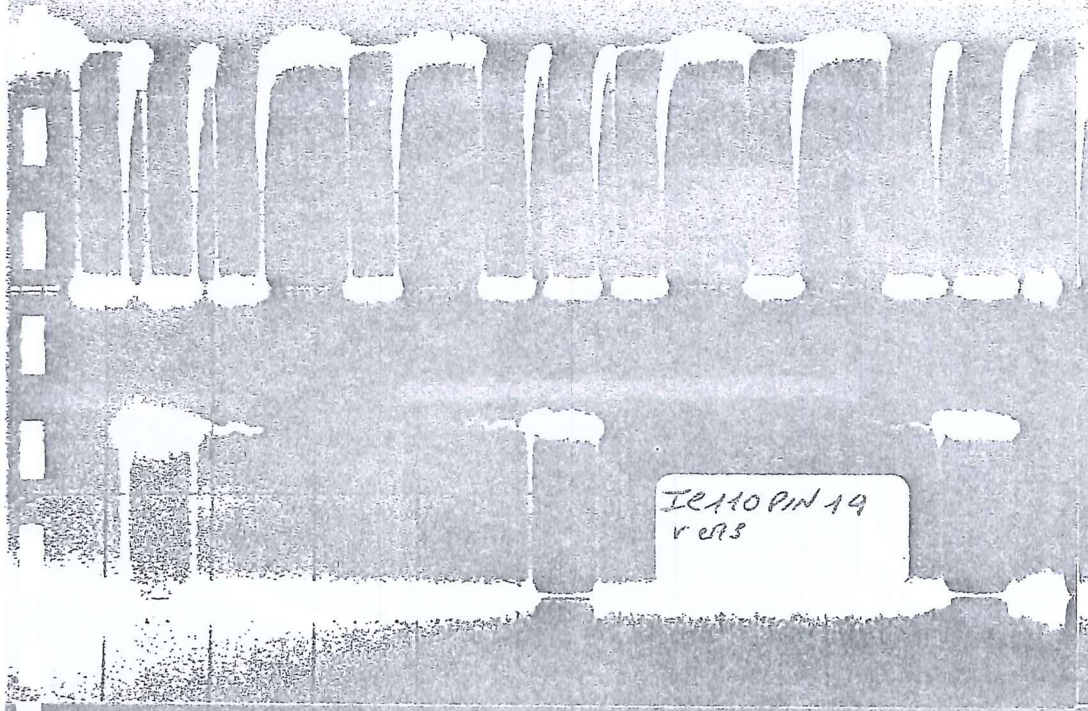
DATA AT RAM
(PIN 14)
V CAS
0.2uSec

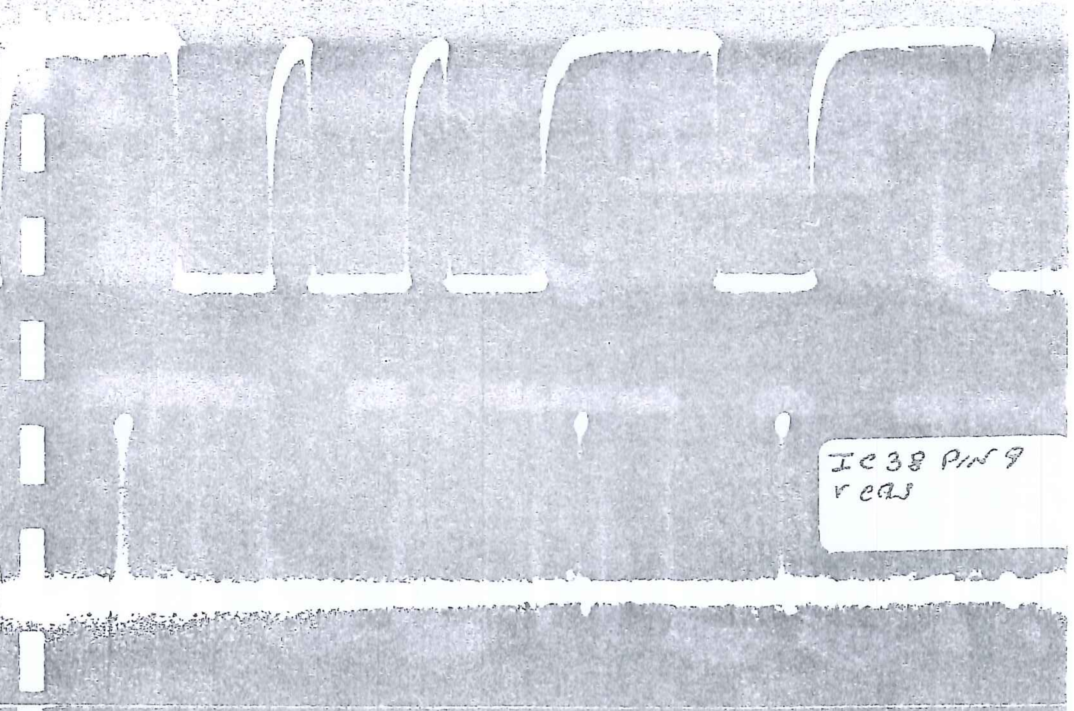


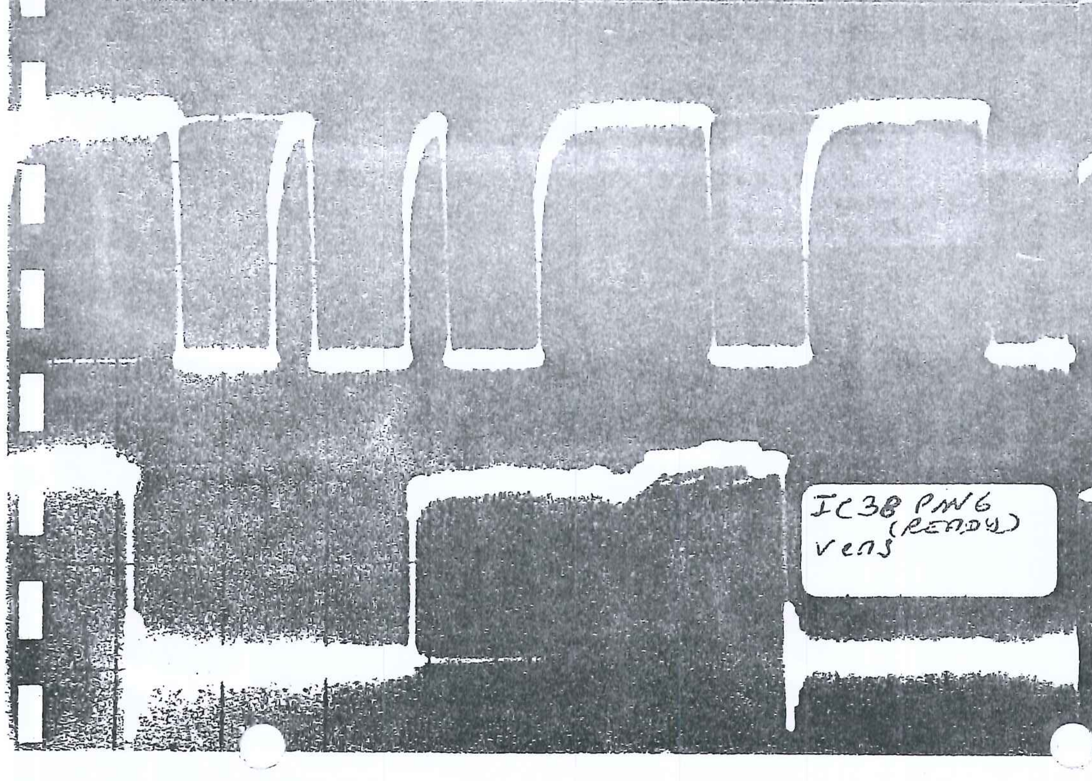
$\overline{\text{WR}}$ AT RAM
(PIN 3)
V CAS
0.2uSec

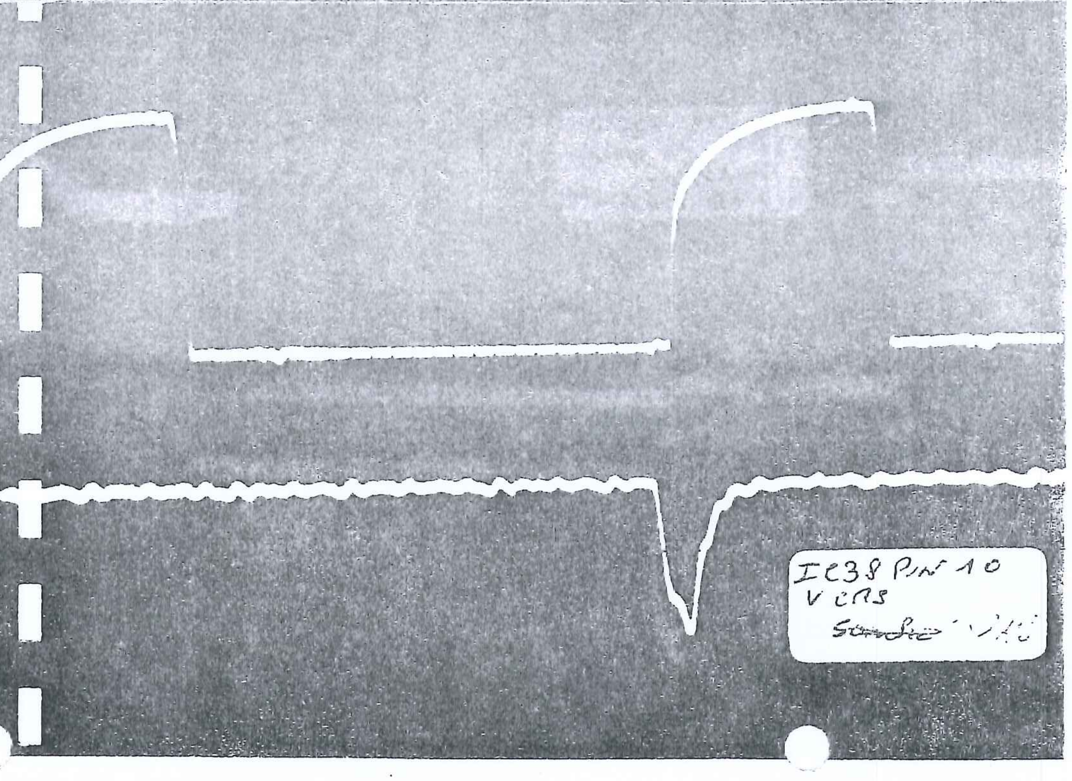


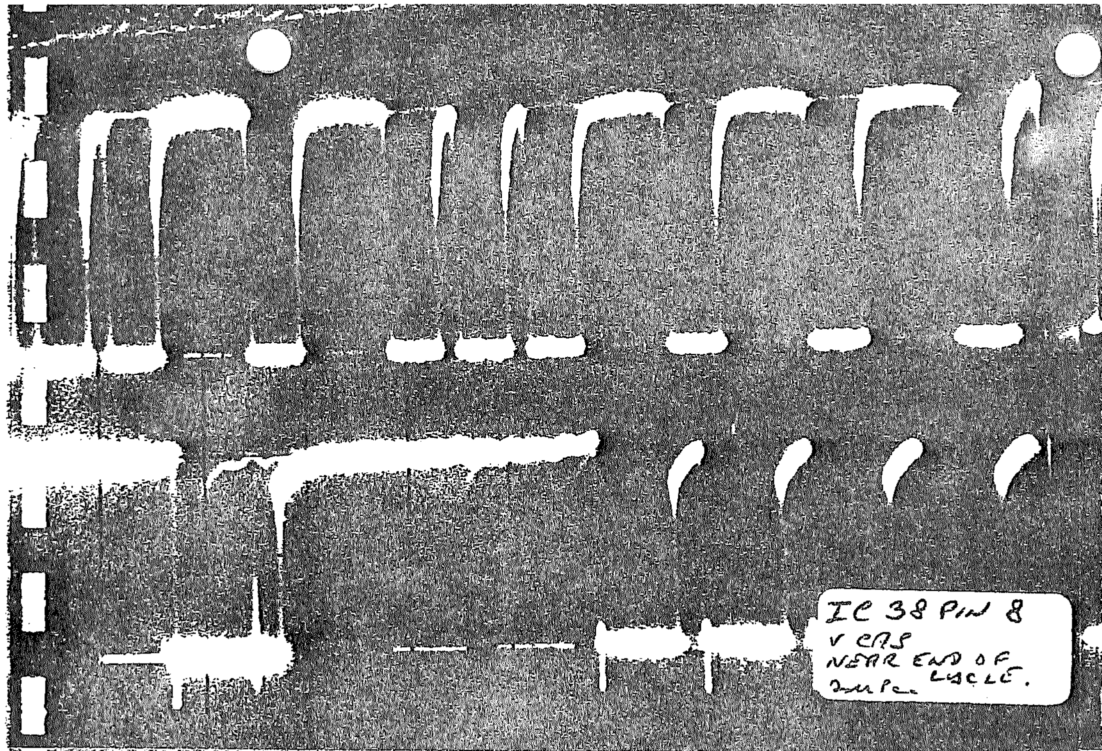
YELLOW PROM
PIN 9
V CAS

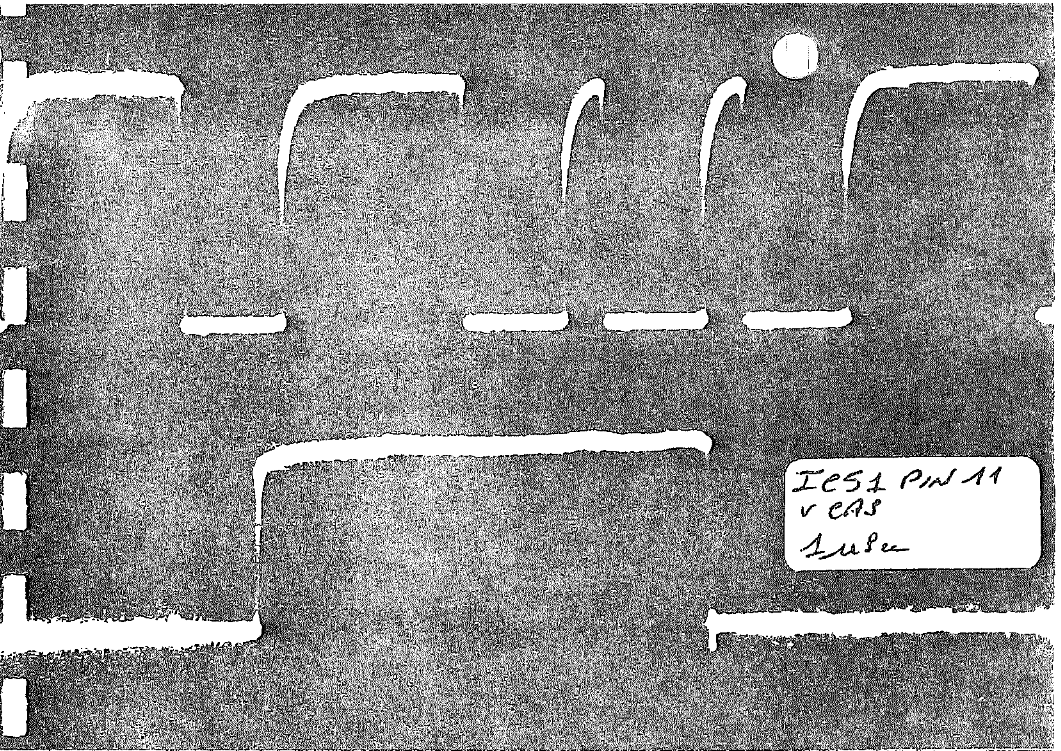

IC110 PIN 19
v CAS

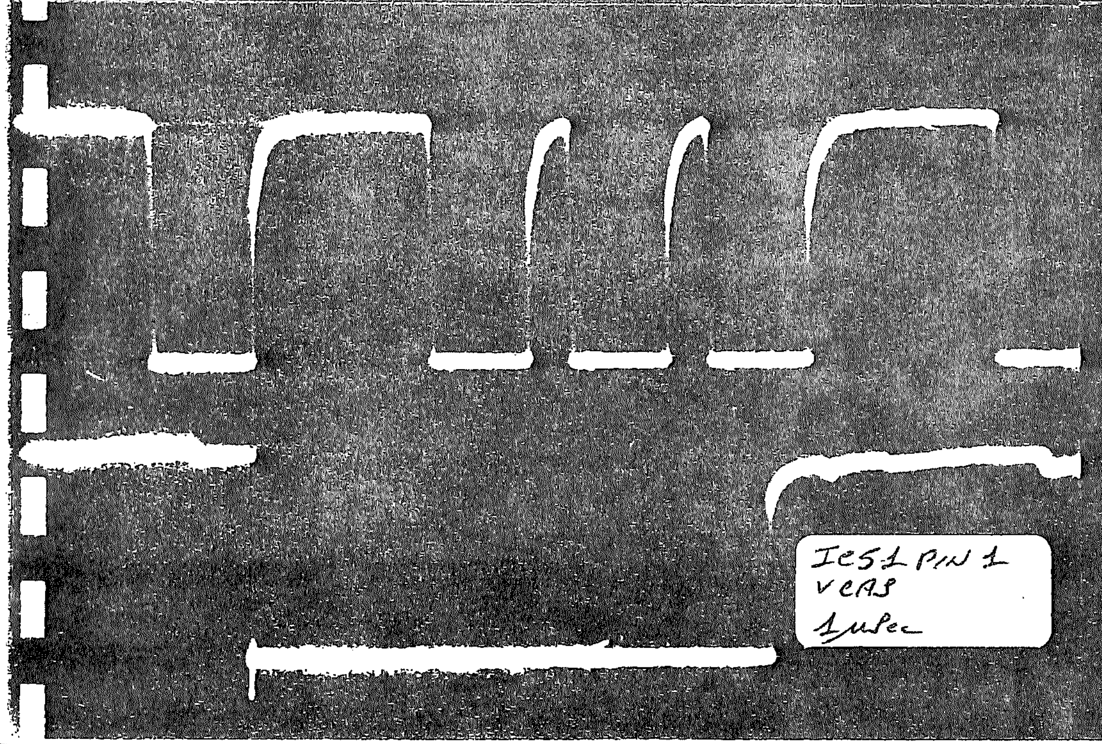

IC38 PIN 9
v CAS

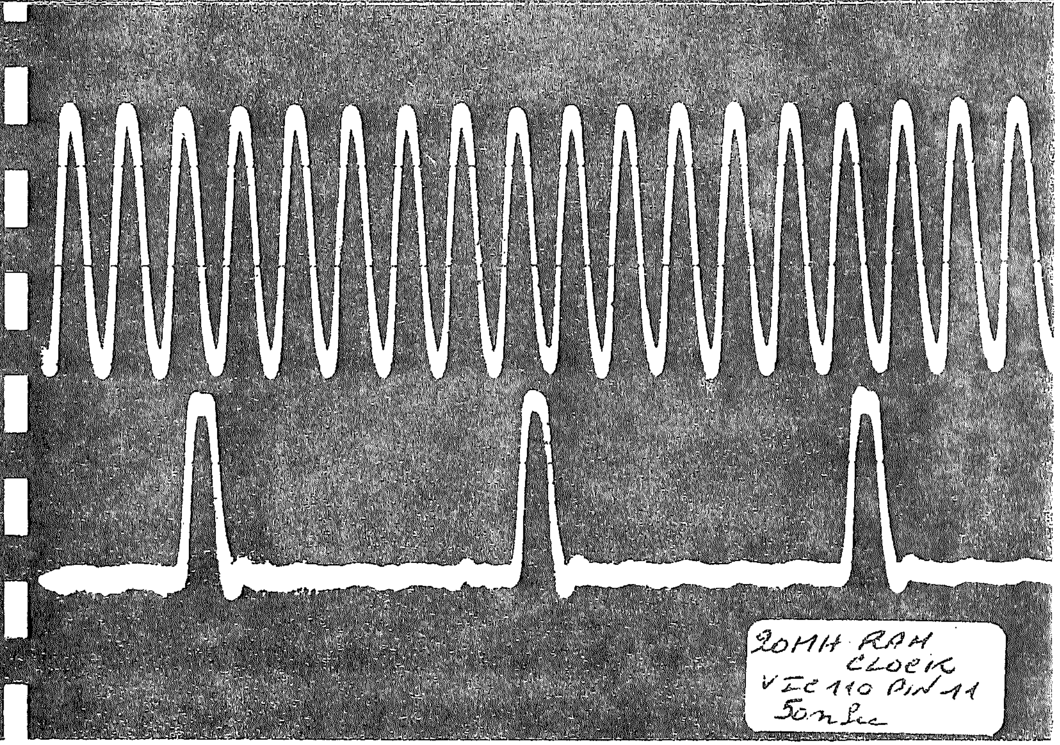

IC38 PIN 6
(READY)
v CAS

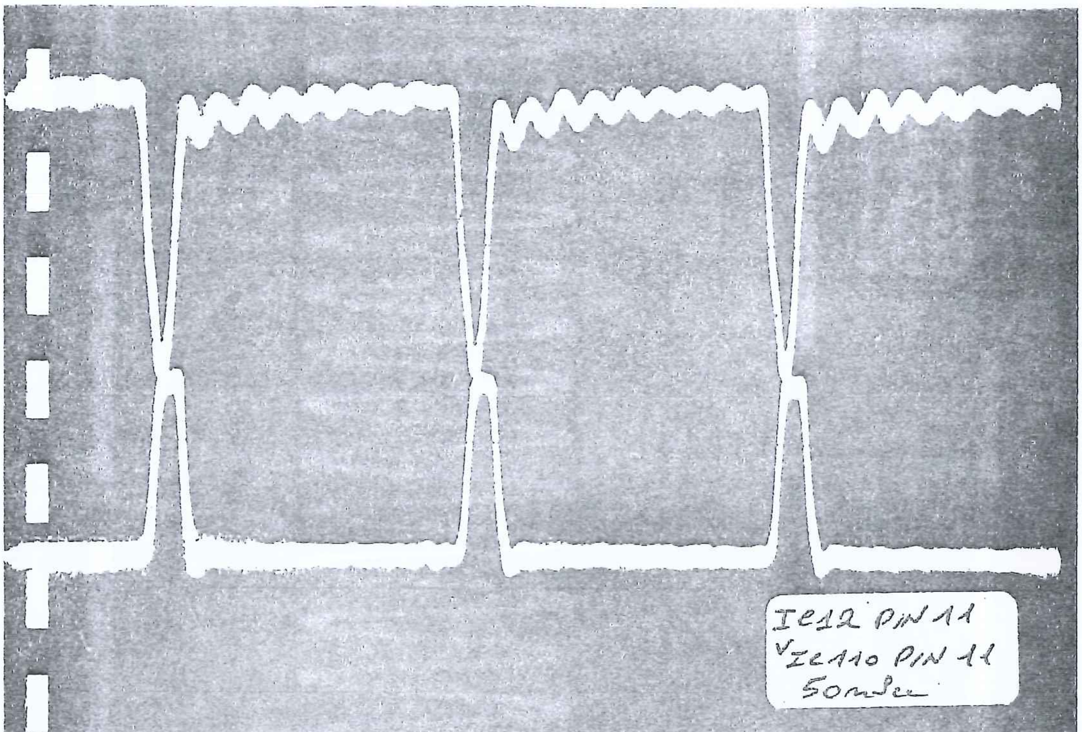

IC38 PIN 10
v CAS
Sonde 1:10

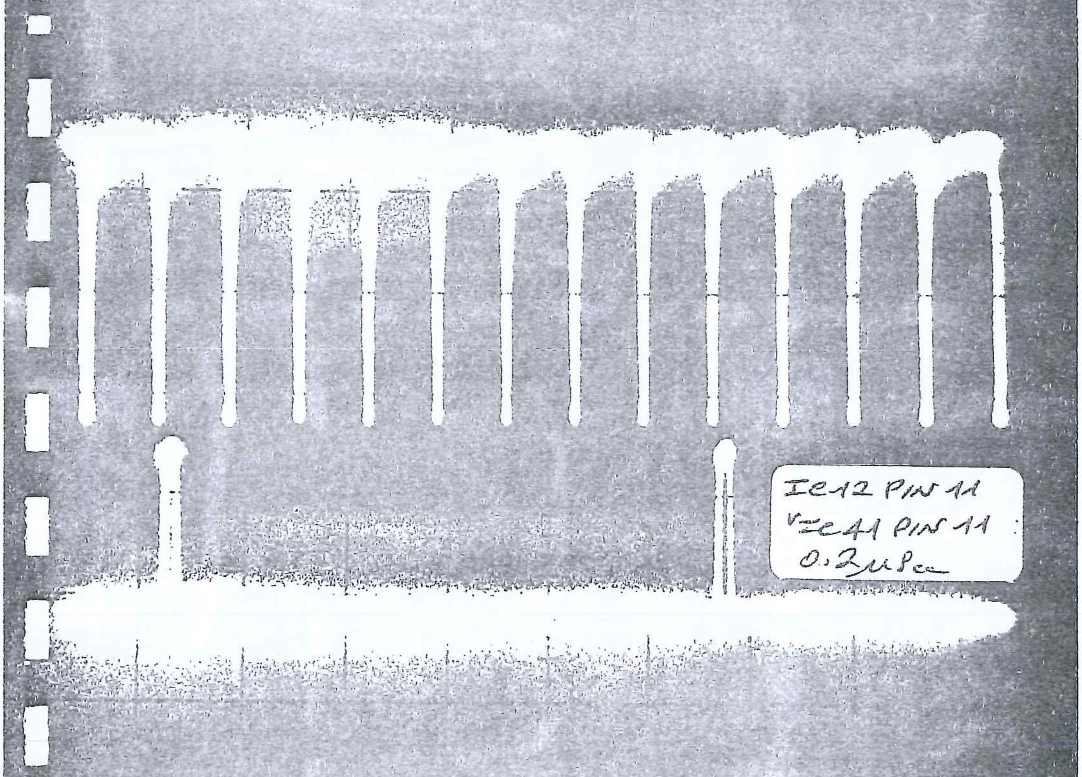

IC 38 PIN 8
V CAS
NEAR END OF CYCLE.
2uSec

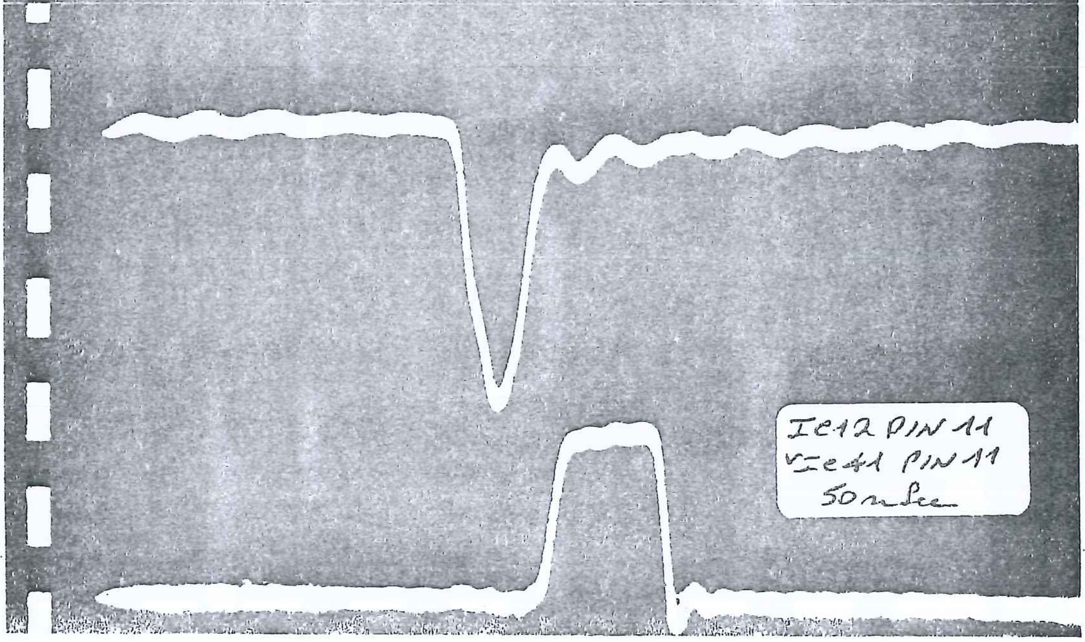

IC51 PIN 11
V CAS
1uSec


IC51 PIN 1
V CAS
1uSec


20MH RAM CLOCK
V IC 110 PIN 11
50nSec


IC12 PIN 11
vIC110 PIN 11
50nSec


IC12 PIN 11
vIC41 PIN 11
0.2µSec


IC12 PIN 11
vIC41 PIN 11
50nSec

4) INPUT PORT CHECK

Requirements :

1) Running Utility

1 SFD00 CF-

Bit	Purpose	Expected state
0	Not used	
1	Not used	
2	PAGE	Mostly high
3	V24 ready	High
4	Paddle event	Low
5	Paddle event	Low
6	Noise input	Variable
7	Cassette input	(Usually high)

Note that if Bit 3 is **not** high then BASIC cannot run.

KEYBOARD CHECK

Requirements

1	Running Utility		
1	SFF07	-FF	activate all key scan lines
2	SFF01	00-	must be zero if BASIC put in (i.e. no keys pressed)

8255 TEST

Requirements

Running UTILITY : type in little test program so the bits of the output port will be 0 or 1 (alternate)

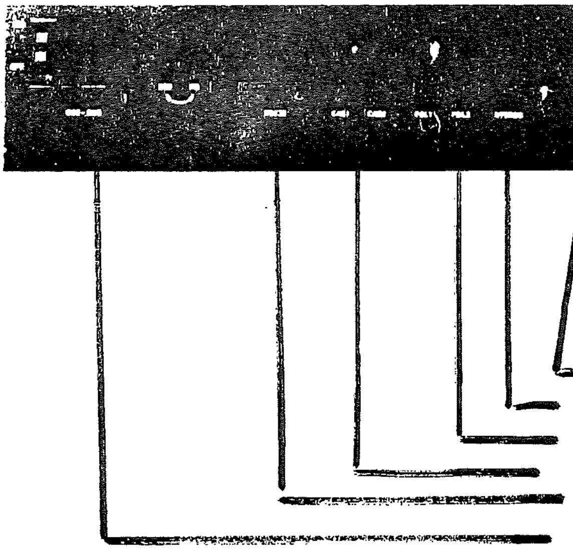
```
SF800  2103FE  36   80  21   00  FE   3655  2101  FE  36  55

       2102FE  3655  C3   05  F8
```

5) Run quick check (see small function check)

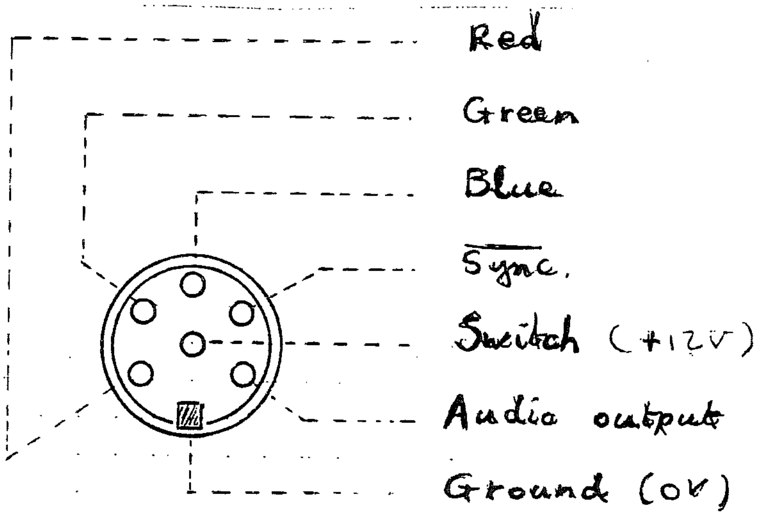
APPENDIX

1) PIN CONNECTIONS



1) PAL OR RGB OUTPUT
2) STEREO OUTPUT
3) PADDLE INTERFACE
4) CASSETTE RECORDER INTERFACE
5) RS232 CONNECTOR
6) DCE-BUS

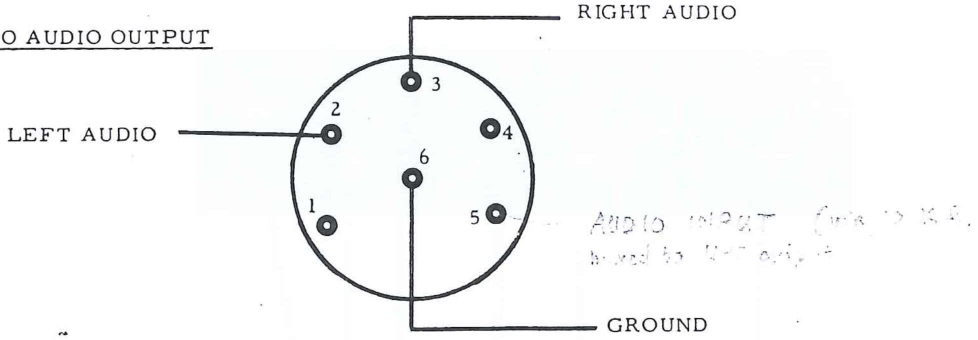
a) RGB output



b) STEREO OUTPUT

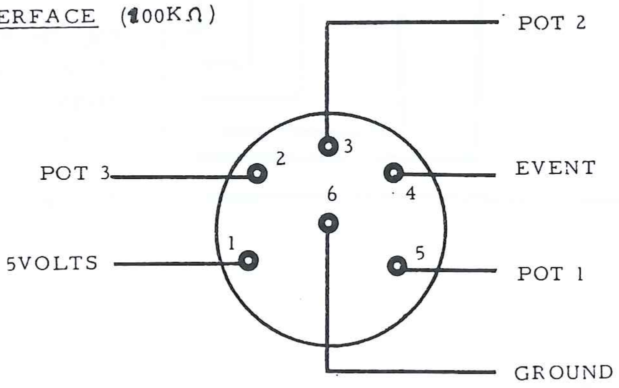
(6 PINS DIN PLUG 240° VIEWED FROM INSIDE OF THE PLUG OR TO THE COMPUTER PLUG)

STEREO AUDIO OUTPUT



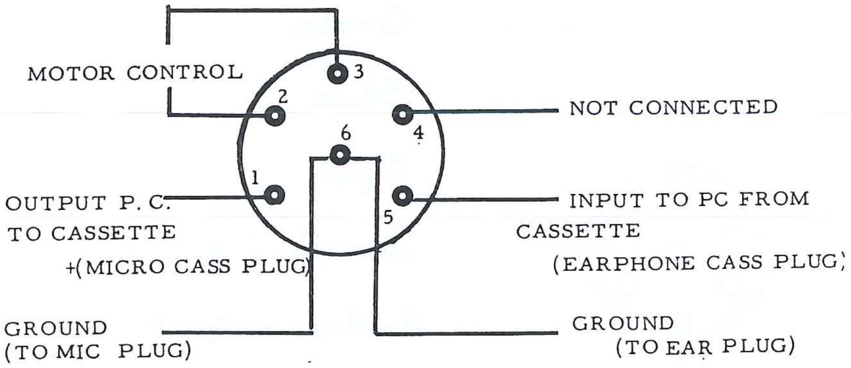
C) PADDLE INTERFACE

PADDLE INTERFACE (100KΩ)



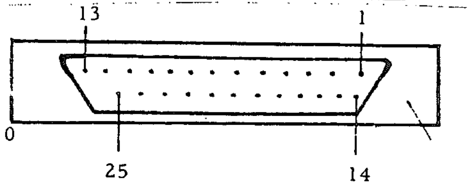
d) CASSETTE RECORDER INTERFACE

CASSETTE RECORDER INTERFACE



e) RS232 INTERFACE

PERSONAL COMPUTER RS-232 CONNECTOR



FEMALE CONNECTOR

(OUTSIDE VIEUW)

PIN	FUNCTION	
1	GND	
2	SERIAL OUT	OUTPUT DATA FORM P.C.
3	SERIAN IN	INPUT DATA TO P.C
4	DATA TERMINAL RDY	INPUT READY HIGH (5V) NOT READY LOW (ØV)
5	+12V*	Note : This connector is wired as for a terminal and signals to pins 2 and 3 may have to be swapped if it is to send data to a terminal/printer.
6	+12V*	
7	GND	
8	+12V*	
16	-5V	
20	via R = 560Ω to +5V	
23)) 24)	RL2 pin 1 & 7	

* 12V THROUGH 220Ω 1/4W

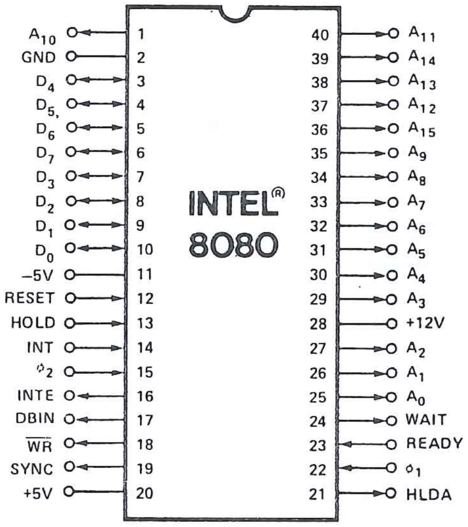
f) DCE-BUS

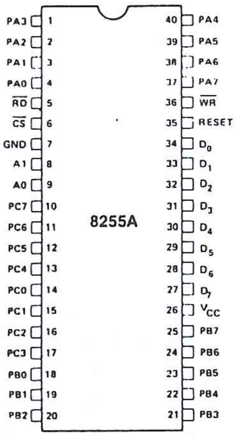
NAME	DESCRIPTION	P.C. PIN#	R.W.C. PIN#	DCE FUNCTION
****	***********	*********	***********	************
P0B0	GIC PORT 0 BIT 0	16	24	DATA BIT 0
P0B1	GIC PORT 0 BIT 1	14	26	DATA BIT 1
P0B2	GIC PORT 0 BIT 2	12	28	DATA BIT 2
P0B3	GIC PORT 0 BIT 3	10	30	DATA BIT 3
P0B4	GIC PORT 0 BIT 4	9	29	DATA BIT 4
P0B5	GIC PORT 0 BIT 5	11	27	DATA BIT 5
P0B6	GIC PORT 0 BIT 6	13	25	DATA BIT 6
P0B7	GIC PORT 0 BIT 7	15	23	DATA BIT 7
P1B0	GIC PORT 1 BIT 0	30	12	DEV. ADDR 0
P1B1	GIC PORT 1 BIT 1	31	10	DEV. ADDR 1
P1B2	GIC PORT 1 BIT 2	32	8	DEV. ADDR 2
P1B3	GIC PORT 1 BIT 3	25	7	DEV. ADDR 3
P1B4	GIC PORT 1 BIT 4	24	9	CARD ADDR 0
P1B5	GIC PORT 1 BIT 5	23	11	CARD ADDR 1
P1B6	GIC PORT 1 BIT 6	22	13	CARD ADDR 2
P1B7	GIC PORT 1 BIT 7	21	15	CARD ADDR 3
P2B0	GIC PORT 2 BIT 0	26	18	BUS EXPAND
P2B1	GIC PORT 2 BIT 1	27	17	WR" (NEG)
P2B2	GIC PORT 2 BIT 2	28	16	RD" (NEG)
P2B3	GIC PORT 2 BIT 3	29	14	NOT USED
P2B4	GIC PORT 2 BIT 4	20	19	NOT USED
P2B5	GIC PORT 2 BIT 5	19	20	NOT USED
P2B6	GIC PORT 2 BIT 6	18	21	NOT USED
P2B7	GIC PORT 2 BIT 7	17	22	NOT USED
EXINTR	EXTERNAL INTERUPT	6	4	EXINTR
IN7	PARALLEL INPUT BIT 7	5	3	IN7
EXRESET"	EXTERNAL RESET (NEG)	7	5	EXRESET"
+12V	+12 V DC	2	2	+12V
+5V	+5 V DC	1	31	+5V
-5V	-5 V DC	3	1	-5V
GND	GND (0V DC)	4	6	GND
INTR	8080 INTR-PIN 14	33	NA	NON EXISTENT
IN7	PARALLEL INPUT BIT 7	34	NA	NON EXISTENT
NC	NOT CONNECTED	8	NA	NON EXISTENT

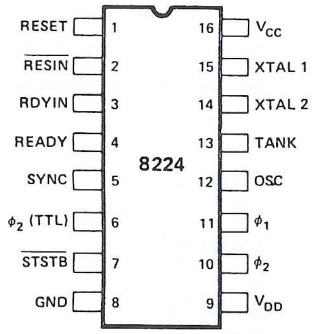
PIN-OUT OF THE 50-PENS CONNECTOR INSIDE DAIpc

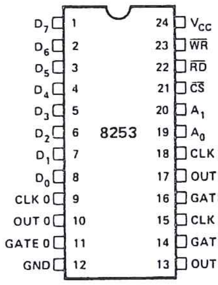
1	GROUND	SCREENING LINE
2	D0	DATA BIT 0
3	GROUND	SCREENING LINE
4	D1	DATA BIT 1
5	GROUND	SCREENING LINE
6	D2	DATA BIT 2
7	GROUND	SCREENING LINE
8	D3	DATA BIT 3
9	GROUND	SCREENING LINE
10	D4	DATA BIT 4
11	GROUND	SCREENING LINE
12	D5	DATA BIT 5
13	GROUND	SCREENING LINE
14	D6	DATA BIT 6
15	GROUND	SCREENING LINE
16	D7	DATA BIT 7
17	GROUND	SCREENING LINE
18	-	NO CONNECTION
19	GROUND	SCREENING LINE
20	MEMW	MEMORY WRITE STROBE
21	GROUND	SCREENING LINE
22	-	NO CONNECTION
23	A10	ADDRESS LINE 10
24	MEMR	MEMORY READ STROBE
25	A14	ADDRESS LINE 14
26	A11	11
27	A12	12
28	A13	13
29	A9	9
30	A15	15
31	A7	7
32	A8	8
33	A5	5
34	A6	6
35	A3	3
36	A4	4
37	A1	1
38	A2	2
39	A0	0
40	INTA	INTERRUPT ACKNOWLEDGE
41	CS LOW ROM	CHIP SELECT LOWER ROM
42	CS LB UPP ROM	CHIP SELECT LOWER BANK UPPER ROM
43	PSEUDE A12	A12 AFTER START-LOGIC
44	CS UB UPP ROM	CHIP SELECT UPPER BANK UPPER ROM
45	+5V	5 VOLT
46	+5V	5 VOLT
47	RAMOP	NOT RAM OPERATION
48	CK2	TTL LEVEL CLOCK (2MHz)
49	HOLD	HOLD REQUEST
50	SYNC	CPU SYNC SIGNAL

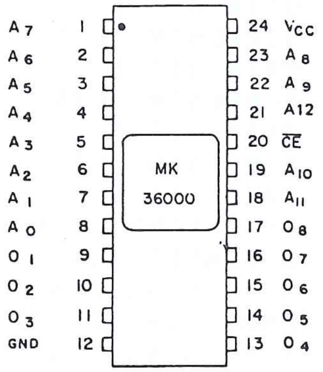
PIN CONNECTIONS IC's



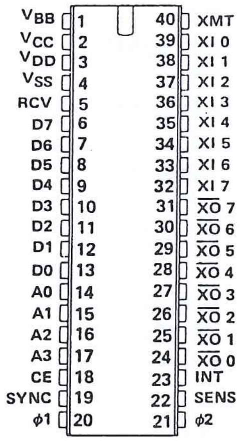


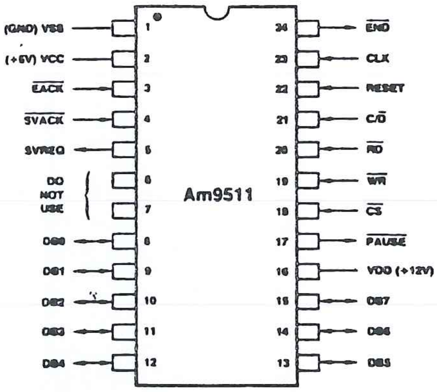


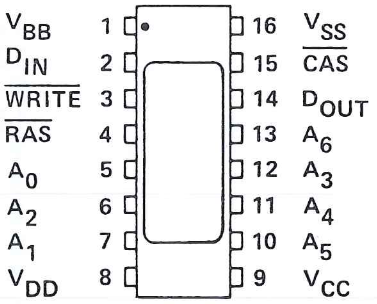


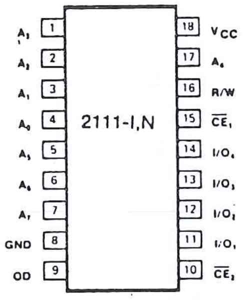


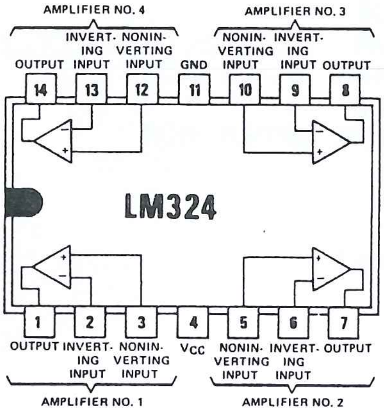
TMS 5501

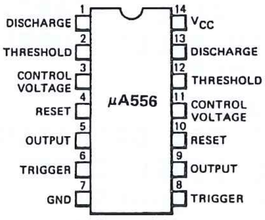


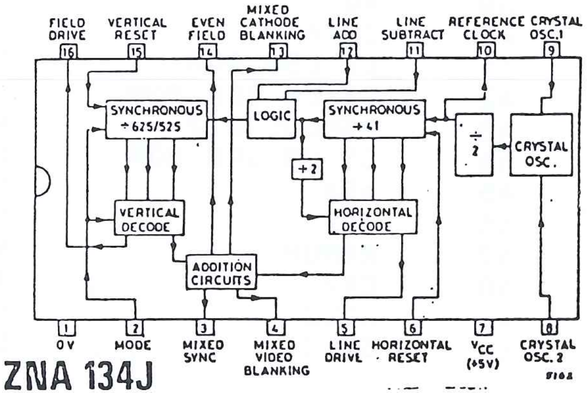






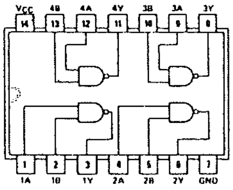




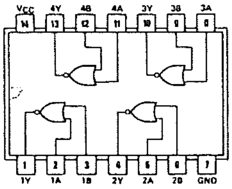


54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

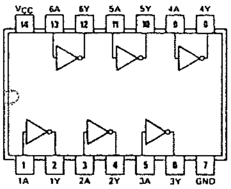
PIN ASSIGNMENTS (TOP VIEWS)



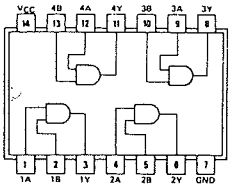
SN5400 (J) SN7400 (J, N)
SN54H00 (J) SN74H00 (J, N)
SN54L00 (J) SN74L00 (J, N)
SN54LS00 (J, W) SN74LS00 (J, N)
SN54S00 (J, W) SN74S00 (J, N)



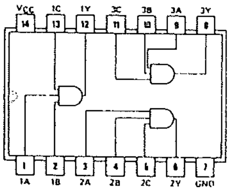
SN5402 (J) SN7402 (J, N)
SN54L02 (J) SN74L02 (J, N)
SN54LS02 (J, W) SN74LS02 (J, N)
SN54S02 (J, W) SN74S02 (J, N)



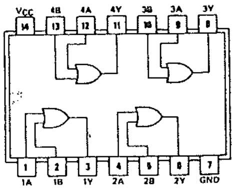
SN5404 (J) SN7404 (J, N)
SN54H04 (J) SN74H04 (J, N)
SN54L04 (J) SN74L04 (J, N)
SN54LS04 (J, W) SN74LS04 (J, N)
SN54S04 (J, W) SN74S04 (J, N)



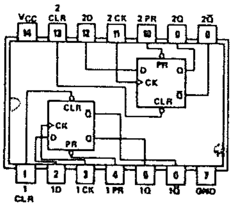
SN5408 (J, W) SN7408 (J, N)
SN54LS08 (J, W) SN74LS08 (J, N)
SN54S08 (J, W) SN74S08 (J, N)



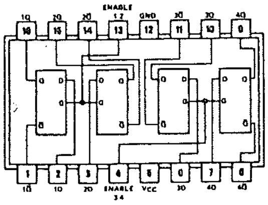
SN54H15 (J, W) SN74H15 (J, N)
SN54LS15 (J, W) SN74LS15 (J, N)
SN54S15 (J, W) SN74S15 (J, N)



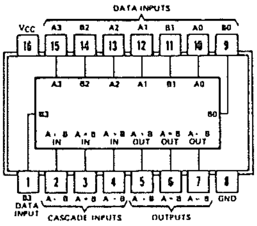
SN5432 (J, W) SN7432 (J, N)
SN54LS32 (J, W) SN74LS32 (J, N)
SN54S32 (J, W) SN74S32 (J, N)



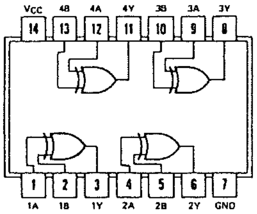
SN5474 (J) SN7474 (J, N)
SN54H74 (J) SN74H74 (J, N)
SN54L74 (J) SN74L74 (J, N)
SN54LS74A (J, W) SN74LS74A (J, N)
SN54S74 (J, W) SN74S74 (J, N)


1Q 2Q 2Q ENABLE 1 2 GND 3Q 3Q 4Q
1Q 1D 2D ENABLE 3 4 VCC 3D 4D 4Q

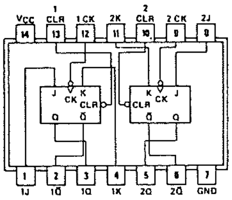
SN5475 (J, W) SN7475 (J, N)
SN54L75 (J) SN74L75 (J, N)
SN54LS75 (J, W) SN74LS75 (J, N)


DATA INPUTS
VCC A3 B2 A2 A1 B1 A0 B0
A<B IN A=B IN A>B IN A>B OUT A=B OUT A<B OUT
B3 DATA INPUT CASCADE INPUTS OUTPUTS GND

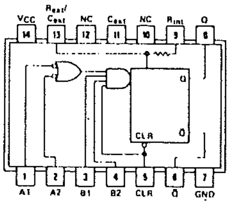
SN5485 (J, W) SN7485 (J, N)
SN54LS85 (J, W) SN74LS85 (J, N)
SN54S85 (J, W) SN74S85 (J, N)



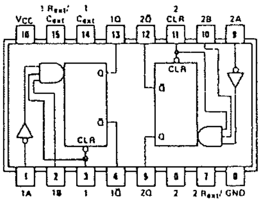
SN5486 (J, W) SN7486 (J, N)
SN54LS86 (J, W) SN74LS86 (J, N)
SN54S86 (J, W) SN74S86 (J, N)



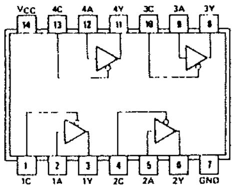
SN54107 (J) SN74107 (J, N)
SN54LS107A(J) SN74LS107A(J, N)



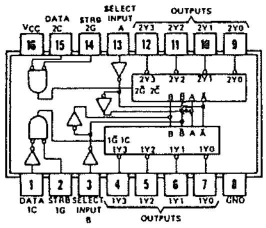
SN54122 (J, W) SN74122 (J, N)
SN54L122 (J, T) SN74L122 (J, N)
SN54LS122 (J, W) SN74LS122 (J, N)


VCC 1 Rext/Cext 1 Cext 1Q 2Q̄ 2 CLR 2B 2A
1A 1B 1 CLR 1Q̄ 2Q 2 Cext 2 Rext/Cext GND

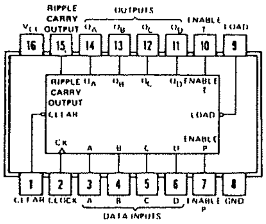
SN54123 (J, W) SN74123 (J, N)
SN54L123 (J) SN74L123 (J, N)
SN54LS123 (J, W) SN74LS123 (J, N)



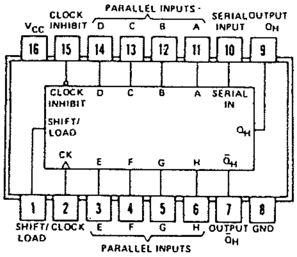
SN54125 (J, W) SN74125 (J, N)
SN54LS125A(J, W) SN74LS125A(J, N)


VCC DATA 2C STRB 2G SELECT INPUT A OUTPUTS 2Y3 2Y2 2Y1 2Y0
DATA 1C STRB 1G SELECT INPUT B OUTPUTS 1Y3 1Y2 1Y1 1Y0 GND

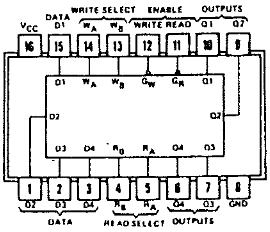
SN54155 (J, W) SN74155 (J, N)
SN54LS155 (J, W) SN74LS155 (J, N)
SN54156 (J, W) SN74156 (J, N)
SN54LS156 (J, W) SN74LS156 (J, N)


VCC RIPPLE CARRY OUTPUT OUTPUTS QA QB QC QD ENABLE T LOAD
CLEAR CLOCK A B C D ENABLE P GND
DATA INPUTS

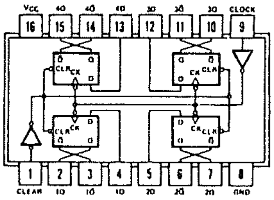
SN54160 (J, W) SN74160 (J, N)
SN54LS160A (J, W) SN74LS160A (J, N)
SN54161 (J, W) SN74161 (J, N)
SN54LS161A (J, W) SN74LS161A (J, N)


VCC CLOCK INHIBIT PARALLEL INPUTS D C B A SERIAL INPUT OUTPUT QH
SHIFT/LOAD CLOCK E F G H OUTPUT Q̄H GND
PARALLEL INPUTS

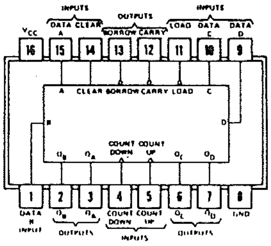
SN54165 (J, W) SN74165 (J, N)
SN54LS165 (J, W) SN74LS165 (J, N)


VCC DATA D1 WRITE SELECT WA WB ENABLE WRITE READ OUTPUTS Q1 Q2
D2 D3 D4 RB RA Q4 Q3 GND
DATA READ SELECT OUTPUTS

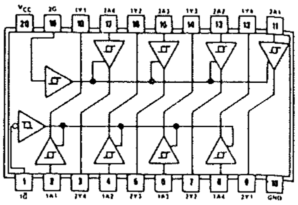
SN54170 (J, N) SN74170 (J, N)
SN54LS170 (J, W) SN74LS170 (J, N)



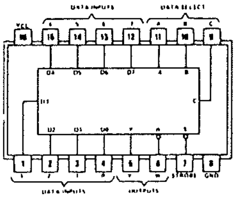
SN54175 (J, W) SN74175 (J, N)
SN54LS175 (J, W) SN74LS175 (J, N)
SN54S175 (J, W) SN74S175 (J, N)


VCC INPUTS DATA A CLEAR OUTPUTS BORROW CARRY INPUTS LOAD DATA C DATA D
DATA B QB QA COUNT DOWN COUNT UP QC QD GND

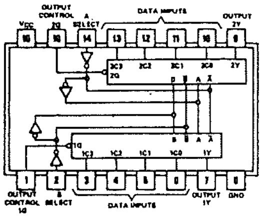
SN54193 (J, W) SN74193 (J, N)
SN54L193 (J) SN74L193 (J, N)
SN54LS193 (J, W) SN74LS193 (J, N)



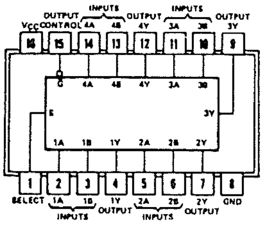
SN54LS241 (J) SN74LS241 (J, N)
SN54S241 (J) SN74S241 (J, N)



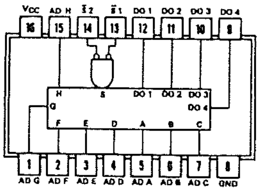
SN54251 (J, W) SN74251 (J, N)
SN54LS251 (J, W) SN74LS251 (J, N)
SN54S251 (J, W) SN74S251 (J, N)


VCC OUTPUT CONTROL 2G A SELECT DATA INPUTS OUTPUT 2Y
OUTPUT CONTROL 1G B SELECT DATA INPUTS OUTPUT 1Y GND

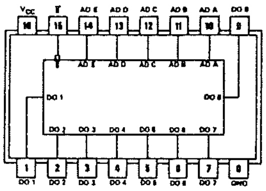
SN54LS253 (J, W) SN74LS253 (J, N)


VCC OUTPUT CONTROL 4A 4B 4Y INPUTS 3A 3B OUTPUT 3Y
SELECT 1A 1B 1Y 2A 2B 2Y GND
INPUTS OUTPUT INPUTS OUTPUT

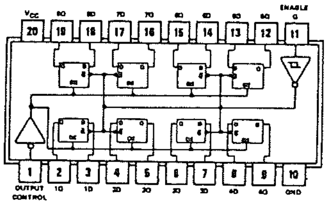
SN54LS257A (J, W) SN74LS257A (J,
SN54S257 (J, W) SN74S257 (J, N)


VCC AD H Ē2 Ē1 DO1 DO2 DO3 DO4
AD G AD F AD E AD D AD A AD B AD C GND

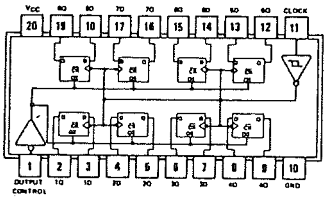
SN54S287 (J, W) SN74S287 (J, N)
(Redesignated TBP14S10)


VCC Ē AD E AD D AD C AD B AD A DO 8
DO1 DO2 DO3 DO4 DO5 DO6 DO7 GND

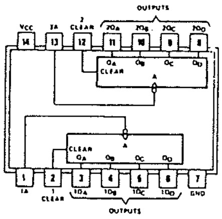
SN54288 (J, W) SN74S288 (J, N)
(Redesignated TBP18S030)


VCC 8Q 8D 7D 7Q 6Q 6D 5D 5Q ENABLE
OUTPUT CONTROL 1Q 1D 2D 2Q 3Q 3D 4D 4Q GND

SN54LS373 (J) SN74LS373 (J, N)
SN54S373 (J) SN74S373 (J, N)

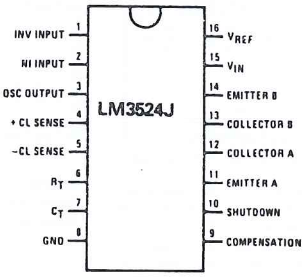

VCC 8Q 8D 7D 7Q 6Q 6D 5D 5Q CLOCK
OUTPUT CONTROL 1Q 1D 2D 2Q 3Q 3D 4D 4Q GND

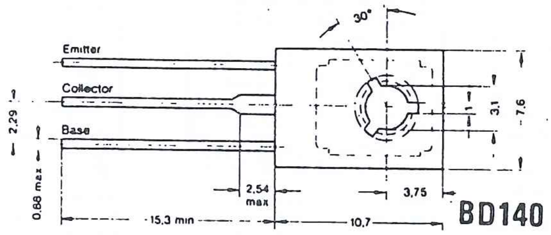
SN54LS374 (J) SN74LS374 (J, N)
SN54S374 (J) SN74S374 (J, N)


OUTPUTS
VCC 2A 2 CLEAR 2QA 2QB 2QC 2QD
1A 1 CLEAR 1QA 1QB 1QC 1QD GND
OUTPUTS

SN54393 (J, W) SN74393 (J, N
SN54LS393 (J, W) SN74LS393 (J,

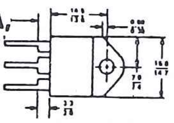
PIN ASSIGNMENTS



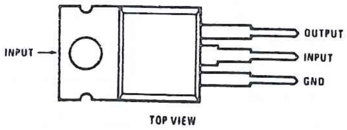


BD140

TIP34A,

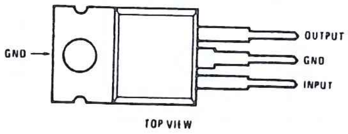


TO-220 Package



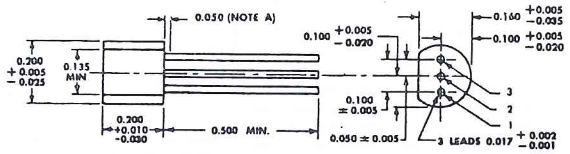
Order Numbers:
LM7905CT
LM7912CT
LM7915CT
See NS Package T03B

Plastic Package
TO-220 (T)



Order Numbers:
LM7805CT
LM7812CT
LM7815CT
See Package T03B

*ALL JEDEC TO-92 DIMENSIONS AND NOTES ARE APPLICABLE



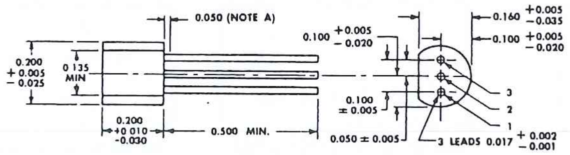
NOTES: A. Lead diameter is not controlled in this area.
B. All dimensions are in inches.

DEVICE	LEADS		
	1	2	3
2N3702, 2N3703	Emitter	Collector	Base
A8T3702, A8T3703	Emitter	Base	Collector

2N3702
2N3703

ECB

*ALL JEDEC TO-92 DIMENSIONS AND NOTES ARE APPLICABLE



NOTES: A. Lead diameter is not controlled in this area.
B. All dimensions are in inches.

DEVICE	LEAD		
	1	2	3
2N3704, 2N3705, 2N3706	Emitter	Collector	Base
A8T3704, A8T3705, A8T3706	Emitter	Base	Collector

2N3704
2N3705
2N3706

ECB

2) SMALL FUNCTION CHECK

* MODE TEST

```
        1Ø DRAW Ø,Ø  YMAX,YMAX 15
        2Ø GOTO 1Ø
       RUN this program in the modes 1 to 6
```

* COLOUR TEST (RUN in mode 3)

```
        1Ø FOR A=ØTO15:FILL A*1Ø,Ø  A*1Ø+9,YMAX  A:NEXT
```

* SOUND TEST (first ENVELOPE Ø 15)

```
                                                 3
        1Ø FOR A=Ø to 15:SOUND Ø)ØAØ FREQ(333):WAIT TIME 5:NEXT
        2Ø GOTO 1Ø                1) → channel select
                                  2)
```

* NOISE TEST

```
        NOISE Ø 15
```

* PADDLE TEST

+ PADDLE 1

```
        1Ø FORA=ØTO2:?PDL(A);: NEXT:?
        2Ø GOTO 1Ø
```

+ PADDLE 2

Change in line 1Ø:FORA=3TO5:...

* CASSETTE LOAD : put first on the cassette a little program (10 times the same program)

```
        then for cassette 1 : LOAD:RUN
             for cassette 2 : POKE #13D,#20
                              LOAD:RUN
```

* KEYBOARD CHECK : (touch each key 2 times)

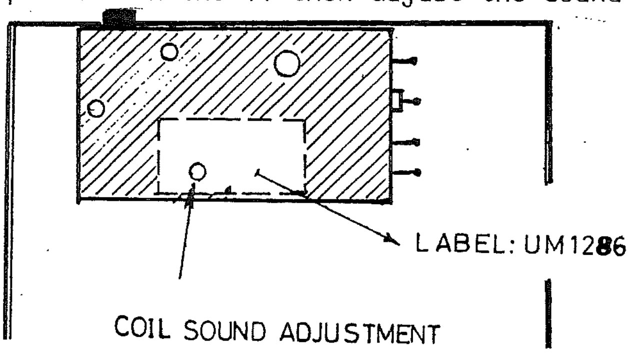
for the keys ↑, ←, ↓, → and TAB go to EDIT mode.

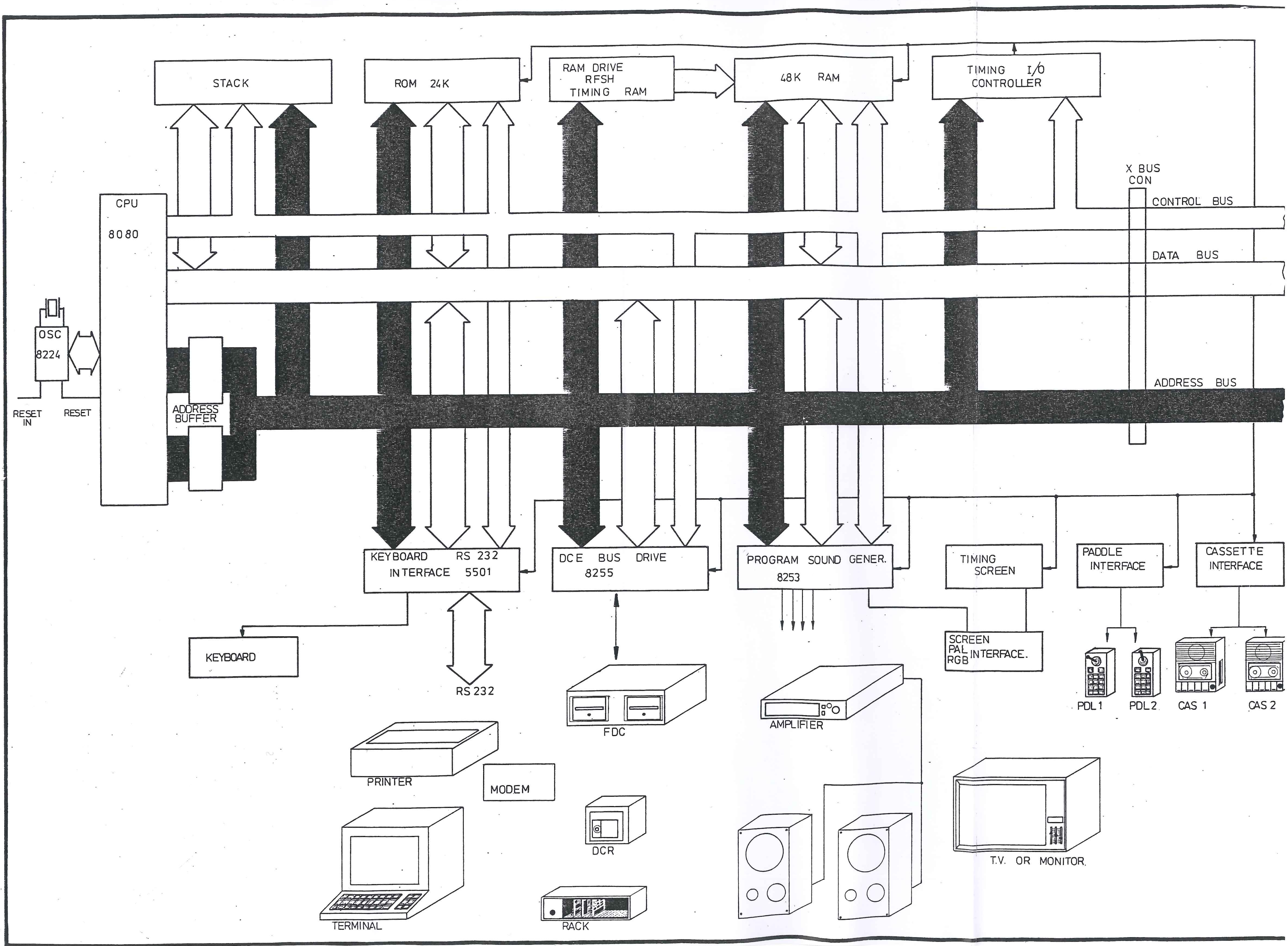
* If you have a FDC or digital cassette recorder : check for correct loading.

* Adjust the sound on the PAL card :

```
        type : * ENVELOPE Ø 15
               * SOUND Ø Ø Ø Ø 2E3
```

first adjust the picture on the TV then adjust the sound on the UHF modulator.




STACK
ROM 24K
RAM DRIVE
RFSH
TIMING RAM
48K RAM
TIMING I/O
CONTROLLER
X BUS
CON
CONTROL BUS
DATA BUS
ADDRESS BUS
CPU
8080
OSC
8224
RESET
IN
RESET
ADDRESS
BUFFER
KEYBOARD RS 232
INTERFACE 5501
DCE BUS DRIVE
8255
PROGRAM SOUND GENER.
8253
TIMING
SCREEN
PADDLE
INTERFACE
CASSETTE
INTERFACE
KEYBOARD
RS 232
FDC
AMPLIFIER
SCREEN
PAL
RGB
INTERFACE.
PDL 1
PDL 2
CAS 1
CAS 2
PRINTER
MODEM
DCR
TERMINAL
RACK
T.V. OR MONITOR.