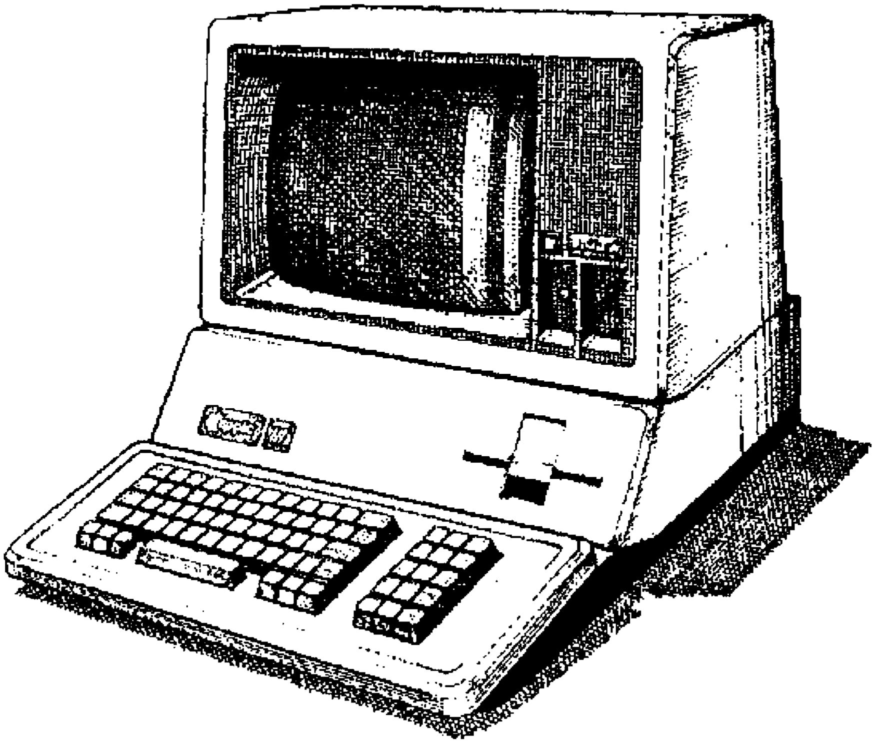
Apple /// Computer Information

# Apple /// Service Reference Manual



## Section I of II • Theory of Operation

## Chapter 2 • Memory & Memory Addressing

Written by Apple Computer • 1982

# MEMORY & MEMORY ADDRESSING

## INTRODUCTION TO THE APPLE /// MEMORY

In looking at the Apple /// and its memory we are immediately posed with the problem of how the 6502 processor can handle 128K of RAM, 4K of ROM, and a heavy array of internal and external I/O devices. The Apple /// does indeed do just that, and, further, has the hardware capability of controlling an additional 128K of memory (for a total of 256K).

This "magic" is accomplished by Bank Switching technology. At any one time, the processor can directly address 65K locations. With the addition of the Bank Switch Register, an extended addressing register, the program can call up different banks of 32K RAM space. With other software switches, ROM and all I/O locations can be replaced with RAM. [See Figure 2.1]

The first 8K of memory, from locations 0000 to 1FFF, are fixed. The 32K memory from locations locations 2000 to 9FFF are electrically switchable. The Apple /// can choose any of 15 banks to place in this area at any one time. The maximum amount of storage on the Apple ///, therefore, is equal to 15 Banks x 32K per bank + 32K fixed storage. By comparison, a 128K system would have three banks, a 256K system would have seven, etc.

In addition, the area in the fixed bank from location C000 to CFFF can be switched from RAM memory to I/O space for the slots. The area from F000 to FFFF is also switchable. When the machine is turned on, this area is ROM containing the startup program. This program runs a quick system check then loads SOS in from the internal drive. SOS then switches this area back to RAM.

The extended addressing mode allows any two adjacent banks, N and N+1, to be addressed as a contiguous 64K RAM space. Addresses 0000 to 7FFF are mapped into bank N while addresses 8000 to FFFF are mapped into bank N+1.

The Apple /// has the capability for variable Zero Page locations and Alternate Stack locations. All these features give the Apple /// great flexibility. They also provide means for very large application programs, or applications that need large amounts of RAM space for data crunching.

## SIMPLIFIED MEMORY LOGIC

If we simplify the memory and memory address logic we get three basic elements:

- the processor
- the RAM address circuits
- the RAM array

apple computer inc.

# Memory Map

128 K

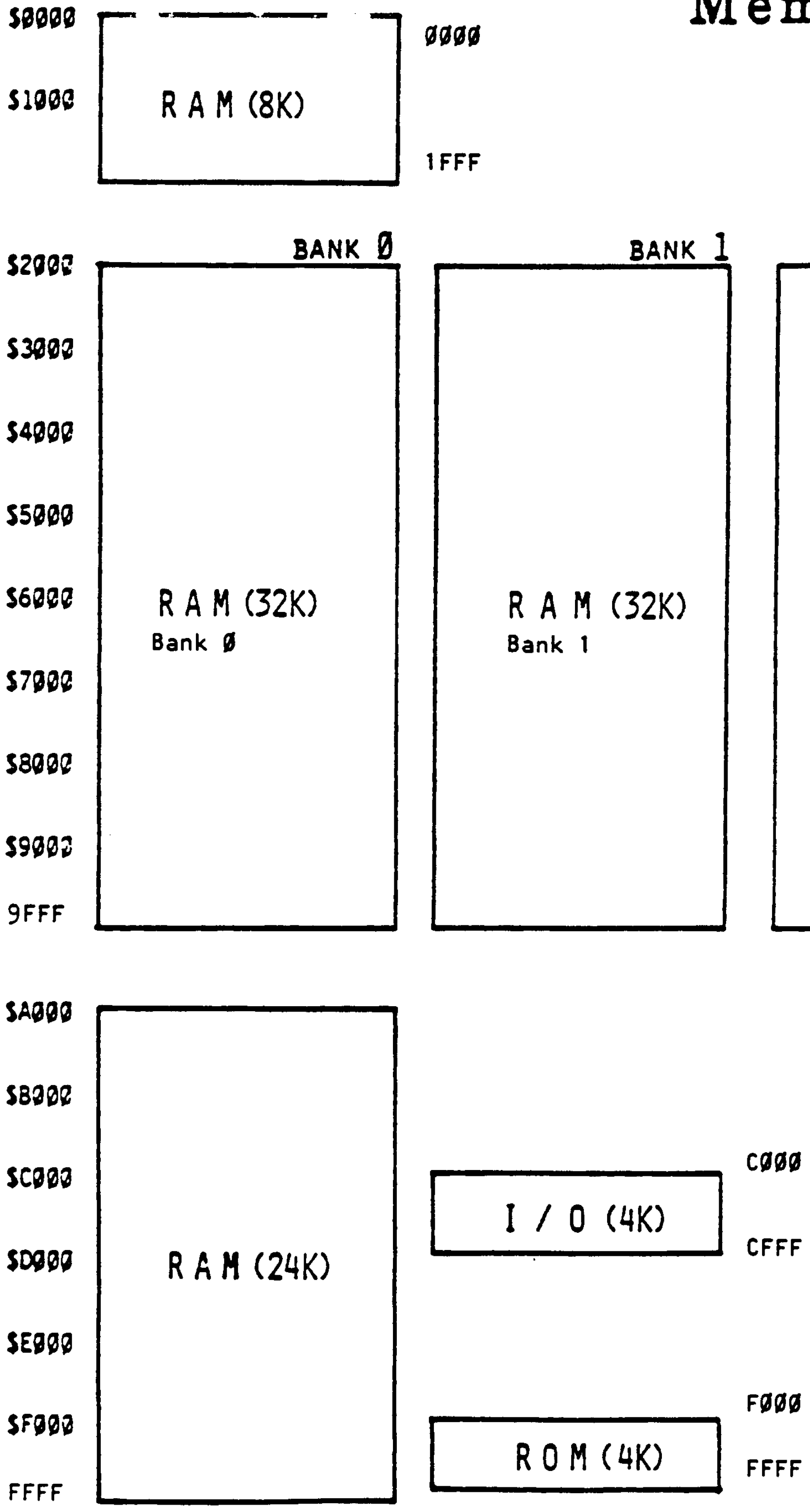


FIG 2.1

## SIMPLIFIED MEMORY ADDRESS BLOCK DIAGRAM

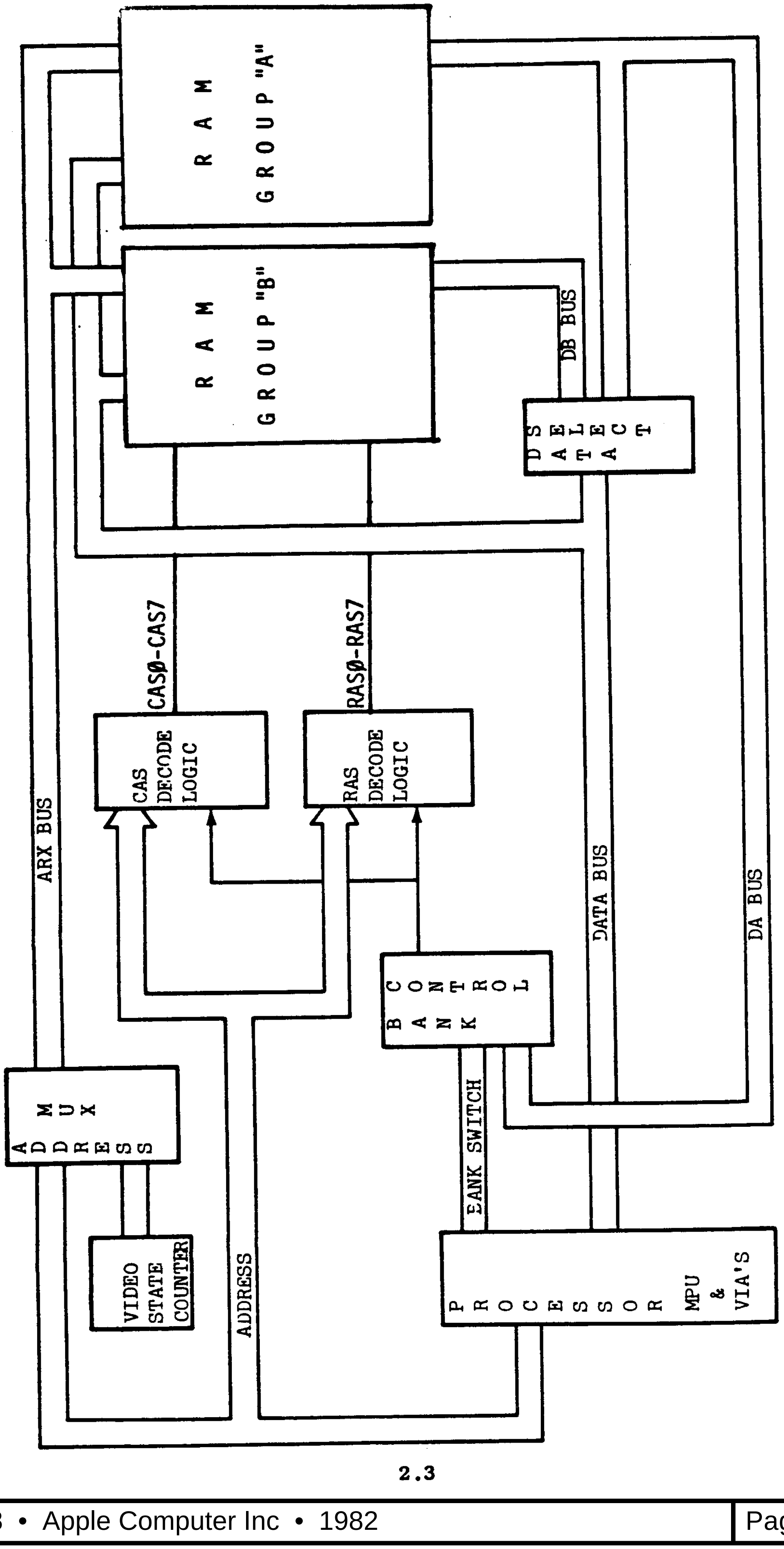


FIG 2.2

In this discussion, the processor is more than just the microprocessor chip; it also contains various external registers and control ROMs which enable the extended addressing and bank switch modes. [Refer to Figure 2.2.]

Very simply, the processor presents an address for memory cycle, which is multiplexed into the address bus, ARX, and is decoded to develop the RAS (Row Address Strobe) and CAS (Column Address Strobe) to the RAM array (RAM group A & B). The direction of the data is controlled by Read/Write*. The selection of which RAM group is being gated to the data bus is controlled by the CAS decode circuits.

The display is memory mapped, the Screen time-shares the RAM on the opposite phase of the processor clock. Both the screen and the processor are running at a 1MHz rate, which means that the RAM is running at a 2MHz rate. One new feature of the Apple /// is that the processor is able to make use of the other "phase" while the screen is off. In other words, any time the screen is off, the processor may run at a full 2MHz rate.

## MEMORY ADDRESSING: BLOCK FLOW

As more detail is added, it is possible to see the basic elements of the complete memory system (For now, we are not considering any to the hardware or I/O).

- In the memory addressing logic of Figure 2.3, the processor now shows the MPU and the two registers for expanded addressing capacity:
  - the bank address register
  - zero page register
- The address circuit is comprised of two sections:
  - the address multiplexer
  - the RAS/CAS decoder
- The RAM array is expanded to show the eight RAMs.

It should be noted that this diagram depicts a 128K system (with a 12V Memory board), and that each of the RAMs shown actually represents eight RAM chips, one for each bit. Each RAM contains 16K bytes. The dotted lines indicate the row of chips that contain the 32K RAM chips. These are actually two 16K RAMs which reside on the one IC package.

## THE PROCESSOR

The MPU is isolated from the rest of the memory by various buffers, muxes, and registers. The mux switches in the Zero Page register whenever the MPU is attempting to reference the zero page.

## BLOCK DIAGRAM MEMORY ADDRESS LOGIC

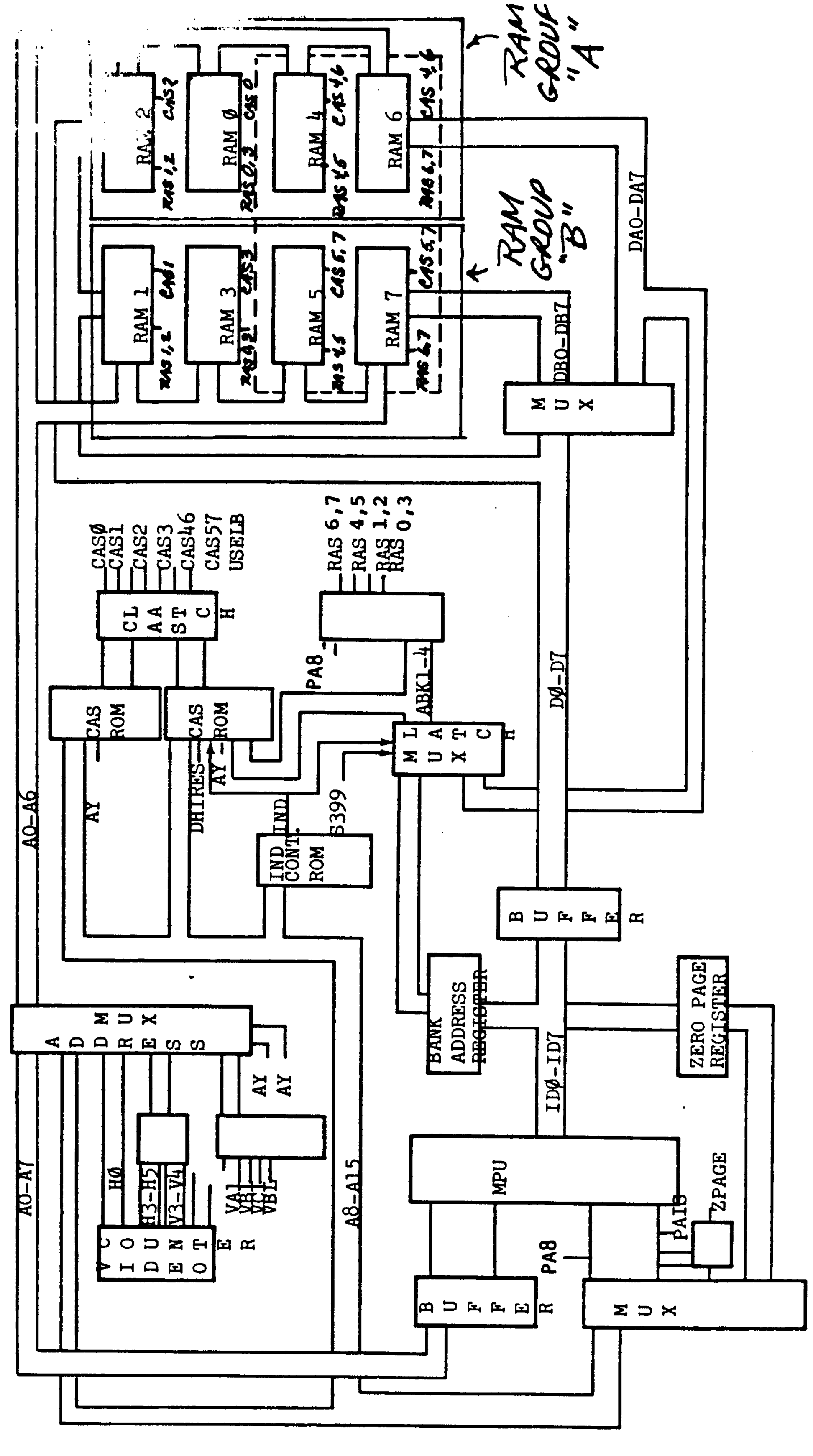


FIG 2.3

- o The register may contain the true zero page or may be set to any of 255 other values, under program control.

- o The zero page register resides in the VIA and is accessed at FFD0. (On the Main Logic Board this is the VIA at location B6.)

- o The Bank Register is located in the other VIA and is accessed at FFEF (at location B5). Zero page selection is independent of bank selection.

## MEMORY ADDRESS MULTIPLEXER

The Memory Address Mux provides the four sets of addresses to the RAM array. They are:

- o MPU RAS
- o Video RAS
- o MPU CAS
- o Video CAS

These are time multiplexed by the four states determined by AX* line which is held at a steady state, allowing the processor full access to the RAM.

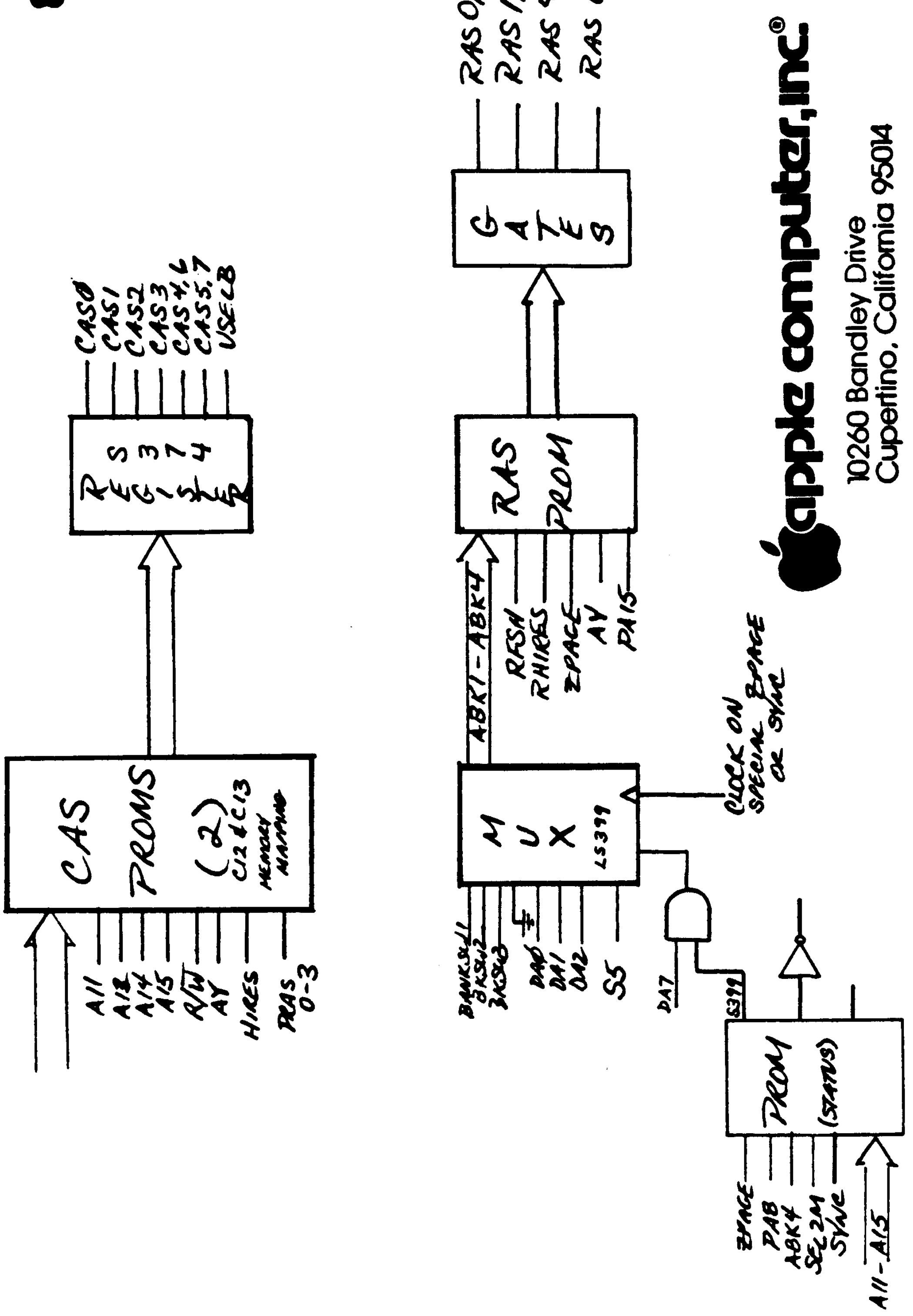
The Video addresses are much the same as in the Apple ][. There is a minor difference in the Summing Circuit, but the technique of condensing undisplayed addresses is the same. There is an additional consideration in the Apple ///, which has a feature requiring additional control of the Video lines. This new feature is called slow scrolling of the screen.

Slow scrolling is accomplished in the Video Mux ROM by the arithmetic offset of the VA, VB, and VC lines. This offset causes characters to be fetched from memory in advance of where the screen actually thinks it is. The character array on the screen shifts up the number of dots determined by the binary weight of the VBX lines. The processor, by monitoring the Vertical Blanking, can then step the VBX lines and scroll the screen by moving in new lines at the bottom, removing the top line, and placing it at the bottom, thus rolling the display.

## RAS/CAS DECODE

The RAS/CAS decode circuit is made from four ROMs, a latch, and a latching mux. The basic inputs to the circuit are:

- o the Address Bus
- o the Bank Switches
- o the Zero Page Select



apple computer, inc.®
10260 Bandley Drive
Cupertino, California 95014

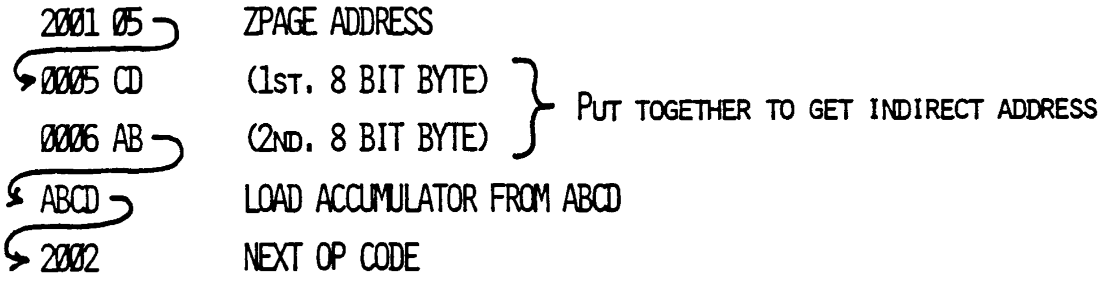
RAS/CAS DECODE LOGIC

BLOCK DIAGRAM

FIG 2.4

## INDIRECT ADDRESSING

```
LDA (Z PAGE), Y                    Y REG = Ø
```

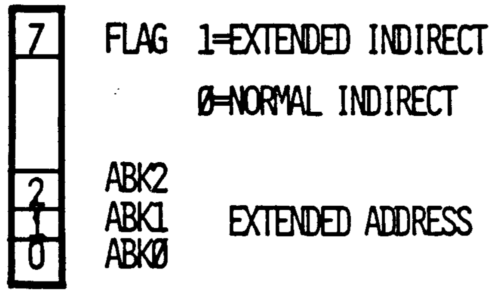


```
2ØØØ B1        LDA (Z PAGE), Y OP CODE
2ØØ1 Ø5        ZPAGE ADDRESS
ØØØ5 CD        (1ST. 8 BIT BYTE) \
                                  } PUT TOGETHER TO GET INDIRECT ADDRESS
ØØØ6 AB        (2ND. 8 BIT BYTE) /
ABCD           LOAD ACCUMULATOR FROM ABCD
2ØØ2           NEXT OP CODE
```

FIG 2.5

IF ZPAGE = 18-1F (IF ZERO PAGE FALLS BETWEEN THE RANGE OF 18-1F)

```
              2ØØØ  B1     LDA (ZPAGE). Y OP CODE
              2ØØ1  Ø5     ZPAGE ADDRESS
              18Ø5  CD
16-BIT READ   18Ø6  AB     14Ø6  82
             2ABCD         LOAD ACCUMULATOR
              2ØØ2         NEXT OP CODE
```



[Refer to Figure 2.4]

Normal or Direct Addressing looks at the address and current bank selection, and enables the appropriate array of 64K RAM. This Direct Addressing always uses RAM 0 and RAM 3. Bank switches determine which of the RAM pairs is used for the other 32K of RAM.

It should be noted that on any read cycle, one RAS line and two CAS lines are selected. In this way, two bytes are always presented to the video circuits and the data selector. During a write cycle only one of each is selected.

The dual byte read usually provides only the information for the new video modes, but there is a new special memory fetch cycle built into the hardware. It is a special extended Indirect Addressing scheme which places the entire memory in virtual access.

By using Indirect Addressing through the zero page containing the 16 bit address, an instruction can address any of the 64K bytes contained in the bank pair. Thus any of the 32K byte RAM banks can be paired with any of their neighbors to form a 64K byte virtual address space.

If, during a zero page reference the zero page register has a value between $18 and $1F ($ means hexadecimal), a special Indirect Mode is called up. This mode looks at the sister fetched data byte on the RAM address bus and also looks at the high order bit. See Figures 2.5 and 2.6.

This special Indirect Mode is determined by the zero page register (X page = Z page EOR $0C) If the bit is zero, the mode is not actualized and the reference continues in a normal manner in the presently selected bank arrangement. But if the High Order Bit (DA7) is high, the bank control mux latch switches to the state determined by the state of the DA0-DA2 lines. This allows the program to have access to another array of special zero pages.

When the system is in this special extended Indirect Mode, the I/O and LSI are totally disabled and the RAM is enabled to the data bus.

## ALTERNATE STACK

Alternate Stack, the new feature of the Apple ///, is not shown in the block diagrams. One of the bits of the Environmental Register (from one of the VIAs) is the Alternate Stack Switch. If the Alternate Stack Switch is selected, the stack associated with that zero page is either the one after the zero page, if the zero page reference is even, or the one previous if the reference is odd (i.e., if zero page is 2B then the stack is located in 2C; if the zero page is 31 then the stack is in 30).

## OTHER BANK SWITCHING

Earlier it was mentioned that the processor can access RAM associated with the addresses that are normally with I/O, ROM, and other circuits. Looking again at the Environmental Register, there are several switches that enable or disable I/O, ROM, and other circuit address decoding. If these switches are selected to disable their associated function, the control ROM, which develops the enable for the RAM data selector, senses the fact that no hardware is being selected and allows RAM data to be read on the bus. No other special enables are needed since RAM is always read for every address presented. It should be noted that a special RAM write enable is used to prevent inadvertant writing into the RAM space associated with the hardware while the hardware is enabled.

## MEMORY & MEMORY ADDRESSING APPENDIX

**The attached figures and illustrations are provided for your reference. Little or no explanation has been provided.**

This Appendix contains:

- THE APPLE /// MEMORY MAP
- MEMORY MAP SPACE ALLOCATIONS
- ADDRESS LOGIC TRUTH TABLE
- 128K 12V MEMORY BOARD: PHYSICAL MEMORY
- THE 5V MEMORY BOARD: PHYSICAL MEMORY
- ADDRESSING LOGIC EXPRESSIONS
- MPU REGISTERS

apple computer inc.

# APPLE /// MEMORY MAP

## SOS MEMORY ALLOCATION

| Location | Assignment |
|---|---|
| 0000-1FFF | SOS and Interpreter Workspace |
| 2000-9FFF | Bank 0 Graphics Page 1 and 2 |
| 2000-9FFF | Bank 1 Program |
| 2000-9FFF | Bank 2 Driver and Interprter |
| A000-BFFF | Interpreter |
| C000-CFFF | I/O or SOS Kernal (Bank switchable to RAM) |
| D000-EFFF | SOS Kernal |
| F000-FFFF | Boot ROM OR SOS Kernal |

## ADDRESS ASSIGNMENT

| ADDRESS<br>(HEX) | ASSIGNMENT<br>(FUNCTION) |
|---|---|
| 0000-00FF | Zero Page |
| 0100-01FF | Stack |
| 0200-02FF | Input Buffer |
| 0300-03FF | Open |
| 0400-07FF | Lo-Res Display (Primary) and text |
| 0800-0BFF | Lo-Res Display (Secondary) and text |
| 0C00-0FFF | Open-Reserved for system space |
| 1000-1FFF | Open |
| 2000-3FFF | Hi-Res Pg1 (Primary) switchable to RAM |
| 4000-5FFF | Hi-Res Pg1 (Secondary) switchable to RAM |
| 6000-7FFF | Hi-Res Pg2 (Primary) switchable to RAM |
| 8000-9FFF | Hi-Res Pg2 (Secondary) switchable to RAM |
| A000-BFFF | Open |
| C000-C07F | System I/O |
| C000 | Keyboard "A" bus data |
| C001-C007 | Same as 0000 but not used |
| C008 | Keyboard "B" bus data |
| C009-C00F | Same as C008 but not used |
| C010 | Keyboard reset |
| C011-C02F | Not used in Apple III |
| C030 | Toggle the speaker like in A-11 |
| C031-C03F | Same as C030 but not used |
| C040-C040 | Sound hardware beeper |
| C04E | Character Ram Disable |
| C04F | Character Ram Enable |
| C050 | Clear Text Mode |
| C051 | Set Text Mode |
| C052 | Clear Mix Mode |
| C053 | Set Mix Mode |
| C054 | Clear PG2 Mode |

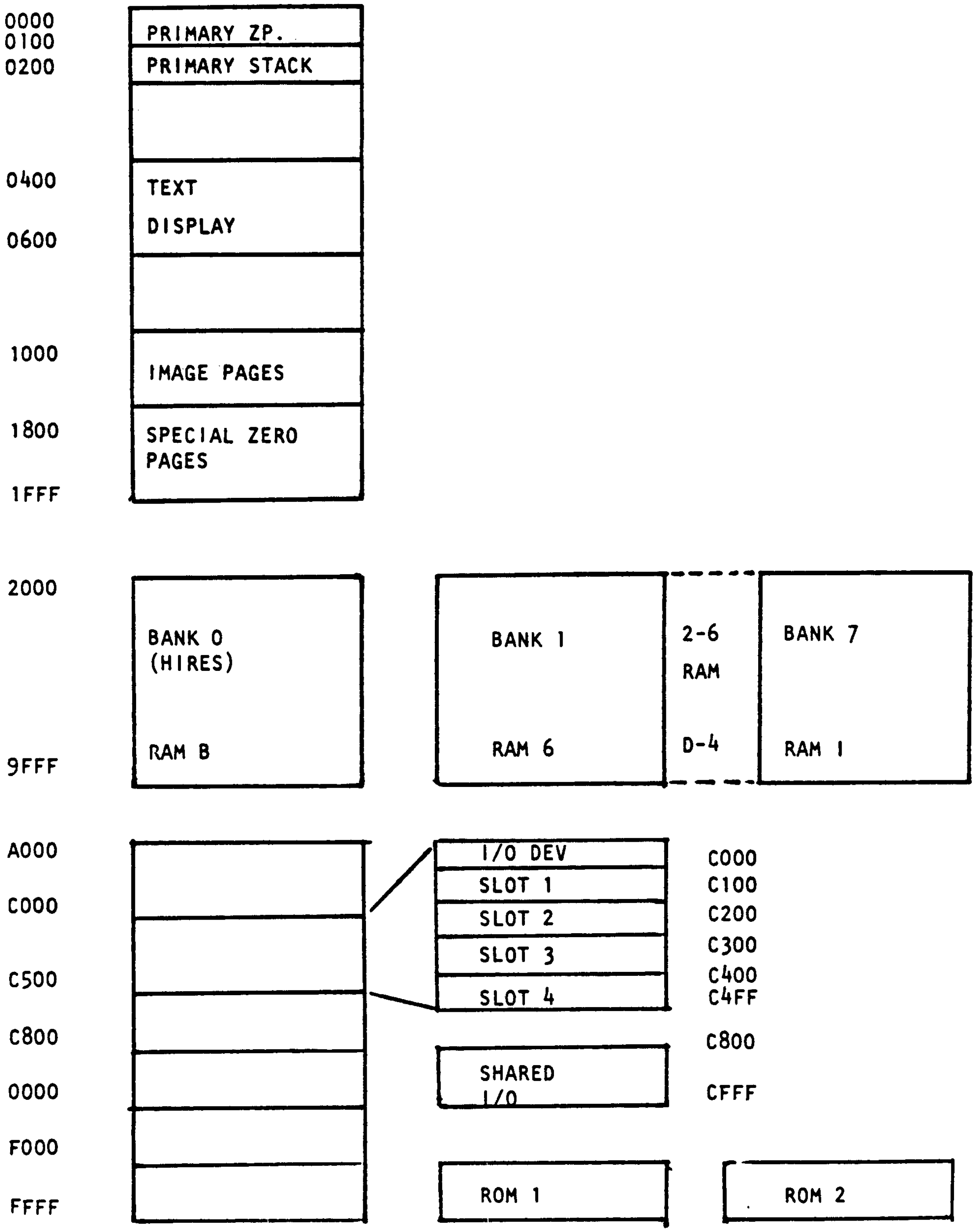
| | | |
|---|---|---|
| C055 | Set PG2 Mode | |
| C056 | Clear HIRES Mode | |
| C057 | Set HIRES Mode | |
| C058 | Clear EMSOT PDL0 | |
| C059 | SET ENSOT PDL0 | |
| C05A | Clear PDL2 (A/D Addr 2) | |
| C05B | Set PDL2 | |
| C05C | Clear PDLEN (A/D Ramp Start) | |
| C05D | Set PDLEN | |
| C05E | Clear AXC0 (A/D Addr 1) | |
| C05F | Set AXC0 | |
| C060,C068 | Read SW0 | |
| C061,C069 | Read SW1/MGNSW | |
| C062,C06A | Read SW2 | |
| C063,C06B | Read SW3/SC0 | |
| C064,C06C | Read IRQ3 | |
| C065,C06D | Read IRQ4 | |
| C066,C06E | Read PDLOT (A/D Ramp Stop) | |
| C067,C06F | Read MUXI (PRAS Control) | |
| C070 | Access Real Time Clock | |
| C071-C07F | Sames as C070 but not used | |
| C080-C0FF | I/O Scot Device Enable | |
| C08F | | |
| C09X | NDevice Select 1 | |
| C0AX | NDevice Select 2 | |
| C0BX | NDevice Select 3 | |
| C0CX | NDevice Select 4 | |
| C0D0 | Clear DS A0 | A0,A1=0,0=no select |
| C0D1 | Set DS A0 | 1,0=Ena 1 Exit |
| C0D2 | Clear DS A1 | 0,1=Ena 2 Exit |
| C0D3 | Set DS A1 | 1,1=Ena 3 Exit |
| C0D4 | Clear Enable 1 Int | |
| C0D5 | Set Enable 1 Int | |
| C0D6 | Clear Side 2 | |
| C0D7 | Set Side 2 | |
| C0D8 | Clear SCR | |
| C0D9 | Set SCR | |
| C0DA | Clear ENCWRT | |
| C0DB | Set ENCWRT | |
| C0DC | Clear ENSEL | |
| C0DD | Set ENSEL | |
| C0DE | Clear ENSIC | |
| C0DF | Set ENSIO | |
| C0E0 | Clear DPH0 (also VAL) | |
| C0E1 | Set DPH0 | |
| C0E2 | Clear DPH1 (also VBI) | |
| C0E3 | Set DPH1 | |
| C0E4 | Clear DPH2 (also VCI) | |
| C0E5 | Set DPH2 | |
| C0E6 | Clear DPH3 | |
| C0E7 | Set DPH3 | |
| C0E8 | Disable Motor Drive (strt 2 sec to) C0E9 Enable Motor Drive | |
| C0EA | Enable Ext | |
| C0EB | Enable Int | |

| | |
|---|---|
| C0EC | Clear Q6 Note:Q6,Q7 control read |
| C0ED | Set Q6 write, and sense write |
| C0EE | Clear Q7 protect |
| C0EF | Set Q7 |
| C0F0r | 6551 Rec Data Reg |
| C0F0w | 6551 Xmit Data Reg |
| C0F1r | 6551 Status Reg |
| C0F1w | 6551 Program reset |
| C0F2r/w6551 | Command Reg |
| C0F3r/w6551 | Control Reg |
| CIXX | NIO Select 1 |
| C100-C7FF | I/O Slot individual ROM Space |
| C100 | Slot 1 Firmware |
| C1FF | |
| C2XX | NIO Select 2 |
| C200-C2FF | Slot 2 Firmware |
| C3XX | NIO Select 3 |
| C300-C3FF | Slot 3 Firmware |
| C3FF | |
| C4XX | NIO Select 4 |
| C400-C4FF | Slot 4 Firmware |
| C500-C7FF | Run Space only |
| C800-CFFF | Expansion Rom Firmware |
| D000-DFFF | Open (system software) |
| E000 | Bank switchable between Rom and Ram |
| FFCX | Always Ram |
| FFD0 | Port B VIA-73 "Zero Page Reg" |
| FFD1 | Port A VIA-73 |
| FFD2 | DDR B VIA-73 |
| FFD3 | DDR A VIA-73 |
| FFD4 | Timer 1 low Latch (w)/Counter (r) |
| FFD5 | Timer 1 High Counter |
| FFD6 | Timer 1 Low Latches |
| FFD7 | Timer 1 High Latches |
| FFD8 | Timer 2 Low Latch (w)/Counter (r) |
| FFD9 | Timer 2 High Counter |
| FFDA | Shift Register (serial print) |
| FFDB | Aux Control Reg VIA-73 |
| FFDC | Peripheral Control Register |
| FFDD | Interrupt Flag Register (73) |
| FFDE | Interrupt Enable Register (73) |
| FFDF | ORA/IRA With no handshake |
| FFE0 | Port B (97) (sound and slot NMI) |
| FFE1 | Port A (97) Banksw and IRQ's |
| FFE2 | DDR B (97) |
| FFE3 | DDR A (97) |
| FFE4 | Timer 1 Low Latch/Counter |
| FFE5 | Timer 1 High Counter |
| FFE6 | Timer 1 Low Latches |
| FFE7 | Timer 1 High Latches |
| FFE8 | Timer 2 Low Latch/Counter |
| FFE9 | Timer 2 High Counter |
| FFEA | Shift Register (97) |
| FFEB | Aux Control Register (97) |

| | |
|---|---|
| FFEC | Peripheral Control Register (97) |
| FFED | Interrupt Flag Register (97) |
| FFEE | Interrupt Enable Register (97) |
| FFEF | ORA/IRA with no handshake |
| FFE0 | Ram/Rom Bank |
| FFF1 | E/O Bank Switch |
| FFF2 | 2 MHz/MHZ Mode Switch |
| FFF3 | Hires Bank Switch |
| FFF4 | Screen Enable |
| FFF5 | Display Modes |
| FFF6 | Zero Page Register |
| FFF7 | Interrupt Control |
| FFFF | |

apple computer inc.

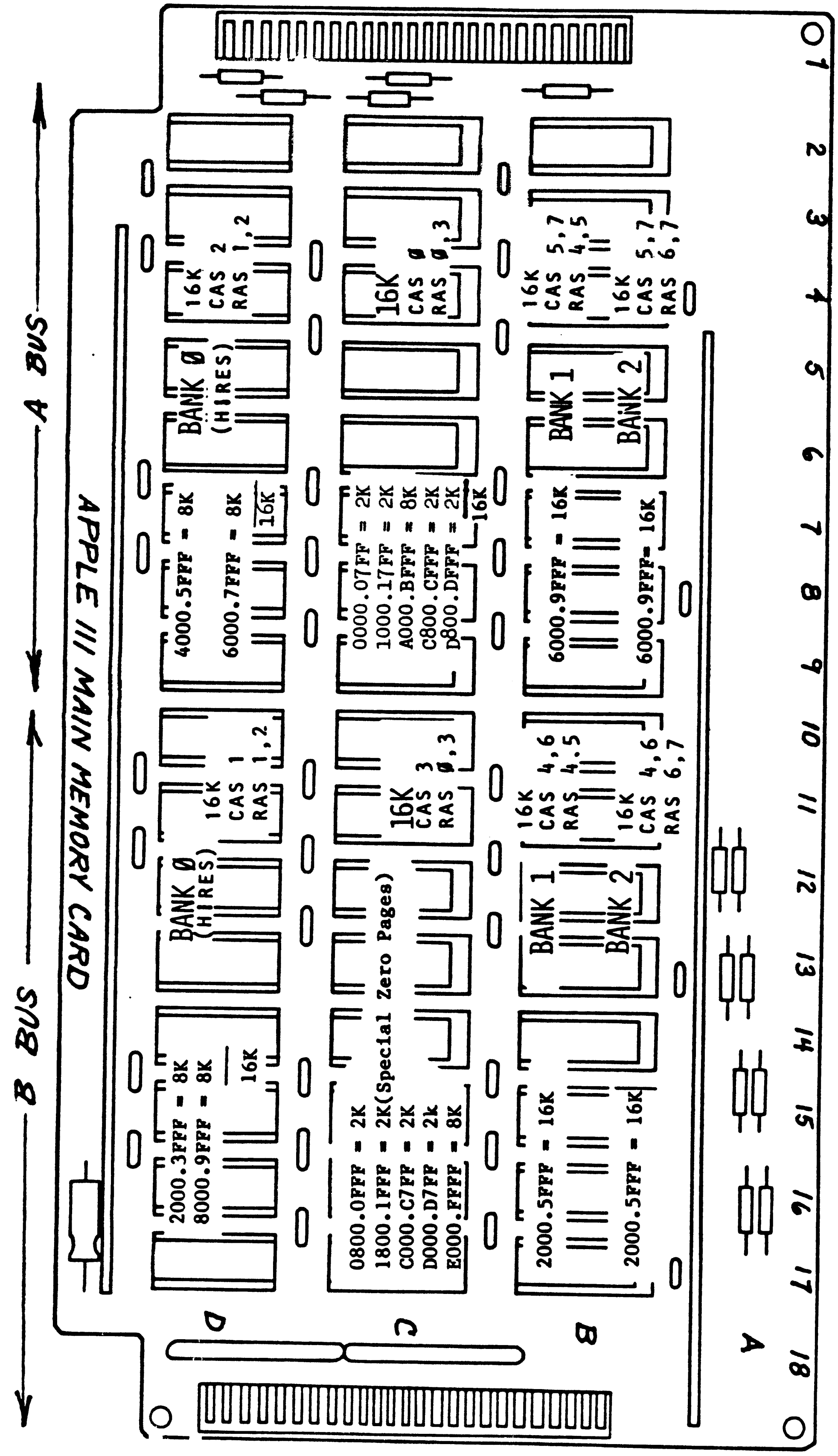
# MEMORY MAP SPACE ALLOCATIONS



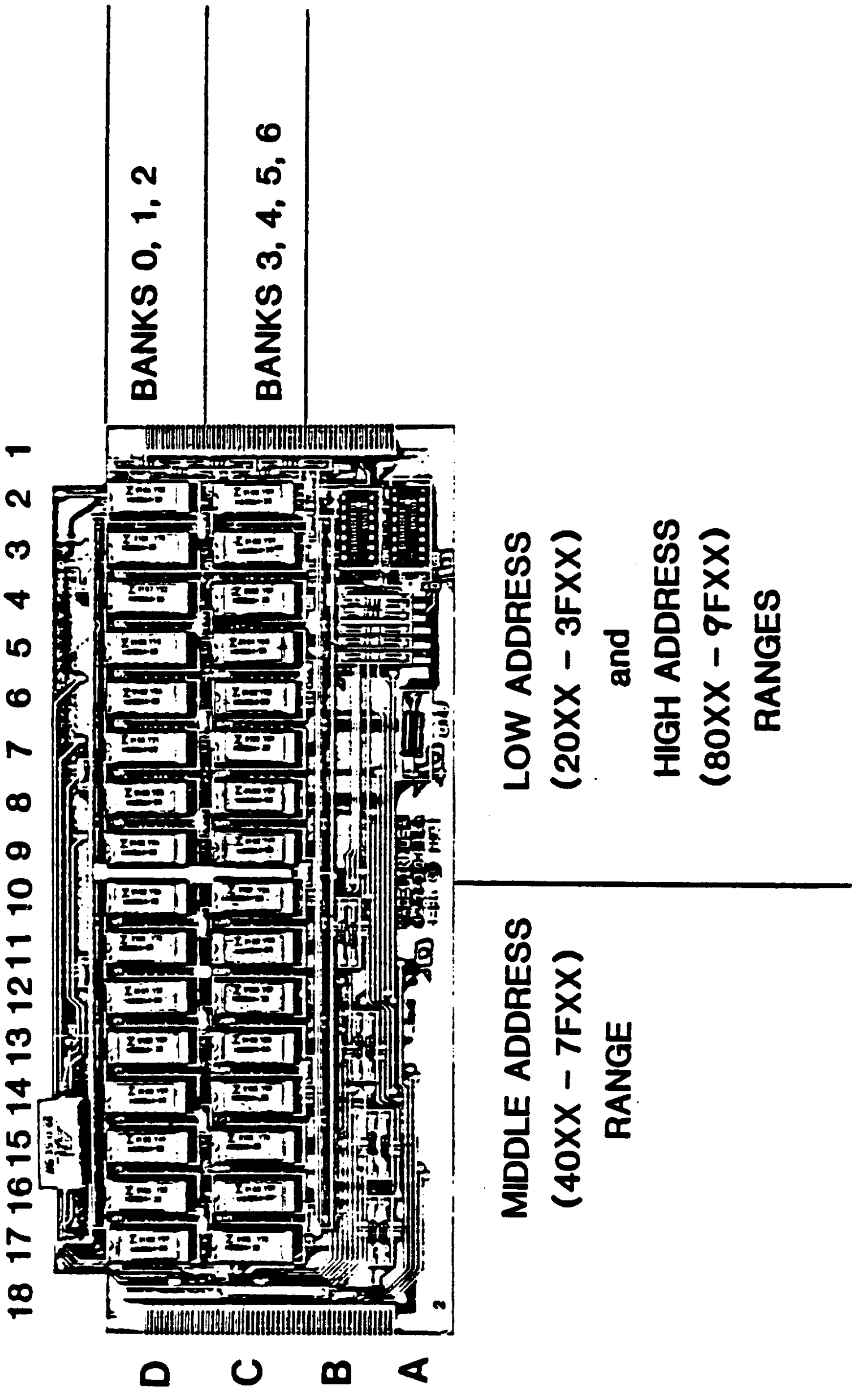
** FFCX, FFDX, FFEX ARE ENVIRONMENT ADDRESSES

| $\overline{AX}$ | $\overline{AY}$ | ARφ | AR1 | AR2 | AR3 | AR4 | AR5 | AR6 | STATE FUNCTION |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | Aφ | A1 | A2 | A3 | A4 | A5 | A7 | MPU GEN RAS ADD. |
| 0 | 1 | Hφ | H1 | H2 | Σ1 | Σ2 | Σ3 | Vφ | VIDEO GEN RAS ADD |
| 1 | 0 | A6 | A8 | A9 | $A10 \oplus \overline{PCAS\phi}$ | $A11 \oplus \overline{PCAS\phi}$ | A12 | $\overline{PCAS1} \oplus (A15 \oplus \overline{PCAS\phi,3})$ | MPU GEN CAS ADD |
| 1 | 1 | Σ4 | V1 | V2·V5 | MUX1 | MUX2 | MUX3 | $\overline{PG2}$ | VIDEO GEN CAS ADD |

ADDRESS LOGIC TRUTH TABLE

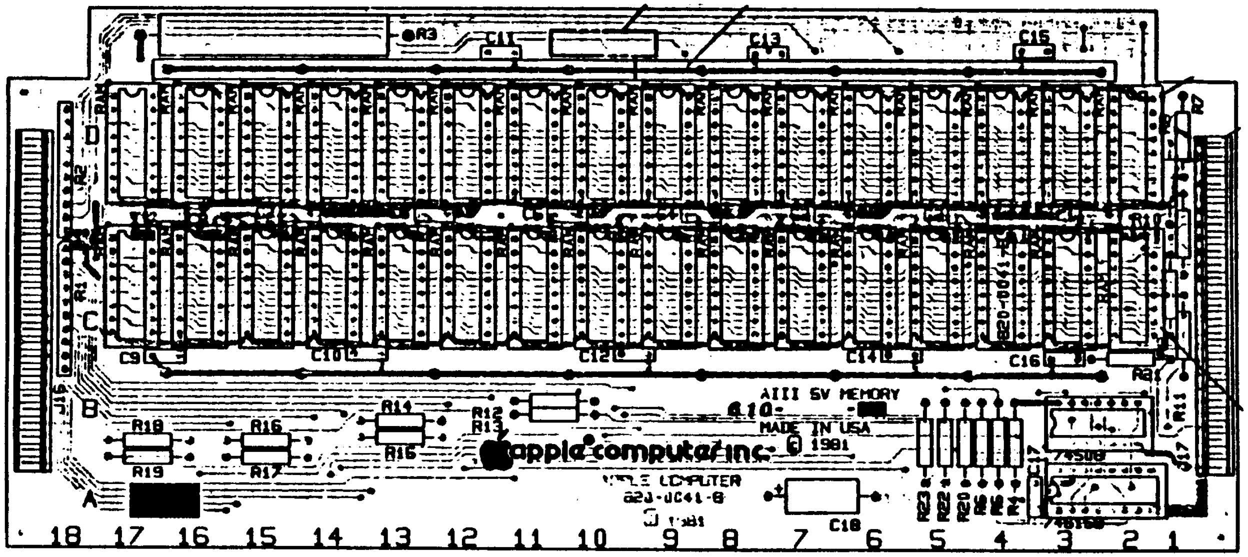

APPLE /// MAIN MEMORY CARD
A BUS
B BUS
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
D
C
B
A
16K CAS 2 RAS 1,2
BANK Ø (HIRES)
4000.5FFF = 8K
6000.7FFF = 8K
16K
16K CAS 1 RAS 1,2
BANK Ø (HIRES)
2000.3FFF = 8K
8000.9FFF = 8K
16K
16K CAS Ø RAS Ø,3
0000.07FF = 2K
1000.17FF = 2K
A000.BFFF = 8K
C800.CFFF = 2K
D800.DFFF = 2K
16K
16K CAS 3 RAS Ø,3
0800.0FFF = 2K
1800.1FFF = 2K(Special Zero Pages)
C000.C7FF = 2K
D000.D7FF = 2k
E000.FFFF = 8K
16K CAS 5,7 RAS 4,5
16K CAS 5,7 RAS 6,7
BANK 1
BANK 2
6000.9FFF = 16K
6000.9FFF = 16K
16K CAS 4,6 RAS 4,5
16K CAS 4,6 RAS 6,7
BANK 1
BANK 2
2000.5FFF = 16K
2000.5FFF = 16K

2.19


BANKS 0, 1, 2
BANKS 3, 4, 5, 6
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
D
C
B
A
LOW ADDRESS
(20XX – 3FXX)
and
HIGH ADDRESS
(80XX – 9FXX)
RANGES
MIDDLE ADDRESS
(40XX – 7FXX)
RANGE

apple computer inc.

## THE 5V MEMORY BOARD ( 256K )



USES: PROM 342-0061
342-0063

64K RAM uT4264-20
D264A-15

## HOW TO READ PROM (ROM) LOGIC EXPRESSIONS

X′ -- The single quote at the end of an expression means that the state of the signal is true when low, or it may represent the inversion of the state of the signal.

* -- Defines a logic AND operation. The expressions on either side of the asterisk is ANDed.

+ -- Defines a logic OR operation. The expression on either side of the asterisk is OR′d.

() -- Defines the boundaries of a logic expression. A new expression is defined by what ever is inside of the brackets.

RULES FOR INTERPRETATION

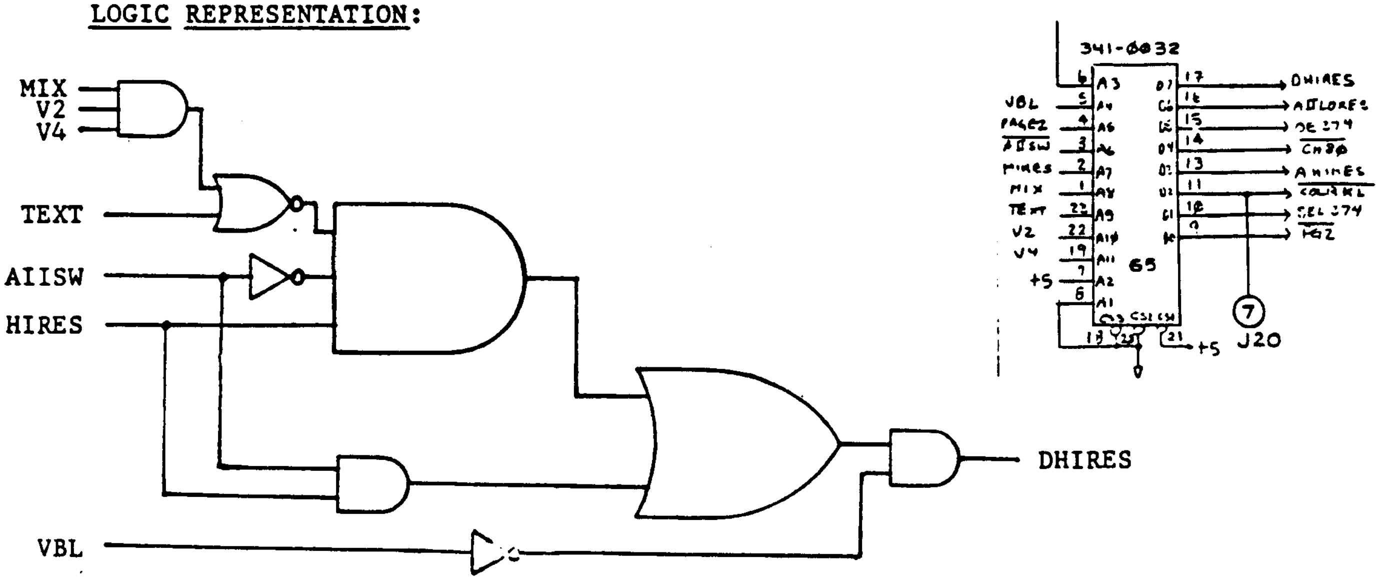
1. Always interpret (transform) the expression within the brackets first.
2. Interpret AND (*) logic operations before OR (+) operations.

EXAMPLE:

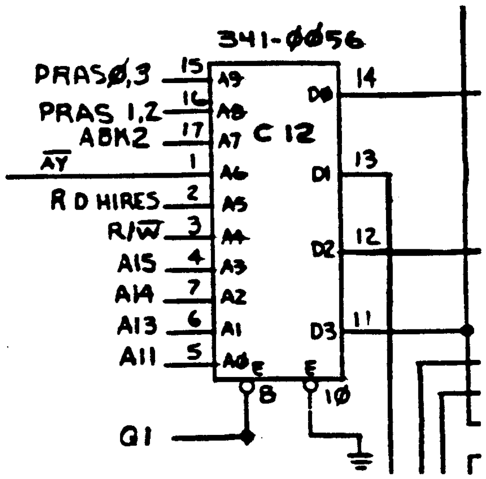
| Given: | INPUTS: | AIISW′ | HIRES | TEXT |
|---|---|---|---|---|
| | | MIX | V2 | V4 |
| | | VBL | | |

OUTPUT: DHIRES = (AIISW*HIRES*(TEXT+MIX*V2*V4)′+AIISW′*HIRES)*VBL′

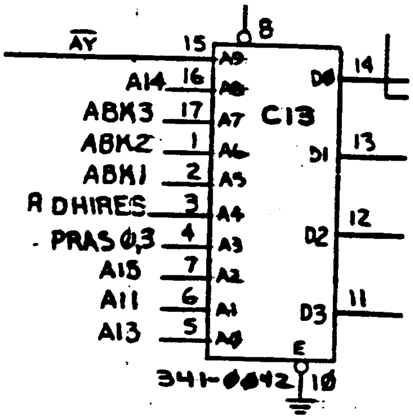
LOGIC REPRESENTATION:



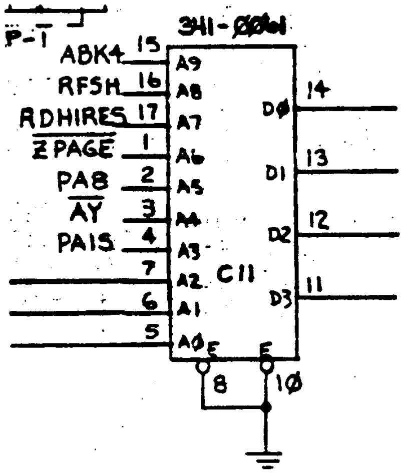
```
A=A11
B=A13
C=A14
D=A15
E=R/WN
F=DHIRES
G=AY'
H=ABK2
I=PRAS1,2
J=PRAS0,3
D0=PCAS0'
D1=PUSELB
D2=PCAS3
D3=PCAS3'

PCAS0'=(PRAS0, 3 * (DHIRES' *AY' +AY* (A15'* A14' * A13' *A11' *R/WN' +
      A15' *A14' *A13'* R/WN+A15*A14'*A13' *A11)))'


PUSELB=PRAS0, 3 * (A15' *A14' *A13' *A11 +A15 *A14 *A13' *A11' +A15 *A14
      *A13) +PRAS0, 3 *PRAS1, 2 *(A15' *A14' *A13+A15* A14' *A13') +PRAS0,
      3 *PRAS1,2 '*(A15' *A14' *A13+A15' *A14 *A13') +PRAS0, 3 '* PRAS1, 2
      * (A14' *A13' +A14 *A13) +PRAS0, 3 ' *PRAS1, 2 ' *A14'

PCAS3=PRAS0, 3 * (DHIRES ' *AY ' + AY* (A15' *A14' *A13' *A11 + A15 *A14
     A13' *A11' +A15 *A14 *A13))

PCAS3'=(PRAS0,3 * (DHIRES ' *AY' +AY* (A15' *A14' *A13' *A11+A15 *A14 *A13'
      *A11' +A15* A14* A13)))'
```



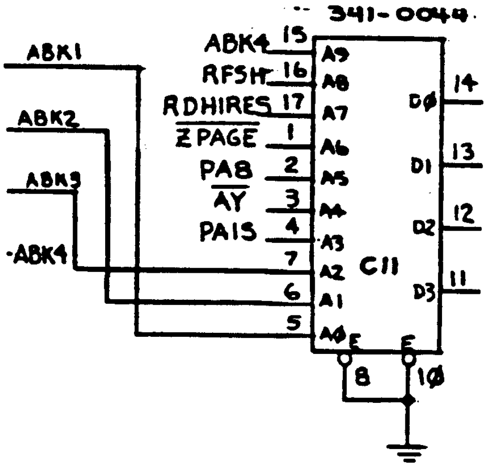
```
A=A13
B=A11
C=A15
D=PRASO,3
E=DHIRES
F=ABK1
G=ABK2
H=ABK3
I=A14
J=AY'
D0=PCAS4,7'
D1=PCAS5,6'
D2=PCAS1'
D3=PCAS2

PCAS4,7'= (AY*PRASO, 3 * (ABK1*ABK2*ABK3'+ABK3* (ABK1' + ABK2'))* (A15'
        *A14) +AY* PRASO,3 '* (ABK1' *ABK2*ABK3'*A14+ABK1*ABK2*ABK3'+ABK2'
        *ABK3+ABK1' *ABK2*ABK3*A15') *(A14' *A13+A14*A13))'

PCAS5,6'= (AY8PRASO,3 *(ABK1*ABK2*ABK3' +ABK3* (ABK1' +ABK2'))* (A15'*A14'
        *A13+A15*A14' *A14'*A13') +AY*PRASO, /3 '*(ABK1'*ABK2*ABK3' *A15+
        ABK1*ABK2*ABK3*'+ABK2'*ABK3+ABK1'*ABK2*ABK3*A15') *(A14'*A13'A14*
        A13))'

PCAS1' = (DHIRES *AY'+AY*PRASO, 3 * (ABK3' * (ABK1'+ABK2') +ABK1*ABK2*ABK3)
       * (A15'* A14' *A13+A15*A14' *A13') +AY*PRASO, 3 '*(ABK2' *ABK3'+ABK1'
       *ABK2*ABK3'*A15') * (A14'* A13'+A14*A13))'

PCAS2' = (DHIRES *AY' +AY*PRASO,3 *(ABK3'* (ABK1' +ABK2') +ABK1*ABK2*ABK3)
       *A15'* A14+AY*PRASO,3 '*(ABK2'*ABK3'+ABK1'*ABK2*ABK3'*A15) *(A14'
       *A13+A14*A13'))'
```



```
RAS 258

A=ABK1
B=ABK2
C=ABK3
D=PA15
E=AY'
F=PA8
G=ZPAGE'
H=DHIRES
I=RFSH
J=ABK4
D0=PRAS0,3
D1=PRAS1,2
D2=PRAS4,5
D3=PRAS6,7

PRAS0,3 = AY'* (DHIRES' +RFSH) + ((ABK4* (ZPAGE*PA8')')'+ABK4* (ZPAGE*PA8)'
        '*ABK1* ABK2*ABK3)*AY PRAS1,2=AY'+AY PRAS4,5=AY'+AY*(ABK4* (ZPAGE*
        PA8')')'*(ABK1*ABK2*ABK3'+ABK1'*ABK2*ABK3' *PA15+ABK1'* ABK2'*PA15'
        +ABK1*ABK2*ABK3)  PRAS6,7 +AY'*DHIRES'+AY* (ABK4*(ZPAGE*PA8')')'*
        (ABK1* (ABK2' +ABK3')) +AY*ABK4* (ZPAGE*PA8')'* (PA15'*ABK1* (ABK2'
        +ABK3') +PA15*ABK1'* (AKB2' + ABK3'))
```



```
RAS.2.TEXT

A-ABK1
B=ABK2
C=ABK3
D=PA14 5
E-AY'
F=PA8
G=ZPAGE
H=DHIRES
I=RFSH
J=ABK4
D0=PRAS0,3
D1-PRAS1,2
D2=PRAS4,5
D3=PRAS6,7

PRAS0, 3 =AY'* (DHIRES'+RFSH) + ((ABK4* (ZPAGE*PA8')') ' +ABK1*ABK2*ABK3)
          *AY

PRAS1,2  =AY'* (DHIRES+RFSH) =AY* (ABK1'* AKB2'*ABK3'* (ABK4* (ZPAGE*PA8')
          '**PA15)'+ABK1*ABK2*ABK3) +AY*ABK3'* (ABK1'* ABK2* ABK4* (ZPAGE*
          PA8') '*PA15+ABK1*ABK2*(ABK4* (ZPAGE*PA8') *PA15)')

PRAS4,5 =RFSH*AY'+AY*ABK2'*ABK3'* (ABK1'*ABK4* (ZPAGE*PA8') 'PA15+ABK1*
        (ABK4* (ZPAGE*PA8') '*PA15)')

PRAS6,7 =RFSH*AY'+AY*ABK3'* (ABK1*?BK2'*ABK4* (ZPAGE*PA8') '*PA15+ABK1'*
        ABK2* (ABK4* (ZPAGE*PA8') '*PA15)')
```

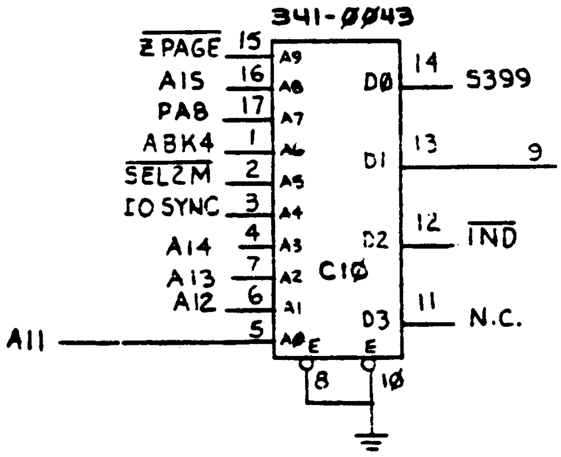


A=A1<br>
B=A12<br>
C=A13<br>
D=A14<br>
E=IOSYNC<br>
F=SEL2M'<br>
G=ABK4<br>
H=PA8<br>
I=15<br>
K=PAGE'<br>
D0=S399<br>
D1=PRDY'<br>
D2=IND'

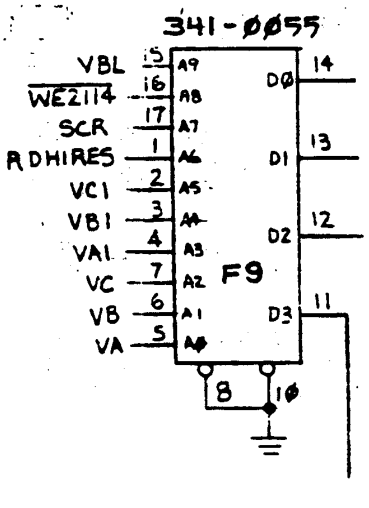
S399=ZPAGE*PA8' *A15' *A14' *A13' *A12* A11

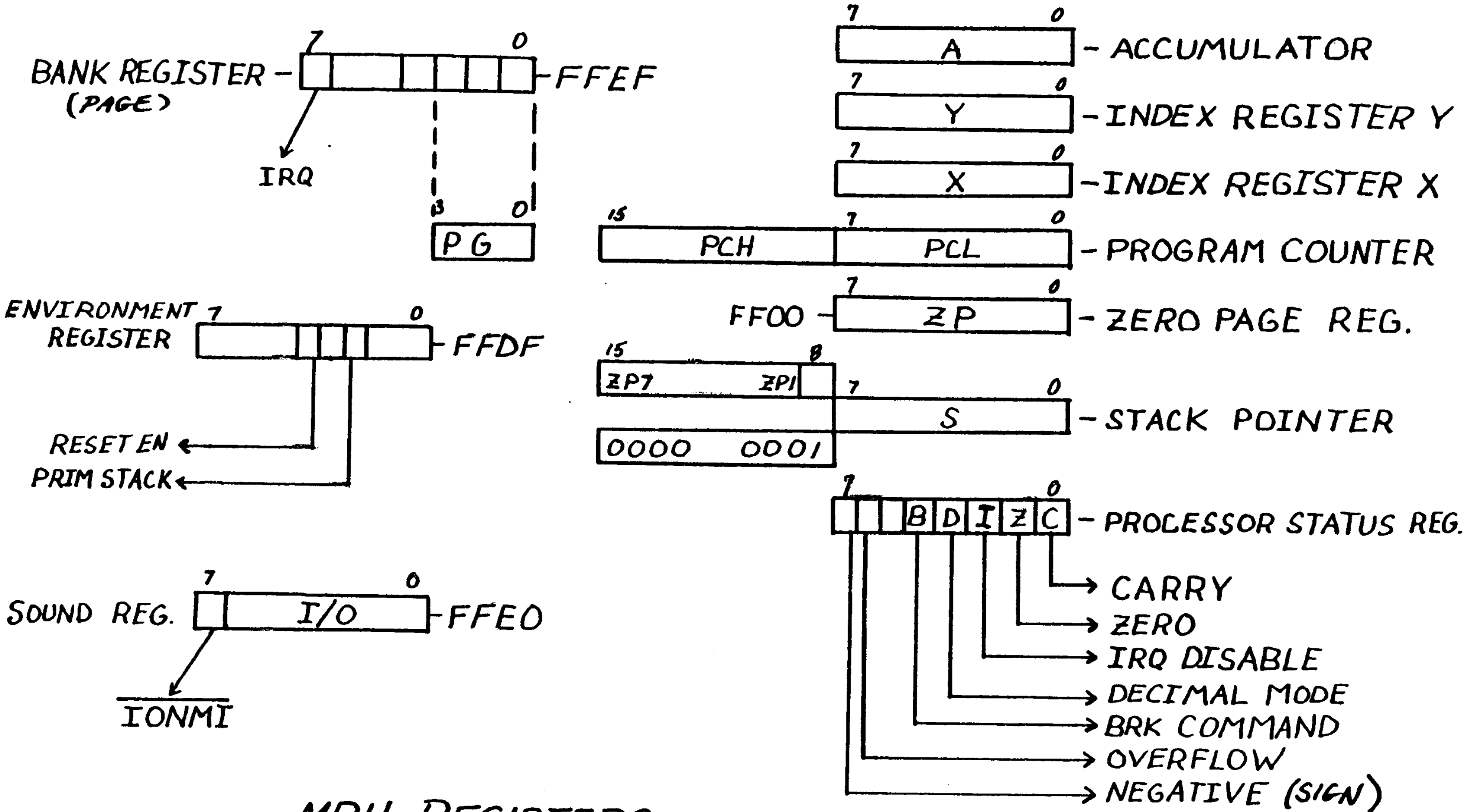
PRDY'=IOSYNC *SEL2M *ABK4

IND'= (ABK4* (ZPAGE* PA8') ')'



```
U175 1. TEXT

A=VA
B=VB
C=VC
D=VA1
E=VB1
F=VC1
G=DHIRES
H=SCR
I=WE2114'
J=VBL
DO=MUX1
D1=MUX2
D2=MUX3
D3=ENHREG'


MUX1=(DHIRES' +SCR * (VA* VA1' +VA' *VA1) + SCR' *VA) * (VBL' + WE2114)
    +VBL * WE2114'

MUX2=DHIRES* (SCR* (VA* VA1* (VB* VB1' +VB'* VB1)'+ (VA* VA1)'* (VB* VB1'
    +VB * VB1) ) +VB* SCR')

MUX3=DHIRES * (SCR* ((VA* VA1* (VB+ VB1) +VB* VB1) * (VC* VC1'+VC'* VC1)
    '+(VA* VA1* (VB+ VB1) +VB * VB1) '*(VC * VC1) ) +VC* SCR')

ENHREG'=DHIRES' *WE2114'
```





## MPU REGISTERS

## I/O DEVICE STROBES

| C0XX | |
|---|---|
| 0X | KEYBOARD READ |
| IX | RESET KEYBAORD |
| 2X | N.C . |
| 3X | Speake Toggle |
| 4X | Beeper |
| 5X | Display Regiater |
| 6X | Switch Register |
| 7X | Real Time Clock |
| 8X | N.C. |
| 9X | |
| AX | Slot 1-4 |
| BX | DEVSEL |
| CX | |
| DX | I/O CONTROL REGISTER |
| EX | DISK REGISTER |
| FX | ACIA |

## KEYBOARD DATA

| C000 | |
|---|---|
| 0 | ASCII0 |
| 1 | ASCII1 |
| 2 | ASCII2 |
| 3 | ASCII3 |
| 4 | ASCII4 |
| 5 | ASCII5 |
| 6 | ASCII6 |
| 7 | KBO FLAG |

## KEYBOARD STATUS

| C008 | |
|---|---|
| 0 | ANY KEY DOWN |
| 1 | SHIFT |
| 2 | CONTROL |
| 3 | CAPSLOCK |
| 4 | APPLE I |
| 5 | APPLE II |
| 6 | KEYBOARD PRESENT |
| 7 | ASCII7 |

## VIA

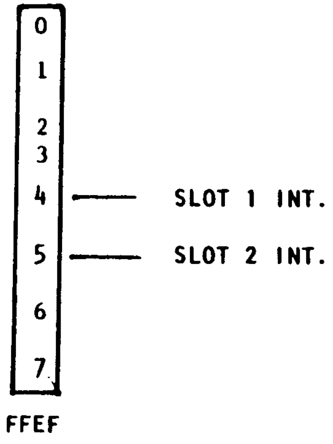
| FFDX | |
|---|---|
| CA1 | SLOT INT. |
| CA2 | SWI/MARGIN INT. |
| CB1 | SERVICE CLOCK |
| CB2 | SERVICE DATE/PDLO |

$\left.\begin{array}{l}\text{SWI/MARGIN INT.}\\ \text{SERVICE CLOCK}\\ \text{SERVICE DATE/PDLO}\end{array}\right\}$ SILENTYPE

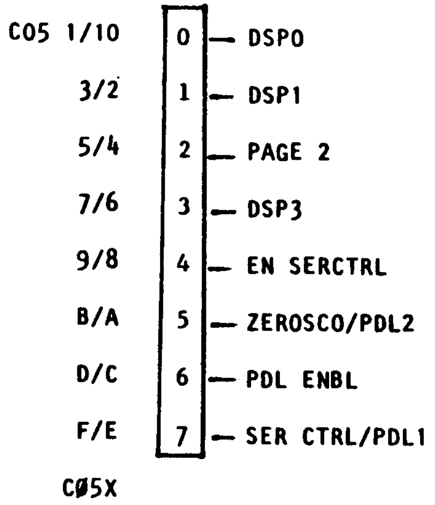
## VIA

| FFEX | |
|---|---|
| CA1 | RT CLOCK INT. |
| CA2 | KEYBOARD UNIT |
| CB1 | VERTICAL BLANKING |
| CB2 | VERTICAL BLANKING |

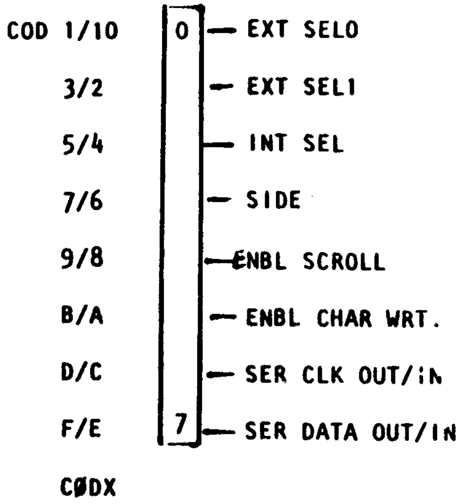
## BANK PAGE REGISTER



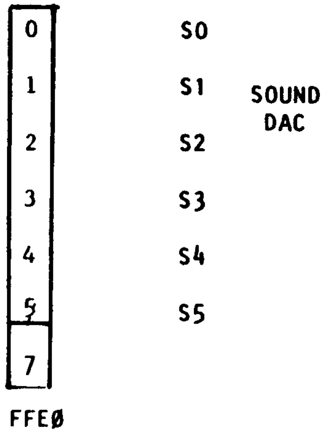
## DISPLAY REGISTER



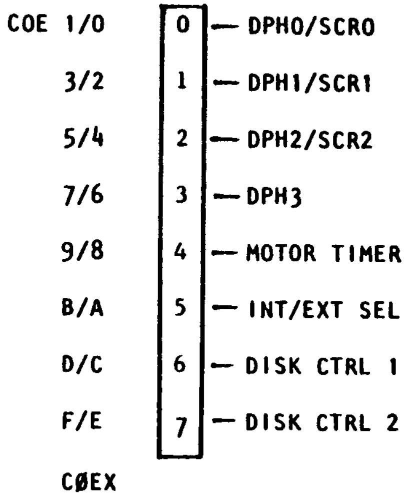
## I/O CONTROL REGISTER



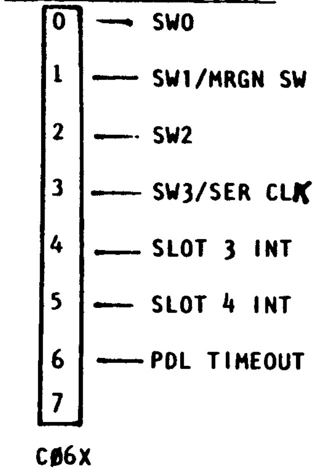
## SOUND REGISTER



## DISK REGISTER



## SWITCH REGISTER


0 — SW0
1 — SW1/MRGN SW
2 — SW2
3 — SW3/SER CLK
4 — SLOT 3 INT
5 — SLOT 4 INT
6 — PDL TIMEOUT
7
CØ6X