## The IM Underground Volume #1

#### Introduction

The Geneva-12, Geneva-9, Athens-18, and Monaco-9 fonts are needed to correctly view and print the files on this disk. The applications MacPaint, MacWrite, and Consulair's Edit are needed to fully view and print the files. MacWrite could be substituted for Edit as long as the ruler is set up to emulate 8 character tab settings and a font such as Monaco-9 is used.

The disk contains the following files:

### **Schematics**

Memory (MacPaint)
Control Logic (MacPaint)
CPU & ROM (MacPaint)
IWM & VIA (MacPaint)
Serial Interface (MacPaint)
Address Multiplexer (MacPaint)
Keyboard & Mouse (MacPaint)
128K/512K (MacPaint)

## <u>Hardware</u> Info

Keyboard Hardware (MacWrite) Video Hardware (MacWrite) Sound Hardware (MacWrite) VIA Hardware (MacWrite) Mouse Hardware (MacWrite)

# <u>Keyboard ROM</u>

8021PAGEO (Edit) 8021PAGE1 (Edit) 8021PAGE2 (Edit) 8021PAGE3 (Edit) Register Usage (Edit)

### <u>Charts</u>

Video Timing (MacPaint) Hacker Chart (MacWrite) Empty Folder (Yes -- it is empty!)

## System Info

The System Revealed (MacWrite)
The Small System (MacWrite)
Resource Flags (MacWrite)
512K Upgrade (MacWrite)
Beyond 512K (MacWrite)

We strongly suggest that the reader also obtain (if they don't already have it) the software documentation *Inside Macintosh* from Apple as it provides valuable information about the inner workings of the operating system and device drivers.

Please notify us of any errors and/or omissions so that future releases can be corrected.

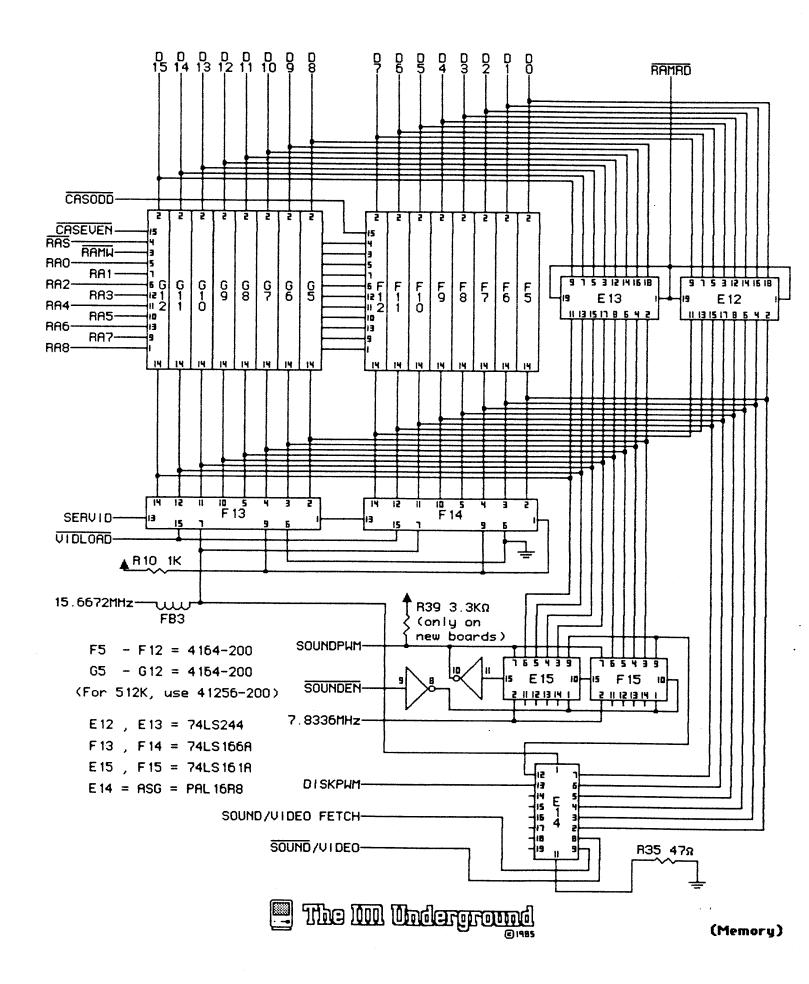
The keyboard ROM listings are from an 8021 MPU that did not have Apple's copyright in or on it. The circuit board schematics are traced from several Macintosh logic boards.

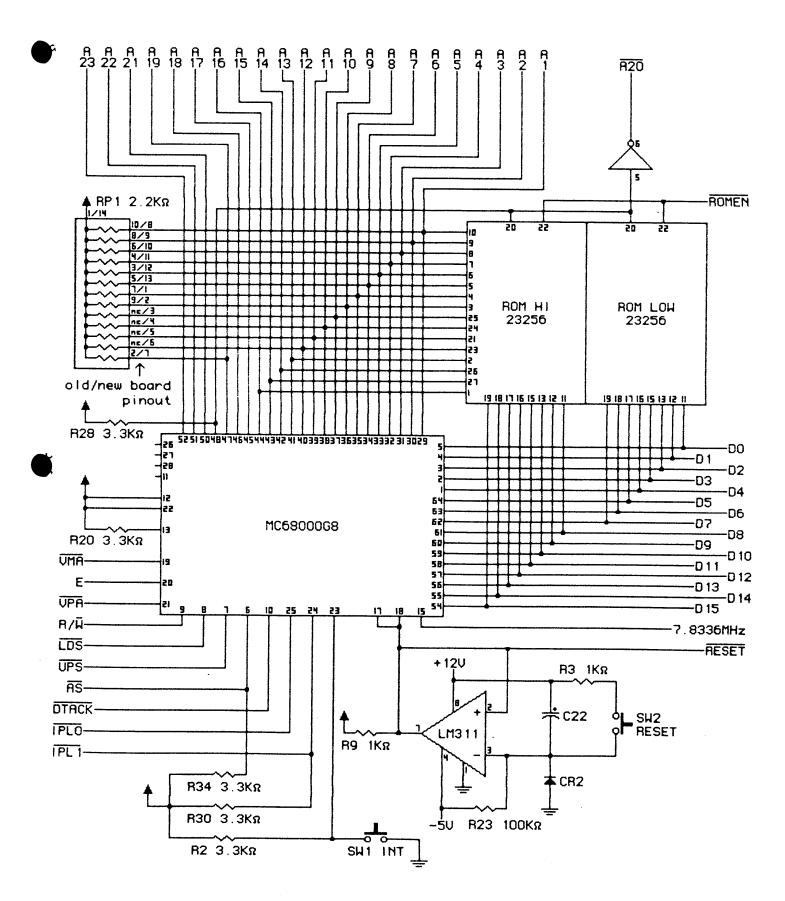
Apple, Apple Care, Finder, ImageWriter, Inside Macintosh, Lisa, Mac, Macintosh, Macintosh XL, MacPaint, MacsBug, MacWorks, MacWrite, QuickDraw, Resource Editor, and Resource Mover are trademarks of Apple Computer, Inc.

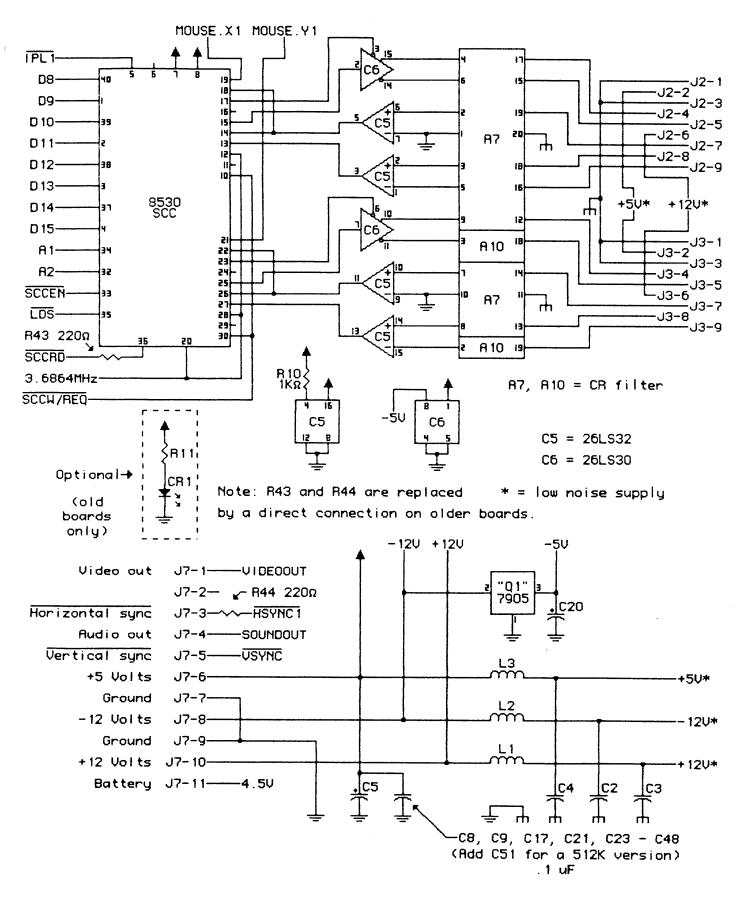
Consulair is a trademark of Consulair, Inc.

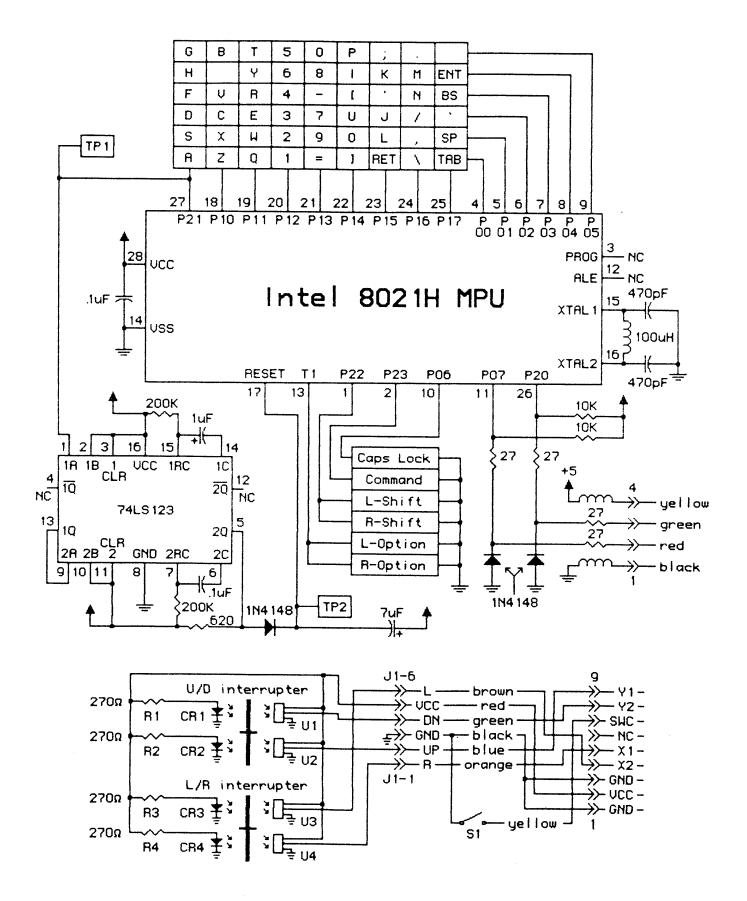
Intel is a trademark of Intel Corporation.
Fedit is copyright by John Mitchell.
Motorola is a trademark of Motorola, Inc.
Rockwell is a trademark of Rockwell International.
Synertek is a trademark of Synertek, Inc.
Zilog is a trademark of Zilog, Inc.

The IM Underground Volume #1 is available for \$25 from:
The IM Underground
715 Church Street #16
Ann Arbor, Michigan 48104









### Keyboard Hardware

This document covers the Macintosh keyboard and it's communications protocol. Electrically, the keyboard appears as a two dimensional array of open/closed switches that can be scanned by sending a signal out one set of wires and then checking a second set for the signal to return. If a key is depressed, it will connect the two wires attached to it thus causing a signal sent out one of the wires to be returned via the other. In the Macintosh, the actual arrangement consists of a 6 by 9 array where the 9 wire set is used to output a signal and the 6 wire set is used to detect this same signal. This array provides locations for up to 54 keys of which 52 are used in the US version (53 on international versions).

Such an arrangement has one intrinsic problem – when more than two keys are depressed, it is possible for "ghost" keys to appear. For example, when three keys in an L shaped pattern are depressed, a fourth key will "appear" at the last vertice of a square containing these four keys. Therefore, certain keys that are used in conjunction with others may not be in this array. The Shift, Option, Caps Lock, and Command are thus connected directly to the keyboard controller. Programatically, they can be thought of as being in the same array as the rest of the keyboard even though they are physically separate.

The keyboard is controlled (and scanned) by an Intel 8021 single chip microcomputer. This device contains 1024 bytes of program ROM and approximately 65 bytes of internal RAM. It can execute up to 100,000 operations per second. Electrical power is supplied via two of the four wires in the keyboard cable. The other two wires are used to communicate in serial with the Macintosh. An almost identical arrangement can be found in Apple's numeric keypad. The biggest difference is that the keyboard array size has been changed.

The keyboard serial interface operates in an isosynchronous mode where a clock signal is supplied on one wire while data is sent (or received) over the other. The reason for the separate clock is that the keyboard does not contain a precise frequency source.

The format of the data sent is the same as a 7 bit asynchronous link that has one start bit and a one stop bit. The only difference is that the high order data bit is sent first. When the Macintosh wishes to send to the keyboard, it sets it's serial output to a logic 0 and then loads a register with the data to send. Eventually, the keyboard will notice the start bit and respond by supplying the clock signal to clock the data byte to it one bit at a time. Should the keyboard wish to send to the Macintosh, it simply places

it's data on the serial data line and generates the clock which allows the Macintosh hardware to receive the data. Note that the keyboard always generates a clock for either data transfer direction. Also, data can be sent in both directions on the same wire. (The data rate used in this interface is near 3300 bits per second.)

The software in the keyboard operates by scanning all of the keys repeatedly until a change is detected. This key is then "debounced" and the Macintosh told about the change. In greater detail: The software starts at the beginning of the key array and compares the present key status (pushed/not pushed) of each key with what it saw previously. When a change is detected, the software waits several milliseconds and then checks if the key is still at it's new state. If so, a code byte for this change is constructed and placed in a short buffer. Eventually, the keyboard will be able to send this code to the Macintosh. Once per keyboard checking loop, the keyboard checks the serial data line for a start bit. If there is one, it tries to receive the data. Also, the keyboard will be checking it's buffer for any data to send that it is allowed to send. If there is some, it will be sent.

The communications format used by the keyboard, keypad, and Macintosh between themselves is a loose daisy-chained master/slave type of architecture. For a slave to send data, it must have permission to send given to it by the master. Since the interface is a bidirectional system without collision detection, the rules for its operation need to be closely followed. There are several cases where they may be broken due to a time-out. These are listed below. The system is configured such that the Macintosh is always a master and the keyboard is always a slave. Any keypads (there could be more than one – although the Mac could get confused) in between act as a slave to the Macintosh (or next higher element) and as a master to the keyboard (or next lower element).

There are 5 different commands that may be sent by the master. Of these, the Permission to send and Self test keyboard commands (hex 10 and 76) are only sent by the Macintosh itself. If a slave receives any commands other than these, the slave ignores them. The commands are:

# Request to send (Hex code 14)

This code is sent when the master wants data. The slave has permission to respond with one byte of data which it should do immediately. If it has no real data, a hex 7B should be sent.

## Permission to send (Hex code 10)

This code is sent by the master to give the slave permission to send one byte of data when it is generated. After this one byte is sent, (it need not be immediately) permission will once again be with the master.

## Reset and identify (Hex code 16)

This code is sent by the master to reset the slave and have it identify itself. The slave has permission to respond with one of the following 3 codes. Hex 02 if it is just a keyboard, hex 10 if it is just a keypad, and hex 12 if both a keypad and a keyboard are present. (The result of this command can be found in the system global KbdType at memory location \$21E in the Macintosh.)

### Perform self test (Hex code 36)

This code is sent by the master to have the slave perform a test of its RAM. The slave has permission to send a single byte result code. Hex 77 is sent if there was an error and a hex 7D is sent if the RAM tested out OK.

# Remote self test (Hex code 76)

This code is only sent by the Macintosh when both a keypad and keyboard are present. The keypad will pass a self test command to the keyboard and then pass the result to the Mac. If no keyboard is present or if it does not respond, the keypad will send back a hex 77 as an error code. As normal, permission will revert to the Macintosh (the master) after this result is sent. In reality any byte with bit D5 set will be passed through with this same bit cleared.

If the slave does not respond in a short time period to a command requesting immediate response, permission to send will revert back to the master. This is so that if a device is not present, the master will not get stuck in a loop waiting for the (non-existent) slave. This also allows the slaves to be disconnected/connected at any time without destroying the interface integrity. The Macintosh normally operates any slaves below it in the Permission to send mode. Thus the slaves have a timeout mode where they send a hex 7B to say that they have no data if no activity takes place for about a quarter second. The keyboard will load its FIFO with a 7B after

the timeout but will not send the byte unless it has permission to send. Thus, if it is operated in Request for data mode only, these requests should be at a reasonably fast rate. If the keypad times out, it will send a byte of data even if does not have permission. Thus it should be polled with request for data at a reasonable rate or the master should assume that it does not have permission to send. When a keypad is in the system, it will operate the slave under it in a request for data mode. This polling occurs at a reasonable rate. The data sent to the keypad is stored in a FIFO so that it can later be sent upon request to the Macintosh.

# <u>Data sent by a slave</u>

01 - 68	Codes for keys as they are pushed down.
81 - EB	Codes for keys as they are released.
6D/ED	Unused by keyboard/keypad.

6F/EF CTRL push/release. 71/F1 SHIFT push/release.

73/F3 CAPS LOCK push/release.

75/F5 OPTION push/release.77 Error from memory test.

79 Prefix to flag following key code as a

keypad key.

78 No change in status.7D Memory test passed.

7F/FF Unused by keyboard/keypad.

F7-FD Unused by keyboard/keypad.

# Commands sent by a master

10	Permission to send.
14	Request to send data.

7 16 Reset and identify all devices.

36 Perform self test in first receiving device.

76 Pass a self test command to the next device.

#### Video Hardware

A roster-scon video display (such as the one in the Macintosh) operates by scanning a beam of electrons across the screen (CRT), normally from left to right and top to bottom. The beam is modulated by the video signal – thus selectively activating the phosphor in the tube and causing it to emit light. Scanning must be repeated at approximately 60 Hertz since the phosphor quickly stops emitting light. This article describes how the modulating signal is made and the side affects that are produced.

Since the electron beam scans each dot (pixel) on the screen individually, the information to control it must be sent one unit at a time. In the Macintosh, each pixel can be either black or white, so one binary bit of information is sufficient for each pixel. As the electron beam scans, it needs a continuous supply of bits (one at a time). This bit stream is produced on the digital logic board by the video display hardware. This hardware also produces two other signals – one to tell the electron beam when to return to the left edge of the screen and one to tell it when to return to the top. While the electron beam does this, it is kept turned off to prevent white lines on the screen in it's path. (In a black and white monitor, there really aren't individual dots on the screen – rather, the shape of the modulating signal delineates the boundaries between pixels. The entire process is similar to dragging a pen across a piece of paper and selectively lifting and dropping it.)

To continually scan the display, the present image on the screen is kept in memory (RAM) and read out by the video hardware as it is needed. The Macintosh memory is 16 bits wide - thus this many bits are read at a time. A device called a shift register is used to convert the 16 bits into a bit stream that is 16 units of time long - thus defining 16 pixels on the CRT. Incidentally, reading the video display performs another vital function - 'refreshing' the memory. This is needed because dynamic memory (as used in the Macintosh) consists of many small, leaky capacitors. These must be continuously discharged or recharged to retain their data. A read (or write) operation on the memory is enough to refresh 128 (or 256) memory locations.

Any description of the video timing must also include the memory timing since the hardware for these is strongly intertwined. Proceeding with this – the memory cycles 1958400 times per second. Each cycle is therefore about about 511 nanoseconds (nS) long. These cycles are divided evenly between the CPU and video with each device using alternate cycles. While video refresh is active (i.e., not in a retrace) it's cycles are used to fetch

video data for the CRT. Since these words are read every 1.021 microseconds (uS), just enough bits will be fetched to meet the needs of the 15.6672 MHz video shift rate. At the end of every horizontal retrace, the video cycle is used instead to read the sound and disk speed information from their RAM buffers. The remaining video cycles are given to the CPU to allow the system to run faster. This can be done because the electron beam is in retrace and needs no video signal. (See the MacPaint document 'Video Timing')

The figure has been arranged such that it contains all the memory cycles that happen during one complete screen refresh (frame). Including the retrace times, there are a total of 86\*370 or 32560 memory cycles in one frame. This results in a screen refresh rate of 1958400/32560 or 60.147 Hz. As can be seen from the figure, each of the first 342 scan lines of the CRT contain 32 video accesses and 32 CPU accesses in the active part, and 23 CPU accesses and 1 sound/disk-speed access in the inactive (retrace) part. Also, the last 28 scan lines (which occur during vertical retrace) contain 87 CPU accesses and 1 sound/disk-speed access.

One important result of the above is that the 68000 CPU cannot run at it's full 7.8336 MHz speed from the RAM since at that rate, it could use all 1958400 memory cycles in each second. If the CPU requests access to the RAM while it is being used by the video hardware, it will be forced to wait for it's turn. However, internally, the CPU can still run at it's full speed. This can be used to advantage by assembly language programmers by careful arrangement of operations. For example, suppose the contents of all 8 data registers (DO-D7) need to be multiplied by some constant and then a different constant needs to be added to the results. One immediately obvious way of doing this is to have 8 MULU (or MULS) operations in a row and then 8 ADD opcodes in a row. This arrangement is the not the fastest possible - the ADD operations will be limited by the speed that these opcodes can be fetched by the 68000. A better method would be to alternate the MULU and ADD opcodes. This way, the ADD opcode will be read into the 68000's instruction queue while the preceding multiply is taking place. The result is that the 68000 runs at it's full speed internally.

The memory used by the Macintosh for it's video display normally starts at \$01A700 (\$07A700 in 512K versions). However, knowledge of this address is not needed to program the Macintosh since QuickDraw will automatically set up it's data structures to use the correct range of memory. The hardware itself supports the use of a different range (page) of memory as an alternate source of video data - this video page begins exactly 32K bytes below the first. Since the hardware can instantly switch

between the two pages, they could be used in extremely fast graphics applications. (None of the 'standard' application programs out yet use the second video page - mainly because it leaves very little memory left for the program on a 128K machine.)

There is almost no support in the operating system to allow use of the second video page. It is limited to the 'Launch' and 'Chain' operating system calls. Both of these allow a selector code to determine which video page will be used by the application. However, this only allocates an extra 32K of memory after the stack memory if the second video page is selected. It does nothing to switch the actual memory used by the hardware. Furthermore, when a program is run from the Finder, it will be run with the first video page selected – the only way to actually run a program with the second video page selected is to write a short program that calls the main program with the correct page setup. Finally, the 'Launch' and 'Chain' routines do not insure that the second video page is actually usable – a debugging program such as MacsBug may occupy that memory.

The best way use the second video page is to launch the program normally. Then check if the present system is a Macintosh (i.e., not a Lisa running MacWorks) by checking such things as the screen size and operating system version numbers stored in global RAM memory variables and exiting if they are incorrect. The presence of MacsBug (or a RAMdisk) can be detected by checking if the system globals BufPtr (at \$010C) and ScrnBase (at \$0824) are the same. After this, the program can set up to use the second video page in one of two ways. If the memory space between the two pages (10880 bytes) is not enough for the stack space needs of the program, it can manually set the stack top just below the start of the second video page. Otherwise, the stack can be left where it is and the memory space for the second page reserved. Setting the application heap limit to a point just. below that memory would suffice to reserve it - however, this does not prevent the CPU stack from growing down into the video display. A better way would be to manually (i.e., not via the operating system) make the second video page a non-relocatable memory block at the very end of the application heap. This will allow the stack-overflow sniffer to operate normally and detect any stack overflows.

### Sound Hardware

Overall, the sound generator can be broken down into four pieces of which three are hardware. These four pieces can then be arranged in a hierarchy through which "commands" flow. At the highest level is a program that controls all the hardware below it. This is part of the Macintosh Operating System and thus not a subject of this document. The level just below this is an arbitrary waveform generator. This can be thought of as a black box that converts input data into any output desired. If the output of this box were graphed on the y-axis of a graph while the x-axis was time, the data supplied to the box would be all the y coordinates needed to produce the desired output. However, because this could require an infinite amount of data to produce certain sounds, certain limitations are imposed by the hardware. The main one is that the y values are only supplied at discrete (not continuous) time intervals. A smaller limitation is that the y values can have only a limited amount of precision. When this data is provided to the "black box", the intervals between the points are filled by interpolation. The output of the black box is then sent to the third part of this hierarchy which is an audio amplifier. The amplification (or resulting output volume) that this section provides can be controlled by software. The final element of this hierarchy is the speaker or external stereo system connected to the audio output jack of the Macintosh.

The arbitrary waveform generator is essentially a Digital to Analog Converter (DAC). In operation, the data that is sent to this DAC is held in memory (RAM) in a buffer - so that it can be generated before the hardware actually needs it. As data is needed, the DAC is given the contents of successive locations of this buffer. This buffer holds enough data to generate 1/60th of a second of sound.

The DAC itself is built in a somewhat unconventional way. The sound data coming from the buffer is first converted into variable width pulses by a counter circuit. These pulses are then approximately integrated over time by an analog circuit. The output of this is a voltage that roughly relates to the pulse width (which was originally controlled by the data from the buffer). The pulse width can be from 0 to 255 units of time. A width of 255 will result in a positive peak to the speaker while a pulse width of 0 makes a negative peak.

As the system is running, data values are taken (by hardware) from the buffer at the end of every horizontal retrace of the video monitor. This is a rate of 15.6672/704 MHz or about 22254.5 Hz. This data is then loaded into an 8 bit "up" counter. During the next horizontal scan line, this counter will

the next section of sound. The main or default buffer used most often is called the First Sound Page while the second buffer is called (naturally) the Second Sound Page. Each buffer contains 1/60th of a second of sound or 370 samples. The buffer used is determined by an output bit that is controllable by the program making the sound. In a 128K Macintosh, the first sound page starts at \$1FD00 or 130304 decimal while the second page starts at an address of \$1A100 or 106752 decimal. After this starting address, the sound data occupies the even numbered memory locations. The odd locations are used to control the disk speed and thus should never be modified in the process of producing sound. The last address used in the first sound page is \$1FFE2 or 131042 decimal while the last used address in the second page is \$1A3E2 or 107490 decimal.

The above method for producing sound output is very good at producing high quality sounds but it also will take a large amount of computer power to generate the 22255 samples needed every second to produce sound. Although it is possible to generate 1/60th second of sound and then allow this data to repeat continually, doing this will limit the sound output to those frequencies that can be evenly divided into 1/60 second intervals. Therefore, there are two other ways to produce sound that are less demanding on the microprocessor. Both of these involve a special control line to the 8 bit up counter used to produce the varying pulse widths. Both methods will also result in square wave output. A square wave signal is not a "pure" sound but rather it is the sum of a large number of sine waves which each are mathematically a pure sound. While the control line to the counter is active, it will force the counter to a value of 0. This will result in a pulse width output of indefinite length for as long as the control line is active. If the sound buffer is entirely filled with the value of 255 and the control is turned on and off, sound will be produced. This is because the pulse width will essentially be changed from 0% to 100% depending upon the control line.

This control line can be changed in two ways. The simplest is for the microprocessor to change it as needed to produce the desired sound. Another way is to program one of the large scale integration chips in the Macintosh to produce a square wave frequency there. Thus the frequency of a sound can be controlled with minimal processor control. An interesting effect can also be performed with this circuitry. Instead of loading the sound buffer with all 255s, a real sound can be stored there. The square wave being driven into the counter will then toggle the sound on and off. This essentially adds a second sound signal to the output. Since the only way to terminate a pulse from the counter is for it to count up to 255, there

will be a delay of about 32 microseconds as the counter does this counting. Normally though, this will be of no affect. The control line also has a second use; when set up to be controlled directly by the CPU, it is used to turn the sound on and off. Thus between sounds, all audio output can be shut off.

Several precautions should be observed when designing sound output. The first of these is the limited frequency range of the audio output circuitry. At low frequencies, the gain of the amplifier is reduced. This should not be a big problem since most of these low frequencies won't be heard. However, for frequencies that approach the low end of the range, some extra amplitude must be given to account for the loss in the amplifier. At the high end of the spectrum, more problems can result. Since each audio cycle of output will need at least two samples to produce the frequency, the highest possible frequency that can be produced is 11127 Hz. Also, since the filter circuit converting the pulse width signal will be filtering high frequencies, a loss of amplitude will occur. This loss will be large enough so that it can not be made up with a larger magnitude of values stored in the sound buffer. Another precaution to observe is to use the control line to turn the sound off instead of just filling the sound buffer with a constant value (The buffer could be filled with 255's with no ill affect though). This is due to the fact that while the counter is producing any sort of pulse output, the filter will allow some of it to appear at the speaker output in the form of a very quite high pitched noise. The internal speaker will not reproduce this well, but a home stereo system attached via the audio output jack might. Another precaution is that when the control line to the counter is active, the effective output pulse width is 352 units. Since the range of outputs for the counter (when it is being fed data from the sound buffer) is only from 0 to 255 units, activating the control line will [in all cases] cause a large sudden change in the audio output signal. This will sound like a click or pop and is the reason that certain sounds start and end sharply. There is no solution to this problem, however, having sounds end at a pulse width of 255 (by filling the sound buffer with 0's) will reduce the magnitude of the change to the smallest possible value.

The three volume control bits each have approximately a binary weight. In other words, as the value represented by these bits is increased in a binary manner, the volume will increase in approximately equal steps. However, this pattern is not exact – writing the control bits most significant to least significant and assigning the 000 setting to have the relative volume level of 1, the following relationship holds: 000 = 1.0, 001 = 2.0, 010 = 4.1, 011 = 5.1, 100 = 7.9, 101 = 8.9, 110 = 11.0, 111 = 12.0

Thus the volume setting of 7 is twelve times louder than the setting of 0. Note that a control panel setting of 0 results in the sound being turned off in some programs while other programs will use it as just another volume setting.

All the control signals for the sound hardware come from a 6522 Versatile Interface Adapter chip. This component is itself as complicated as the sound circuitry so only a minimal amount of information about it will be presented here. For programming purposes, it can be treated as a set of sixteen memory addresses of which two are of interest for this column. At the memory address of 1110-1xxx-xxx1-111x-xxxx-xxx0, DO through D2 of the byte are the sound volume control bits. D2 is the most significant bit, D1 the next, and D0 the least significant bit. In the above address, the x's represent bits that can be either 0 or 1 (Apple seems to prefer to set these to 1's.) Thus the address is \$EFFFFE or 15728638 decimal. Since the other bits in this byte are used for other system uses, the present value should be read, modified, and then written back (preferably with interrupts turned off). In BASIC to set the volume to 5, this would be: IRA = 15728638!: POKE IRA,PEEK(IRA) AND &HF8 OR &HO5. Note: this does not change the control panel (master) volume setting.

At the same location that the volume can be modified, D4 selects which sound page will be used. Setting this bit to a 1 will select the first sound page. Putting a 0 into this bit will select the second sound page. This bit should probably never be modified since the disk speed control data also comes from this same page. When this bit is changed from BASIC, the operating system will still be using the first sound page and if any disk usage is needed, the system will "hang".

At the address of 1110-1xxx-xxx0-000x-xxxx-xxx0 (\$EFE1FE or 15720958), the control line to the counter is found. A 1 will allow the counter to operate and thus produce sound while a 0 will force the counter to the 0 state. This bit is found in D7 and can be read/written just like the volume control bits. Several other locations in the 6522 chip can be used to command this bit to be connected to the output of a counter. This is how a square wave audio output can be generated.

#### VIA Hardware

This document describes all the inputs and outputs available through the Macintosh's 6522 Versatile Interface Adapter (VIA) chip. Since this is a rather complicated chip, the various other functions it performs internally will not be described. (For further information on this chip's internal operations, consult the Synertek or Rockwell manual about it.)

The entire chip consists of 16 different byte wide registers through which all of it's operations can be controlled. Because of the way that it is attached, each of the 16 registers is located 512 bytes apart in the Macintosh memory map. To find the exact address of each register, the register number should be multiplied by 512 and a base address added to this result. The final result is the memory address to use. Normally the offset (register number \* 512) is fixed so that it can be coded directly into a program. The base address is also fixed (in the Macintosh).

Assembly language programmers can use the global variable "VIA" at memory location \$1D4 to find the base address of the VIA. The code to do this is actually 2 bytes shorter than loading the base address as an immediate value (using the global variable takes one to two microseconds longer though). This global variable normally contains the value \$EFE1FE. Once the base address is in a register, the address register with displacement mode can be used to get to the desired register in the VIA.

Register number 15 (offset of \$1E00 and address of \$EFFFFE) is used to access the input and output bits located in Port A (PAn). A read operation will return the current value on the chip's pins. A write operation will only affect the current output bits. Bits that are inputs will still latch the value written, however it won't appear as an output until the line is changed to be an output. Register number 0 (offset of 0 and address of \$EFE1FE) is used to access those bits found on Port B. Most of the 14 remaining registers are not needed while doing simple operations with the VIA.

**SVO**, **SV-1**, and **SV2** control the sound volume. A binary value of 000 produces the lowest volume while a binary value of 111 produces the maximum volume. Normally these bits will be altered only by the operating system. (It will set them to match the volume selected from the control panel.) However, as there is no routine to change the volume, a program would need to alter these bits directly should it need to change the current volume setting. Apple recommends that this be done with the 68000's interrupt priority level set to "3" so that an interrupt cannot occur while the bits are being changed.

When **OVERLAY** is a 1, ROM memory is available at memory address 0. This is used during system reset to allow the 68000 to read a reset vector

line. At present, this input is not used by the operating system. A possible use is to perform flicker free video page switching at a certain point in the video refresh. Another use is as a time base upon which some input sampling could be performed. Since it is of exactly the same frequency as the sound output sampling rate, it might be used to help digitize an input sound into the same number of samples needed to output it at a later time.

The **SOUNDEN** output is used to turn the sound hardware on and off as described in the Sound Hardware document. A value of 0 is the "on" state and a value of "1" is the off state. Changing this bit from within a user program is harmless as the worst possible event is simply a constant sound output.

The **VSYNC** input is used as an interrupt upon which many time related tasks are performed by the Macintosh operating system. It will go low during every vertical retrace. (A rate of 60.1474 Hz.) This input could also be used to perform such things as flicker free video page switching.

The KBD.CLK is generated by the keyboard/keypad to transfer data to and from the shift register in the 6522. Data enters and exits via the KBD.DATA line. These two lines are of use only when writing a replacement keyboard driver.

Overall, attempt to use an operating system routine to alter these I/O bits to retain compatibility with different hardware (such as the Lisa). Where this is not possible, make sure that the system is a Macintosh and carefully alter the needed bits.

PAO PA 1	Output Output	SVO SV1	Low order bit of volume select
PA2	Output	SU2	Middle order bit of volume control High order bit of volume control
PR3	Output	SND.PG2	Sound/disk page select
PR4	Output	OVERLAY	ROM/RAM select
PA5	Output	DISKOUT	Disk command output
PR6	Output	VID.PG2	Video page select
PA7	Input	SCCW/REO	DMA request signal from SCC
PB0	Input/Output		Data to/from RTC pin 6
PB1	Output	RTC.CLK	Data clock to RTC pin 7
PB2	Output	RTC.EN	Enable signal to RTC pin 5
PB3	Input	MOUSE . SWC	Mouse button input
PB4	Input	MOUSE.X2	Mouse quadrature input
PB5	Input	MOUSE.Y2	Mouse quadrature input
PB6	Input	HSYNC2	Horizontal sync signal
PB7	Output	SOUNDEN	Sound PWM enable/reset
CR 1	Input	USYNC	Vertical sync interrupt input
CA2	Input	RTC. 1HZ	1 Hz square wave from RTC pin 1
CB1	Input	KBD.CLK	Data clock input from keyboard
CB2	Input/Output	KBD.DATA	Data to from keyboard
			=

#### Mouse Hardware

At any moment of time, the position of a mouse can undergo three possible state changes (for each axis of movement). These are: move left, stay in place, and move right. Information about these state changes must be sent out of the mouse. Basically, two ways exist to do this: either the computer can be told of each change or the mouse can output it's present position from which each change can be deduced. The first method is difficult to do with a limited amount of hardware since two (or more) successive "move left" operations must be distinguishable from just one. Thus the second method is employed in most mice.

Encoding the full mouse pointer position in the mouse hardware itself is not practical because of the amount of data this would require. Thus, the mouse sends only the minimum amount of information needed to decode each movement. (This movement is then used to control the pointer movement.) Three distinguishable positions are enough to express all of the three possible state changes. These three positions are: the next left, the present, and the next right positions. Because these must be encoded as a binary number, at least 2 bits are needed. However, 2 bits of data can encode a total of four values so thus four positions are used. Note that at any time, only three of the four will be used.

Four consecutive positions of the mouse could be numbered 0, 1, 2, and 3. When the mouse is in positions 1 or 2, it always has another position available to the right or left. This is not true for locations 0 and 3. Therefore, the number codes repeat as the following endless sequence: ...  $2\ 3\ 0\ 1\ 2\ 3\ 0\ 1\ 2\ ...$  As the mouse moves in either direction, it traverses this set of numbers.

The encoding of these position numbers in binary must be done in a special way. The initially obvious way of mapping them in the standard numerical binary sequence (0 to 00, 1 to 01, 2 to 10, and 3 to 11) will not work. Two of the transitions (1 <-> 2 and 3 <-> 0) in this type of coding have more than one bit change. Since the binary values will be generated mechanically in the mouse, the two transitions cannot be guaranteed to be simultaneous. What is needed is a coding where one bit changes at a time. (This is known as a Gray code.) The only 2 bit Gray code is: ..., 10, 00, 01, 11, 10, 00, ... read in either direction. The Macintosh mouse uses this code. When this sequence is graphed with respect to time, the result looks like two square waves that are 90 electrical degrees offset from each other (one cycle is 360 degrees). For this reason, such signals are often said to be in quadrature.

## Listing 1 - Mouse signal demonstration program

```
10 IRB=15720958#
20 MOUSEX1 = 10485754#
30 MOUSEY1 = 10485752#
100 IF PEEK(MOUSEY1) AND &H8 THEN PRINT "1"; ELSE PRINT "0";
110 IF PEEK(IRB) AND &H20 THEN PRINT "1"; ELSE PRINT "0";
120 IF PEEK(MOUSEX1) AND &H8 THEN PRINT "1"; ELSE PRINT "0";
130 IF PEEK(IRB) AND &H10 THEN PRINT "1"; ELSE PRINT "0";
140 IF PEEK(IRB) AND &H8 THEN PRINT "1" ELSE PRINT "0"
```

© 1985

```
;Disassembly listing of the Macintosh Keuboard and
                 ;Keypad controller (intel 8021 MPU) ROM. May 1984.
                 ;This code continually scans the keyboard and reports
                 ;any changes to the Macintosh.
                 ;The chip from which this code was read was marked:
                 ;P8021H 2173. It had a date code of 8352. It did
                 ;not have any copyright in or on it other than the
                 ; Intel copyright for the CPU design.
                 ;The approximate clock rate of the MPU is 3 Mhz.
                 ;This results in a CPU cycle time of slightly under 10 uS.
                 ;All of the quasi-bidirectional pins have internal
                 ;pullup resistors.
                 ;Entry point of hardware (power-on) RESET
000
        54 89
                CALL 289
                                 ;init system RAM and timer
                 ;Relooping entry point - check for operation mode
002
        00
                NOP
                                 ;Delay
903
        ΠR
                 IN A,PO
                                 ;Find out if keyboard or keypad
        53 20
004
                ANL A, #$20
                                 ;Mask for only 205 input
                JZ $088
005
        05 A8
                                 ;If jumpered to ground, we are a keypad
                 ;Keyboard main loop
008
        34 92
                CALL $192
                                 ;Scan the keys
        B8 30
                MOV RO. #$30
                                 ;Load pointer to outgoing FIFO
        E6 14
                JNC $014
                                 ; If no keypush, then skip
OÕE
                CALL $288
        54 A8
                                 ;Place key stroke in the FIFO
010
        54 B6
                CALL $286
                                 ; Init timer
012
        04 28
                JMP $028
                                 ;Go attempt to send the byte
014
        F0
                MOV A, PRO
                                 ;Get first byte in FIFO
015
        96 29
                JNZ $029
                                 ; If old data is present, try to send it
                ;Check for inactive timeout
017
        16 18
                JTF $018
                                 ; If timeout
019
        04 28
                JMP $028
                                 ;Figure what there is to do
018
        98 2E
                MOV RO, #$2E
                                 ;Load pointer to flag
0 1D
        50
                MOV A,@RO
                                 ;Get the value
01E
        53 02
                ANL A, #$02
                                 ;See if inactive for too long
                                 ;Yes
020
        96 25
                JNZ $025 -
622
        10
                 INC PRO
                                 ;Keep track of inactive time
023
        04 28
                JMP $02B
                                 ;Go figure out what to do
025
        B8 30
                MOV RO. #$30
                                 ;Load pointer to F!F0
027
        80 78
                MOU @RO,#$78
                                 ;Load "I'm awake" code and try to send it
029
        54 86
                CALL $286
                                 ;Init timer and reset idle counter
                ;Decide what needs processing and go do it
028
        B8 2F
                MOU RO. #$2F
                                 ;Load pointer to received data
020
        FO
                MOV A, ERO
                                 ;Get present value
```

;8021PAGEO - Comments by The IM Underground (C) 1985

```
;Send a byte if our turn still
                :Attempt to receive a command and process it
        54 BF
                CALL $28F
                                ;Sample serial input
032
        F6 02
                JC $002
                                ; If inactive, then reloop
034
        74 CF
                                ;Receive a data byte
                CALL $3CF
036
        FO
                MOU A, 9RO
                                :Get received data
037
        03 14
                XRL 8.#$14
                                ;See if request for data command
                JZ $05F
039
        C5 5F
                                :Respond with result
03B
        F0
                MOV A, @RO
                                ;Get received data
                                ;See if permission to send
030
        D3 10
                XRL A. #$10
                JZ $04E
03E
        C5 4E
                                ;Try to send a data byte
040
        FO
                MOV A, ERO
                                ;Get received data
041
        D3 16
                XRL A.#$16
                                ;!s it reset command?
043
        C6 61
                JZ $061
                                ;Reset the keyboard and return $03
                MOV A,€RO
                                ;Get received data
045
        F0
        D3 36
046
                XRL A.#$36
                                ;!s it a self test command?
048
        C6 63
                JZ $063
                                ;Go do the self test routine
04A
        BO 00
                MOV @RO, #$00
                                ;Clear receive bute
040
        04 02
                JMP $002
                                ; If none of those, reloop
                ;Try to send a byte from the $30 F!F0
04E
        B8 30
                MOV RO.#30
                                ;Load pointer to FIFO
050
        FO
                MOV A, 2RO
                                ;Get first byte
051
        06 02
                JZ $002
                                :Reloop if empty
053
        54 BA
                CALL $28A
                                ;Set timer to timeout in_70ms
                                Exit loop after timeout
055
        16 30
                JTF $030
057
        54 BF
                CALL $2BF
                                 ;Sample F20 serial data input
        E6 55
                JNC $055
059
                                ; Wait until timeout or serial line inactive
        74 80
                CALL $38C
                                ;Send a byte of data
        04 02
                JMP $002
                                :Reloop
                ;Vectors to short code blocks in page 3
05F
        64 1B
                JMP $318
                                ;Link to respond with data code
061
        64 26
                JMP $326
                                ;Reset keyboard (return $03)
                ;Perform RAM memory test.
                Entry Mone
                :Exit
                        R0 = $36
                        (80) = result flag $70 = OK $77 = Error
                        Rest of RAM reinitialized
                       ALL
                :Uses
                ;Fill RAM with one bit pattern
        98 02
                MOV RO, #$02
053
                                ;Load starting test address
        89 3E
                MOV R1,#$3E
065
                                ;Load size to test (62)
067
        BO AA
                MOV @RO,#$AA
                                ;Put value into RAM
                INC RO
069
        18
                                ;Advance pointer
06A
        E9 67
                DUNZ R1,$067
                                ;Loop through all locations
                ;Check if all locations held this value
```

02E

96 4E

JNZ \$04E

```
060
        89 3E
                 MOV R1,#$3E
                                  ;Load size again
06E
        88 02
                 MOV RO. #$02
                                  :Load start again
070
        FO
                 MOV A.@RO
                                  ;Fetch value there
         18
                 INC RO
                                  ;Advance pointer
        D3 AA
                 XRL A.#$AA
                                  ;See if it was $AA
074
        96 8D
                 JNZ $08D
                                  ; If error, exit
076
        E9 70
                 DUNZ R1,$070
                                  ;Loop for rest of RAM
                 ;Fill RAM with other bit pattern
078
        89 3E
                 MOV R1,#$3E
                                  ;Load size again
078
        88 02
                 MOV RO,#$02
                                  ;Load start again
070
        80 55
                 MOV @RO,#$55
                                  ;Put (NOT $AA) in RAM
07E
        18
                 INC RO
                                  ;Advance pointer
07F
        E9 70
                 DJNZ R1,$07C
                                  ;Loop for all of memory
                 ;Check if it held everywhere
081
        89 3E
                 MOV R1,#$3E
                                  ;Load size
        88 02
083
                 MOV RO, #$02
                                  ;Load start
                 MOV A,@RO
085
        FO
                                  ;Get value from RAM
086
        18
                 INC RO
                                  ;Advance pointer
        D3 55
                 XRL A,#$55
                                  ;See if error
087
089
        96 SD
                 JNZ $08D
                                  ; If so, exit
088
        E9 85
                 DJNZ R1,$085
                                  ;Loop for rest of RAM
                 ;Reset back to operational state
080
        F9
                 MOV A,R1
                                 ;Get final value of counter
08E
        AA
                 MOV R2,A
                                 ;Save it in R2
08F
        54 89
                 CALL $289
                                 ; Init system
        54 B6
                 CALL $285
                                 ;Init timer to O
        FĤ
                 MOV A,R2
                                 ;Get result back
094
        96 9C
                 JNZ $090
                                  ;|f there was an error...
095
        B8 36
                                 ;Load pointer
                 MOV RO.#$36
098
        80 70
                                  ;Load OK code
                MOV @RO.#$7D
09A
        04 53
                 JMP $053
                                 ;Send the result code
09C
        88 35
                MOV RO, #$36
                                 ;Load pointer
09E
        BO 77
                MOV @RO, #$77
                                 ;Load error code
0A0
        04 53
                JMP $053
                                 ;Send the result code
0A2
        00
                NOP
                                 ;Take up space
0A3
        00
                NOP
                                 ;Take up space
0A4
        00
                NOP
                                  ;Take up space
0A5
        00
                MOP
                                  ;Take up space
086
        00
                NOP
                                  ;Take up space
0R7
        00
                NOP
                                 ;Take up space
                 ;Keypad main loop entry point
088
        23 01
                MOV A,#$01
                                 ;Load mask
ORA
        54 00
                CALL $2CC
                                 ;Set keypad mode flag
                ;See if keyboard communications must be done
        B8 2C
OAC
                MOV RO. #$20
                                 ;Load pointer to flag
ORE
        FO
                MOV A.@RO
                                 ;Get the value
```

```
ORF
        F7
                RLC A
                                 ;Test keyboard absent flag
080
        E6 BC
                JMC $0BC
                                 ; If present then talk to keyboard always
082
        88 28
                MOV RO.#$28
                                 ¡Load a flag pointer
                                 ;Count loops
        10
                 INC PRO
Obo
        F0
                ORS.A VOM
                                 ;Get result
086
        97
                CLR C
                                 ;Set up for 6502 type ADD
087
        13 F8
                ADDC A,#$F8
                                 ;See if result was less than 8
089
        E6 D7
                JNC $007
                                 ;Try to find keyboard only every 8 passes
088
        AO.
                MOV @RO.A
                                 ;Load new result (0)
                ;Ask keuboard for data
OBC.
        23 14
                MOV A.#$14
                                 ;Load command bute
OBE
        74 2E
                CALL $32E
                                 ;Send the bute to the keyboard
000
        C6 C8
                JZ $008
                                 ;If the keyboard took the byte
002
        23 80
                                 ;Load mask
                MOV A.#$80
0C4
        54 00
                CALL $200
                                 ;Set keyboard absent flag
006
        04 07
                JMP $007
                                 ;Skip ahead
820
        23 7F
                                 ;Load mask
                MOV A, #$7F
OCA:
        54 D1
                CALL $201
                                 ;Clear keyboard absent flag
000
        74 60
                CALL $36D
                                 ;Receive return code from the keyboard
OCE
        FD
                MOV R5,A
                                 :Save it
OCF
                XRL 8,#$79
        03 78
                                 ;See if no changes code
0D1
        C6 D7
                JZ $0D7
                                 ; If so, then don't give to Mac
0D3
        B8 30
                MOV RO,#$30
                                 ;Load FIFO pointer
0D5
        54 88
                CALL $288
                                 ;Save the returned value
                ;Scan Keys and put result in the keypad FIFO
007
        54 53
                CALL $253
                                 ;Scan keus
        E6 DF
009
                JNC $ODF
                                 ; If nothing changed
        B8 3A
                MOV RO, #$3A
                                 ;Load FIFO pointer
        54 A8
                CALL $288
                                 ;Save result
ODF
        74 OE .
                CALL $30E
                                 ;Check FIFOs for data
0E 1
        F6 FD
                JC $OFD
                                 ;Don't bother with awake stuff if data to send
                ;Account for extra time taken to talk to keyboard
0E3
        B8 20
                MOV RO.#$20
                                 ;Load flag pointer
0E5
        FO
                MOU A, @RO
                                 ;Get the flag byte
0E6
        F7
                RLC A
                                 ;Check the keyboard present flag
0E7
        E6 F1
                JNC $0F1
                                 ;!f present, send awake codes at faster rate
0E9
        23 02
                MOV A,#$02
                                 ;Load mask
0EB
        DO
                XRL A,0RO
                                 ;Flip D1 of flag byte
0EC
        HO.
                MOV PRO,A
                                 ;Replace result
0ED
        53 02
                ANL A, #$02
                                ;Check new status of D1
0EF
        96 FD
                JNZ $OFD
                                 ;Send data at half rate if keyboard absent
                ;Check if "awake code" (or "no changes") should be sent
0F 1
        B8 2E
                MOV RO, #$2E
                                 ;Load pointer to another flag
0F3
        10
                INC PRO
                                 ;Count loops
0F4
        FO
                MOV A,@RO
                                ;Replace result
0F5
        97
                CLR C
                                 ;Set up for 6502 type ADD
OF6
                ADDC A,#$E4
        13 E4
                                ;Essentially see if A was less than 28
OF8
        E6 FD
                JNC $OFD
                                ; If so, then continue
OFA
        AO.
                MOV @RO,A
                                ;Load new result (0)
```

```
24 1F
                JMP $11F
OFB.
                                 ;Send an awake code
                 ;Check for what needs to be done
ΘÃ
ഗുമ
        88 2F
                MOV RO, #$2F
                                 ;Load pointer to SIO flag byte
OFF
        FO
                MOV A, ERO
                                 Get the value
100
        95 08
                JMZ $108
                                  if it contained a command
102
        54 BF
                CALL $2BF
                                 ;Test P2O input from Mac
104
        F6 10
                JC $11D
                                 ;If inactive, reloop
105
        74 CF
                CALL $3CF
                                 ;Receive the bute from the Mac
108
        D3 14
                XRL A,#$14
                                 ;See if it a data request command
                                 ;Send a response to the Mac
10A
        C5 1F
                JZ $11F
10C
        FO
                MOV A, PRO
                                 ;Get the byte again
10D
        D3 10
                XRL A,#$10
                                 ;Check if it was permission to send
10F
        06 25
                JZ $125
                                 ; If so
111
        FO
                MOU A, PRO
                                 ;Get the bute again
112
        03 16
                XRL A.#$16
                                 ; Is it a reset command
        C5 6E
114
                JZ $16E
                                 ; If so
115
        FÜ
                MOU A, eRO
                                 ;Get command again
117
        53 40
                AML A, #$40
                                 ;Check for special keyboard self-test flag
        95 82
                JNZ $182
                                 ;Tell the keyboard to test itself
119
118
        04 45
                JMP $045
                                 ;Check for keypad self test or NOP
                 ;Branch to top of loop in different page of ROM
        04 90
                JMP $OAC
110
                                 Vector to start of keypad loop
                 ;Set up to send data or an awake code
11F
        74 OE
                CALL $30E
                                 ;Check FIFOs for data
                                 ;If some data there
121
        F6 25
                JC $125
        80 78
                MOV @RO,#$78
133
                                 ;Place response at ($30)
50
                 ;Send a byte to the Mac if it exists
125
        88 3A
                MOV RO,#$3A
                                 ;Load pointer
127
        FO
                MOV A, ERO
                                 ;Get the byte
        C6 5B
                JZ $15B
128
                                 ;If nothing in keypad FIFO
128
        B8 2C
                MOV RO. #$20
                                 ;Load flag pointer
120
        FO
                MOV A, ERO
                                 ;Get contents
        53 04
                ANL A,#$04
12D
                                 ;Mask for D2
12F
        95 47
                                 ;If the prefix was sent, send a data byte
                JNZ $147
                 ;Send a $79 keypad data prefix to the Mac
131
        88 36
                MOV RO,#$36
                                  ;Load pointer
        80 79
                MOV @RO.#$79
133
                                  ;Load data to be sent
        54 BA
135
                CALL $2BA
                                 ;Clear timer
                JTF $102
CALL $2BF
                                  ;On timeout, try to receive a byte
137
        16 02
                                 ;Test P20 input
139
        54 BF
13B
        E6 37
                JNC $137
                                 ;Nait until Mac does not want to send
                CALL $38C
                                 ;Send a byte to the Mac
130
        74 80
13F
        54 86
                CALL $286
                                 ;Reset timer and timeout mask
141
        23 04
                MOV A, #$04
                                 ;Load mask
                CALL $2CC
143
        54 CC
                                 ;Set prefix sent flag
145
        04 AC
                JMP $OAC
                                 ;Reloop
```

```
;8021PAGE1 - Comments by The IM Underground (C) 1985
                ;Scan the keyboard and return the needed codes
                Entry Mone
                        CY = 0 If no valid change
                 Exit
                        CY = 1 If a valid change
                        A = R5 = Result code
                        A,F,R0,R1,R2,R3,R4,R5,R6,R7,&2
                ;Uses
192
       89 34
                MOV R1,#$34
                                 ;Load pointer into FiFO
194
       F1
                MOV A, @R1
                                ;Get last byte in FIFO
195
       05 99
                JZ $199
                                 ;!f room in FIFO for another key
       27
197
                CLR A
                                 :*** THIS IS A BUG! ***
                ;The above line should be CLR C to flag no changes. As is,
                ; if the FIFO is full and CY was set prior to calling this
                proutine, The last entry will be smashed with the present
                contents of R5. However, another pair of bugs that do not
                jallow the entire FIFO to be used, will cause this code to
                ;never find a FIFO full condition and thus this bug is
                ;bypassed.
198
        83
                RET
                                 ;Don't scan if no space
                ;Perform a scan
199
        89 20
                MOV R1,#$20
                                 ;Load pointer to key status array
198
        23 FD
                MOU A, #$FD
                                 ;Load output data
190
        3A
                CUTL P2, A
                                 ;Enable P21 output
19E
        34 EE
                CALL $1EE
                                 ;Test this row of keys
180
        23 FF
                MOV A, #SFF
                                 ;Load inactive data
182
        38
                OUTL P2,8
                                 ;Turn outputs off
        F6 98
                JC $198
                                 ;Exit if there was a change
        23 FE
                MOV A. #$FE
                                 ;Load stanting scan code
187
                                 ;Save it
        88
                MOV RO,A
                                 ;Select a scan line from P1
188
        39
                OUTL P1,R
189
        19
                INC R1
                                 ;Advance key status table pointer
                                ;Test this row of keys
188
        34 EE
                CALL $1EE
                                ;Load inactive value
1AC
        23 FF
                MOV A, #$FF
1AE
        39
                OUTL P1,A
                                ;Deselect the port !ine
1AF
        F8 BD
                JC $1BD
                                ;Exit if there was a change
181
        F8
                MOV A,RO
                                ;Get current scan status
        97
182
                CPL C
                                 :Set CY
                                 :Move the scan line to the left
1B3
        F7
                RLC A
        F6 A7
184
                JC $187
                                 ;Reloop for another line
186
        19
                INC R1
                                 ;Advance table pointer to shift status
187
        54 06
                CALL $206
                                 ;Get the shift key status
189
        D1
                XRL A, PR1
                                 ;See if anything changed
1BA
        98 BE
                JNZ $1BE
                                 ;If there was a change - process it
                                 ;Flag no changes
1BC
        97
                CLR C
        83
180
                RET
                                 ;Done
                ;Compute the key codes of a SHIFT type key
                ;Entry A = Change mask
                        R1 = Pointer into status table
                ;Exit
                        CV = 0 if no valid change
```

CY = 1 If a valid change

```
54 E7
                CALL $2E7
                                 ;Compute a change mask
        54 R1
                CALL $281
                                 ;Delay 4.2 ms for debounce
                CALL $206
102
        54 06
                                 ;Read shift status again
1C4
                MOV A.@R1
        F 1
                                 ;Get old status
105
        58
                ANL A,R3
                                 ;Mask for only current key
106
        AC.
                MOV R4, A
                                 :Save the result
107
        FD
                MOV 8,85
                                 :Now fetch the shift status
108
        58
                ANL A, A3
                                 ;Mask it for only the current key
109
                                 ;See if old and present status is same
        DC.
                XRL A,R4
1CR
        C5 BC
                JZ $18C
                                 ;Exit if this key "bounced"
100
        FB
                MOV A,R3
                                 ;Get the changed mask
1CD
        D 1
                XRL A, 981
                                 ;Change the bit in the array
1CE
        81
                MOV @R1,A
                                 ;Replace the new result
1CF
        FB
                MOV A.R3
                                 ;Compute key number from the mask
100
        57
                                 ;Test if it was OPTION
                BRC A
1D1
        E6 D7
                JNC $107
                                 ; If not, skip to next test
103
        23 75
                                 ; OPTION = 58
                MOV A,#$75
105
        24 E9
                JMP $1E9
                                 ;Skip to end
107
        67
                RRC A
                                 ;Was it SHIFT?
108
        E5 E0
                JNC $1E0
                                 ;!f not, skip to next test
1DA
        23 71
                MOU A,#$71
                                 ;SHIFT = 56
        24 E9
100
                JMP $1E9
                                 ;Skip to end
1DE
        00
                NOP
                                 ;Take up waste space
1DF
        00
                NOP
                                 ;Take up waste space
1E0
        67
                RRC A
                                 ;See if was CTRL
1E1
        E6 E7
                JNC $1E7
                                 ;If not, assume CAPS LOCK
1E3
        23 6F
                MOV A,#$6F
                                 ;CTRL = 55
1E5
        24 E9
                JMP $1E9
                                 ;Skip to end
        23 73
                MOV A,#$73
                                 ; CAPS LOCK = 57
        48
                ORL A,R2
                                 ;Combine with pushed/released info
1EA
        90
                MOV R5, A
                                 ;Save result
1EB
        97
                CLR C
                                 ;Put a zero there
1EC
        Ĥ7
                CPL C
                                 ;Set CY to show key
1ED
        83
                RET
                                 ;Done
                ;Debounce a keyboard key read from PO
                ;Entry R1 = Pointer to key status table
                        Ports set up to read the keys from PO
                        CY = 0 If no valid keypush
                 Exit
                        CY = 1 If a valid change
                         A = R5 = Result code
                        A,F,R2,R3,R4,R5,R6,R7,&1
                :Uses
1EE
        00
                NOP
                                 ;Delay
1EF
        08
                IN A,PO
                                 ;Get key status
1F0
        43 00
                OR A, #$CO
                                 ;Remove non-scan lines
1F2
                                 ;Save result
        AD
                MOV R5,A
1F3
        D1
                XRL A, eR1
                                 ;See if there were any changes
1F4
        96 F8
                JNZ $1F8
                                 ; If so
1F6
        97
                CLR C
                                 ;Flag none
1F7
        83
                PET
                                 ;Exit then
```

A = R5 = Result code

;Debounce a key and compute it's return code

```
R1 = Pointer to key status table
                         R5 = Initial read status of keys
                         Scan line set up to read the keys
                  Exit
                         CY = 0 If no valid change
                         CY = 1 if valid change
                         A = R5 = Result code
                         A,F,R2,R3,R4,R5,R6,R7,&1
                 :Uses
        54 E7
1F8
                 CALL $2E7
                                  ;Compute mask of one of the keys
        54 81
                 CALL $281
                                  ;Delay 4.2 ms
1FC
        F1
                 MOV A, PR1
                                  ;Get the present status
1FD
        58
                 ANL A,R3
                                  ;Mask it for only one key
1FE
        ЯC
                 MOV R4,A
                                  ;Save result
1FF
                                  ;Get contents
        F8
                MOV A,RO
200
                MOV R6, A
        ĤΕ
                                  ;"PUSH" it
201
        B8 2C
                MOV RO, #$20
                                 ;Load pointer
203
                MOV A, 2RO
        FO
                                 ;Get flag byte
204
        67
                RRC A
                                 ;Check Isb
205
        F6 11
                 JC $211
                                  ;!f set (performed as if keupad)
207
        00
                HOP
                                 ;Delay
208
        08
                IN A,PO
                                  ;Get status of keys again
209
        58
                 ANL A.R3
                                 ;Mask for only the present one
20A
        DC.
                XRL A,R4
                                  ;See if status change still present
208
        96 27
                JNZ $227
                                 ;Yes - then a valid change
                MOV A,R6
                                  ;"POP" old RO contents
20D
        FE
20E
        88
                MOV RO,A
                                 ;Replace them
20F
        97
                CLR C
                                  ;Flag no changes
210
        83
                RET
                                 ;Done
        F9
                MOV A,R1
                                 ;Get table pointer
        03 24
                XRL A,#$24
                                 ;See if 24
214
        05 07
                JZ $207
                                 ;Then status comes from PO
216
        D3 07
                XRL A,#$07
                                 ;See if was 23
218
        96 10
                JNZ $210
                                 ;If the scan is multiplexed
218
        99
                IN 8,21
                                 ;Get keys from here
218
        44 09
                JMP $209
                                 ;Join rest of code
                IN A,P2
210
        08
                                 ;Get multiplexed keys
21E
        77
                BR A
                                  ;Toss P20
21F
        43 F9
                OR A,#$F9
                                  ;Mask for only P22,P23
221
        56 09
                JT1 $209
                                 ;If OK, then join rest
223
        53 FE
                                  ;Copy T1 to DO
                AND A, #$FE
225
        44 09
                JMP $209
                                 ;Join rest of code
227
        B8 2C
                MOV RO, #$2C
                                 ;Load pointer to keypad flag byte
229
        FO
                MOV A,®RO
                                 ;Get the flag
22A
        53 08
                AND A,#$08
                                 ;Test a bit (This will always be a ()
220
        96 41
                JNZ $241
                                 ;Skip 2-key rollover limiting
22E
        B8 2D
                MOV RO, #$2D
                                 ;Load pointer to # of keys pushed
230
        FD
                MOV A,R5
                                 ;Get scan result
231
        58
                ANL A,R3
                                 ;Mask for the present key
232
        05 38
                JZ $238
                                  ;If it was pushed
234
        F0
                MOV A,@RO
                                 ;Get present value
235
        C6 41
                JZ $241
                                 ;Don't let it go negative
237
        Ũ7
                DEC A
                                 ;Remove one
```

;Replace result

238

A0

MOV @RO,A

Entry A = "Changed mask"

```
239
        44 41
                JMP $241
                                 ;Skip to remaining code
238
        FÜ
                MOV A. @RO
                                 ;Get flag value
230
        53 02
                                 ;See if 2 keys pushed
                ANL A, #$02
        96 OD
                JNZ $20D
                                 ; If so, then don't take this one
<u>ل. غ</u>
                INC PRO
        10
                                 ;Count keys pushed
241
        FB
                MOV A,R3
                                 ;Get mask
242
        01
                XRL A, @R1
                                 ;Flip status of that one key
243
        A1
                MOV @R1.A
                                 ;Replace result
244
        F9
                MOV A.R1
                                 ;Get table pointer
245
        53 OF
                ANL A,#$OF
                                 ;Compute the offset from start (20)
247
        RΕ
                MOV R6,A
                                 ;Save value for multiply by 6
248
        97
                CLR C
                                 ;Clear to allow 6502 style ADD
249
                RL A
        E7
                                 ;Times 2
24R
                RL A
        E7
                                 ;Times 4
248
        7E
                ADDC A,R6
                                 ;Times 5
240
        7E
                ADDC A,R6
                                 ;Times 6
240
        75
                ADDC A,R7
                                 ;Add position in this row of the key
24E
        87
                CPL C
                                 ;Set it (was still clear)
        F7
24F
                RLC A
                                 ;Make Isb a one (for a stop bit)
250
       48
                ORL A,R2
                                 ;Combine with pushed/released info
251
        24 EA
                JMP $1EA
                                 ;Join exit code
                ;Scan keypad keys
                 Entry None
                        CY = 0 if no change found
                :Exit
                        CY = 1 if change found
                         A = A5 = Result code
                :Uses
                        A,F,R0,R1,R2,R3,R4,R5,R6,R7,&2
\neg = \neg
        B8 3E
                MOV RO, #$3E
                                 ;Load buffer pointer
        FO
                MOU A, 9RO
                                 ;Get last byte of buffer
256
        97
                CLR C
                                 ;Get ready to show no change
257
        C6 5A
                JZ $25A
                                 ; If space in buffer
259
        83
                RET
                                 ;Don't do anything
258
        23 FB
                MOV A.#$FB
                                 ;Load initial mask value
250
        B9 20
                MOV R1,#$20
                                 ;Load pointer to key status
25E
        88
                MOV ROJA
                                 ;Save port value
25F
        90
                CUTL PO.A
                                 ;Select a scan line
260
        54 78
                CALL $27A
                                 ;Scan this group
                                 ;Load inactive value
262
        23 FF
                MOV A, #$FF
264
        90
                OUTL PO,A
                                 ;Deselect the line
265
        F6 79
                JC $279
                                 ; If a key change was found
267
        19
                INC R1
                                 ;Advance table pointer
268
        A7
                CPL C
                                 ;Set it
269
        F8
                MOV A,RO -
                                 ;Get present select bute
268
        57
                RRC A
                                 ;Move over to right
268
        F6 5E
                JC $25E
                                 ; if not done yet
26D
                                 ;Bet status of some more keys
        09
                IN A,P1
26E
        54 82
                                 ;Test these
                CALL $282
270
        F6 79
                JC $279
                                 ;!f a change
272
        19
                INC R1
                                 ;Advance table pointer
273
        00
                NOP
                                 ;Delay
274
       08
                IN A,PO
                                 ;Get remaining keys
       43 E7
275
                                 Mask for only two lines
                ORL A, #$E7
277
       44 82
                JMP $282
```

;Check these

```
Entry Mone
                 ;Exit
                         A = T = $00
                         TF = 0
                         A,TF,T
                 ;Uses
28A
        27
                 CLR A
                                 ;Load 0
                 ;Load a value to the timer
                   This routine can be used to initialize the timer
                 ;and to clear the timer flag.
                 Entry A = Data
                 Exit:
                         T = \tilde{H}
                         TF = 0
                         TF,T
                 ;Uses
28B
        62
                A,T VOM
                                 ;Send it to timer
                 JTF $28E
280
        16 BE
                                 ;Clear timer flag
28E
        83
                 PET
                                 ;Done
                 ;Copy P20 input to CY
                 ; This routine will test the P2O serial line. It
                 ;will do a small amount of checking for noise also.
                 ;Entry P20 = 1 (to allow input)
                        CY = P20
                 ;Exit
                 ;Uses
                        A,F
        97
                CLR C
                                 ;Clear result flag
        08
                IN A.P2
                                 ;Get P2 data
        53 01
                ANL A,#$01
                                 ;Mask for P20 input from Mac
203
        C6 CB
                JZ $208
                                 ;If active, exit
205
        OR
                IN A,P2
                                 ;Get data again
206
        53 01
                ANL A,#$01
                                 ;Check it again in case of noise
208
        C6 CB
                JZ $2CB
                                 ;Exit if active now
2CR
        87
                CPL C
                                 ;P20 was a one
2CB
        83
                RET
                                 ;Done
                ;OR A to flag
                  This routine can be used to set bits in the
                 options flag byte.
                ;Entry A = Mask
                ;Exit A = Result
                        RO = $2C
                        (\$2C) = (\$2C) OR A
                        A,RO,($20)
                :Uses
2CC
        BS 20
                MOV RO, #$20
                                 ;Load address pointer
2CE
        40
                ORL A, PRO
                                 ;Or new value to old
2CF
        80
                MOV @RO,A
                                 ;Replace result
200
        83
                RET
                                 ;Done
```

```
;AND A to flag
                   This routine can be used to clear bits in the
                 joptions flag byte.
                 Entry A =Mask
                 Exit:
                         A = Result
                         R0 = $20
                         ($2C) = ($2C) AND A
                 ;Uses
                         A,RO,($2C)
201
        88 20
                MOV RO, #$20
                                 ;Load address pointer
203
        50
                ANL A, PRO
                                 ;And new value
2D4
        A0
                MOU PRO, A
                                 ;Replace result
205
        83
                RET
                                 ;Done
                 ;Read special keys from keyboard
                   This subroutine will read the status of the
                 special keys on the keyboard and return the
                 result in the format needed for the key
                 ;table.
                 Entry P06 = P22 = P23 = 1
                 Exit A = R5 = Result (0 = pushed)
                          SO = OPTION
                          D1 = SHIFT
                          D2 = CTRL
                          D6 = CAPS LOCK
                          03-05,07 = 1
                 :Uses A,R5
206
        OA.
                IN A,P2
                                 ;Get special key status
207
        77
                RR A
                                 ;Throw away P20 serial input
                OR A,#$F9
208
        43 F9
                                 Mask for only SHIFT and CTRL
2DR
        56 DE
                JT1 $2DE
                                 ;Test OPTION
2DC
        53 FE
                ANL A, #$FE
                                 ;Transfer it to 80 with other keys
2DE
        53 BF
                ANL A, #$BF
                                 ;Clear bit for CAPS LOCK status
2E0
        AD.
                                 ;Save result
                MOV R5.A
2E 1
        08
                IN A.PO
                                 ;Get CAPS LOCK status
        53 40
2E2
                ANL A,#$40
                                 ;Mask for only it
2E4
        40
                ORL A.R5
                                 ;Combine with other status
2E5
        ΑD
                MOV R5,A
                                 :Replace result
2E6
        83
                RET
                                 ;Done
```

;Set up masks for a keupush

This routine will take a mask with 1's at the locations where a key changed state and single out one of them and prepare a mask to use to process it further. Key releases are processed first so that the Mac does not see more keys pushed at any time than there really are.

```
R1 = Pointer into key status table
                         R2 = $00 \text{ (pushed) or } $80 \text{ (released)}
                 Exit:
                         R3 = Bit set where the change occurred
                         R7 = Bit number of the key (0-7)
                 :Uses
                         A,F,R2,R3,R4,R6,R7
2E7
        BB FF
                 MOU R3,#$FF
                                  ; Init for counter
2E9
        RΕ
                 MOV R6,A
                                  ;Save "changed" mask
2EA
        97
                 CLR C
                                  ;Don't shift any ones in
2E8
        67
                 RRC A
                                  ;Pop bits out to test for a logic 1
2E0
        18
                 INC R3
                                  ;Count bit position
                 JMC $2EB
2ED
        E6 EB
                                  ;Loop until a 1 is found
2EF
        AC
                MOV R4,A
                                  ;Save partial result
2F0
        FB
                MOV A.R3
                                  :Get counter
2F 1
        ЯF
                MOV R7,A
                                  ;Save it too
2F2
        27
                                  ;Load a blank workspace
                 CLR A
2F3
        AA.
                 MOV R2.A
                                  ;Clear pushed/released flag
2F4
        18
                 INC R3
                                  ;Adjust for loop
2F5
                 RLC A
        F7
                                  ;Shift the one back to where it was
2F6
        EB F5
                 DUNZ R3,$2F5
                                  ;Loop for correct number of bits
2F8
        ĤΒ
                MOV R3,A
                                  ;Save mask containing a single "change"
2F9
        00
                MOP
                                  ;Delay
2FR
        00
                MOP
                                  ;Dellau
2FB
        00
                NOP
                                  :Delay
2FC
        00
                MOP
                                  :Delay
2FD
        00
                MOP
                                  ;Delau
2FE
        01
                XRL A, PRI
                                  :Invert key status data
2FF
        58
                 ANL A,R3
                                  ;Mask only for this key
300
                 JZ $305
        05 05
                                  ;lf it was just pushed down
302
        BA 80
                 MOV R2,#$80
                                  ;Load "released" mask
        83
                                  ;Done (process release before any pushes)
                RET
        FC
                MOV A,R4
                                  ;Get partial shift result
396
        96 09
                JNZ $309
                                  ; If multiple push, reloop
308
        83
                RET
                                  ;Done ("pushed")
309
        FB
                MOV A,R3
                                  ;Get mask of changes
30A
        37
                CPL A
                                  ;Flip state
308
        5E
                ANL A,R6
                                  ;Remove the first key found
        44 E7
30C
                JMP $2E7
                                  :Reloop
                 ;Set CY if the $30 or $3A FIFO's contain data
                   This subroutine will test both of the FIFOs for
                 data. The carry flag will be cleared if both
                 ;FIFOs are empty.
                 Entry Mone
                 ;Exit
                         CY set if either had data
                 ;Uses
                         A,F,80
30E
        97
                CLR C
                                 ;Init to assume both empty
30F
        B8 3A
                MOV RO,#$3A
                                 ;Load pointer
311
        Fΰ
                MOV A, @RO
                                 ;Get first data byte from this FIFO
312
        96 19
                JMZ $319
                                 ; If some contents
314
        88 30
                                 ;Load pointer
                MOV RO,#$30
316
        FO
                MOV A, ERO
                                 ;Get first data byte from this FIFO
```

;Entry A = "Changed" mask

```
317
                JZ $31A
                                ;Exit if it was empty
319
        87
                CPL C
                                :Flag some contents
31A
        83
                RET
                                :Done
                ;Send an "awake" code if nothing else to send
                 This routine will check the keyboard FIFO for data.
                ilf it is empty, it will place a "NUL" event code
                ;there. In any case, it will reset the idle timer
                 and go try to send the result (or data) to the Mac.
                ;This code is branched to from the command loop.
                :Entry None
                 Exit
                        R0 = $30
                        (\$30) = \$78 if nothing else in FIFO
                        ($2E) = $00
                ;Uses
                        A,F,T,&1
318
        54 86
                CALL $286
                                ;Init timer and reset inactive counter
310
        B8 30
                MOV RO,#$30
                                ;Load pointer into FIFO
31F
        F0
                MOV A, @RO
                                ;Get present value
        95 24
320
                JNZ $324
                                ;If something already present - skip
                                ;Load awake code
        80 78
322
                MOU @RO,#$78
324
        04 55
                JMP $055
                                ;Go send it
                ;Reset keyboard
                  This short piece of code is executed from the command
                processing loop in response to a keyboard/keypad
                reset command. The result is always 03 and is set up
                ;to be sent out the serial port.
                Entry Mone
                        R0 = $36
                Exit:
                        (\$35) = \$03
                ;Uses
                        AII,&2
326
       54 89
                CALL $289
                                ;Reset/initialize RAM
328
        B8 36
                MOV RO.#$36
                                ;Load pointer to result "FIFO"
32A
       80 03
                MOV @RO.#$03
                                ;Load return code saying "We're here!"
320
       04 55
                JMP $055
                                ;Go send it
```

;Transmit a data byte on P21

This routine will send 8 bits out P21 when supplied with an external clock into P06. The msb will be sent first. The code has several time outs. If one of these is reached, the transfer is aborted with the ports left however they were last. Initially, the routine will wait for the clock input to be inactive — it can wait forever for this. It will then set P21 low to act as a processor of the start bit and the wait up to about 2048 cycles for the first clock edge. On all following bits, it will wait up to about 8192 cycles per bit before timing out. The coutput data will change within 8 to 18 cycles of the failing edge of the clock. The clock should be low from 10 to 22 cycles and the maximum data rate is one bit every 22 cycles. Since P21 is left at the last sent data even during a time-out, the calling code should preset it to a 1 to allow future input.

```
;Entry A = Data
; PC5 = P21 = 1
;Exit No timeout
; A = R1 = $00
; P2 = $FF
; Timeout
; A = $40
; R1 = Count of unsent bits
;Uses A,F,TF,R1,R5,P2,T,&1
```

```
AD
                 MOV R5,A
                                  ;Save transmit data
        00
                 NOP
                                  ;Delay
        54 88
                 CALL $28A
                                  :Reset timer
332
        08
                 IN A.PO
                                  ;Set clocking input
333
        53 40
                 ANL A,#$40
                                  :Mask for POS
335
        C5 32
                 JZ $332
                                  ;Loop until inactive
337
        23 00
                 MOV A.#$CO
                                  ;Load new timer data
339
        54 88
                 CALL $288
                                  ;Set it in
33B
        89 08
                 MOV R1,#$08
                                  ;Load data length
33D
        23 FD
                 MOV A,#$FD
                                  ;Load a "0" for P21
33F
        38
                 OUTL P2.A
                                  ;Send start bit on P21 to wake up receiver
340
        FD
                 MOV A,R5
                                  ;Get transmit data
341
        F7
                 RLC A
                                  :Shift over
342
        ad
                 MOV R5, A
                                  :Replace result
343
        F6 59
                 JC $359
                                  ;Send a one bit
345
        08
                 IN A,PO
                                  ;Get clock input
345
        53 40
                 ANL A.#$40
                                  :Mask for clock input on PO6
348
        C6 4E
                 JZ $34E
                                  ;Send the next bit
34A
        15 8B
                 JTF $38B
                                  ;Error timeout
                 JMP $345
34C
        64 45
                                  ;Wait for falling clock edge
34E
        23 FD
                MOV A.#$FD
                                  ;Load output data
350
                                  Send a "0" on P21
        3A
                 OUTL P2, A
351
        27
                 CLR A
                                  ;Load a 0.
352
        52
                MOU T.A
                                  :Reset the timer
353
        16 55
                JTF $355
                                  ;Reset timer flag
```

```
355
        00
                 HOP
                                  ;Delay
356
        E9 40
                 DUNZ R1,$340
                                  ;Loop for full byte
358
        83
                 RET
                                  :Done
        98
                 IN A,PO
                                  ;Get clocking input
        53 40
35A
                 ANL A, #$40
                                  ; Mask for PO6
350
                 JZ $362
        C6 62
                                  ;Send the next bit
                 JTF $38B
35E
        15 8B
                                  ;Error timeout
                 JMP $359
360
        64 59
                                  ;Wait for falling clock edge
        23 FF
                 MOU A,#$FF
362
                                  ;Load output data
364
        3ล
                 OUTL P2,R
                                  ;Set a "1" on P21
365
        27
                 CLR A
                                  ;Load a 0
366
        62
                 MOV T, A
                                  :Reset timer
367
        16 69
                 JTF $369
                                  ;Clear timer flag
369
        00
                 MOP
                                  ;Delay
35A
        E9 40
                 DJNZ R1,$340
                                  ;Loop for full bute
360
        83
                 FIET
                                  ;Done
```

### ;Receive a data byte on P21

; This routine will input a data byte from the P21 line. ;The msb will be read first. A clocking input should be ;provided on POB. The code here will sample the data ;input within B to 1B CPU cycles of the falling edge of ;the clock line. The data should not be sent faster than ;1 bit every 20 CPU clock cycles. If the entire byte has ;not been transferred within about 8192 CPU cycles, the ;routine will abort the transfer. The clock input should ;be low for a maximum of 19 cycles and a minimum of 10.

```
36D
        00
                NOP
                                 :De Lau
36E
        54 SA
                CALL $28A
                                 ;Clear timer for a timeout
370
        AD.
                MOV R5, A
                                 ;Clear receive byte (A = 00)
371
        89 08
                                 ;Load data length to receive
                MOV R1,#$08
                                 Load an $FF for output data
373
        07
                DEC A
374
                                 ;Set outputs inactive
        38
                CUTL P2,A
375
        08
                                 ;Test the clock input
                IN A.PO
376
                                 ;Mask for PO6
        53 40
                ANL A,#$40
        C5 7E
378
                JZ $37E
                                 ;Get the bit if the clock went low
37R
        16 88
                JTF $38B
                                 ;Error timeout if timer times out
370
        64 75
                JMP $375
                                 ;Reloop until timeout or full bute
37E
        OA
                                 ;Input data from source
                IN 8,P2
37F
        53 02
                ANL A,#$02
                                 :Mask for P21
381
        97
                CLR C
                                 ;Get ready for shift
382
        06 85
                JZ $385
                                 :Make CY = P21
384
        87
                CPL C
                                 ;Switch state to a '1'
```

```
385
        FD
                MOV A,R5
                                 ;Get receive shift register
386
        F7
                 RLC A
                                 ;Shift in new data
387
        ĤD
                MOV R5,A
                                 :Replace result
        00
                MOP
                                 ;Delay
        E9 75
                DUNZ R1,$375
                                 ;Fetch full byte
36B
        93
                RET
                                 :Done
                 ;Transmit a byte on P20 from a F1F0
                   This routine will send 8 bits of data that were
                 istored in a FIFO out the P2O 1/O line. A clock is
                 provided out PO7 to clock the data. The msb will be
                 ;sent first. Assuming no hardware errors, the lsb of
                 ;the data sent will end up being a '1'. This routine
                 ;will take 286 CPU cycles to execute (including the CALL
                 ;to get here) when the lsb of the data sent is a '1'
                 ;and the data sent was the only entry in the FIFO.
                 ;The clock will be low 16 cycles and high 17 cycles and
                 ;the data out will change 4 cycles before the falling
                 clock edge. Once the data byte has been sent, the FIFO
                 ; is moved down one byte with the vacated position filled
                 ;with a O. The command flag is also cleared – thus
                 turning off whatever command caused this data bute to
                 ;be sent.
                   As written, the FIFO handling has two bugs. If the
                 ;FIFO contained 5 bytes of data, the last byte will
                ;never get cleared. It will replicate itself forever
                ;much like the sign bit in an ASR operation. Also, in
                ;this situation, the command byte will not get cleared
                ;but rather the last data byte will also replicate
                :itself there.
                ;Entry RO = Start of FIFO to take data from
                 :Exit
                        A = (2F) = 0
                        RO = 2F
                        R4 = 00
                        P0 = P2 = FF
                        A,F,R0,R1,R3,R4,P0,P2,&1
                :Uses
380
        89
                MOV R1.#$08
                                 ;Load number of bits to send
38E
        FO
                MOV A, eRO
                                ;Get a data byte from the FIFO
38F
        ΑB
                MOV R3,A
                                 ;Save it in R3
390
        F7
                RLC A
                                 ;Test the present msb
                MOV R3,A
391
        88
                                 ;Save rotating result for next loop
392
        F6 B9
                JC $389
                                ;Transmit a logic one if a '1'
                ;Transmit a logic zero
394
        23 FE
                MOV A #$FE
                                ;Load pont data
395
        3A
                OUTL P2.A
                                ;Set P20 to a logic 0
397
        23 7F
                MOV A,#$7F
                                ;Load clocking data
```

;Get clock ready

;Delay

;Delay

:Load delay count

;Delay total of 8 CPU cycles

399

39A

390

39E

39F

90

00

00

BC 01

54 83

OUTL PO.A

CALL \$283

MOP

MOP

MOV R4,#\$01

```
380
        23 FF
                 MOV A, #$FF
                                  ;Load clocking data
382
        90
                 OUTL PO.A
                                  ;Clock data in at destination
3A3
        FΒ
                 MOV A,R3
                                  ;Get remaining data to send
        00
                 MOP
                                  ;Delay
        00
                 NOP
                                  ;Delay
386
        E9 90
                 DUNZ R1,$390
                                  ;Loop for rest of the bits
                 ;Move the FIFO down one byte
388
        BC 04
                 MOV R4,#$04
                                  ;Load max size move *** BUS ***
3AA
        F8
                 MOV A,RO
                                  ;Get destination pointer
3AB
        17
                 INC A
                                  ;Advance it 1
3AC
        PA
                 MOV R1, A
                                  ;Set source just above destination
3AD
        F1
                 MOV A, PR1
                                  ;Get a bute
3AE
        80
                 MOV @RO,A
                                  ;Move it down one F!FO entry
3AF
        C6 B5
                 JZ $385
                                  ;Exit if end of data
381
        18
                 INC RO
                                  ;Advance destination pointer
382
        19
                 INC R1
                                  ;Advance source pointer
383
        EC AD
                 DUNZ R4,$3AD
                                  ;Loop for up to 4 bytes of data in FIFO
385
        B8 2F
                 MOV RO, #$2F
                                  ;Load pointer to command byte
387
        80
                 MOV @RO.A
                                  ;Attempt to clear it *** BUG ***
3B8
        83
                 RET
                                  ;Done
                 ;Transmit a logic one
389
        23 FF
                 MOV 9,#$FF
                                  ;Load port data
38B
        3A
                 OUTL P2.A
                                  ;Set P20 to a "1"
3BC
        23 7F
                 MOV A,#$7F
                                  ;Load clocking data
3BE
        90
                 OUTL PO.A
                                  ;Get clock output ready
3BF
        80 01
                MOV R4,#$01
                                  ;Load delay count
301
        54 83
                 CALL $283
                                  ;Delay total of 8 CPU cycles
        ŨŨ
                HOP
                                  :Delau
        00
                MOP
                                  ;Delay
305
        23 FF
                MOV 8, #$FF
                                  ;Load clocking data
307
        90
                 OUTL PO.A
                                 ;Clock data in at destination
308
        FB
                MOV A,R3
                                  ;Get remaining data to send
309
        00
                MOP
                                  ;Delay
3CA
        00
                MOP
                                 ;Delay
3CB
        E9 90
                DJNZ R1,$390
                                  ;Loop for rest of bits
3CD
        54 A8
                 JMP $3A8
                                 ;Go to FIFO move
```

#### ;Receive a data byte from P20

This routine will read 8 bits of serial data from the ;P20 I/O line. It will supply a data clock out the PO? ;output to clock data out of the source. The msb is ;transferred first. The data read will be left in the A ;register and also placed in the active command flag. ;Thus any commands received will automatically be set as ;the current command to execute. The lsb of all received ;data will be pitched so that it can be used as a stop ;flag by the sender. This routine will take 333 CPU cycles ;to execute (including the CALL to get here). The clock ;is low for 18 cycles and high for 22. The data will be ;sampled 8 CPU cycles after the rising edge of the clock.

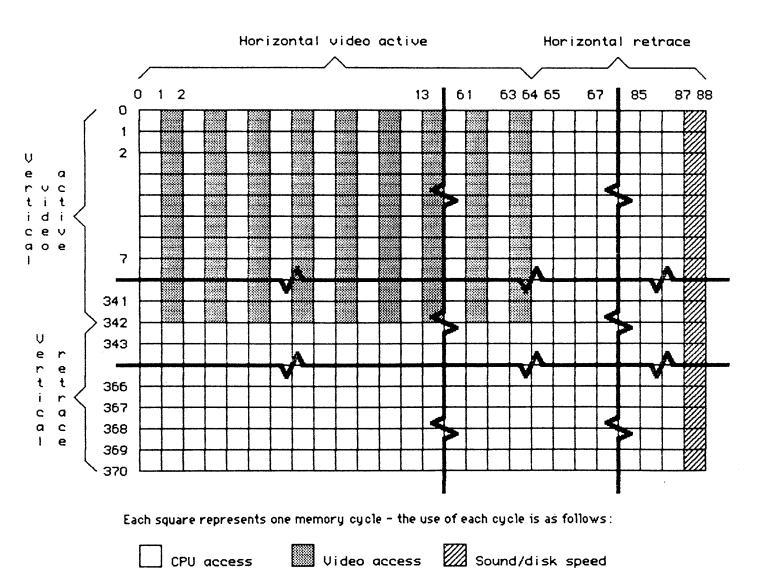
```
:Exit
                         A = (2F) = Data AND FE
                         R0 = 2F
                         R1 = R4 = 00
                         R3 = Data
                         P0 = FF
                 :Uses
                         A,F,RO,R1,R3,R4,P0,&1
3CF
        BA 00
                MOV R3,#$00
                                  ;Clear receive register
3D1
        89 08
                 MOV R1.#$08
                                  ;Load length in bits to receive
        23 7F
3D3
                 MOV A. #$7F
                                  ;Load elocking data
305
        90
                 OUTL PO.A
                                  Set PO7 low to clock out next bit
                MOV 84,#$02
306
        BC 02
                                  ;Load delay count
        54 83
3D8
                 CALL $2A3
                                  ;Delay total of 12 CPU cycles
3DA
        23 FF
                MOV A,#$FF
                                  ;Load elocking data
3DC
        90
                 OUTL PO.A
                                  ;Set PO7 high or inactive
300
        F٩
                MOV A,R3
                                  ;Get serial shift register
SDE
        E7
                BL A
                                 ;Move present data over one bit
3DF
        AA
                MOV R3,A
                                 ;Save result for later OR with new
3E0
        00
                MOP
                                 ;De lau
3E 1
        00
                MOP
                                  :Delau
3E2
        00
                NOP
                                 :Delay
3E3
        OA.
                IN A,P2
                                  ;Get input data from P20
3E4
        53 01
                ANL A,#$01
                                 ;Mask for only serial input in DO
326
        48
                ORL A,R3
                                  ;Combine with previously read bits
3E7
        AA
                MOV R3,A
                                  ;Replace result for next time
3E8
        00
                MOP
                                  ;Delay
3E9
        00
                MOP
                                  ;Delay
3EA
        00
                MOP
                                  ;Delay
3EB
                                 ;De lay
        00
                MOP
3EC
        E9 D3
                DJNZ R1,$3D3
                                 ;Loop for a complete byte of data
        53 FE
                ANL A, #$FE
                                 ;Mask out isb (stop bit) of input data
        88 2F
                MOV RO, #$2F
                                 ;Load pointer to command flag
3F2
        90
                MOU PRO,A
                                 ;Save input data with Isb=O there
3F3
        83
                RET
                                 ;Done
                ;Use up waste space to full 1K
3F4
        FF
                DCB
                         $FF,12 ;Empty space
                END
```

; Entry P20 = P07 = 1

Reg.	Keyboard usage	Keypad usage
06-07 08-17 18-1F 20-24 25-29 28	General purpose registers 8 level processor stack Unused Key status table (O = pushed) Rest of key status table Unused Unused	General purpose registers 8 level processor stack Unused Key status table (0 = pushed) Unused Unused Counts loops if keyboard is not responding (absent).
20	Keypad flag byte. Left at 0 to disable some keypad loop flags. DO = Set to flag keypad mode D1 = Provides divide by 2 for idle timer (keyboard absent) D2 = Set when keypad prefix sent D3 = Set to a one disable 2-key rollover (always zero) D4-D6 = Unused - left at zero D7 = Set if keyboard absent	Keypad flag. See at left for uses of the various bits.
2D 2E	Unused Idle timer. When it reaches 2,	Unused Idle timer. Every 28 loops (56
	a data byte is sent to the Mac. Reset when data is sent. Counts	if keyboard absent) of idle time, a data byte will be sent to the
2F	when the timer times out. Command flag. received commands are placed here for processing. After they are executed, the location is cleared. 10 = Permission to send 1 byte 14 = Request for data 16 = Perform a reset 35 = Perform self test	Mac. Command flag. received commands are placed here for processing. After they are executed, the location is cleared. 10 = Permission to send 1 byte 14 = Request for data 16 = Reset keyboard and self 36 = Self test on keyboard
30-34	Else = NOP Keyboard FIFO. Data is placed in the first zero location and taken out of location 30.	Else = NOP FIFO of data read from the keyboard. The operation is identical to the keyboard mode.
35	Zero to allow the above FIFO	Zero to allow the above FiFO
36	to empty. Single byte FIFO for passing	to empty. Single byte FIFO for passing
37	responses back. Zero to clear the above	responses back. Zero to clear the above
38-39 38-3E	Unused Unused	Unused
Ull UL	or idaded	Keypad FIFO. Operates similar to the other FIFOs.
3F	Unused	Zero to clear the above
Port	Keyboard usage	Keypad usage
00-02 03-04 05 06 07	Scan return lines Scan return lines Scan return line Status of CAPS LOCK Clocking output to Mac or keypad	Scan output lines Non multiplexed key status Tied low to select keypad mode Clocking input from keyboard Clocking output to Mac

10-17 Scan output lines
20 Serial I/O to Mac or keypad
21 Scan output line
Status of SHIFT
Status of CTAL
T1 Status of OPTION

Non multiplexed key status Serial I/O to Mac Serial I/O to keyboard Scan return line Scan return line Scan return line



ASCII Binary Oct Dec Hex NUL 00000000 000 000 00 SOH 00000001 001 001 01 STX 00000010 002 002 02	NII - pull (zero)	ASCII Binary Oct Dec Hex 0 01000000 100 054 40 A 01000001 101 055 41
ETX 00000011 003 003 03 EOT 00000100 004 004 04		B 01000010 102 056 42 C 01000011 103 057 43 D 01000100 104 058 44
ENQ 00000101 005 005 05 ACK 00000110 006 006 06 BEL 00000111 007 007 07	ETX – end of text EOT – end of transmission ENQ – enquiry ACK – acknowledge BEL – bell	E 01000101 105 069 45 F 01000110 106 070 46 G 01000111 107 071 47
BS 00001000 010 008 08 HT 00001001 011 009 09	PS - bookspan	11 -1
LF 00001010 012 010 0A VT 00001011 013 011 0B FF 00001100 014 012 0C	Lt - line feed UT - vertical tab EE - form feed	J 01001010 112 074 48 K 01001011 113 075 48
CR 00001101 015 013 0D SO 00001110 016 014 0E	CR - carriage return SO - shift out	L 01001100 114 075 40 M 01001101 115 077 40 N 01001110 115 078 4E
SI 00001111 017 015 0F DLE 00010000 020 016 10 DC1 00010001 021 017 11	HT - horizontal tab  LF - line feed  VT - vertical tab  FF - form feed  CR - carriage return  SO - shift out  SI - shift in  DLE - data link escape  DC1 - device control 1  DC2 - device control 2  DC3 - device control 3	0 01001111 117 079 4F P 01010000 120 080 50
DC2 00010010 022 018 12 DC3 00010011 023 019 13	=======================================	
DC4 00010100 024 020 14 NAK 00010101 025 021 15 SYN 00010110 025 022 16	BC4 – device control 4 NAK – negative acknowledge SYN – synchronous idle	T 01010100 124 084 54 U 01010101 125 085 55
ETB 00010111 027 023 17 CAN 00011000 030 024 18 EM 00011001 031 025 19	ETB – end of transmit block CAN – cancel	W 01010111 127 087 57 X 01011000 130 088 58
SUB 00011010 032 026 1A ESC 00011011 033 027 1B	SUB - substitute SSC - escape	Y 01011001 131 089 59 Z 01011010 132 090 5A I 01011011 133 091 5B
FS 00011100 034 028 10 GS 00011101 035 029 1D RS 00011110 036 030 1E	FS - file separator GS - group separator BS - record separator	\ 01011100 134 092 50 1 01011101 135 093 50
US 00011111 037 031 1F SP 00100000 040 032 20	SYM - synchronous idle ETB - end of transmit block CAN - cancel EM - end of medium SUB - substitute ESC - escape FS - file separator GS - group separator RS - record separator US - unit separator SP - space	^ 01011110 136 094 5E - 01011111 137 095 5F ` 01100000 140 096 60
" 00100010 042 034 22 # 00100011 043 035 23		a 01100001 141 097 51 b 01100010 142 098 52 c 01100011 143 099 53
\$ 00100100 044 035 24 \$ 00100101 045 037 25 & 00100110 045 038 26		d 01100100 144 100 64 e 01100101 145 101 65
' 00100111 047 039 27 ( 00101000 050 040 28		f 01100110 146 102 56 g 01100111 147 103 57 h 01101000 150 104 68
) 00101001 051 041 29 * 00101010 052 042 28 + 00101011 053 043 28		i 01101001 151 105 69 j 01101010 152 106 68
, 00101100 054 044 20 - 00101101 055 045 2D . 00101110 056 046 2E		1 01101100 154 108 6C m 01101101 155 109 6D
/ 00101111 057 047 2F 0 00110000 060 048 30		n 01101110 156 110 6E o 01101111 157 111 6F p 01110000 160 112 70
1 00110001 061 049 31 2 00110010 062 050 32 3 00110011 063 051 33		q 01110001 161 113 71 r 01110010 162 114 72
4 00110100 064 052 34 5 00110101 065 053 35 6 00110110 066 054 36		t 01110100 154 116 74 u 01110101 165 117 75
7 00110111 067 055 37 8 00111000 070 056 38		∨ 01110110 166 118 76 ₩ 01110111 167 119 77 × 01111000 170 120 78
9 00111001 071 057 39 : 00111010 072 058 3A ; 00111011 073 059 3B		y 01111001 171 121 79 z 01111010 172 122 78 { 01111011 173 123 78
<pre></pre>		01111100 174 124 7C 01111101 175 125 7D
? 00111111 077 063 3F	DEL - delete	~ 01111110 176 126 7E DEL 01111111 177 127 7F

## The System Revealed

The Standard System File (05/02/84) contains both a resource and data fork. The data fork is loaded upon boot-time (system start-up) and installs various ROM patches -- the install entry point is located 512 bytes into the fork. The resource fork of the file is where most of the action takes place, it contains 26 individual resource types:

```
ALRT (Alert template)
4 resources - Used by the SFPackage (3) - each has the [Purgeable] bit set.
   -3994 (12 butes): "Can't find that disk."
   -3995 (12 bytes): "A system error occurred; please try again."
   -3996 (12 bytes): "Replace existing "10" ?"
   -3997 (12 bytes): "Disk is locked."
BNDL (Bundle)
1 resource
            (40 bytes): Links System File with FREF & its icon.
   Û
CDEF (Control definition function)
2 resources - each has the [Purgeable] bit set.
            (0666 bytes): Object code for various button definitions.
   1
            (1268 bytes): Object code for scroll bars definitions.
CURS (Cursor - cursor, mask, & hot-spot)
4 resources
   1
            (68 bytes) : I-Beam cursor
            (68 butes) : Cross cursor
            (68 bytes): Plus cursor
            (68 bytes): Watch cursor
DITL (Item list in a dialog or alert)
ố resources - Used by the SFPackage (3) - each has the [Purgeable] bit set.
   -3994 (058 bytes): "Can't find that disk."
   -3995 (078 bytes): "A system error occurred; please try again."
   -3996 (074 bytes): "Replace existing "10" ?"
   -3997 (052 bytes): "Disk is locked."
   -3999 (144 bytes) : "Save as:"
```

-4000 (168 butes): "Open"

```
DITL cont'd.
1 resource - Used by DiskInit package (2) - has the (Purgeaple) bit set.
   -6047 (422 bytes): "Initialize this disk?"
1 resource - Used by Key Caps - has the [Purgeable] bit set.
   -15936 (030 bytes): Key Caps dialog item list
DLOG (Dialog template)
2 resources - Used by the SFPackage (3) - each has the [Purgeable] bit set.
   -3999 (23 bytes) : "Save as:" - used in SFPutFile
   -4000 (23 bytes): "Open" - used in SFGetFile
t resource - Used by DiskInit package (2) - has the (Purgeable) bit set.
            (23 bytes): DiskInit dialog box
1 resource - Used by Key Caps - has the [Purgeable] bit set.
   -15936 (29 butes) : Key Caps dialog box
DRVR (Desk accessory or other device driver)
8 resources - each has the (Purgeable) bit set.
            (1024 bytes): ".Print" - Handles ImageWriter printing
   12
            (2444 bytes) : "Calculator"
   13
            (3392 bytes) : "Alarm Clock"
   14
            (1586 bytes) : "Key Caps"
   15
            (0932 bytes) : "Puzzle"
   16
            (2418 bytes) : "Note Pad"
   17
            (2824 bytes) : "Scrapbook"
   18
            (4038 bytes): "Control Panel"
DSAT (System startup alert table)
1 resource - [Locked] and [System Heap] bits set.
   0
            (482 bytes): Dialog information for system start-up.
FKEY (Command-Shift-Number routine)
2 resources - each has the [Purgeable] bit set.
   3
            (312 bytes): Code for screen snapshot to disk.
   4
            (094 bytes): Code for screen dump to printer.
FONT (Font)
27 resources - each has the [Purgeable] bit set except (Chicago-12).
   0
            (0000 bytes): "Chicago" font reference number.
   12
            (2940 bytes): (Chicago-12) [System Heap] bit set.
```

(0000 bytes): "New York" font reference number.

256

```
FONT cont'd.
            (2032 bytes) : (New York-9)
   265
            (2200 bytes): (New York-10)
   266
   268
            (2734 bytes) : (New York-12)
   270
            (3352 bytes) : (New York-14)
            (4516 bytes) : (New York-18)
   274
            (5260 bytes) : (New York-20)
   276
   280
            (6832 bytes) : (New York-24)
            (0000 bytes) : "Geneva" font reference number.
   384
            (2152 bytes) : (Geneva-9)
   393
            (2200 bytes) : (Geneva-10)
   394
   396
            (2734 bytes) : (Geneva-12)
            (3568 bytes) : (Geneva-14)
   398
   402
            (4864 bytes) : (Geneva-18)
   404
            (5848 butes) : (Geneva-20)
   408
            (7568 bytes) : (Geneva-24)
            (0000 bytes): "Monaco" font reference number.
   512
            (2026 butes) : (Monaco-9)
   521
   524
            (2464 bytes) : (Monaco-12)
            (0000 bytes): "Venice" font reference number.
   640
            (3604 butes) : (Venice-14)
   654
            (0000 bytes): "London" font reference number.
   768
   786
            (3268 bytes) : (London-18)
            (0000 bytes): "Athens" font reference number.
   896
             (4468 bytes) : (Athens-18)
   914
FREF (File reference)
4 resources
             (13 bytes) : "System" - system file name.
   0
             (13 bytes) : "Finder" - finder file name (start-up application).
    1
             (16 bytes): "ImageWriter" - print driver file name.
   2
             (21 bytes): "Clipboard File" - clipboard file name.
   3
FRSV (Font reserved for system use)
1 resource
    1
             (10 bytes) : Hmmm?...
ICN# (Icon list - icon & mask)
 1 resource
             (256 bytes): System File application icon
    3
```

```
ICON (Icon)
 4 resources - each has the (Purgeable) bit set.
             (128 bytes): Alert face with "!"
    1
             (128 bytes): Alert face with "*"
             (128 bytes): Alert face with "?"
             (128 bytes): DiskInit icon for DITL -6047
 INIT (Initialization resource)
3 resources - each has the [Locked] and [System Heap] bits set.
             (646 bytes): Keyboard software map
    0
    1
             (100 bytes): Keypad software map
    2
             (768 bytes): Dialog information for system "bomb-box".
INTL (International resource)
2 resources - each has the [Purgeable] bit set.
             (032 bytes): International format settings (time, date, etc...)
   0
             (332 bytes): International date names (days, months, etc...)
    1
MACS (Program Information)
i resource
   0
             (13 bytes) : "Mac Software"
MDEF (Menu definition procedure)
1 resource
   0
            (684 bytes): Code standard menus.
PACK (Package)
6 resources - each has the [Purgeable] bit set.
            (2430 bytes) : DiskinitPackage - [Locked] bit also set.
   3
            (3238 bytes): SFPackage - [Locked] bit also set.
   4
            (4540 bytes): Floating Point package
            (4190 bytes): Transcendental functions package
   5
            (1436 bytes): International Utilities - [Locked] bit also set.
   6
   7
            (0208 bytes): Binary-Decimal Conversion
      (Pattern - a space is required after the 'T')
2 resources
            (8 bytes) : Desktop pattern - [System Heap] bit set.
   16
   17
            (8 bytes): Scrollbar pattern
```

```
PAT (Pattern list - as seen in the Control Panel & MacPaint)
1 resource - has the [Purgeable] bit set.
            (306 bytes): 38 patterns
PICT (Picture)
8 resources - each has the (Purgeable) bit set.
   -15800 (477 bytes): Control panel item - caret blink rate
   -15801 (097 bytes): Control panel item - volume slider
   -15803 (877 bytes): Control panel item - keyboard rates
   -15804 (698 bytes) : Control panel item - mouse button rate
   -15805 (383 bytes): Control panel item - cursor scaling rate
   -15806 (917 bytes): Control panel item - volume scale
   -15807 (474 bytes): Control panel item - menu item flash rate
   -15968 (616 bytes): Alarm clock items
PREC (Print record)
1 resource
   2
            (72 bytes): Retains current print settings.
STR (String - a space is required after the 'R')
4 resources - each has the [Purgeable] bit set except -15871.
            (22 bytes): "Version 1.1 24-Apr-84" - system version number.
   -8192 (12 bytes): "ImageWriter" - print driver file name.
   -15871 (80 bytes): "The Note Pad is unavailable unless there are 2K
                         free bytes on the startup disk."
   -15872 (14 bytes): "Note Pad File" - note pad file name.
STR* (String list)
1 resource - has the [Purgeable] bit set.
   -15840 (265 bytes): 7 Scrapbook messages
WDEF (Window definition function)
2 resources - [Purqeable] bit set for WDEF 1.
            (1200 bytes): Object code for square-cornered windows.
   1
            (0838 bytes): Object code for rounded-cornered windows.
```

# Standard System File Data

Version Date

05/02/84

Resource Fork

127488 bytes 5120 bytes

Data Fork

ZSYS

Type Creator

MACS

File Attributes

Bundle & Inited (\$21)

File Flags

\$80 (not protected)

#### The Small System

The Standard System File (05/02/84) can be significantly reduced in size. The following may be removed and the Finder and MDS Edit will still function:

- CURS The thin-cross cursor (#2) may be removed if the Alarm Clock DRVR is also removed. The Wide-cross cursor (#3) may be removed.
- **DITL** The Key Caps dialog item list (#-15936) may be removed if the Key Caps DRVR is also removed.
- **DLOG** The Key Caps dialog template (#-15936) may be removed if the Key Caps DRVR is also removed.
- **DRVR** The .Print driver (#2) may be removed if both the ImageWriter file and the FKEY #4 resource are removed and no printing is to be done. If any hidden drivers (those whose names begin with a period) are left, at least one visible desk accessory must also be left or the Finder will crash. Note that many system resources may be removed if certain of these drivers are removed.
- **FKEY** Either or both of these resources (#3 and #4) may be removed if screen snapshots are not needed.
- **FONT** Almost all fonts may be removed. Don't remove a font name resource without removing all of it's actual fonts sizes too. Leave at least the Chicago-12 font since a font is required by the system.
- **INIT** The keyboard mapping function (#0) may be removed if both the keyboard and Key Caps accessory will not be used. The keypad mapping function (#1) may be removed if the keypad is not present or will not be used
- **PACK** Both the Floating Point package (#4) and Transcendental Functions package (#5) may be removed if the Calculator accessory is removed.

- **PAT\*** The pattern list (#0) may be removed if the Control Panel accessory is removed and MacPaint won't be used with this System.
- **PICT** The Alarm Clock pictures (#-15968) may be removed if the the Alarm Clock DRVR is removed. The remaining pictures (#-15807 through #-15800) may be removed if the Control Panel DRVR is removed.
- **PREC** The printer record (#2) may be removed if the Print driver is removed.
- **STR** The system version number string (#0) may be removed. The ImageWriter file name string (#-8192) may be removed if the ImageWriter file is removed. The Note Pad strings (#-15871 and #-15872) may be removed if the Note Pad accessory is removed.
- **STR#** The ScrapBook strings (#-15840) may be removed if the ScrapBook accessory is removed.
- **WDEF** The round cornered window definition function may be removed if the Calculator accessory is removed and no applications on the disk use this type of window.

The following resources are all used to build the DeskTop file. Once this has been constructed, these resources may be removed. However, if the desktop needs to be rebuilt, the system icons won't appear. The Finder will also pause for a very long time before opening the disk's window. All of these resources should be deleted at once: **BNDL** \*0, FREF \*0 through \*3, ICN\* \*3, and MACS \*0.

#### Resource Flags

This document describes the effects of the various resources flags listed in **The System Revealed**.

Any resource with the <u>system heap</u> flag set will be loaded into the system heap instead of the application heap. This means that these resources will be available even when the application heap is not usable, such as during a program launch or system boot. A constructive use of this flag will be mentioned in the latter half of this document.

Since the INIT and DSAT resources supply code and data that is important to the system's operation, these all must load to the system heap. In operation, these resources are detached from the System file once they are loaded. This allows them to stay in memory even when the system resource file is closed (such as when the startup disk is changed).

Two other resources load to the system heap - the desktop pattern and the system font. The Chicago-12 font must load there since it is used in system bomb and disk switch alerts - which may occur at any time. The desk pattern is placed into the system heap as it is needed at the start of every program when the Window Manager is initialized. If it were not in memory at this time, a disk switch might be needed before the screen was cleared.

The <u>preload</u> flag does not have much usefulness in the System file. In fact, it is generally ignored for system resources during boot time. Also, if the System file is opened by a resource editor, many unneeded resources might be preloaded into memory.

Only a few resources use the <u>locked</u> flag. As several low memory pointers reference the INIT and DSAT resources, they need to be locked once in memory. Several of the packages also have the locked bit set. There is no particular reason for this as they will be locked upon being called and unlocked when the package code terminates.

The <u>purgeable</u> bit by far has the most varied usage. It's use can profoundly affect system performance when memory space is low. Items that should not be purgeable are those that are needed almost continually by an application. Hence MDEF O, WDEF O, and the various CURS resources are not purgeable. If these were purged at the wrong time, the system would just have to purge something else and reread these resources back into memory. Also, a heap's grow zone function might be able to make space at very little cost but this does not happen until all purgeable memory blocks have been removed.

Items that should be purgeable are those that are used only occasionally such as the disk initialization and standard file package dialog resources. Resources that are used with desk accessories should also be surgeable so that an application need not be permanently burdened with the remains of a desk accessory after it has been closed. Hence all of the ALRT, DITL, DLOG, DRVR, PICT, and FONT (except Chicago-12) resources are purgeable. The WDEF 1 resource is also purgeable since it is mainly used in some desk accessories. The Note Pad message STR -15871 is flagged as not purgeable. This is probably a mistake and should be changed to being purgeable.

## The Fast System

By careful use of the system heap flag, a System file that is only used on 512K (or more) computers can be changed to provide a noticeable speed up. This involves having the commonly used resources load into the system neap instead of the application heap. Thus when a new application launches, many of the resources that it needs will already be in memory - bypassing the delay in reading them from a disk.

There are several disadvantages to this: Some programs modify a resource (in memory) for their own use. Since normally any subsequent program would get a new version of the resource, the resource is not changed back. However, when it is in the system heap, the old modified copy will be used by the following program. An example of this is the Fedit program – it changes the date format in the INTL resources. Another disadvantage to placing resources in the system heap is that the disk will not be usable on a 128K computer. The last disadvantage is that the Finder will occasionally purge the wrong data from memory – leaving a rather odd looking (but still functional) desktop – i.e., there is a bug someplace.

Since there is not enough space to hold every resource, the greatest benefit can be gained by placing only the most used ones into the system heap. Essentially, this amounts to those that are left after every step mentioned in **The Small System** has been performed. A few others may be placed into the system heap – such as a font or accessory that is used often. (Note that the Finder uses the Geneva-9 font.) Don't set any flags for any of the BNDL, FREF, ICN\*, and MACS resources as these are only used by the Finder to build the DeskTop file. The following is a working list of all the resources that load to the system heap from an extremely cut down System file: (The only other resources in this file are the Finder resources mentioned above.)

- ALRT -3997 to -3994
- CDEF 0 and 1
- CURS 1 to 4
- DITL -6047, and -4000 to -3994
- DLOG -6047, -4000, and -3999
- DSAT 0
- ICON -6047, and 0 through 2
- INIT 0 and 2
- INTL 0 and 1
- MDEF 0
- PACK 2, 3, 6, and 7
- PAT 16 and 17
- WDEF 0
- The FONTs Chicago-12, Geneva-9, and Monaco-9.

There are two ways to change all the the resource flags. The easiest is to use the Resource Editor program and simply open the System file and change the needed flags. If this is not available, then the RMover and Fedit programs can be used to accomplish the same result by following the following steps:

- Use the Finder to duplicate the System file on the target disk. (Use a disk without any important data on it. The disk also **must** have RMover on it and should have Fedit available.)
- After the Finder finishes, reset and reboot the system from this same disk. (Either by pushing the reset button or turning the computer off and then back on.) This is needed so that the new file can be renamed.
- Rename the copy of the System file to a 6 character long name. **System** is a good choice.
- Using Fedit, **Open** the copy and get to the resource fork. Scan to near the end until the resource type names in the resource map become visible. Now change all of them to something the system won't recognize but that can be easily changed back later. The best choice would be to simply change the letters to lower case. The Resource Manager will consider these as different resources but their true name will still be distinguishable. Note that the ICN#, BNDL, FREF, and MACS names can be ignored as these won't be touched later.

- Now use RMover (it **must** be run from the startup disk which should be same one that contains the duplicate System file) and **Open** the copy. Select all of the resources to place into the system heap and select **Open** from the menu. A dialog box should appear that contains a check box marked System Heap. Click it so that it is selected. For the fonts, don't bother with the resources that simply define the font name (these will be 0 butes long).
- Run Fedit again and change the resource type names back to their originals. (The entire reason for changing the names in the first place is so that the system won't get confused between the real System file and the one being modified.)
- The following step requires that some File Manager data be modified while the system is running and has one of the files in question open. Perform this step to completion all at one time. Use **Open Volume** in Fedit to open the startup volume (this is where all the above changes have been made). Scan along several sectors until the volume directory (which begins in sector number 4. Find the file name **System** and use ASCII-modify to change it to something else. (**Systen** is fine.) Now look for the name of the copy of the system (**Systen** if the original suggestion was followed) and use ASCII-modify to change it to be **System** (this is the reason for making it 6 characters long). Make sure to use the **Write** command to write each of these changes to disk. This entire step is needed to replace the original system file with the modified copy.
- Now immediately reset and reboot the system (don't even exit the Fedit!). If this is not done immediately, the system might write some of the old directory back or even worse - crash.
- The system should now boot up normally. As time goes on, it should speed up somewhat. If the debugger program MacsBug is available, use the HX and HD commands to examine the resources currently in the system heap.

The above steps should result in faster operation of the computer. It is recommended that disks with important data not be modified in this way.

#### 512K Upgrade

This document describes the procedure to upgrade a 128K Macintosh to 512K. This operation should only be attempted by those with considerable skill with soldering tools and integrated circuits. It will also void Apple's warranty and Apple Care. Finally, there is the possibility that after-market hardware enhancements will be incompatible. (The IM Underground cannot assume any responsibility for problems caused by this procedure.)

In principle, the upgrade involves removing the present 64K RAMs, installing new 256K parts and adding an extra multiplexer chip along with some discrete components. The parts needed are:

- Sixteen 256K bit RAM chips @200nS or faster (41256-200).
- A 74AS253 or 74F253 multiplexer.
- Two 2.2KΩ resistors.
- One 47Ω resistor.
- One .1 μF capacitor (preferably glass).
- 16 pin sockets for the above mentioned circuits.

The following tools and supplies will be needed during the upgrade:

- A Torx T-15 screwdriver.
- Desoldering tools.
- A soldering iron with a small tip.
- A tool to help in removing and inserting chips from sockets.
- Diagonal cutters to clip excess lead length.
- Solder.
- Fine wire such as wire-wrapping wire.

Even though sockets are not absolutely required, installing them is recommended because the desoldering process can weaken the traces on the circuit board. The first time they probably won't come off if care is taken. However, the second time that a chip is removed, the traces might easily come off. This would need to be done if one of the new RAM chips is defective.

Precautions against static damage should be observed during the entire upgrade procedure. If an anti-static workstation is not available, covering a desktop with aluminum foil should suffice. If foil is used, be careful not to short any live circuits to it. Avoid walking over a carpet and then directly touching any circuitry. It is advisable to remain seated at the work area until the entire upgrade is finished. If available, use a soldering iron with a grounded tip.

The first step in any upgrade is opening the Macintosh case. Start by disconnecting all cables and accessories. Also remove the programmer's

switch if it is installed. This can be done by prying it up using a small screwdriver. Now place the Macintosh face down on a smooth surface. Using the Torx screwdriver, remove the 5 screws holding the case on. Two are located at the bottom of the back near all of the connectors. These go into metal and thus will be hard to twist at first. (If a screwdriver other than the Torx is used, the heads of these screws will be simply stripped making it almost impossible to open the case.) Another screw is located inside the battery compartment while the last two screws are at the top under the carrying handle. All of these go into plastic - it may be possible to use a standard flat head screwdriver to remove these. Now pull the back of the case off. This is an extremely tight friction fit and will require that the top and bottom be gently worked loose. Do not use a screwdriver to "lever" it open - this will only gouge the plastic. Eventually, the case will literally pop off. Slide it up and off carefully so as not to scratch the inside which is coated with conductive paint. Once the case is opened, remove the foil shield from the connectors. Next unplug the 20 wire ribbon cable going to the disk drive and the 10 wire power connector. Now grab the logic board by the small metal frame on it and slide it out of the chassis.

The next step is removing the present memory chips. These are all located in the box marked "RAM" at the front of the circuit board. The grid coordinates are F5 through F12 and G5 through G12. These parts should be marked either with the number 4164 or something close. If a desoldering station is to be used, desolder each pin of each RAM and then gently lift it out of the circuit board. As a last resort (or if a desoldering station is not available), snip every pin of the chips and remove the body. Then desolder the pins individually. Do not allow the circuit board to heat up much as this might lift some of the fine traces. Also be careful about pulling the plated through holes out. After all 16 chips are removed, examine the board for any-lifted traces and solder bridges. Remove any solder bridges. For lifted traces, note which pads they connect. This will be fixed later. Remove any loose traces that might short other traces. Pay particular notice to the top side of the board as this will be inaccessible after the sockets are installed.

Now install the sockets. If they have any orientation, place the pin one end (this generally has a notch) towards the front of the circuit board. The silkscreened pattern on the board should match. As the sockets are installed, it may help to bend pins on two opposing corners out slightly to hold the socket while the board is turned upside down to tack solder these two pins. Now reheat these pins and at the same time push the socket all the way against the circuit board. Finally solder the remaining pins and

perhaps add some extra solder to the first two pins. Do not use excessive amounts of solder as this will tend to cause solder bridges. It may also cause the solder to flow down through the board and into the socket - preventing a pin from being insertable. With all sockets installed, check once again for solder bridges and remove any. At this time use the fine wire to replace any traces that were lifted during the desoldering phase.

(The following step may be cmitted if desired.) Now plug the new memory chips into the sockets. Pin 1 on all of them is towards the corner of the board with the keyboard plug. In other words, the pin 1 end is towards the front of the circuit board (which is normally toward the front of the computer). It should also line up with the notch drawn on the board and probably the sockets. Make sure that all pins are seated correctly in the sockets and not bent under the chips or perhaps outside the socket. Double-check for any scrap solder or solder bridges. Now place the board back into the chassis and reconnect the power and disk cables. Set the Macintosh upright and plug the power cable in (without putting the cover on). Be careful not to touch any of the inside circuitry while the power is on. Turn the system on and verify that it operates (still at 128K though). Once operation has been verified, turn it off and unplug the power cable. Wait several minutes and then remove the logic board again. Be careful around the CRT as it will still contain a high voltage charge.

The purpose of the preceding step is to verify that the sockets were installed correctly and that there are no solder bridges. It also partially tests the new memory chips. If there was a problem, the Macintosh will place a sad Mac on the screen and show an error code below it. A memory error was found if the first two digits of the code are 02 through 05. In this case the last four digits are a 16 bit number written in hexadecimal. Each suspected memory chip has the bit in it's position set to a 1 in this number. Consult the MacPainting **Memory** to map this value to the actual chips involved. (For example, if DO was set in the mask, then consult the schematic to find that DO connects to the chip at F5.) If there is a problem, check very carefully for shorts and/or breaks in traces. Perhaps use an ohmmeter to verify the board against the schematics included elsewhere in this disk.

The next step is adding the extra multiplexer chip. This can procede in one of several ways. The newer circuit boards have a place for the multiplexer. The older (original) boards only have a row of pads which can be used to tap into the needed signals. With these boards, the extra chip can be either piggy-back onto an existing one or mounted on a custom printed circuit board that attaches to the row of pads.

#### For the newer circuit boards:

There is an unused location at G13 (very near the keyboard connector). This is designed to hold the extra 74AS253 chip needed. Simply mount a socket there and then plug the chip in (it has the same pin 1 orientation as the memory). Also, add the two  $2.2 \mathrm{K}\Omega$  and the  $47\Omega$  resistors as shown in the schematics. These go in the locations marked R40, R41, and R42. Add the .1 $\mu\mathrm{F}$  capacitor at the location marked C51. Finally, cut the jumper marked as W1 and labeled 128K only.

Now reassemble the computer and procede with testing as above. The computer should now operate as a 512K system. If everything works, then reattach the case and the various other peripherals. Remember to place the foil shield over the connectors. Also, the user preferences selected via the control panel will need to be reentered.

• For older circuit boards that will have the multiplexer piggy-backed:

Several pins of the multiplexer will need to be soldered to the chip directly below it. These are: 2, 8, 14, and 16. Bend all other pins straight out so that they can be connected separately. Now place this multiplexer directly on top of the one at F3. (It could also be placed on top of F4, G3, or G4 but F3 is closest to the row of 7 pads with the needed extra signals. Also, it might be desirable to place the two chip assembly in a socket.) Now wire up pins 1 and 3 through 7 to mimic the circuitry on Apple's newer logic boards. Don't forget to add the  $47\Omega$  and  $2.2K\Omega$  resistors. The signals A17, A18, and RA8 are available at various positions in the row of 7 pads near the end of the 68000 CPU. Finally, cut the trace on the bottom side of the board that connects pins 1 and 2 of the pads. Now procede with system-check out as above.

• For older circuit boards that will have an extra board added:

Make a printed circuit board that provides all the needed connections. If a 74AS253 (or 74F253) device is not available, this method can allow the use of other similar multiplexers. Continue by attaching the board to the pads. Cut the trace connecting pins 1 and 2 there and procede with system check out as above.

Note that adding the extra board allows for some interesting changes to be made. Read the next page of the document for more data.

## Upgrade info

The multiplexer added during the upgrade functions to send the A17 and A18 address lines from the CPU to the memory chips. For video and sound hardware memory references, the multiplexer output should be at the level it is when both A17 and A18 are high. For a non-inverting multiplexer, this is a high logic level. It is a "low" for inverting multiplexers. Note that at no time should the multiplexer tri-state it's output. The only reason that the multiplexers driving RAO through RA7 tri-state is that they need to multiplex a total of 6 signals onto each memory row address.

The  $47\Omega$  resistor on the output is needed to reduce ringing caused by the highly capacitive load that the memory chips create. Without it, the effective access time will increase. EMI/RFI emissions might also increase. The two pullup resistors on the address lines are there to keep them at a known state when the 68000 is currently in an inactive bus cycle. This can be important for the row address as the hardware will still cycle the memory even during idle time intervals.

The 74ASxxx parts are not used primarily for their speed, but rather for their reduced input currents. (Witness the 74LS257 parts used to multiplex the sound address.) 74LS253 parts would have been too slow while 74S253 parts would have required too large of an input current on several of the address lines. The AS parts are thus a useful compromise.

All the multiplexers available are not used to their fullest when emulating Apple's circuitry. Most have at least half unused. By choosing the correct part, this extra capability can be used to create a system that can be both a 126K and a 512K computer. This can be useful to a software developer who needs 512K to develop a program but would like to test it in a 126K environment. Consult the MacPaint document 128K/512K for the schematic of such a modification. In this circuit, the D flip-flop is used to select between the two memory sizes. It latches the state of the interrupt button at the end of each system reset. Simply pressing this button during a reset is enough to select 128K. Note however that the reset signal has a very long debounce time interval (more than 1 second). To connect this circuit, interrupt and reset are tapped by soldering a wire to one end of the pullup resistors for these signals. A 74S151 part could be used since A17 and A18 did not connect to any other loads in the computer.

#### Beyond 512K

This document does not contain any specific information about the Macintosh. Instead, it contains several raw ideas for expanding memory beyond the standard limit of 512K. There are several software problems with this. The first is that the ROM (and some other code) checks for how much memory is available and chooses either 128K or 512K. Other sizes will not be recognized without operating system patches. The second problem is that during system boot, the ROM addresses video memory at the locations where it is found in a 512K system. In a 1 megabyte system, this would require that the video memory be placed in the middle of the memory address space. This will interfere with a linear growth of memory heaps. There are two solutions. The first is to use the extra 512K as a RAM disk. The second is to patch the InitApplZone system trap to allocate the video memory as a non-relocatable block inside the application heap. (Of course this problem can be easily fixed by a ROM change.) The final problem with large memory spaces is that the BlockMove system trap can only correctly move a maximum of 3 megabytes at a time.

The memory expansion described here would be performed by soldering a second 512K of memory on top of the present memory chips and generating a separate RAS (and possible CAS) for this new bank. To avoid a doubling in power dissipation, only one bank would be given RAS at a time. Because of the characteristics of dynamic memory, the power dissipation of 1M of memory connected this way would be only slightly higher than that of the standard 512K. Memory refresh is the biggest concern in such a design. The most critical point is during vertical retrace as video reads do not happen in this time interval. Below is a summary of the refresh intervals with various memory bank interleaving schemes.

# Some system constants:

- 44 word times per scan line
- 370 scan lines per frame
- 28 inactive scan lines per frame
- 32\*342 screen memory reads per frame

All numbers below are either scan line indexes or time intervals in scan lines. The values in parentheses specify a range of video scan lines (which are not necessarily active lines) in the format: (startline ... endline).

If alternate words are located in alternating banks, the inactive time in scan lines is:

(A1 selects which bank to use.)

(for 256 row refresh chips)(16 scan lines for all rows) (336 ... 341) to (0 ... 5) = 6 + 28 = 34 scan lines

 $(326 \dots 335)$  to  $(6 \dots 15) = 6 + 28 + 16 = 50$  scan lines

(for 128 row refresh chips)(8 scan lines for all rows)

 $(336 \dots 341)$  to  $(0 \dots 5) = 6 + 28$  scan lines = 34 scan lines

 $(334 \dots 335)$  to  $(6 \dots 7) = 6 + 28 + 8 = 42$  scan lines

If 256 words are found in one bank with the following 256 words in the other bank, the inactive time in scan lines is:

(A9 selects which bank to use.)

(for 256 row refresh chips)(16 scan lines for all rows)
(8 scan lines from one bank, then 8 from the other bank)
(326 ... 335) to (6 ... 15) = 6 + 16 + 28 = 50 scan lines
(336 ... 341) to (0 ... 5) = 6 + 28 - 34 scan lines

If 128 words are found in one bank with the following 128 words in the other bank, the inactive time in scan lines is:

(A8 selects which bank to use.)

(for 128 row refresh chips)(8 scan lines for all rows) (4 scan lines from one bank, then 4 from the other bank) (336 ... 341) to (0 ... 5) = 6 + 28 scan lines = 34 scan lines (334 ... 335) to (6 ... 7) = 6 + 28 + 8 = 42 scan lines

The conversion from scan line times to actual time is:

34 scan lines = 1.528 ms

 $42 \operatorname{scan} \operatorname{lines} = 1.887 \operatorname{ms}$ 

50 scan lines = 2.247 ms

44.5 scan lines = 2.000 ms

From the above it can be seen that if banks are switched by A1, refresh timing will be met for both 128 (2 mS) and 256 (4 mS) row refresh chips. If a larger number of contiguous words reside in the same bank, then the actual arrangement needed depends on the chips used. (Note that the above assumes that the low order address bits are given to the memory chips as a row address (as they are in the Mac).) In practice it might be preferable to place a larger number of words in the same bank. This so that the minimum number of multiplexer inputs need to be rerouted.

One final item needs mentioning. The system can be speeded up somewhat by changing the memory timing slightly. Since this involves changes to the same PALs as with the extra memory, it might be reasonable to perform both changes at once. The present timing performs two memory cycles in every ~1 $\mu$ S interval. This could be increased to 4 cycles: 2 for the CPU, one for video, and one idle. (During video retrace, it's cycle would also be idle – i.e., the memory would not receive any control signals.) If this is done, the cycles will be around 255 nS long. This requires some fast memory chips that will be run at their limits.