Handy Specification

Original June-87	Dave Needle
20-Jan-89	Rev M
6-Mar-89	Rev M-2
27-Mar-89	Rev M-3
13-Apr-89	Rev N
22-May-89	Rev N-1
30-May-89	Rev N-2
16-June-89	Rev N-3
9-Aug-89	Rev P (DevCon)

Private, confidential, secret, and don't look at it under P. of V.P.

1 General Overview

Handy is a portable hand held programmable video game. It is totally self contained with its own power source, display screen, fluorescent backlight speaker, and data input reader.

1.1 Marketing Feature Set

The unit is portable, allowing its use in many areas previously untouched by arcade games. The unit is programmable thru the use of small data cartridges. The unit can be privately used with earphones. The marketing advantages are obvious. They can be detailed in a marketing document. Certain advantages, however, are worth mentioning here.

In addition to using the game in the usual 'play' areas, the player can easily carry the game and several of the compact data cartridges to nearly any 'play' location. For example, long car rides, airplane flights, sales meetings, and visits to relatives. With the use of an earphone, the game can be played without audible disturbance of the other car passengers, relatives, or classmates.

By adding the expansion module (an infra-red communications pod), many players may play simultaneously and interactively in a common game (car race, space battle, snow ball fight, etc.) Additionally, a matching receiver unit could plug into the joystick connector of a C64 or other game machine and Handy could be used as a wireless joystick.

Intergame communications can also be done with a wire. Many players can be connected in parallel on the same wire.

1.2 External Specification

1. The display is a 3.4 inch diagonal color LCD with a resolution of 160 horizontal triads and 102 vertical lines. The LCD can display 16 levels (each) of red, green, and blue, for a total of 4096 colors.
2. There is a fixed intensity backlight for the display.
3. There is a contrast control (viewing angle) for the LCD.
4. The display rate is programmable and includes 50 and 60 Hz.
5. The display can be used in 3 orientations, allowing for both left handed and right handed landscape as well as portrait operation.
6. The audio system is mono with 4 voices and a frequency response from 100Hz to 4KHz.
7. There is a 2 inch speaker, a volume control and an earphone jack for private listening.
8. The required batteries are 6 'AA' cells.
9. There is a power jack for use with an external power supply.
10. The game input controls are a thumb controlled joy stick, two sets of two independent fire buttons, pause, and two flablode buttons.
11. The data input is a ROM cartridge with 1 Mbyte of address space and can contain writable elements.
12. The expansion connector interfaces to a bi-directional serial communications port. The only expansion devices considered at this time are an infra-red communications pod (or wire) to allow for multiple player games, a steering wheel adaptor, and a wireless joystick adaptor.

2 Hardware Overview
(see the Handy Block Diagram, appx 1)

The system hardware consists of:

'Mikey' and 'Suzy', custom digital ICs.
A 16 MHz crystal.
Two 64k by 4 DRAMs.
A 2 inch speaker with an earphone jack and volume control.
An LCD and its related drivers, backlight, and contrast control.
A data input system. (Tape or ROM)
Batteries, power supply, and external power jack.
An expansion port.
A joystick, 2 fire buttons, and other switches.

The division of circuitry between the 2 digital ICs is that Mikey contains all of the non-sprite hardware and Suzy is only a sprite generation engine. Some non-sprite functions (the switch readers and the ROM reader) are in Suzy due to pin limitations. In addition the math functions are part of Suzys sprite engine.

The crystal is the only source of timing information in the system. The basic timing 'tick' of the system is 62.5 ns. Let us now define the term 'tick' to be 62.5 ns.

The system RAM is 64K bytes. This RAM houses the video buffer(s) and collision buffer (total maximum of 24K bytes) in addition to the game software (worst case minimum of 40K bytes). The RAMs have a 120ns RAS access time and 60ns page mode CAS access time. This allows us to have a 125ns (8MHz) page mode memory access rate and a 250ns (4MHz) normal memory access rate.

The speaker is a 2 inch diameter 8 ohm speaker. The volume control range includes zero. The earphone jack is the standard stereo 'Walkman' style (only mono sound, however).

The LCD has a resolution of 480 horizontal pixels by 102 vertical pixels. Three pixels, one of each color, form a square triad with a resultant screen resolution of 160 triads by 102 lines. The column drivers can generate 16 levels of intensity for each pixel, resulting in a palette of 4096 colors. The LCD circuitry includes the power generation for the LCD driver ICs, the decoding of the Mikey strobes for the LCD driver ICs, and the power generation for the backlight.

The data input system is either a ROM cartridge or a magnetic tape reader. The system hardware will support both, but units will be made with either one or the other. The data input systems are explained elsewhere.

The power system provides raw power to the regulator, and if used, the motor. The regulator has a soft on/off function so that the system can power itself off when not in use. This is required so as to avoid the customer frustration of expensive battery replacements which could then cause loss of software revenue. The motor is separately powered so that its load is not part of the regulators problem. The soft off function also disconnects it from the power source.

The expansion port has a bi-directional serial port operating asynchronously at a programmable speed with a maximum of 62500 baud. This is approximately 104 bytes per 60 Hz frame. By allowing all of the games in a multiple player game to operate one frame behind their control functions, up to 104 bytes of data can be communicated. Using an example maximum of 8 players in a group and for some overhead, 12 bytes per player are available.

The human input controls consist of a 4 switch (8 position) joy stick, two sets of 2 independent fire buttons, game pause button, 2 flablode buttons, power on, and power off. The two sets of fire buttons are wired together, there are only 2 'fire' signals. Two sets are available to allow for left and right handed operation. Flablode is a Jovian word meaning a device or function that we know is required or desired but for which we don't have an actual definition (noun: flabloden, verb: to flablode).

3 Software Related Hardware Perniciousness

(or why some software people hate some hardware people)

There are certain things that the software ought not to do to the hardware. While these things are not physically destructive on their own, they will cause the hardware to act unpredictably strange, which may cause the user to be physically destructive.

While we certainly could have protected the system from many of the following problems, I doubt that we would have found them all. In addition, the act of software doing one of these things means that there is a problem in the software anyway and the intended function would probably not happen. Additionally, this is only a toy. If this unit were a bio-medical device I would have found a safer solution.

The things that software MUST NOT do and the precautions software MUST take are neither intuitive nor obvious, so I will detail them here.

3.1 Don't Do These Things.

If you do any of the things that I tell you not to do, the hardware will act unpredictably and will not be expected to recover without a complete system initialization. So don't do them.

3.1.1 Suzy SCB Register Accesses

All of the registers in the Suzy SCB are unsafe to touch while the sprite engine is in operation. This is true for BOTH read and write accesses. Prior to accessing any of those registers (they are identified in the hardware address appendix as 'unsafe') you must first check to see if the sprite engine is running (either for sprites or math). Those status bits are in SPRSYS.

3.1.2 Sequential Memory Accesses

CPU instructions that result in a Suzy access followed immediately by another Suzy access will probably break Suzy. Please do not do them.

3.1.3 Compiler Stuff

Some compilers do stupid things when they convert complex statements. For example, "A = B = C = 0" may get converted to:

store 0 to C,
read C and write it to B,
read B and write it to A.

This will break if you try it on any of my hardware, so unless you are real sure that your compiler will never do it, don't write your code that way.

3.1.4 Writes to the Game Cartridge

Writes to Suzy are blind, but will complete before any other CPU function can disturb them EXCEPT in the case of writes to the game cart. For 12 ticks after a write to the game cart is started, the CPU MUST NOT access Suzy (read or write). If it does, the write to the cart will be trashed.

3.1.5 Palettes and Page Breaks

This one is an actual hardware bug of mine. The hardware signal that ends the loading process for any SCB element comes 2 bytes prior to the actual end of the element. If a page break comes at the same time as the 'end' signal, then that end signal needs to be delayed by one cycle. If not, the loading process will end and not start up again at the end of the page break. Well, I screwed up the logic for the pen index palette loader. Therefore if a pen index palette starts at address xxFA, a page break will occur at the same time as the 'end' signal and this page break will be ignored by the palette loader. The last 2 bytes of this 8 byte palette will not be loaded from RAM, and the values loaded by the previous SCB will still be in the hardware. Pen index numbers C,D,E, and F will be incorrectly unchanged. Sometimes I am such a jerk.

3.2 Please Don't Do These Things.

There are also things that software people do that merely annoy the hardware people rather than annoy the end user. This includes seemingly harmless activities like sub-decoding bit values from byte descriptions (sprite types, for instance), or assuming address continuity between areas of hardware (like SPRCTL0 follows the Sprite Control Block).

Please don't do them.

It would be difficult to list all of the possibilities, so I will list the general considerations and ask you to be sensible about them. In addition, please feel free to submit a request for an exemption on a specific case. If it is granted, we will change the specification so as to make the special case forever legal. The price will be small.

The list:

Do not decode any bits within any byte definitions.
Do not assume any address continuity between hardware sections.
Do not assume that any hardware addresses that are multiply mapped will remain so.
Do not use un-defined bits within a byte for anything.

If you notice an unspecified but clever interaction between two or more sections of hardware, please come and tell me about it. It is more likely that it is a mistake rather than an omission from the spec.

I will try to list the approved exemptions in the spec.

Thank You

3.3 Do It This Way

Some of the hardware functions, as designed by us mindless brutes, require special handling. As we discover these requirements, I will list them here.

3.3.1 Timer Reload Disable

If a particular timer reload bit is set to disable, the timer ought to count down to zero, stop counting, and set its 'timer done' bit. However, if the 'timer done' bit was already set, the timer will not even do its first count. Therefore, when the setting of 'timer reload' is 'disabled', you must clear the 'timer done' bit in order for the timer to count at all.

This 'clear' can be accomplished by setting the 'reset timer done' bit, but that can cause a new and exciting problem. The 'clear' can also be accomplished by writing directly to the byte that contains the 'timer done' bit.

The answer of choice is to write a 0 to bit 3 of the control byte (TIMnCTLB). Since the other bits are adjusted by hardware, I recommend writing a full byte of 0 to that register when you need to do the 'clear' function.

3.3.2 Reset Timer Done

The 'reset timer done' bit in the hardware registers was mistakenly designed as a 'level' signal rather than a 'pulse' signal. Therefore, to use it correctly, you must 'pulse' it in software. This means first setting it high, and then setting it low. The problem with just leaving it high is that the dumb hardware may (depending on other conditions in that particular timer) send a continuous interrupt to the interrupt logic. In addition, since a lucky interrupt might actually slip in during this 'software pulse', the recommended process is to clear the interrupt enable bit at the same time that the reset timer done bit is set. Happily, both bits are in the same register. Please remember to restore the 'enable' bit as you release the 'reset' bit.

4 CPU/ROM

The CPU is a 65C02 cell imbedded in the Mikey IC. The specifics of its instruction set and register operations are found in the VTI specification.

4.1 CPU Cycle Timing

While the number of CPU cycles for each instruction are defined in the VTI spec, the number of system 'ticks' used by each CPU cycle are defined here. Some cycles will require a fixed number of ticks, some will require a variable number of ticks based on the instruction that preceded it, and some will require a variable (and sometimes non-deterministic) number of ticks based on the functioning of other pieces of hardware in the system. In addition, the CPU can be paused by the video and refresh accesses, and it can be interrupted by the timers and the serial port. The point of listing all of these variances to the CPU timing numbers is to advise the programmer that CPU cycles can not be used as the timing elements in small software timers. Under certain specific circumstances, the environment will be sufficiently controlled to allow for small software timers (initial mag tape reading), but in general, that practice should be avoided.

4.1.1 RAM Page Mode

The RAMs used in the system have a page mode operation in which one of the control signals (RAS) is not repeated for each memory access. This allows the cycle to be shorter than a normal access cycle. The requirement for using a page mode cycle is that the current access is in the same 256 address page of memory as the previous access. While comparing current and previous addresses is certainly a valid method of determining if a page mode cycle can be used, it usually takes the same or more time than just repeating the control signal. As a result, page mode is often not used by many CPU designers.

We use the method of decoding the current op-code to see if the next cycle could be a page mode cycle and then observing the other pertinent states of the system to see if some reason exists to NOT allow a page mode cycle. While this method is not 100% efficient in allowing all possible page mode opportunities, its silicon requirements are small and when properly designed it does not permit false page mode cycles.

The CPU makes use of the page mode circuitry in its op-code reads. All writes and all data reads are done in normal memory cycles. A page mode op-code read takes 4 ticks, a normal read or write to RAM takes 5 ticks.

4.1.2 Hardware Accesses, DTACK

CPU accesses to hardware require an acknowledge signal (DTACK) from that hardware in order for the CPU cycle to proceed. These CPU accesses fall into 2 classes. One is for hardware that is always available and the other is for hardware that becomes available at a time not related to the CPU.

For always available hardware , the DTACK can be generated from the address decode and not cause the cycle to be longer than a RAM read or write cycle (5 ticks). Writes to Suzy are handled as 'always available' whether or not Suzy is actually available.

For eventually available hardware, (some of Suzy and some of Mikey), the DTACK is generated as a combination of the address decode, sometimes the column strobe, and the particular hardware being accessed. This cycle has a minimum requirement of 5 ticks and a maximum requirement of 128 ticks (probable actual maximum of 40). The maximum is not related to the CPU, it is required to prevent video and refresh underflows. All writes to Suzy are 'Blind' in that the write cycle is always 5 ticks long and does not get a DTACK. Suzy will accept the data immediately and then place it internally as required. The maximum time required for this internal placement is 6 ticks (except for writes to the game cart). This is less than the fastest turnaround time for 2 sequential writes to Suzy, so no collisions will occur. Poof for unsafe addresses.

Some of the hardware in Mikey is constructed of Dual Ported RAMs addressed in a cyclical manner. These DPRAMs will have a maximum latency equal to their cycle time (1.25 us). Some of the hardware in Suzy is also constructed of Dual Ported RAMs. These RAMs are not cycling, but their latency is still slightly variable (+1 tick, -0) due to clock synchronization.

The CPU accessible addresses in both Mikey and Suzy are not all readable and writeable. See the hardware address appendix (appx 2) for the specifics.

4.1.3 CPU Cycle Tick Summary

Cycle	Min	Max
Page Mode RAM(read)	4	4
Normal RAM(r/w)	5	5
Page Mode ROM	4	4
Normal ROM	5	5
Available Hardware(r/w)	5	5
Mikey audio DPRAM(r/w)	5	20
Mikey color palette DPRAM(r/w)	5	5
Suzy Hardware(write)	5	5
Suzy Hardware(read)	9	15

4.1.4 CPU NMI Latency

The NMI signal path from the pin of Mikey to the pin of the internal CPU contains clocked delays. This will make the usability of the pin questionable in the debug environment.

Fortunately for us, Howard and Craig arranged the diagnostic hardware to still use it effectively. See the Howard board spec for details.

4.1.5 Suzy Bus Request-Bus Grant Latency

The maximum allowable latency at Suzy is constrained by the needs of the video DMA circuit. As a compromise between bigger FIFOs in the Mikey video DMA and reduced performance in Suzy, we are setting the maximum latency from Bus Request to Bus Grant at 2.5 us.

The time between Mikey requesting the bus and Suzy releasing it is dependant on the state of the currently running process inside of Suzy. The longest process is 30 ticks. Adding the overhead of accepting the bus request and releasing the bus grant brings the total to 40 ticks.

4.2 ROM

The system ROM is imbedded in Mikey. Its size is 512 bytes. Its main and perhaps only function is to read in the initial data from the data input system and then execute that data. In the case of the magnetic tape system, some error correction will be performed.

4.3 CPU Sleep

BIG NOTE: Sleep is broken in Mikey. The CPU will NOT remain asleep unless Suzy is using the bus. There is no point in putting the CPU to sleep unless you expect Suzy to take the bus. We will figure out how to save power some other way.

I will still keep the following paragraphs pertaining to sleep in this spec.

The CPU can put itself to sleep by writing to the CPU disable address. The CPU wants to sleep for two reasons. The first is that it is done with all of its work and we would like to conserve battery power. In this case, the CPU would pre-arrange for an interrupt to wake it up at the next time that work needs to be done (probably vertical restart). When awoken by an interrupt, normal interrupt vectoring takes place. Note that if no interrupt occurs, the CPU sleeps forever. An interrupt will wake you up even if they are disabled inside the CPU. The second reason to sleep is that Suzy has been requested (by the CPU) to do work. In that case, it is up to Suzy to awaken the CPU when it is done. When awoken by Suzy, processing continues from where it left off.

The CPU can not fall asleep unless it puts on its pajamas. This is accomplished by disabling all interrupts (register in Mikey) that you do not want to wake you up, then clearing them (in case they came in while you were disabling them), and then writing to the Suzy Wakeup Acknowledge address IF you were woken up by Suzy (or if this is the first time you are going to sleep). The purpose of the Suzy Wakeup Acknowledge address is to prevent missing the wakeup request from Suzy should it occur exactly at the same time as an interrupt.

5 Display

5.1 Frame Rate

We have a programmable frame rate to accommodate either 50 or 60 Hz lighting systems. We may also want to vary the frame rate for 'game' reasons.

While no vertical blanking is actually required by the LCD, the system likes to have some non-display time in which to cleanly change the display characteristics. In addition, multiple player games need some dead time between frames to re-synchronize to the master. Since reloading the color palette could take 150 us, at least 1 and perhaps 2 scan lines of time should be allocated as vertical blank time. The current method of driving the LCD requires 3 scan lines of vertical blank.

During vertical blank time, none of the display lines are allowed to be on. If they were, that line would be noticeably 'brighter' than the others.

Additionally, the magic 'P' counter has to be set to match the LCD scan rate. The formula is: INT((((line time - .5us) / 15) * 4) -1)

Don't forget to set the display control bits. Normal 4 bit color is:
'DISPCTL' (FD92) = x'0D'

Some frame rate choices are:

60Hz:

159 us x 105 lines =16.695 ms [59.90 Hz], 3 lines of Vertical Blank

For normal 60Hz video operation, set the relevant timers as follows:

Timer 0: clock=1us, backup=158.
FD00=x'9E', FD01=x'18'
Timer 2: clock=linking, backup=104.
FD08=x'68', FD09=x'1F'
Pcount: 'PBKUP' = 41.
FD93= x'29'

50Hz:

190 us x 105 lines =19.950 ms [50.13 Hz], 3 lines of Vertical Blank

For 50Hz video operation, set the relevant timers as follows:

Timer 0: clock=1us, backup=189.
FD00=x'BD', FD01=x'18'
Timer 2: clock=linking, backup=104.
FD08=x'68', FD09=x'1F'
Pcount: 'PBKUP' = 49.
FD93= x'31'

75Hz:

127 us x 105 lines =13.335 ms [74.99 Hz], 3 lines of Vertical Blank

For 75Hz video operation, set the relevant timers as follows:

Timer 0: clock=1us, backup=126.
FD00=x'7E', FD01=x'18'
Timer 2: clock=linking, backup=104.
FD08=x'68', FD09=x'1F'
Pcount: 'PBKUP' = 32.
FD93= x'20'

The maximum frame rate occurs when Hcount is 121 (backup = 120) which results in a vertical frequency of 78.7 Hz. 75 Hz (a useful rate) is achieved by setting the H backup value to 126 (x'7E'). The vertical values do not change.

We notice that 50Hz operation causes a massive flicker that is probably due to the speed of the LCD itself. 50Hz may not be usable.

8 Magnetic Tape

The circuit for reading magnetic tape is in Mikey. However, the Mikey ROM does not currently activate the tape mode. When the decision to use tape is made, the ROM may have to be changed. The alternative would be to use an external ROM that would activate the tape circuits.

9 ROM Cart

9.1 ROM Cart Address Space

In a ROM Cart unit, the addresses for the ROM Cart are provided by an 8 bit shift register and a 11 bit counter. A particular ROM Cart will be wired to the address generator such that the upper 8 bits of its address will come from the 8 bit shift register and the remaining lower bits of its address will come from the lower bits of the counter. A 64k byte ROM Cart will have 8 bits of counted address and 8 bits of shifted address. A 128k byte ROM Cart will have 9 bits of counted address and 8 bits of shifted address. The maximum address size is (8+11) 19 bits which equates to 1/2 megabyte of ROM Cart address space. Since there are 2 strobes available to the cart, there is a total of 1 megabyte of address space without additional hardware support.

The 8 bit shifter is controlled by 2 signals from Mikey, 'CartAddressData' and 'CartAddressStrobe'. 'CartAddressData' is the data input to the shift register and 'CartAddressStrobe' is the clock to the shift register. The shift register accepts data from the 'CartAddressData' line on rising (0 to 1) transitions of the 'CartAddressStrobe' line. Data is shifted into the register most significant bit first. The 'CartAddressStrobe' line is also the reset signal for the 11 bit counter. The counter is reset whenever the line is high (1). The software must remember to set the 'CartAddressStrobe line low after shifting in the address so as to release the reset of the counter.

9.2 ROM Cart Data Read

In order to prevent a reduction of battery life when using the ROM Cart, we only enable the ROM itself when we wish to read a data byte. We do it in such a manner as to guarantee that no RAM access can occur simultaneously with a ROM access. This is achieved by driving one of the 'chip enable' lines from SUZY. The trailing edge of this chip enable signal also advances the address counter by 1. Only one address is available to the Cart and the activation of either chip enable signal will increment the address.

The 8 tri-state data bits of the ROM are tied to 8 of the switch reading lines on SUZY. The switches are isolated from the ROM by resistors. When a read request comes from MIKEY, Suzy will hold off MIKEYs DTACK, enable the ROM (which overdrives any pressed switches), wait 437.5 ns (for ROM access time), and disable the ROM. Towards the end of the wait, SUZY passed the data from the ROM to Mikey and released the DTACK. The actual access time available at the pins of the Cart is 392 ns. The CPU cycle that performed the actual read uses 15 ticks of the clock.

9.3 ROM Cart Data Write

The ROM Cart can also be written to. The addressing scheme is the same as for reads. The strobe is also self timed. The length of the strobe is 562.5 ns, the data is stable for 125 ns prior to the strobe and for 62.5 ns after the strobe. This is a 'blind' write from the CPU and must not be interrupted by another access to Suzy until it is finished. The CPU must not access Suzy for 12 ticks after the completion of the 'blind' write cycle.

9.3 ROM Cart Power-Up

Since some types of ROM do not have a useful power-down mode, we provide a switched power pin to the cartridge. This pin is controlled by the state of the 'CartAddressData' signal from Mikey. Yes, this is the same pin that we use as a data source while clocking the address shift register and therefor, we will be switching ROM power on and off while loading that register. Unless the software is poorly arranged, that interval of power switching will be short. The switched power pin is powered up by setting the 'CartAddressData' signal low. It is suggested that the pin be powered up for the read of any ROM cart since carts that do not need it will not be wired to that pin. Additionally, information in that ROM cart can tell the software if it needs to further manipulate the pin.

10 Timers/Interrupts

10.1 Timers

There are 8 independent timers. Each has the same construction as an audio channel, a 3 bit source period selector and an 8 bit down counter with a backup register. This gives a timer range of 1 us to 16384 us. Timers can be set to stop when they reach a count of 0 or to reload from their backup register. In addition, they can be linked, with the reload of one timer clocking the next timer.

The linking order is as follows:

Group A:
Timer 0 -> Timer 2 -> Timer 4.

Group B:
Timer 1 -> Timer 3 -> Timer 5 -> Timer 7 -> Audio 0 -> Audio 1-> Audio 2 -> Audio 3 -> Timer 1.

As with the audio channels, a count of 0 is valid for 1 full cycle of the selected clock. A backup value of 5 will result in 6 units of time. Actually, due to hardware limitations, the first utilization of that timer after it has been set will result in a time value between 5 and 6 units. Subsequent utilizations (reload is 'on') will result in a value of 6 units.

10.2 Timer Utilization

Two of the timers will be used for the video frame rate generator. One (timer 0) is set to the length of a display line and the second (timer 2) is set to the number of lines.

One of the timers (timer 4) will be used as the baud rate generator for the serial expansion port (UART).

Note that the hardware actually uses the bits in these timers and it would be dangerous to arbitrarily fiddle with them in software.

The other 5 timers are for general software use. POOF. See the mag tape description.

10.3 Interrupts

7 of the 8 timers can interrupt the CPU when it underflows. Each interrupt can be masked. The value of the interrupt bit can be polled independent of its mask condition. The interrupt bit for timer 4 (UART baud rate) is driven by receiver or transmitter ready bit of the UART.

If an interrupt occurs while the CPU is asleep, it will wake up the CPU.

Since one of the timers is the vertical line counter, the useful 'end of frame' interrupt can be generated there.

The interrupt signal comes from the timer when the timer value is zero AND the timer is attempting to perform a 'borrow'. Based on the control bits, the borrow may not actually occur, but the interrupt signal will. This signal then requests the bus control circuit to give the bus to the CPU. If the CPU already has the bus, then this function causes no delays. If Suzy has the bus, then the maximum Suzy latency time could be incurred. Then, the interrupt signal waits for the end of the current CPU cycle before actually interrupting the CPU.

11 UART

The Mikey UART is a pretty standard serial kind of thing. However, since most of the communicating world has managed to bolix up the definitions and use of the so-called standards of serial communications, I will attempt to explain all of our UARTs functions. Yes, I will leave something out so as to continue to trick earthlings into thinking that I am human.

11.1 Connector Signals

The UART connects to the REDEYE connector which has pins. There is 1 data pin. This signal is bi-directional serial data. Its idle state is open-collector with a pull-up resistor to +5. There is one ground pin, and one VCC pin.

(a design error causes the power up state of the output to be TTL high, ALL code must set the TXOPEN bit in order to fix this. Its not my fault.)

There is also a signal called NOEXP which indicates the presence of a plug in the REDEYE socket.

Any devices other than RedEye will need to have an intelligent protocol to distinguish themselves from a RedEye type device. See the RedEye specification for protocol details.

11.2 Baud Rate

The baud rate is generated by TIMER4 according to the equation

CLOCK4 / (TIMER4 + 1) / 8

The minimum number that can be used in TIMER4 is 1, the maximum number is 255. The fastest CLOCK4 is 1MHz, the slowest is 15625 Hz. This gives a baud rate maximum of 62.5 Kbaud and a minimum of 7.63 baud. The settings for the common baud rates are:

Baud Rate	CLOCK4	TIMER4	Actual
62500	1us	1	62500
9600	1us	12	9615
2400	1us	51	2404
1200	1us	103	1202
300	2us	207	300.5

11.3 Data Format

The serial data format is the standard 11 bits. We do not offer a choice.

The standard bits are:

1 start bit (binary 0);
8 data bits, LSB first;
1 parity bit (or 9th bit as defined by the control byte);
1 stop bit (binary 1).

The parity (or 9th) bit operates as follows:

Receive:

The state of the 9th bit is always available for read in the control byte. In addition, the parity of the received character is calculated and if it does not match the setting of the parity select bit in the control byte, the parity error bit will be set. Receive parity error can not be disabled. If you don't want it, don't read it.

Transmit:

The 9th bit is always sent. It is either the result of a parity calculation on the transmit data byte or it is the value set in the parity select bit in the control register. The choice is made by the parity enable bit in the control byte. For example:

If PAREN is '1' and PAREVEN is '0', then the 9th bit will be the result of an 'odd' parity calculation on the transmit data byte.

If PAREN is '0', then the 9th bit will be whatever the state of PAREVEN is.

We have just discovered that the calculation for parity includes the parity bit itself. Most of us don't like that, but it is too late to change it.

11.4 Break

A break of any length can be transmitted by setting the transmit break bit in the control register. The break will continue as long as the bit is set.

A 'break' is defined as a start bit, a data value of 0 (same as a permanent start bit), and the absence of a stop bit at the expected time.

The receiver requires that a break lasts 24 bit times before it is recognized. There are many 'standard' break lengths, this is the one we chose.

11.5 Transmitter Status Bits

There are 2 status bits for the transmitter, TXRDY (transmit buffer ready) and TXEMPTY (transmitter totally done).

If TXRDY is a '1', then the contents of the transmit holding register have been loaded into the transmit shift register and the holding register is now available to be loaded with the next byte to be transmitted. This bit is also set to '1' after a reset.

If TXEMPTY is a '1', then BOTH the transmit holding register and the transmit shift register have been emptied and there are no more bits going out the serial data line.

11.6 Errors

There are 3 receive errors, parity error (already explained), framing error, and overrun error. Once received, these error bits remain set until they are cleared by writing to the control byte with the reset error bit set. Writing to the control byte with the reset error bit cleared has no effect on the errors. Note that the reset error bit is NOT an error enable bit. Receive errors are always enabled.

Framing error indicates that a non-zero character has been received without the appropriate stop bit.

Overrun error indicates that a character (of any kind or error) has been received and the previously received character has not yet been read from the receive buffer.

11.7 Unusual Interrupt Condition

Well, we did screw something up after all. Both the transmit and receive interrupts are 'level' sensitive, rather than 'edge' sensitive. This means that an interrupt will be continuously generated as long as it is enabled and its UART buffer is ready. As a result, the software must disable the interrupt prior to clearing it. Sorry.

11.8 left out

I left something out. I know what it is but by the next time I revise this spec, I may have forgotten.

I have forgotten.

12.8 General I/O Port

In the beginning, there was a general purpose 8 bit I/O port. As pins on Mikey became unavailable, the number of bits was reduced. Now all we have are the 5 bits of IODAT and they are not even pure read/write.

The direction of the pins (in or out) still needs to be set even though all but one are forced in the PCB to be either an in or an out but not both. The function of the bits are not apparent from the description in the address appendix, so I will explain them here.

1. External Power.

This bit detects the presence of a powered plug. The ROM sets it to an output, so the system code must set it to an input.

2. Cart Address Data.

This bit must be set to an output. It has 2 functions. One is that it is the data pin for the shifter that holds the cartridge address. The other is that it controls power to the cartridge.. Power is on when the bit is low, power is off when the bit is high.

3. Noexp.

This bit must be set to an input. It detects the presence of a plug in the expansion connector.

4. Rest.

This bit must be set to an output. In addition, the data value of this bit must be set to 1. This bit controls the rest period of the LCD display. The actual Rest signal is a high except during the last 2 scan lines of vertical blank and the first scan line after vertical blank. It is generated by the vertical timing chain in Mikey but its output on this pin can be disabled by the incorrect setting of this bit. In addition, when reading this bit, you will get the actual state of the Rest signal 'anded' with the value of the bit set in IODAT. Yes, we know that the polarity of the name is wrong, but that is the way it came to us.

5. Audin.

This bit can be an input or an output. In its current use, it is the write enable line for writeable elements in the cartridge. It can also be used as an input from the cartridge such as 'audio in' for 'talking-listening' games. Whether it is set to input or output, the value read on this pin will depend on the electronics in the cartridge that is driving it.

6. The other bits.

The other 3 bits in the byte are not connected to anything specific. Don't depend on them being any particular value.

13 Expansion Connector

The expansion connector is a 3 wire stereo earphone jack. The pin assignments are:

Shield: Ground
Tip: VCC
Center: Data

There are certain considerations for data transfer, more later.

Note that when using a wire, any unit that is powered off and connected to the link will disable communications for all of the units in the link.

14 System Reset/Power Up

The state of all of the control bits during and immediately after system reset is given in appendix 2. During reset, the main clocks are still running and are used to propagate the reset condition thru the logic. Reset must be long enough to insure a stable and consistent startup.

The process that takes place upon release from system reset and the steps required to initialize the hardware are described below.

14.1 Suzy Reset Recovery.

At release from reset, all of Suzy is disabled and remains so until enabled by the CPU. During reset, the bus handshake logic has stabilized with bus grant asserted. Any incidental internal activity will be listed when the logic design is completed. Suzy is not expected to require any software initialization until a specific function of Suzy is desired.

14.2 Mikey Reset Recovery.

At release from system reset, the video bus request is disabled and will remain so until enabled by the CPU. The CPU is requesting the bus. During reset, the bus handshake logic has stabilized and is ready to give the bus to the CPU. The ROM overlay is enabled and the CPU will fetch its reset vector from ROM at the normal 6502 address of FFFC and FFFD.

Since there are some pieces of hardware that are running at rates slower than the main system clock, there is the possibility that there will not be a deterministic relationship between the phases of these slower clocks (eg. CPU and audio/timer). To prevent possible peculiarities of operation (due to hardware bugs) and to assist in truthful software emulation, it is suggested that these individual hardware chunks get synchronized at system start. It may be expensive to do it all in silicon, so some of them may need to be done by software. At this time, Glenn says that the audio section is done in silicon. There are other hardware registers that must be initialized correctly in order to let the system come up consistently. They are stated in appendix 2 (hardware addresses).

The current process that must be followed at system initialization is:

1. Disable interrupts
2. Clear decimal mode
3. Read SUZYHREV
4. If = 0, jump to test code
5. Read 8 locations in RAM (each must cause a new RAS to occur)

(more ??)

15 Definitions and Terminology

The polarity of signals and their effect on the circuitry is not always apparent from the name of the signals. For this document and its related documents and appendices, the following conventions will be used:

Set = On = Asserted = Active = Occurs

These mean that the signal is in the state that causes or allows its named function to occur. For example, if bus grant is set, we are granting a bus. If an interrupt mask bit is set, that interrupt is masked.

Reset = Off = Cleared = De-asserted = Inactive = Dropped

These mean that the signal is in the state that does not allow its named function to occur. For example, when bus request is dropped, we are no longer requesting the bus.

The terms 'hi' and 'low' are well applied to actual schematic referenced signals, but should be avoided in generally descriptive text.

Acquired = Taken

These relate to tri-state busses and indicate that the bus is now being driven by the 'acquirer'.

Released = Let Go

These also relate to tri-state busses and indicate that the 'releaser' is no longer driving the bus.

SimWait = Boil

The act of waiting for garbage collection on the VTI tools

16 Known Variances From Anticipated Optimums

This is a list of the known bugs in the hardware.

16.1 Mikey

1. Sleep does not work if Suzy does not have the bus.
2. The UART interrupt is not edge sensitive.
3. The UART TXD signal powers up in TTL HIGH, instead of open collector.
4. The lower nybble of the audio out byte is processed incorrectly.
5. The Reset Timer Done bit is not a pulse.
6. The Timer Done bit requires clearing in order to count.
7. The Mikey ROM sets the External Power Detect pin to output.
8. The REST pin on Mikey is initialized as an input and its data is set to 0. Both are wrong. You must set it to an output with a data value of 1.
9. The IODAT register is not really a R/W register.

16.2 Suzy

1. Remainder does not correctly handle the upper bit in two ways.
2. Signed multiply thinks 8000 is a positive number.

3. Auto clear of the upper byte of an SCB word does not also auto clear the sign flag.
4. The page break signal does not delay the end of the pen index palette loading.
5. The polarity of the 'shadow' attribute is inverted.
6. Signed multiply thinks that 0 is a negative number. (delayed effect)
7. The circuit that detects a '0' in the SCB_NEXT field of an SCB only looks at the upper byte. Therefore, we can't have scabs in page 0. I'm sorry.
8. A data packet header of '00000' can be used to indicate end of data. There appears to be a bug with it. I don't understand it yet. Beat me. Kick me.

17 Approved Exemptions

1. Only the upper byte of the 'NEXT' word in the SCB needs to be set to zero in order to indicate that this is the last SCB in the list. The lower byte of that word can then be used for any other function since the hardware will ignore it if the upper byte is zero.

2. Some of the sprite values (H size, V size, etc) are re-usable. The normal way to reuse them is to have the first sprite in the local list actually initialize the values, and then have the remaining sprites in the local list re-use them. One of the difficulties in doing it this way is that it is not always reasonable to arrange the list and the SCABs appropriately. One of the ways to simplify the problem is to use an initializing sprite with the desired numbers in the reusable registers.

I have been asked to provide an exemption that would allow the software to write directly to the hardware registers in the Suzy SCB and thus avoid all of the overhead of arranging the lists or creating null sprites.

Since this section of hardware is firmly tied to the sprite software process, I believe that it will be OK to write directly to the hardware registers. I could be wrong, so I am going to approve the following conditional exemption.

It will be OK to write to the twentyfour 16 bit registers of the Suzy SCB PROVIDING that you only do it via the MACRO PROVIDED TO YOU BY RJ MICAL. In addition, you must understand that the contents of this macro may change if future revisions of the hardware so require. In addition, you must understand that future hardware may make the process not work and therefore the macro will be changed to be a warning that you can't use it anymore.

Don't cheat.

18 Common Errors

There are errors that many first time programmers and some experienced programmers make with this hardware. Some of these errors may be difficult to identify due to the complexity of the ICs. I will list some of them here to aid in the debugging of 'Mystery Bugs'.

1. The presence of an interrupt in Mikey, regardless of the state of the CPU enable interrupt bit, will prevent the CPU from going to sleep, and thus prevent Suzy from functioning. So if sprites stop working, unintentional interrupt bits can be the hidden cause.

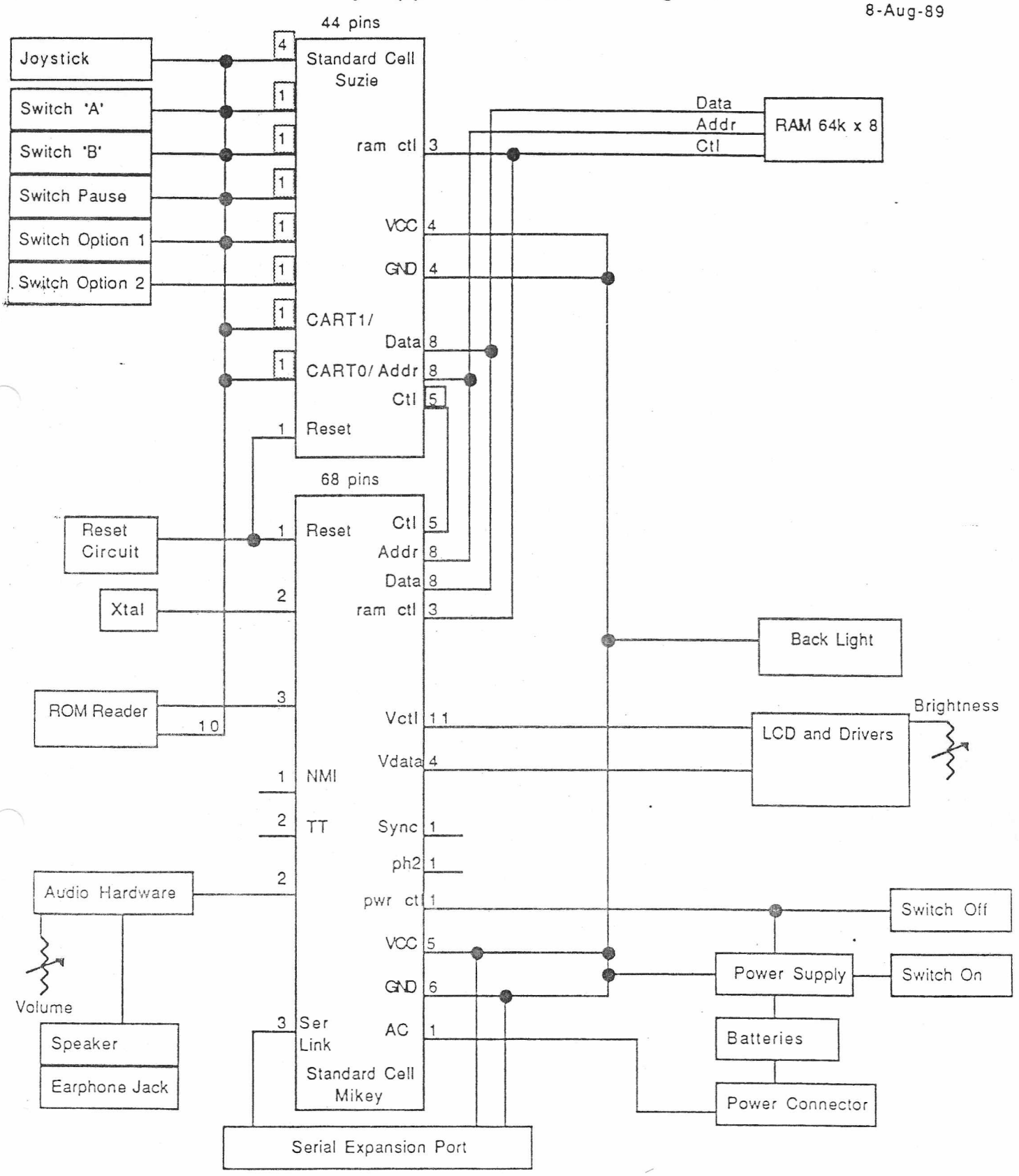
2. The Suzy Done Acknowledge address must be written to prior to running the sprite engine. This is required even prior to the first time the sprite engine is activated. If it is not written to in the appropriate sequences, the CPU will not go to

sleep when so requested. In addition, if some software accidentally allows a Suzy operation to complete without then following that completion with a write to SDONEACK, the CPU will not sleep. So if sprites stop working, something may have gone wrong with your SDONEACK software.

3. Writes to the cartridge are blind. If you accidentally access Suzy before the required delay period, you will modify some internal bus in Suzy. The results are not definable.

Handy Appendix 1, Block Diagram

8-Aug-89



Handy Appendix 5
System Bus Interplay

Original 20-Dec-87	Dave Needle	
21-Dec-87	Rev A	
18-Jan-88	Rev A-1	
20-Jan-88	Rev A-2	
20-Apr-88	Rev B	Saved copy
7-Mar-89	Rev C	

1. System Bus Interplay

The system bus is defined as the inter-chip address, data, and control signals. This system has several bus masters. Two of them (Suzy and the CPU) perform functions that do not require the system bus but do use an internal bus. It is defined that these activities on an internal bus will be reflected on the system bus whenever possible and without using the pertinent activation strobes (no RAS or CAS).

To allow simultaneous operation of bus masters on their internal busses and singular operation on the system bus would have resulted in interleaving the bus masters addresses. This eliminates the performance advantage of page mode. Any performance gained by simultaneous operation is related to the amount of non-system bus activity required and is still limited by the system bus bandwidth. Analysis of the activities of the two masters indicates that we will have higher performance with singular bus operation using page mode than simultaneous bus operation at reduced system bus efficiency.

There is, however, a situation in Suzy where the bus may be relinquished to Mikey while Suzy is processing an internal function. In this case, the internal function in Suzy will be allowed to continue while Mikey has the bus. At the completion of the internal function, Suzy will stop.

We will allow only one bus master at a time to be operational on the system bus.

The bus masters (in order of priority) are:

1. Video
2. Refresh
3. CPU
4. Suzy

Video and refresh requests should occur together whenever possible. This will reduce the overhead of acquiring the bus from Suzy.

There is no default owner of the bus, If no one wants it, it is un-used. When in this un-used condition, the system is 'asleep'.

BIG NOTE: Sleep is broken in Mikey. See the hardware spec.

There are two inter-chip bus handshake lines, bus request and bus grant. Bus request comes from Mikey and is the logical 'or' of the internal CPU, video, and refresh bus requests. Bus grant comes from Suzy and is generated by Suzys bus controller.

1.1 Bus Controllers

There are two bus controllers, one in Mikey and the other in Suzy. The Mikey controller handles the CPU, video, refresh, and the bus grant from Suzy. The controller in Suzy handles the Suzy internals, the bus request from Mikey, and the bus grant to Mikey.

1.2 Suzy Bus Controller

Internals/Externals

Inorder to maintain high bus efficiency in Suzy, it can not give up the bus on any arbitrary cycle. Suzy will have certain cycles in which it can give up the bus

in response to a request from Mikey. Knowledge of those cycles is available to the bus handshake logic.

1.2.1 Suzy External Bus Handshake

Suzy has a bus enable flip-flop that is controlled by the CPU. If the flip-flop is reset, Suzy will release the bus at her next appropriate cycle and will not try to re-acquire it. This flip-flop is reset by the CPU and by the system reset signal. When the flip-flop is set by the CPU, Suzy can now monitor the bus request line from Mikey and when it is off, Suzy can acquire the bus. As soon as Suzy begins acquiring the bus, she will set the bus grant line off. Suzy will only do this if in fact she wants the bus.

After Suzy has the bus, she will relinquish it for only two reasons. The first is that Suzy is done with her task and no longer needs the bus. The second is that the bus request signal from Mikey has come on. In both cases, the bus grant to Mikey is set on when the appropriate cycle in Suzy occurs.

The off state of the Suzy bus enable flip-flop does not modify or reset the internal functioning of Suzy. It may be set or reset by the CPU at any time without adversely affecting the operation of the Suzy internals. This flip-flop only controls Suzys access to the bus. When Suzy is performing a function that requires the bus, it will merely pause until the bus becomes available.

1.3 Mikey Bus Controller

The Mikey bus controller has 3 requestors for its bus, the CPU, the video, and refresh. After system reset, the video and refresh are disabled and the CPU request is on. The CPU, when appropriate, can enable the video and refresh circuits to request the bus. See the state machine description (appendix 7) for cycle by cycle details of bus acquisition.

1.3.1 CPU Bus Release

When the CPU wishes to release the system bus (either to go to sleep or to allow Suzy to have the bus), it resets its bus request flip-flop. This flip-flop can only be reset by the CPU. It can be set by system reset, any unmasked interrupt, and Suzys bus grant coming on (oops !! the return of bus grand from a request generated by a ref or vid will look like a suzy done edge. I dont know how to separately identify them !!!!!)(Perhaps we can pulse the bus grant line from off to on to off to on. This is an unnatural signal and can be detected by the CPU flip flop.). Care must be taken in the hardware design to never miss the bus grant edge nor to falsely identify one.

When the CPU resets its bus request flip-flop, and if no other Mikey master is requesting the bus, the bus request line will drop. If Suzy wants the bus, she can now take it. If not, then no one wants the bus and the system goes to sleep.

BIG NOTE: Sleep is broken in Mikey. See the hardware spec.

1.3.2 CPU Bus Request

Once the CPU has released the bus, it will only get it back when its bus request flip-flop is set as described above. When it gets set, one of the following situations is occuring:

System Reset.	That condition is described elsewhere.
Suzy Bus Grant.	Suzy is done, and the CPU can now continue processsing. Suzy has already set bus grant, so when the CPU requests the bus, he will get it.
Interrupt.	If no one has the bus (the system was asleep), the CPU will get the bus as soon as it requests it. If Suzy has the bus, she is monitoring the bus request line and will grant the bus to Mikey when appropriate.

1.3.3 Video/Refresh Request

The video and refresh circuits will regularly request the bus unless they are disabled. When this request occurs, one of the following will happen:

The CPU has the bus.	At the appropriate point in the state machine cycle, the bus controller will pause the CPU clock and give control to the video or refresh requestor.
Suzy has the bus.	Mikey sends the bus request line to Suzy. At the appropriate point in Suzys cycle she sets the bus grant line. At that point the Mikey bus controller can give control to the video or refresh requestor.
System was asleep.	No one has the bus and the bus controller will give it to the requestor.

1.4 Bus Request/Grant Summary

In summary, the functioning of each of the bus players can be isolated to its particular surroundings as follows:

1. Suzy has a bus enable flip-flop. If it is on, Suzy can participate in the bus game. If not, then Suzy ignores bus request and always provides bus grant.

2. If the bus enable flip-flop is on and if Suzy wants the bus, then she will monitor the bus request line. When that line is off, Suzy will then acquire the bus and reset the bus grant line. When the bus request line comes on, Suzy will (eventually) relinquish the bus and set the bus grant line on.

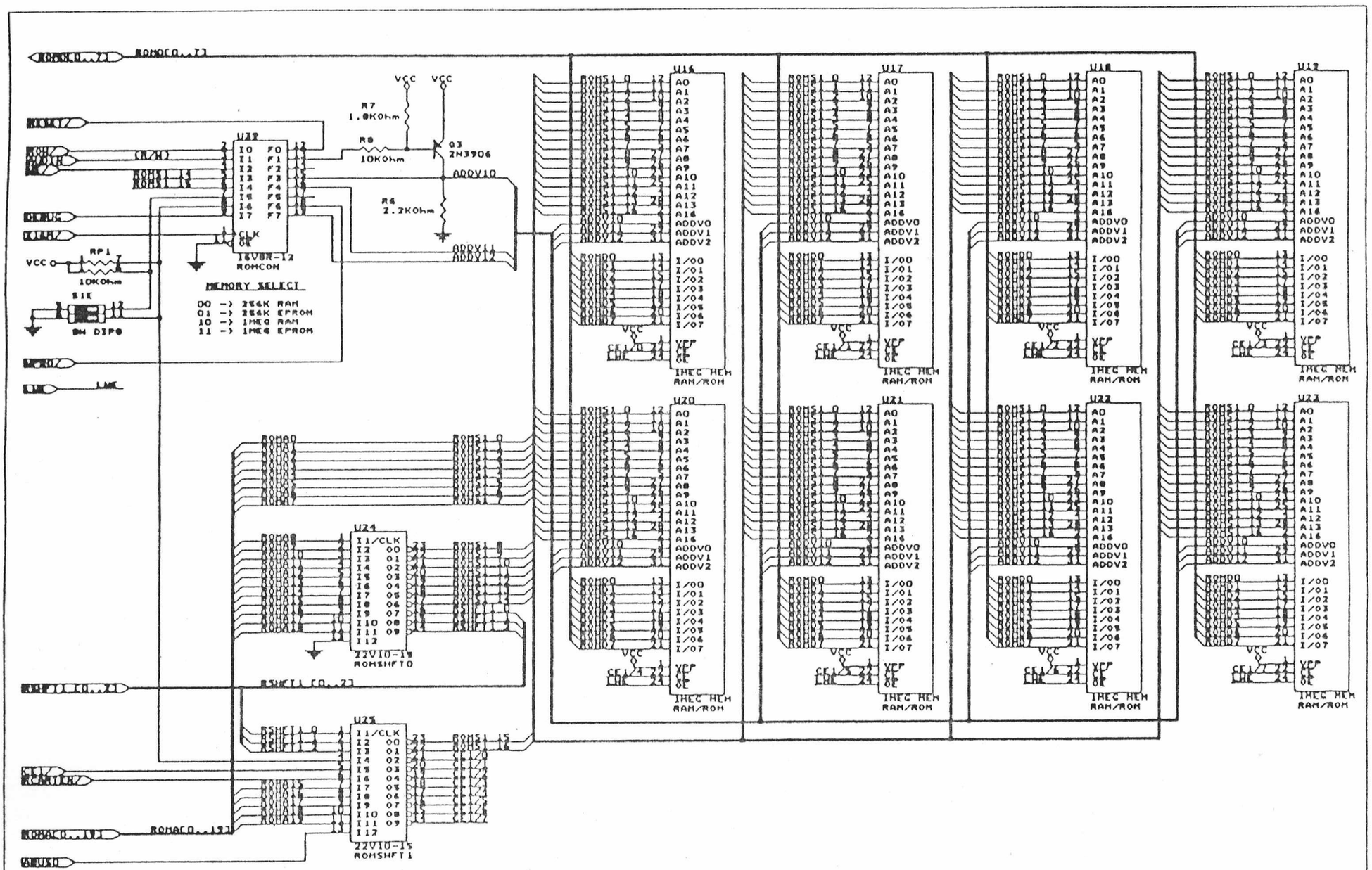
3. When Suzy is done with the bus, she will give it back by setting (or pulsing) the bus grant line.

4. When the video wants the bus, it will set its bus request line. Eventually, it will see that it has been made owner of the cycle and it will proceed. When it is done, it will reset its bus request line.

5. When the refresh wants the bus, it will set its bus request line. Eventually, it will see that it has been made owner of the cycle and it will proceed. When it is done, it will reset its bus request line.

6. When the CPU needs to get the bus, the decider of the need will set the CPU bus request line. Eventually, it will see that it has been made owner of the cycle and it will proceed. When it is done, it will reset its bus request line.

7. The logical 'or' of the CPU, the video, and the refresh 'bus request' signals is sent from Mikey to suzy as the inter-chip bus request signal.

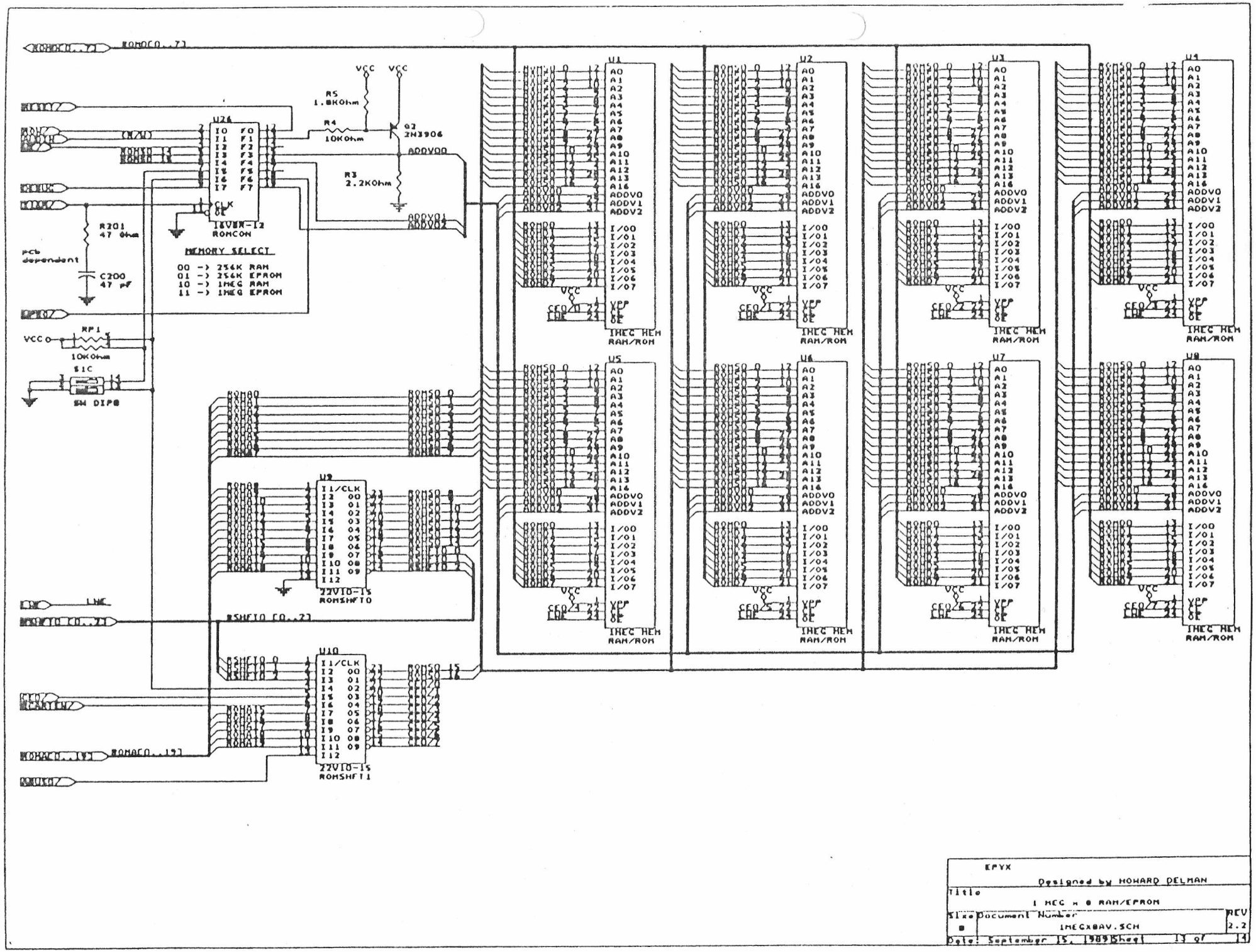


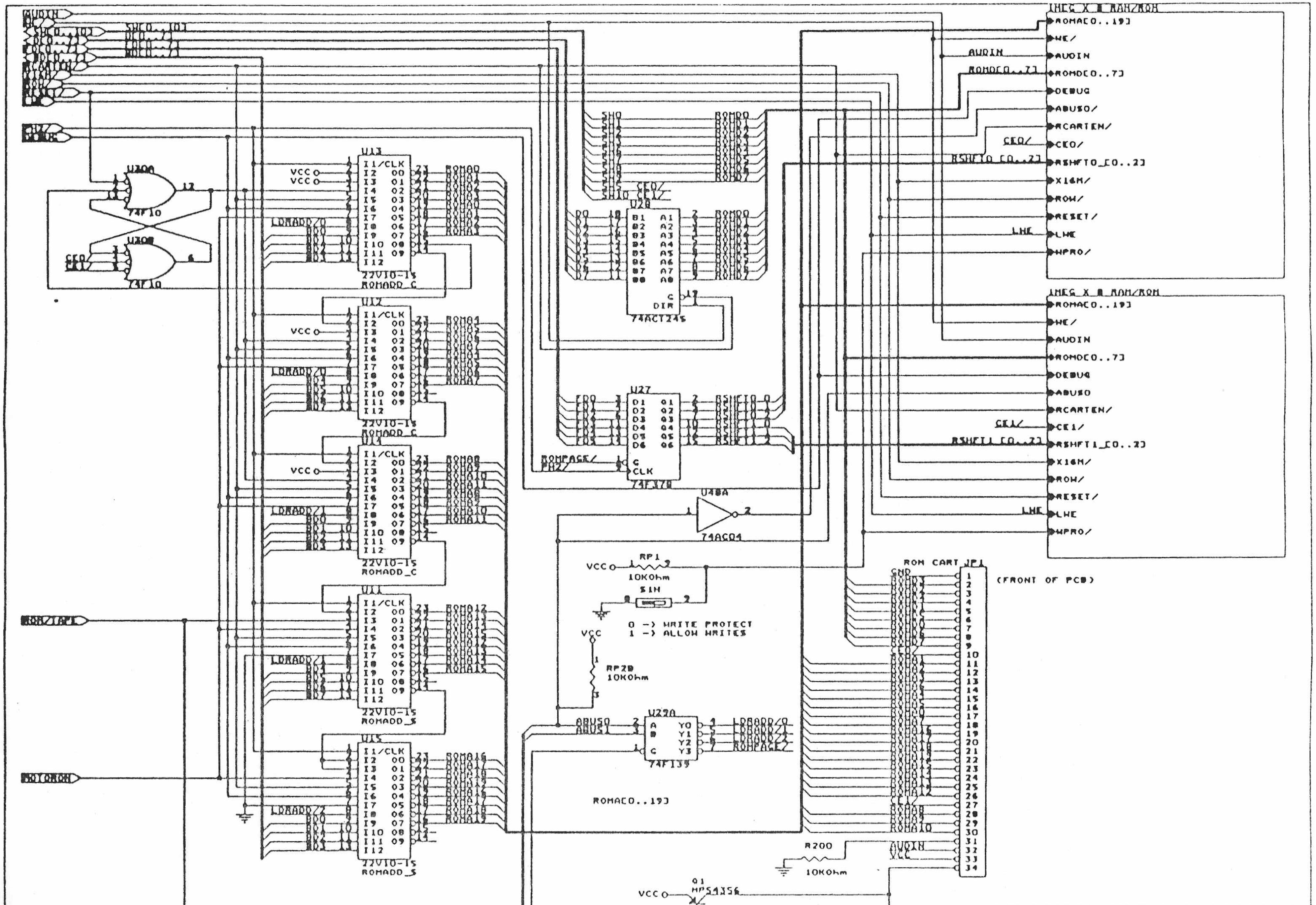
EPYX

Designed by HOWARD DELMAN

Title: 1 MEG x 8 RAM/EP

Size: Document Number: REV


ROMDC0..7]
VCC
R5
1.8KOhm
R4
10KOhm
Q2
2N3906
ADDV00
R3
2.2KOhm
ADDV01
ADDV02
U26
I0 F0
I1 F1
I2 F2
I3 F3
I4 F4
I5 F5
I6 F6
I7 F7
CLK
OE
I&VEN-I2
ROMCON
MEMORY SELECT
00 -> 256K RAM
01 -> 256K EPROM
10 -> 1MEG RAM
11 -> 1MEG EPROM
R201
47 Ohm
pcb dependent
C200
47 pF
RP1
10KOhm
SIC
SW DIP8
U1
U2
U3
U4
U5
U6
U7
U8
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A16
ADDV0
ADDV1
ADDV2
I/O0
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
CE0
CE1
VPP
OE
1MEG HEM
RAM/ROM
U9
U10
I1/CLK
I2 O0
I3 O1
I4 O2
I5 O3
I6 O4
I7 O5
I8 O4
I9 O7
I10 O8
I11 O9
I12
22V10-15
ROMSHFT0
ROMSHFT1
RSHFI0 [0..2]
ROMAC0..19]
EPYX
Designed by HOWARD DELMAN
Title
1 MEG x 8 RAM/EPROM
Size
B
Document Number
1MEGx8AV.SCH
REV
2.2
Date: September 15, 1989
Sheet 13 of 14

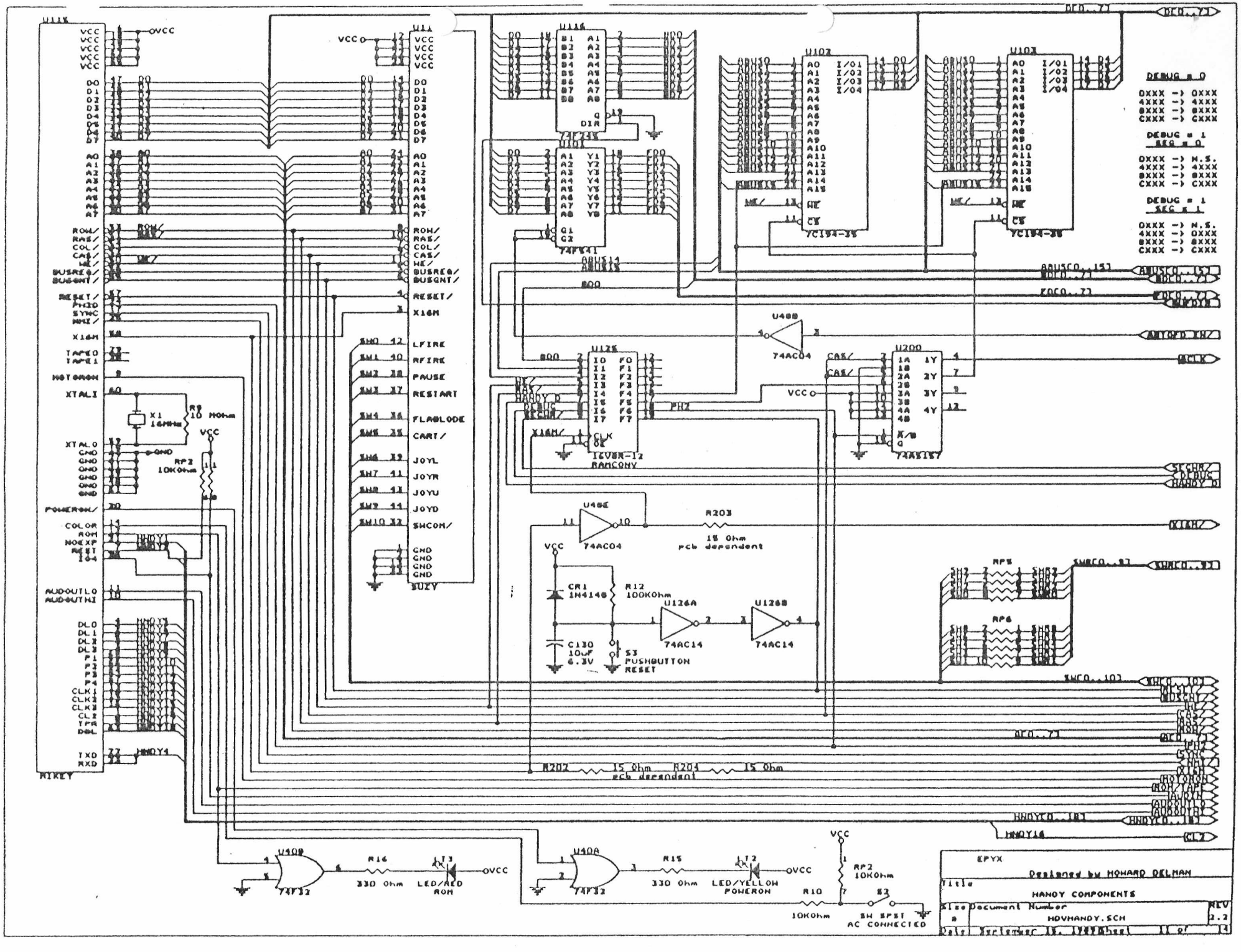


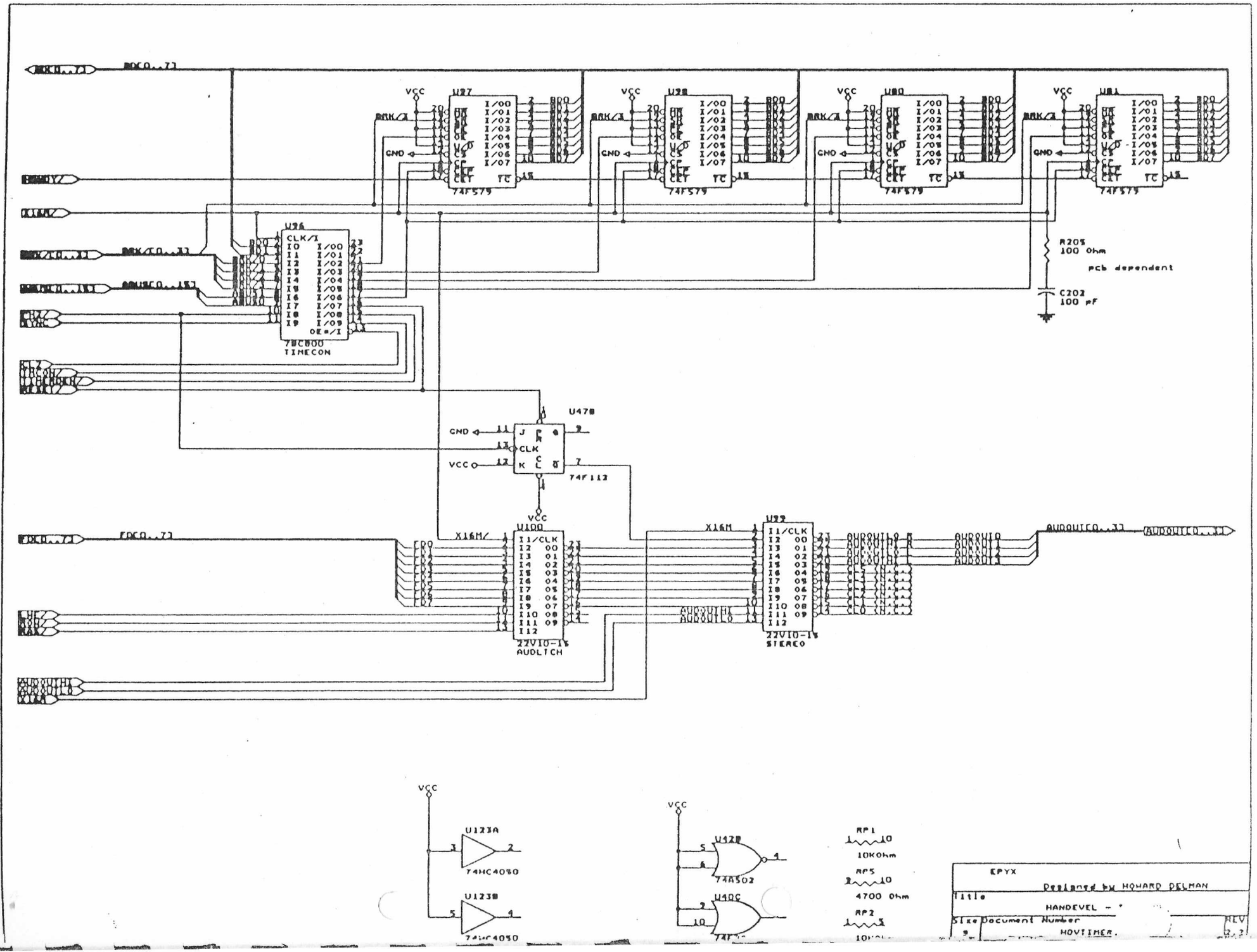
EPYX

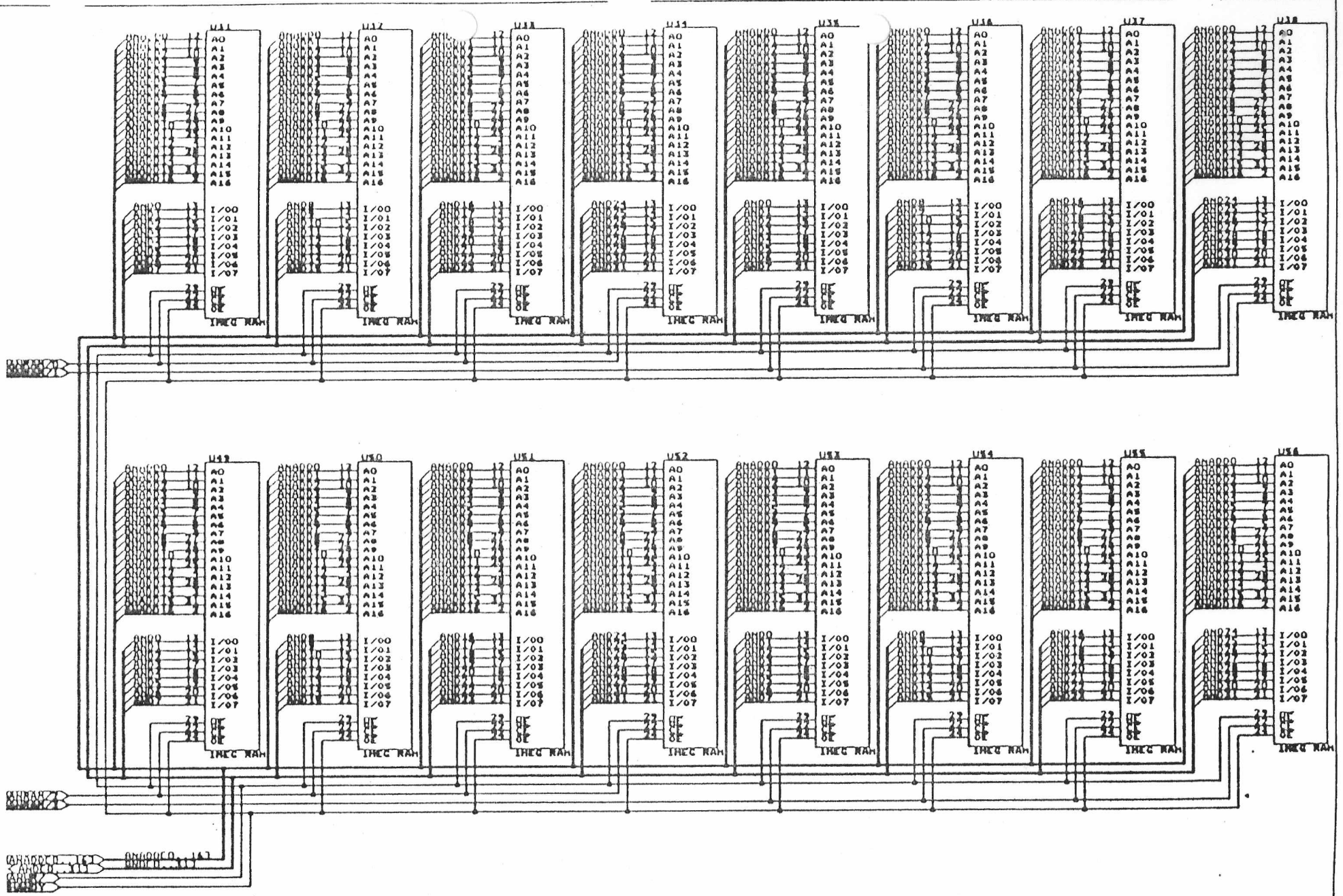
Drawn by ... DELMAN

Title

HANDEVELOP ROM (...)GE


DEBUG = 0
0XXX -> 0XXX
4XXX -> 4XXX
8XXX -> 8XXX
CXXX -> CXXX
DEBUG = 1
SEG = 0
0XXX -> M.S.
4XXX -> 4XXX
8XXX -> 8XXX
CXXX -> CXXX
DEBUG = 1
SEG = 1
0XXX -> M.S.
4XXX -> 0XXX
8XXX -> 8XXX
CXXX -> CXXX
R203
15 Ohm
pcb dependent
R202
15 Ohm
pcb dependent
R203
15 Ohm
330 Ohm
330 Ohm
10KOhm
SW SPST
AC CONNECTED
EPYX
Designed by HOWARD DELMAN
Title
HANDY COMPONENTS
Size
Document Number
HDVHANDY.SCH
REV
2.2
Date
Sheet
of


VCC
U27
U28
U80
U81
74F579
GND
BRK/I
R205
100 Ohm
pcb dependent
C202
100 pF
U26
78C800
TIMECON
U47B
74F112
CLK
X16M
X16M/
U100
22V10-15
AUDLTCH
U99
STEREO
U123A
74HC4050
U123B
U42B
74AS02
U42C
RP1
10KOhm
RP5
4700 Ohm
RP2
EPYX
Designed by HOWARD DELMAN
Title
HANDEVEL - *
Size
Document Number
HOVTIMER
REV



EPYX

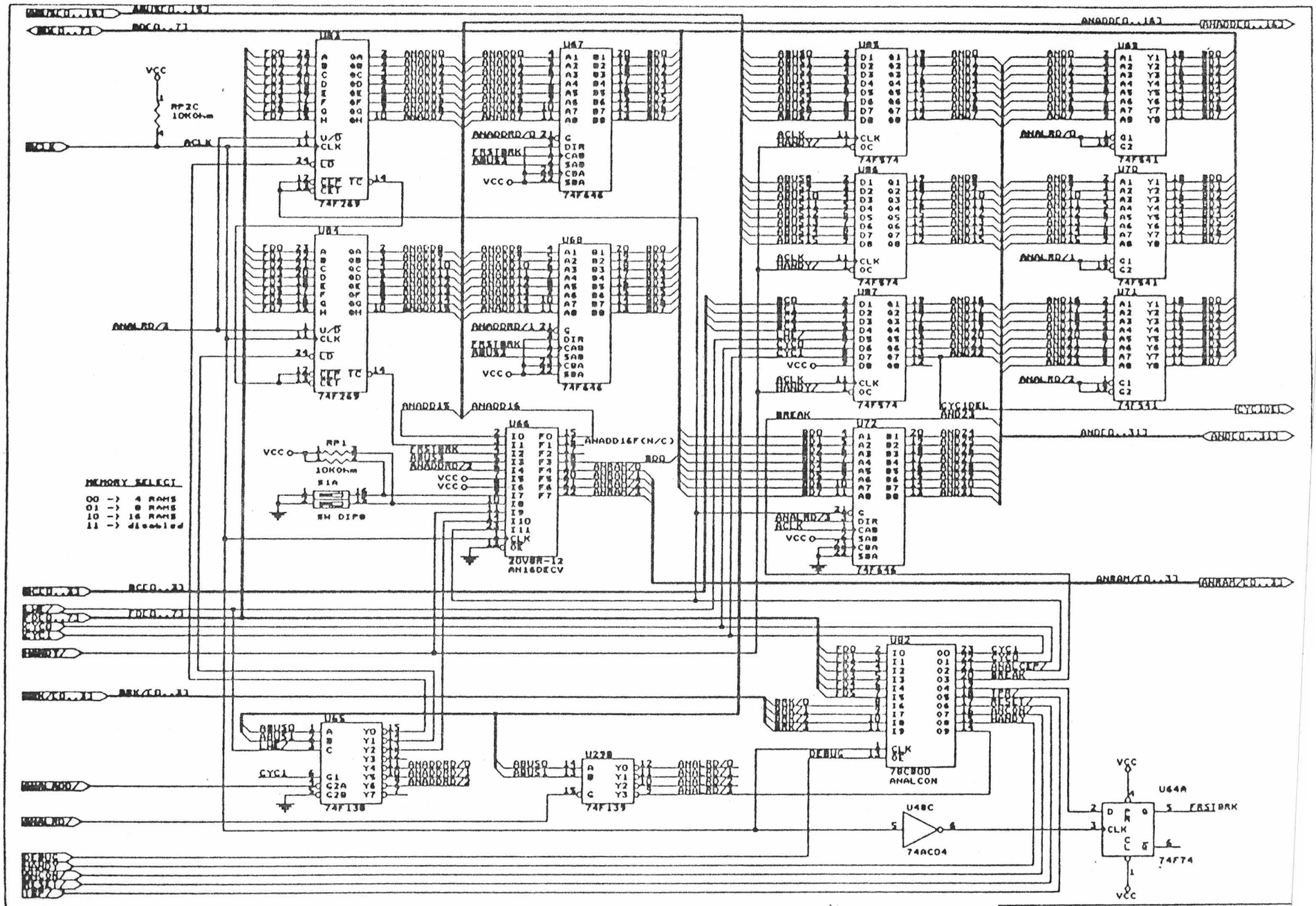
Designed by HOWARD DELMAN

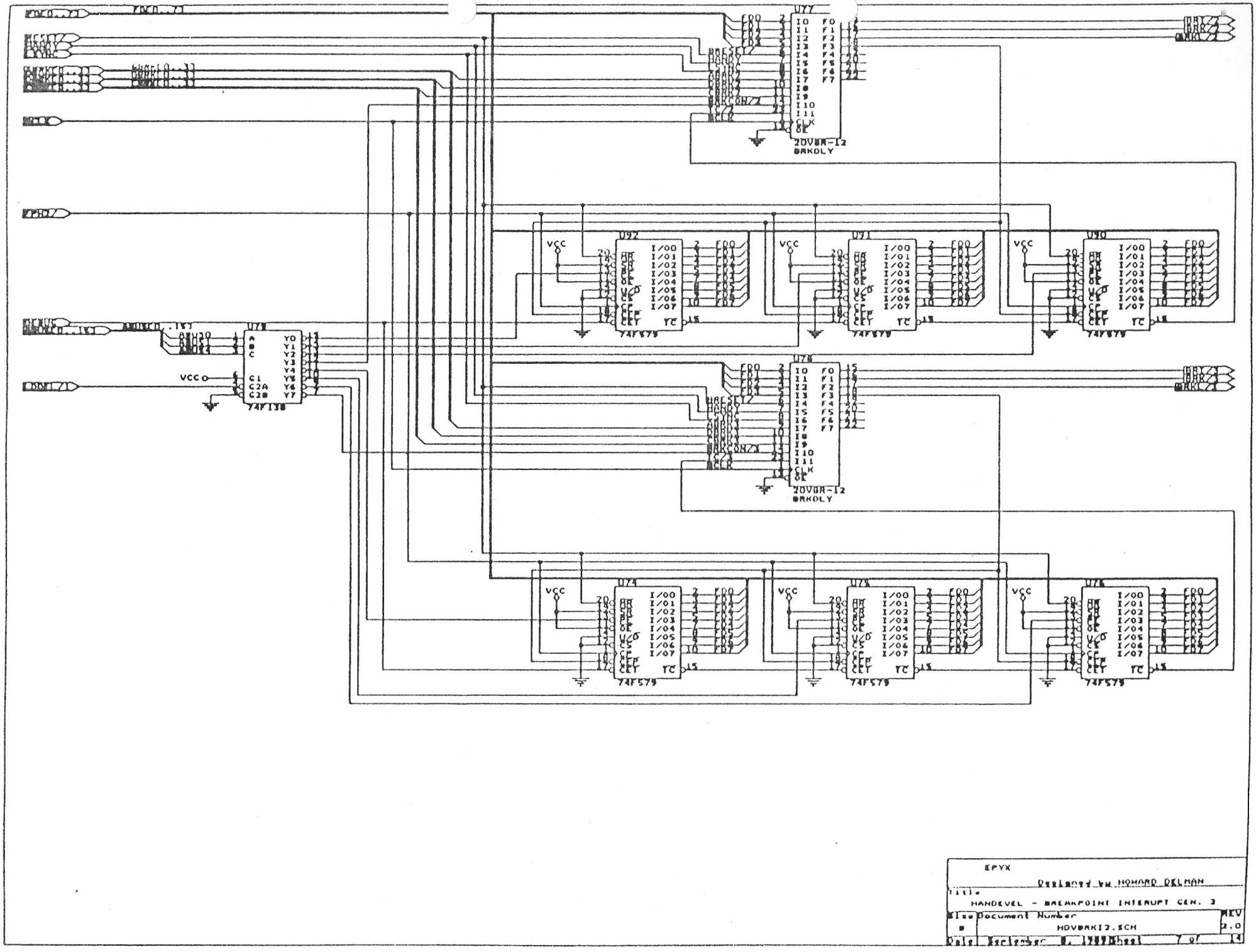
Title

HANDEVEL - ANALYZER RAM

Size: B | Document Number: HOVANRAM.SCH | REV: 2.0

Date: September 8, 1988 | Sheet | of


MEMORY SELECT
00 -> 4 RAMS
01 -> 8 RAMS
10 -> 16 RAMS
11 -> disabled
RP2C
10KOhm
RP1
10KOhm
SW DIP8
20V8A-12
AN16DECV
74F269
74F646
74F574
74F541
74F646
74F138
74F139
78C800
ANALCON
74AC04
74F74
U64A
EPYX
Designed by HOWARD DELMAN
Title
HANDEVEL - ANALYZER
Size
Document Number
Date
September 15



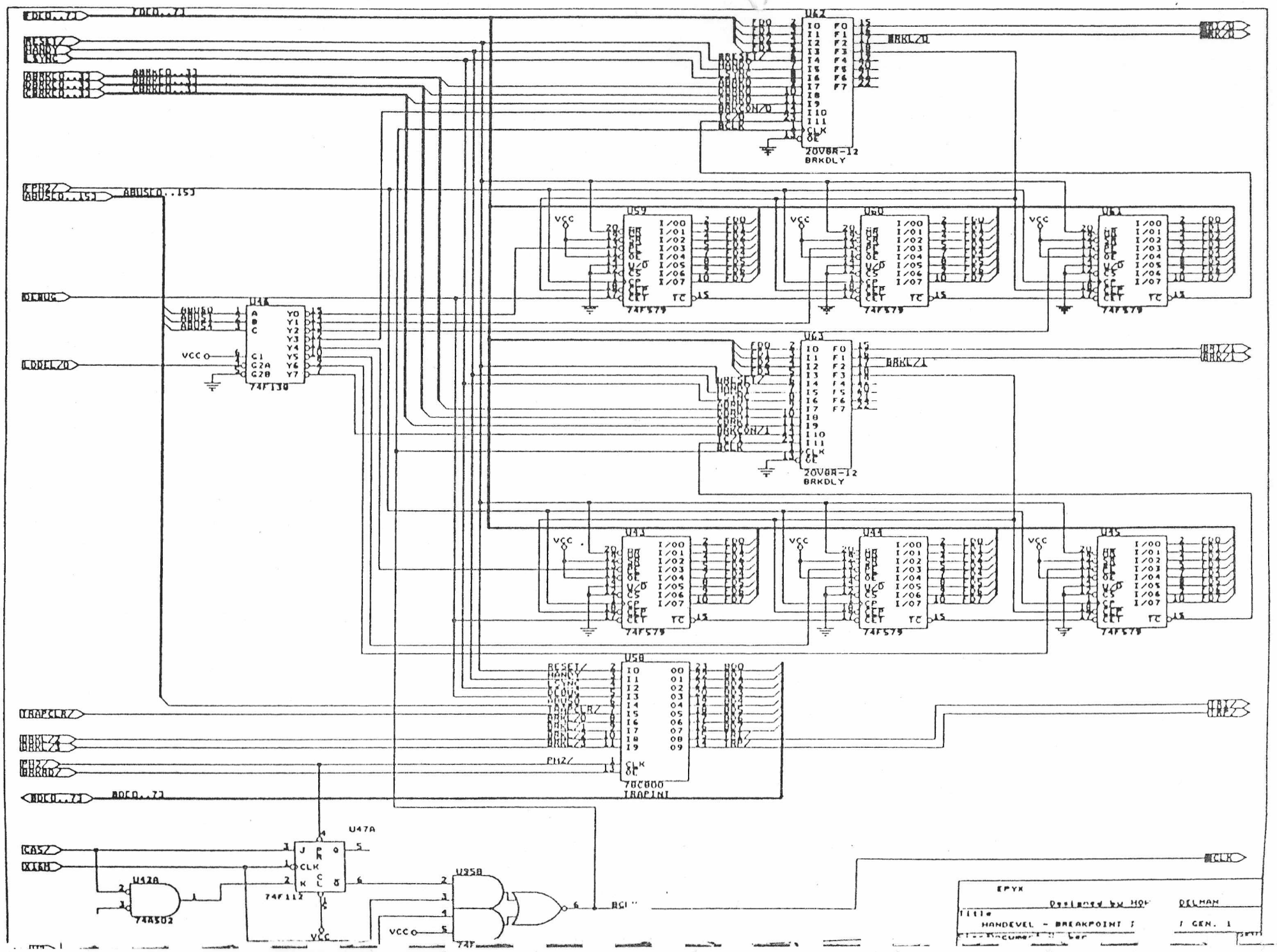
EPYX

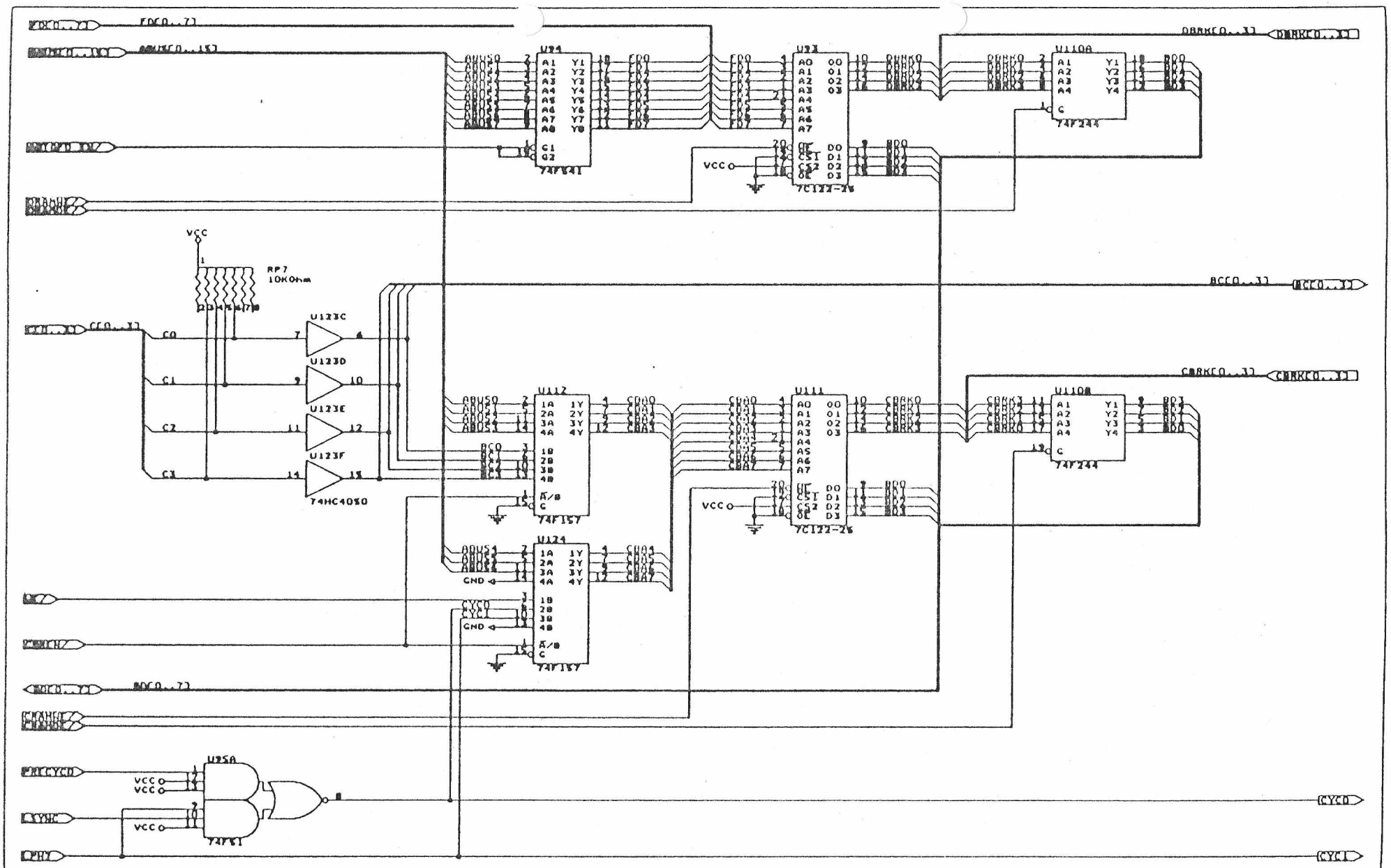
Designed by HOWARD DELMAN

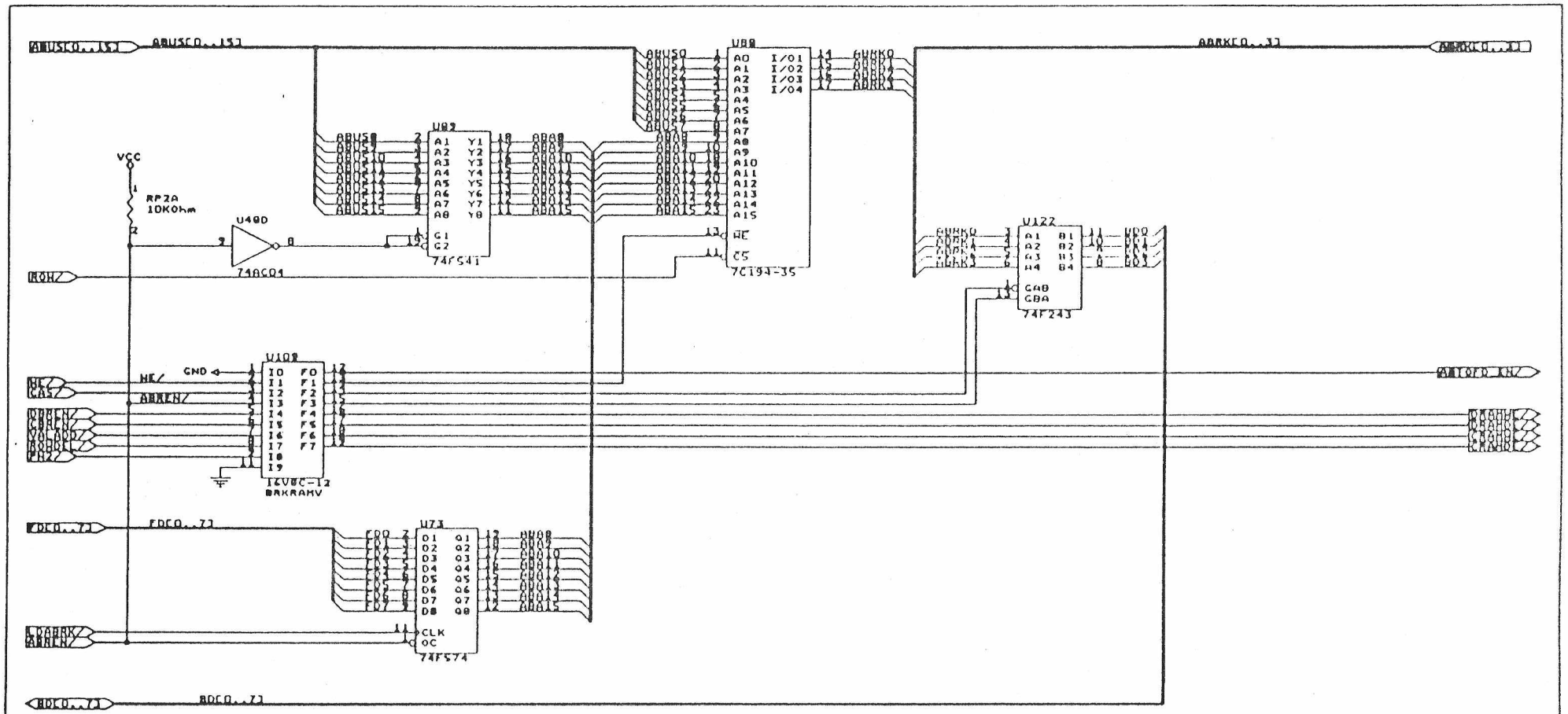
Title: HANDEVEL - BREAKPOINT INTERUPT GEN. 3

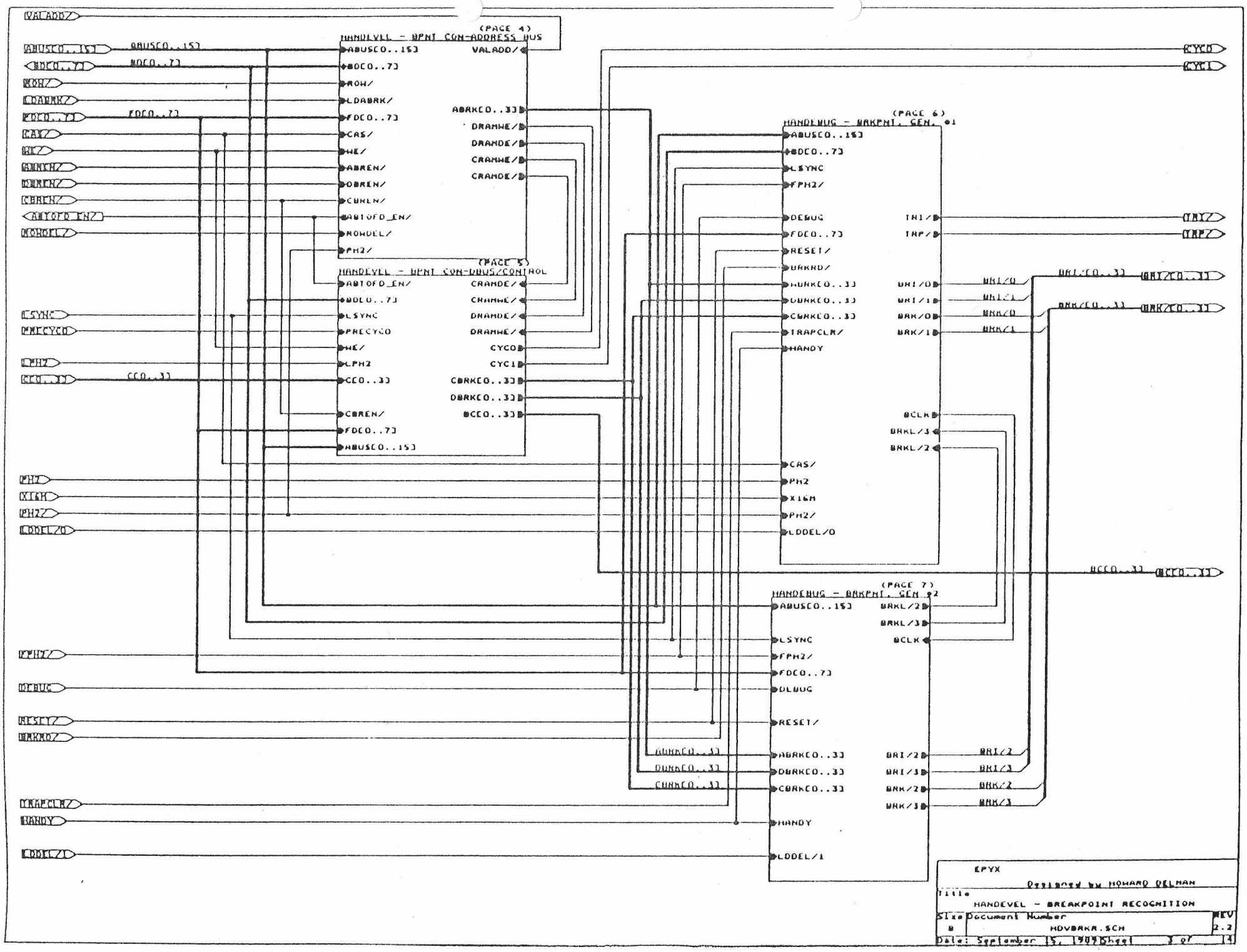
Size	Document Number	REV
B	HDVBRK12.SCH	2.0

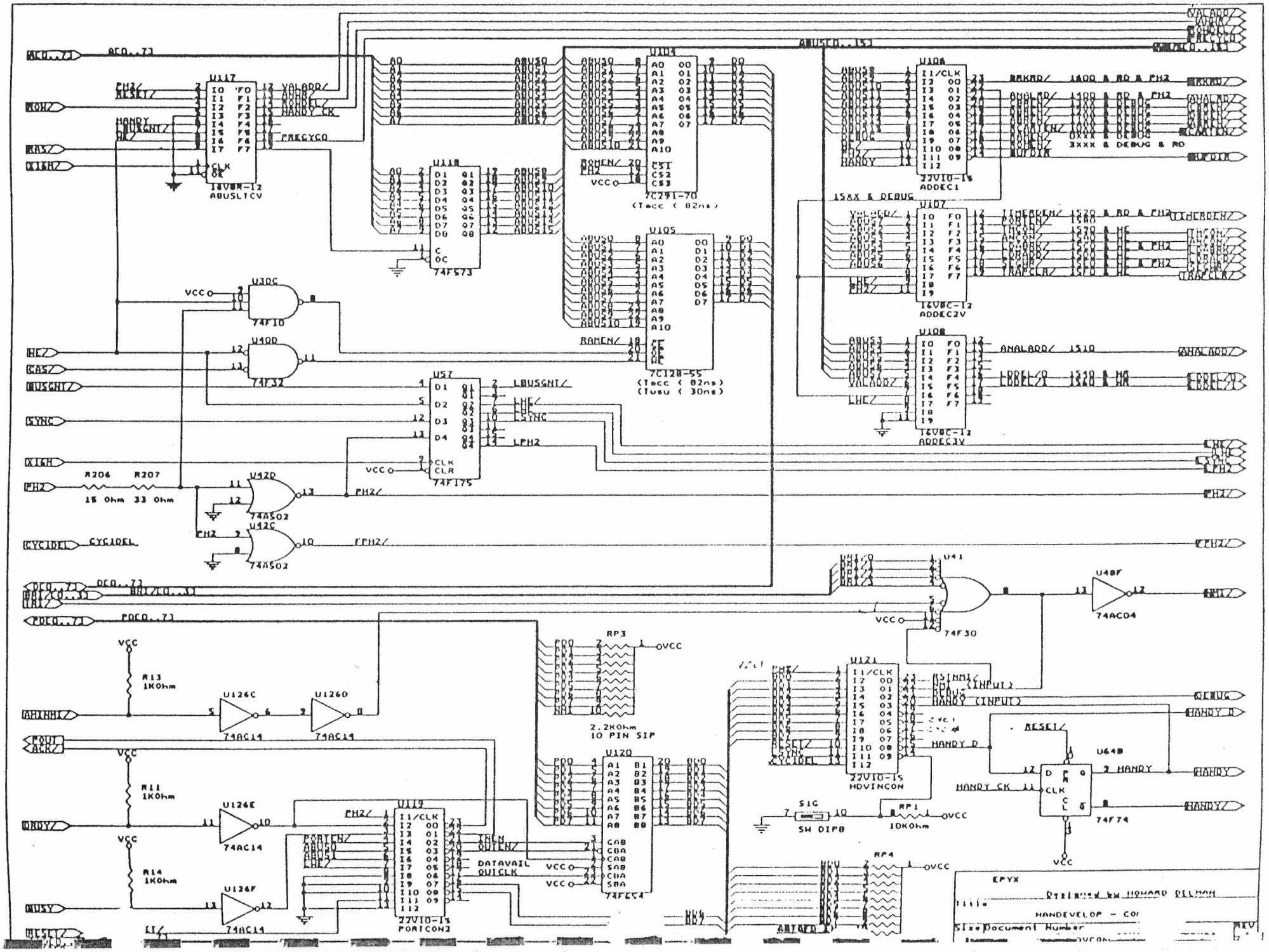
Date: Sheet of

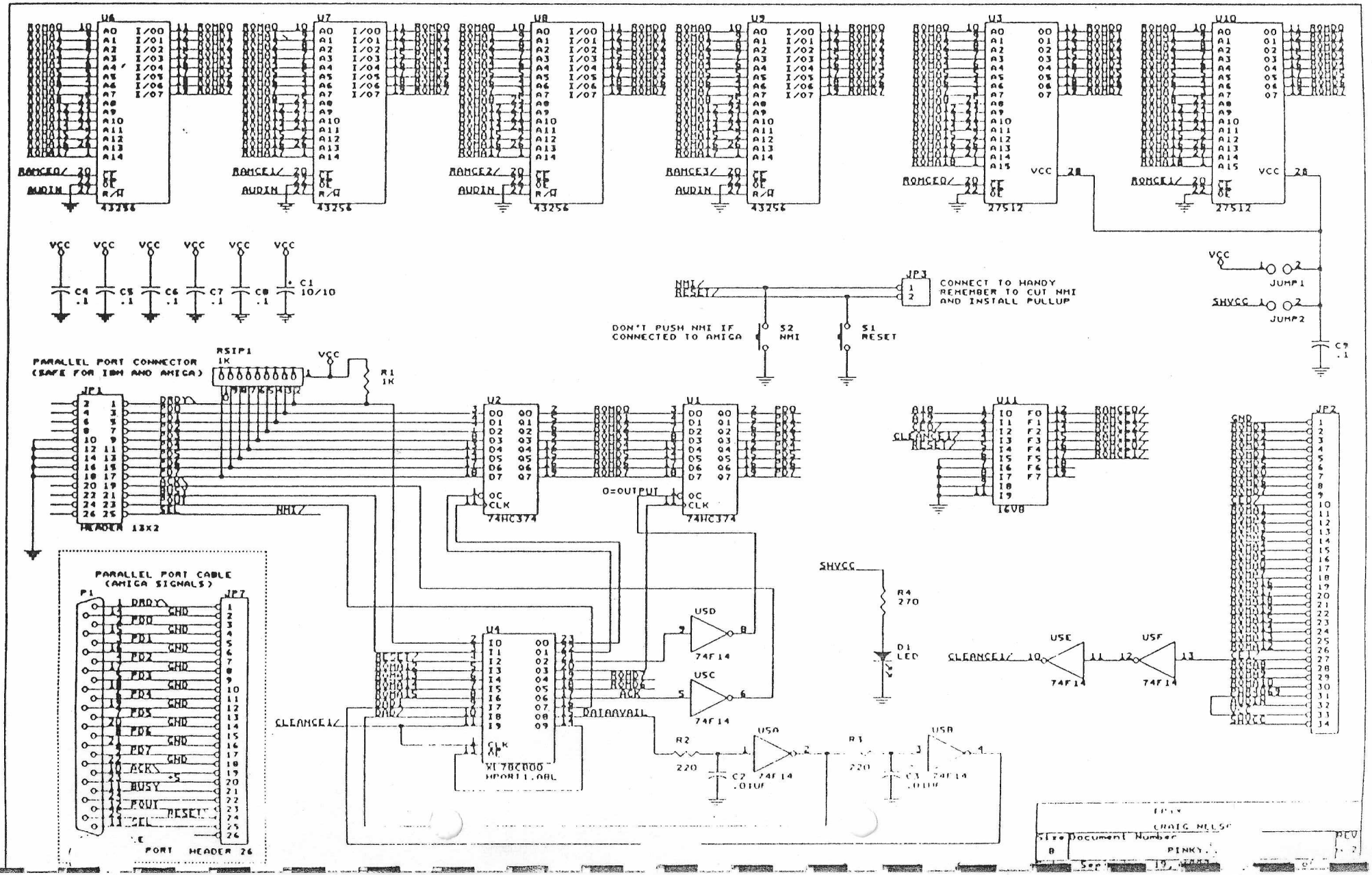

U62
20V8A-12
BRKDLY
74F579
U59
U60
U61
U63
U43
U45
U58
74C800
IRQPINT
U46
74F138
U47A
74F112
U42A
74AS02
U95B
74F
EPYX
MANDEVEL - BREAKPOINT I
I GEN. I

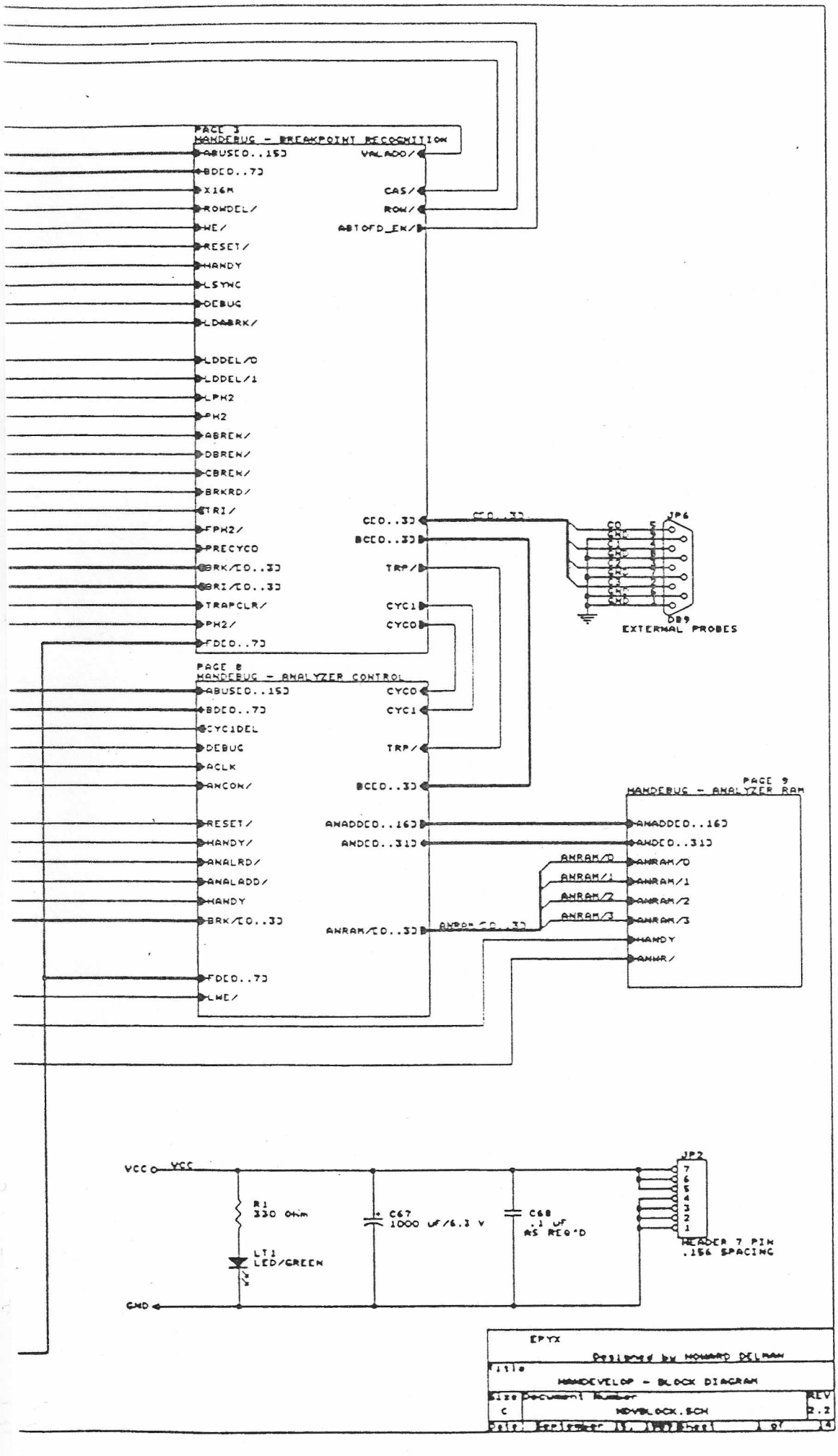

EPYX
Designed by HOWARD DELMAN
Title
HANDEVEL - BREAKPOINT CONTROL DATA/CONTROL
Size
B
Document Number
HDVBRKD.SCH
Rev
2.1
Date: September 15, 1989
Sheet 5 of 14

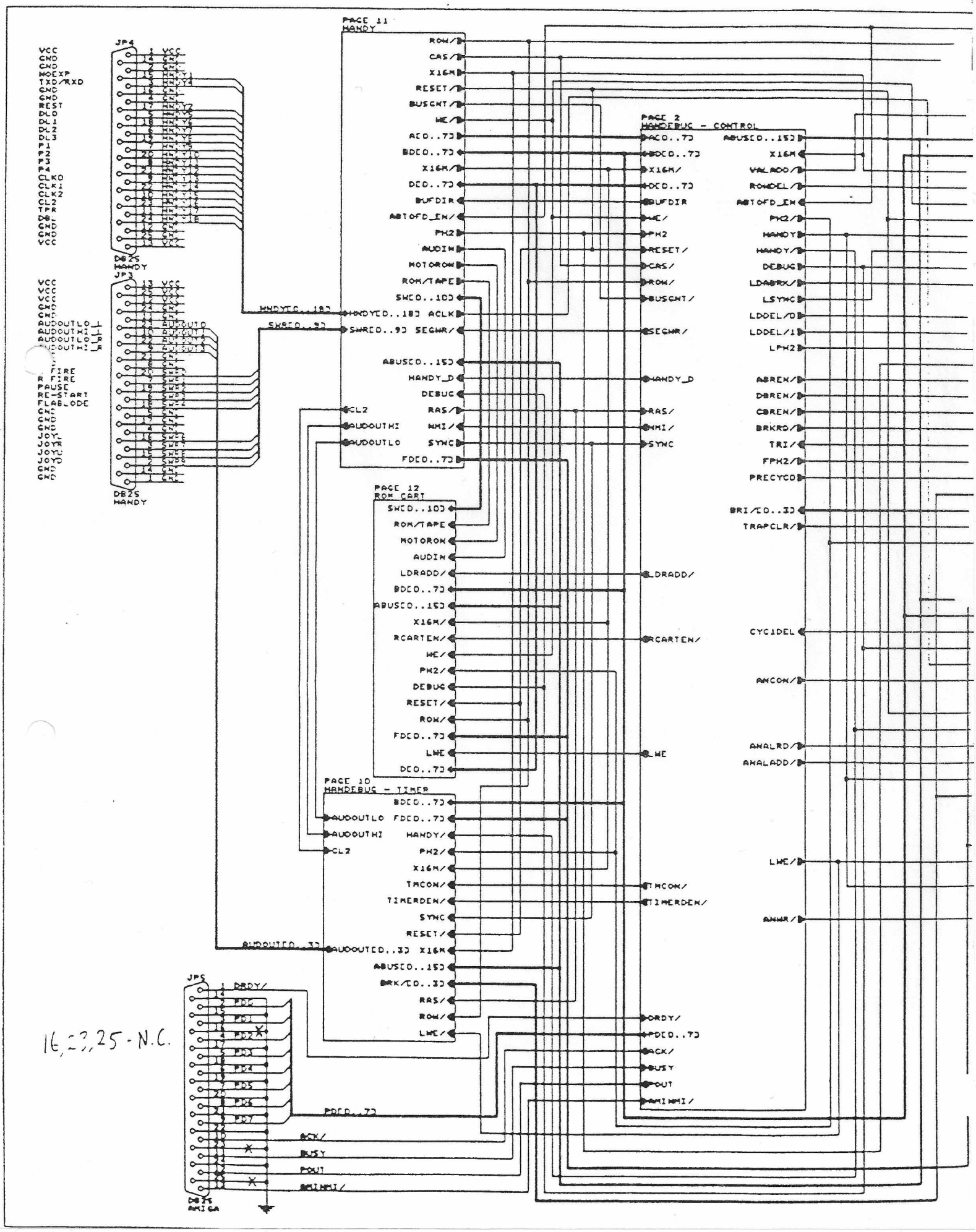

ABUS[0..15]
ABUS[0..15]
VCC
RP2A
10KOhm
U49D
74AS04
RQW/
U83
74FS541
U88
74C194-35
U122
74F243
ABRK[0..3]
ABRK[0..3]
U109
GND
WE/
CAS/
ABRKEN/
IDT7BC-12
BRKRAMV
FDI[0..7]
FDI[0..7]
U73
74FS74
CLK
OC
BDI[0..7]
BDI[0..7]
EPYX
Designed by: NO'
DELMAN
Title:
HANDEVEL - BREAKPOINT CONT
ADDRESS BUS


VALADD/
ABUSCO..15
BDCO..7
ROW/
LDABRK/
FDCO..7
CAS/
WE/
ABREN/
DBREN/
CBREN/
ABTOFD_EN/
NOMDEL/
PH2/
(PAGE 4)
HANDEVEL - BPNT CON-ADDRESS BUS
ABRKCO..3
DRAMWE/
DRAMDE/
CRAMWE/
CRAMDE/
(PAGE 5)
HANDEVEL - BPNT CON-DBUS/CONTROL
CYC0
CYC1
DBRKCO..3
BCCO..3
CBREN/
CSYNC
PRECYCO
LPH2
CCO..3
(PAGE 6)
HANDEHUG - BRKPNT. GEN. #1
LSYNC
DEBUG
TRI/
TRP/
RESET/
BRKRD/
UBRKCO..3
LBRKCO..3
CBRKCO..3
TRAPCLR/
HANDY
BCLK
BRKL/3
BRKL/2
BRI/0
BRI/1
BRK/0
BRK/1
BRI/CO..3
BRK/CO..3
X16M
LDDEL/0
(PAGE 7)
HANDEHUG - BRKPNT. GEN #2
BRI/2
BRI/3
BRK/2
BRK/3
LDDEL/1
EPYX
Designed by HOWARD DELMAN
Title
HANDEVEL - BREAKPOINT RECOGNITION
Size
B
Document Number
HDVBRKR.SCH
REV
2.2
Date: September 15, 1989
Sheet 14


ABUS[0..15]
U117
U118
U103
U106
U107
U108
U105
U57
U32D
U32C
U41
U48F
U121
U120
U119
U64B
U126C
U126D
U126E
U126F
RP3
RP1
RP4
R206
R207
15 Ohm
33 Ohm
R13 1KOhm
R11 1KOhm
R14 1KOhm
74F10
74F32
74F175
74AS02
74F30
74AC04
74AC14
74F374
74F654
74F573
7C291-70
7C128-55
2.2KOhm 10 PIN SIP
10KOhm
SW DIP8
22V10-15 ADDEC1
16V8C-12 ADDEC2V
16V8C-12 ADDEC3V
22V10-15 HDVINCON
22V10-15 PORTCON2
EPYX


PARALLEL PORT CONNECTOR
(SAFE FOR IBM AND AMIGA)
PARALLEL PORT CABLE
(AMIGA SIGNALS)
DON'T PUSH NMI IF
CONNECTED TO AMIGA
CONNECT TO HANDY
REMEMBER TO CUT NMI
AND INSTALL PULLUP
0=OUTPUT
74HC374
74F14
43256
27512
HEADER 13X2
PORT HEADER 26
CRAIG NELSO
PINKY..


PAGE 3
MAKDEBUG - BREAKPOINT RECOGNITION
ABUSE[0..15]
BDE[0..7]
X16M
ROWDEL/
WE/
RESET/
HANDY
LSYNC
DEBUG
LDABRK/
LDDEL/0
LDDEL/1
LPH2
PH2
ABREN/
DBREN/
CBREN/
BRKRD/
TRI/
FPH2/
PRECYCO
BRK/E[0..3]
BRI/E[0..3]
TRAPCLR/
PH2/
FDE[0..7]
VALADD/
CAS/
ROW/
ABTOFD_EN/
CE[0..3]
BCE[0..3]
TRP/
CYC1
CYC0
PAGE 8
MAKDEBUG - ANALYZER CONTROL
ABUSE[0..15]
BDE[0..7]
CYC1DEL
DEBUG
ACLK
ANCON/
RESET/
HANDY/
ANALRD/
ANALADD/
HANDY
BRK/E[0..3]
FDE[0..7]
LWE/
CYC0
CYC1
TRP/
BCE[0..3]
ANADDE[0..16]
ANDE[0..31]
ANRAM/E[0..3]
ANRAM/E[0..3]
JP6
DB9
EXTERNAL PROBES
PAGE 9
MAKDEBUG - ANALYZER RAM
ANADDE[0..16]
ANDE[0..31]
ANRAM/0
ANRAM/1
ANRAM/2
ANRAM/3
HANDY
ANWR/
VCC
VCC
R1
330 ohm
LT1
LED/GREEN
C67
1000 uF/6.3 V
C68
.1 uF
AS REQ'D
GND
JP2
READER 7 PIN
.156 SPACING
EPYX
Designed by HOWARD DELMAN
MAKDEVELOP - BLOCK DIAGRAM
Size C
Document Number
MDVBLOCK.SCH
REV 2.2
Date: September 15, 1987 Sheet 1 of 14


PAGE 11
HANDY
PAGE 2
HANDEBUG - CONTROL
PAGE 12
ROM CART
PAGE 10
HANDEBUG - TIMER
JP4
DB25
HANDY
JP3
DB25
HANDY
JP5
DB25
AMIGA
16,23,25 - N.C.