THE COMPUTER HOBBYIST

Number 9 February, 1976

TCH SUPER SIMPLE FLOPPY DISK INTERFACE Part 1.

As the title implies, TCH has developed a simple, inexpensive controller for floppy disk drives. In fact, it has been running in our 8008 system since December 1975. Like the cassette interface, graphics display, and other projects, the floppy disk controller will run on any kind of computer. This part 1 will describe the characteristics and operation of floppy disk drives in detail as well as give an accurate estimate of the costs involved. The experience gained in transferring 2300 subscriber records to floppy disk and then updating several hundred of them for address changes as well as experience with a graphic image storage and retrieval system convinces us that the design is bug-free and reliable. Nevertheless, circuit diagrams will be held until part 2, to be in issue #10.

Floppy disk systems are not expensive! At least not as expensive as the $1200 to $3000 tags on commercially available systems would lead one to believe. Here are the hard facts on the cost of floppy disk system components. The drive is the worst part. As will be detailed later, drives from the various manufacturers are nearly identical and this applies to the single quantity price tag too; $625 to $750. The first quantity price break is about $100 and occurs at 25 units for most manufacturers. In 100-up quantities, prices range from $400 to $450. This is an example of what standardization and competition can do. How about the controller? With TCH's interface and less than 400 bytes of software, the controller is at most $60 worth of components, circuit board included! Twenty-five packages of very mundane TTL, two 2102 memory chips, and a little software that fits neatly into two 1702's is all it takes. The power supply is not a hassle either. Most drives require +5 volts at about an amp, and +24 volts at 1.5 amps, loosely regulated. Some also need a negative voltage at 50 MA or so. Twenty dollars more worth of components should take care of the power supply. All floppy disk drives have attractive front bezels. Packaging can be as simple as sitting the unit on the table. Mounting in a panel or large box however is just a matter of cutting a rectangular hole in the box front. Floppy disks to keep the critter fed are typically $7.50 each. Minimum order quantities are either 5 or 10 disks. This is not bad considering that the 300K byte capacity of one disk is equivalent to two C-60 cassettes recorded on both sides with a TCH audio cassette interface.

What kind of applications do the characteristics of floppy disks make possible? An obvious one is maintaining lists of information that are subject to frequent change and which need to be printed periodically, such as address lists. Maintaining such a list on tape is a pain at best and certainly requires two drives. With a floppy, any record can be retrieved, updated, and stored again in a few seconds. Text editing, whether it be manuscripts, or programs can be readily handled. The 300K byte capacity of the disk allows massive program text files of 100 to 200 pages of assembly language code to be on one disk. Assembling such programs is very rapid due to the fast transfer rate. Of course keeping extensive program files for debugging and demonstration purposes is a natural application. All kinds of special purpose applications present themselves. For example, Jim parker has written a graphics operating system for use with a floppy. Files of images, parts of images, and character shapes are stored on the disk. New images may be created as a combination of stored images and new graphic input. The stored images can be moved around in the new image and in some cases expanded or shrunk also. The new images can likewise be stored on the disk for later recall or incorporation into other images. The same concepts could also be applied to electronic music composition. A practical small business accounting system virtually requires disk storage, especially to implement the automatic functions that really make such systems pay off. Game applications where the program "learns" by maintaining a file of previous experience are almost endless. In each of these application suggestions, the fast random access and update capability of the floppy disk is central with the other characteristics being of less importance.

That should be enough for vague generalities, lets look now at the individual system components and some of their typical specifications. First we have the disk itself, often called a "diskette" which is an IBM term. The disk is a thin circular sheet of mylar (actually die cut from a wide roll of computer tape) coated on one or both sides with magnetic oxide material (see figure 1). The outside diameter is 7.88 inches and a 1.5 inch round hole is punched in the center for slipping over the drive spindle. An additional small hole is punched 1.5 inches from the center and is called the index hole. For environmental protection, this flimsy disk is enclosed in a sealed, semi-rigid envelope lined on the inside with a felt-like material. The envelope is 8 inches square and has a center punchout so the drive spindle can reach the disk. There is also a radial slot to allow the recording head to reach the disk, and a small hole to allow a lamp-photocell assembly to see the index hole as it rotates by. Finally, an optional write protect hole is punched into one edge of the envelope. Like cassettes, the physical dimensions of floppy disks are standardized, so all manufacturer's products are the same although price and quality could vary (they don't really, the only significant difference is purchasing hassle). There is one major variation however. The single index hole disk just described is the IBM standard but also available are disks which have 32 "sector holes" evenly spaced around the disk on the same circumference as the index hole (see figure 2). These are frequently called "hard sectored" diskettes because the extra holes define 32 distinct, physical sectors on the disk. Every major manufacturer except IBM makes both styles and sells them for the same price. TCH will be using the hard sectored variety for reasons that will be made clear later.

It presently costs manufacturers about $1.50 to make a diskette. Much of this cost is due to the "initializing" process where a predefined data pattern is written all over the disk. This pattern is required by systems using the one-hole diskettes but not by hard sectored systems even though the manufacturers write it on their 33 hole disks anyway. There is great potential for lower cost diskettes ($1.98 each) as usage and competition increases.

Next we have the diskette drive (see figure 3) The drives available from the 8 or so major manufacturers are all so much alike that any differences are usually confined to how the door closes and how fast the salesman talks. There are a few significant differences however which will be noted. First there is a door which opens to expose a narrow slot. The disk envelope is pushed all the way into the slot and the door is closed, trapping it there. Since the door is about all the operator sees, there has been considerable innovation in this area. Door action ranges from a simple hinge and handle to push-button latches which pop the diskette out like a toaster when released. A prime example of poor human engineering is the fact that the disk can be inserted into the drive in 8 different ways and only one of them works. When the door closes, cam action moves the disk against a cone shaped spindle which pokes through the large center hole. A spring loaded female cone then engages from the other side clamping the disk against the drive spindle. The spindle is driven at a constant speed of 360 RPM by a synchronous motor causing the disk itself to rotate inside the felt lined envelope. Except for one manufacturer, there is a noticable lack of innovation in the spindle drive. Typically, a large wheel fixed to the spindle is belt driven by an 1800 RPM motor mounted in the corner of the drive. The belt is best described as a precision rubber band. Several drive manufacturers even use the same identical Bodine 4-pole, split capacitor, sync motor! Pertec has taken a different approach on the FD-400. They use a 24-pole 3-phase motor driven by a transistor inverter powered from the 24 volt DC supply. The motor runs at 360 RPM, thus it drives the spindle directly. The only drawback is a higher acoustic noise level.

The head is on a carriage that can be moved radially, in towards the center or out towards the edge of the disk by a lead screw driven by a stepping motor. This carriage can be positioned at any one of 77 discrete points spaced about .02" apart by stepping the motor one direction or the other. Seventy-seven concentric tracks are thus defined on the disk surface with track 0 being the outermost and track 76 the innermost. A sensor, either a microswitch or lamp-photocell detects when the head is positioned at track 0. A step count must be kept to ascertain the other 76 possible positions. Some drives have two mechanical stops that prevent the head from being positioned beyond the allowable range while others have only one or none. Stepping speed is really the only parameter where there is significant competition. Many drives can step 100 times per second but several advertise a rate of 166 steps per second. Sycor beats them all at 400 steps per second for a maximum positioning time of 190 milliseconds.

CONTINUED ON 4

LETTERS TO THE EDITOR

Gentlemen:
In regards to TCH's IMP-16 CPU, my thanks for a much needed construction article. Several friends and I are convinced this is the CPU we have been waiting for and have started building one.
After spending many hours searching supplier catalogs for hard to find IC's; looking for the lowest cost source on other IC's and parts via price breaks, etc., we decided to compile a list of parts by supplier. This list may be of benefit to other hobbyists desiring to build the TCH IMP-16 CPU. I would be glad to send a copy to other hobbyists for $1.00 to cover postage and copying costs.
Also, we would be interested in communicating with other hobbyists constructing the TCH IMP-16 CPU about additional hardware and software plans. It would seem that a TCH IMP-16 User's Group would be a real possibility.
Thanks again for leading the way with what promises to be a superb series of construction articles for the home computer hobbyist.

Fred Holmes
101 Brookbend Ct.
Maudlin, SC 29662

It is easy to spot the enthusiasm in letters such as the above. Many thanks for the letters of encouragement from all quarters on our construction projects. Send parts list orders directly to Mr. Holmes at the above address.

Gentlemen:
Your editorial in the Vol. 1 No. 8 issue deals with octal VS hexadecimal. Your decision is to adopt a "split octal" notation. The decision is based upon accommodation to the Intel microprocessors.
Several years ago we faced the same call for decision. We went base-16. Reason--the square roots of round numbers are round numbers. Also the fourth roots. For our Altair computer we have changed all instructions over to base 16. This makes the instruction set difficult to use--as you have noted. But this instruction set is transitory. There is little doubt in this quarter that the base-16 system is the system of the future and that the octal system will fade like an old soldier.
Appended is a photo of our Altair with the front panel altered to accommodate the base-16 Computer Compatible Digit. Also a copy of the instruction set. Also a reprint of an EDN article discussing the system.
Incidentally we have interfaced a combinational keyboard to the Altair. With eight keys we send in the operational codes directly. No encoding. Cuts costs drastically. It took perhaps a day to learn the keyboard. We also have the keyboard interfaced to a printer for Computer Compatible Characters. On the next page appear the digits of the base 16 numbering system, followed by the lower case alphabet, followed by some text.

R. O. Whitaker

The article referred to appeared in the February, 1974 issue of EDN magazine. I read the article back then and was frankly intrigued with the ideas presented. However two years of time and experience have taken their toll. First, the advent of the $10 calculator proved that 7-segment displays and associated decoding can be done economically. Second, with the present configuration of the digit, there could be great confusion between 9 (ꓘ) and 12 (/|) when handprinted by a sloppy person. One only has to read as many scrawled addresses as we do to comprehend that problem. Finally, the economics of change and the general conservatism of people are tremendous obstacles to overcome. Look at the acceptance of the Dvorak typewriter keyboard (invented in 1932 by Dr. August, proven in tests to be 50 to 100% faster than conventional keyboard with fewer errors) for one example. Also look at the ongoing metric conversion effort in this country. It is barely started, will take 10 to 20 years to complete, will ultimately "cost" billions, and, sadly, had to be administered by the government.

EDITORIAL

By now nearly everyone should be aware of the results of the BYTE Standards Conference that was held in Kansas last November. We won't attempt to explain in detail the recording method agreed upon since this information will be available in Byte and Popular Electronics. However since Hal Chamberlin and Richard Smith were there for the whole meeting and took an active part in the proceedings we can give a "behind the scenes" report on just what transpired and how.
First, a rundown on the participants. Two groups were very conspicuous by their abscence. Scelbi Computer Consulting wasn't there. If they had been, they probably would have influenced the discussions considerably since they were making audio cassette systems long before the rest of us had thought of the concept. Another absent organization was The Digital Group of Denver, only about 300 miles from Kansas. As most of us know, they have been quite successful to date in penetrating the audio cassette interface market. Just before the conference they announced a speed tolerant version of their interface which would certainly have been of interest to the other participants. HAL Communications, well known for TV Radioteletype gear, came but left upon realizing the mismatch between their equipment offerings and the purpose of the conference. SPHERE and Popular Electronics left after the first two sessions (there were 4 in all). Les Solomon of P.E. was vocal in his support of the HIT format but unfortuantely could not answer the numerous technical questions flying about. He was particularly concerned about tape and recorder certification procedures. Other active groups in attendance who lasted through all 4 sessions were Godbout, Processor Technology, Southwest Technical Products, Lee Feldenstein (Pennywhistle 103), Ed Roberts of MITS, and Harold Mauch of Pronetics. MITS did not really try to infulence the proceedings but did occasionally bring up terms like "defacto standard" in relation to their recording method. Harold Mauch probably had the greatest influence since he has participated in digital cassette standards activity and has had much experience in the design of tape interfaces for industrial applications. We feel that TCH had a significant influence in issues such as record format and motor control.
Oddly enough, Carl Helmers, editor of BYTE was not there to run the show. Instead, Virginia and Manfred Peschke, the publishers were. Virginia essentially took notes and Manfred coordinated the sessions. While having considerable programming experience with large-scale computer equipment, Manfred had some difficulty with all of the technical terms being thrown around. In a way this was good since it forced the participants to be clear and unambiguous in their explanations and arguments. Incidentally, the conference was anything but a fun and games exercise. It was straight work all day Friday and Saturday morning with very little time off for socializing.
In the first session an organized engineering approach was taken to the problem at hand. Goals were stated and intended applications of the standard were established. Then the characteristics of low cost recorders were discussed and listed. Finally, a workable set of performance specifications was established. Up to this point, nothing had been said about any particular recording

THE COMPUTER HOBBYIST
Founded October, 1974

Stephen C. Stallings - Managing Editor
Hal Chamberlin - Contributing editor
Jim Parker - Contributor
Clyde Butler - Photographer
Richard Smith - Programming consultant

THE COMPUTER HOBBYIST is published at intervals approaching monthly near Raleigh, N.C. The subscription rate for 12 issues is $6.00 for persons in the U.S.A. and U.S. possessions. For Canada and Mexico the rate is $7.50 for 12 issues. Subscriptions in all other countries are $11.50 for surface mail and $16.00 for air mail. All back issues are available and cost 65 cents each for persons in the U.S.A., U.S. possessions, Canada, or Mexico. For all other countries back issues cost $1.00 each by surface mail or $1.50 each by air mail. Remittances from countries other than the U.S.A should be in the form of a bank or postal money order.
THE COMPUTER HOBBYIST is seeking paid contributors. Material to be submitted should be typed or neatly written and must not appear to be soliciting business for any firm. For details, please inquire.
Advertizing space is available to interested firms. please inquire for rates and formats.
All correspondence should be addressed to:

THE COMPUTER HOBBYIST
Box 295
Cary, NC 27511

method or existing system. The result of the morning session was a fairly complete body of engineering data needed to develop a recording method to meet the stated goals.

After lunch, each of the participants was asked to explain his existing or proposed system in about 5 minutes. TCH was first and there were approximately 8 people in all who spoke. Three of the formats were so similar that they could be considered identical (all variations of the "Lancaster format" finally adopted). The others were different and familiar to most of us. After some random discussion, someone pointed out that the formats which are based on the 11 unit teletype code (standard asynchronous communication) would not need a computer to encode and decode them. Within a matter of minutes, a vote was taken and the decision made that the standard would be asynchronous, 8 bit character oriented data. This of course essentially eliminated the HIT format and the TCH format from further consideration. We also realized that because of the inherent speed sensitivity of the asynchronous format and the fact that the Lancaster method is the only reasonable way to recover a speed tracking clock for a UART that the eventual outcome would in fact be the Lancaster method.

The first part of the evening session was devoted to other possible modulation methods that would be speed tolerant and still give an 11 unit asynchronous character structure. TCH still advocated the single pulse/double pulse method of encoding zeroes and ones but in the end endorsed Harold Mauch's position paper detailing the Lancaster method. The remainder of the evening and the Saturday morning session was spent formalizing the results and discussing other issues such as preambles, headers, checksums, interblock gap lengths and garbage rejection during gaps. Unfortunately due to lack of time and energy no decisions were reached on most of these data formatting issues. Thus, at this time, the BYTE standard consists of a method of putting 8-bit bytes on tape; essentially a paper tape analog. As a result, there is still some work to be done before hobbyists can freely exchange anything except ASCII text.

In interpreting the results of the conference it is important to remember that it is an interchange standard and not necessarily the only acceptable or even the best way to store data. After all, paper tape is the interchange standard for minicomputers but most all serious data storage takes place on some other medium. The BYTE standard does not seem to have slowed the development and introduction of new cassette interfaces. However, compatibility with the BYTE standard, perhaps by flipping a switch, will become an important interface attribute.

CLUBS ETC.

Each issue we will publish pertinent information about new computer clubs and significant changes in the status of old ones.

LONG ISLAND, NEW YORK
Long Island Computer Association
Contact - Jerry Harrison, Chairman
36 Irene Lane E.
Plainview, NY 11803

SOUTH MIAMI, FLORIDA
South Florida Computer Group
Contact - Terry Williamson
P.O. Box 430852
So. Miami, FL 33143
305/271-9909

WISCONSIN
Wisconsin Area Computer Hobbyists
Contact - Don Stevens
P.O. Box 159
Sheboygan Falls, Wis. 53085

DETROIT, MICHIGAN
Tenative formation (Dearborn, Dearborne Heights, Detroit)
Contact - Robert Tater
8476 Nightingdale
Dearborn Heights, MI 48127
313/279-0099

PITTSBURGH, PENNSYLVANIA
Pittsburgh Area Computer Club
400 Smithfield St.
Pittsburgh, PA 15222
Contact - Eric S. Liber
1156 Pennsbury Blvd. N.
Pittsburgh, PA 15222
Officers: Eric S. Liber, President
Fred Kitman, Secretary-treasurer

NORTH READING, MASSACHUSETTES
Alcove Computer Club
Contact: John P. Vulla, President
230 Main St.
North Reading, Mass. 01864

NEW YORK, NEW YORK
N.Y. City Micro Hobbyist Group
Contact - Robert Schwartz
1E, 375 Riverside Dr.
New York, NY 10025

MIAMI, FLORIDA
Miami Computer Club
Contact - John Lynn
13431 SW 79TH ST.
Miami, FL
305/271-2805

It is about time that the Raleigh, NC area had a hobby computer club. With this aim TCH is sponsering a "seed" meeting. It is not our intent to run a club but merely to start one, so come with organizational ideas in hand. The meeting will take place at the Plantation Inn on US highway # 1 approximately 2 miles north of Raleigh. Please park and enter at the rear of the main building. The get-together will start at 1:00 PM Sunday, March 14 and will last until folks get tired. For the sake of those who want to find out what hobby computing is about TCH will have both its 8008 system and an Altair on demo. The graphics display, cassette interface, and floppy disk will also be demonstrated.

NOTES ON TCH

Starting with this issue all mailing labels will contain your subscription number and the issue with which your subscription will expire. This is possible because all subscription records are now maintained on floppy disk. In the past records had been a combination of flip-file cards, two paper tape formats, and cassette tape. Now that the disk system is implemented all address changes to date have been verified and entered. If your newsletter is incorrectly addressed and you have not changed address in the last two weeks, then we missed your change of address, please resubmit it.

Effective immediately rates for back issues and foreign subscriptions are increasing. As you might suspect this is partially due to increased postal rates. The other factor is that TCH was originally planned for 1 ounce editions, but recent editions have consistantly been larger. Regular subscriptions in the U.S.A will not increase bacause they are mailed third class where-as the back issues and foreign cannot be. This change affects only remittances from this date foreward. Items already paid for will be shipped at the original price. New rates for backissues and foreign are shown below:

Backissues in U.S.A., Canada, Mexico	$ .65
Backissues foreign, surface mail	$1.00
Backissues foreign, air mail	$1.50
12 issue subscription U.S.A.	$6.00
12 issue subscription Canada, Mexico	$7.50
12 issue subscription foreign, surface	$11.50
12 issue subscription foreign, air	$16.00

Cassette interface boards, 8080 wirewrap boards, and regulator kits are still being shipped. Some orders for the 8080 wirewrap board have been delayed however. This is because TCH was faced with back orders from both our vendors on the heatsinks for the regulator kits. Hopefully this is now cleared up. Also cassette ROM programming and IMP-16 documentation is still being offered.

TCH is still seeking articles from outside authors. So far only two articles have been submitted (one of them was published last issue), but many people have inquired about writing for TCH. Rather than send out a flock of letters to potential authors, the needed information will be presented right here.

What is TCH looking for? General interest articles about both software and hardware. General interest means that it is not about your personal unique system or device which no one else would care to duplicate. Also articles which primarily describe some company's product are not suitable for TCH. We will leave those to the big guys who have more space to fill. What does that leave? Plenty! Stories about the application of readily available devices such as joysticks, oscilloscopes, TV sets, electronic music devices, tape recorders, and miscellaneous nifty IC's. Programs for any of the common microprocessors are of course interesting to many people.

What is it worth and how should it be submitted? TCH will pay $20 per page as printed for material. This includes drawings, listings, and photos. Material submitted should be typed or neatly written. If you must send your only copy of something, please say so or it may not be returned. It is desirable for programs to be accompanied by a flowchart. Any pictures should be black and white. If any editing other than grammatical corrections is needed, the author will be notified before publication, therefore please include your phone number.

What about advertising? TCH is now accepting paid advertising. Four formats are offered. Full page (7.5" X 10.5"), half page (7.5" X 5" or 3.75" X 10.5"), and quarter page (3.75" X 5") are available and cost $70, $40, and $25 respectively. This price is for "camera ready" copy. Ads can be set at extra charge. TCH reserves the right to reject any ad and prices are subject to change as our subscriber base of 2,300 continues to expand. For further details write to: TCH, Box 295, Cary, NC 27511.

CONTINUED FROM 1

Normally the head surface protrudes through one of the slots in the disk envelope and is held at a barely grazing distance from the disk surface. To read or write, a pressure pad is lowered to the other side of the disk directly opposite the head to press the magnetic coating against the head. All drives use the guts of a relay to move the pressure pad back and forth. The so-called head load and unload then is really pad load and unload and is accompanied by a resonant clank. The Pertec FD-400 is an exception since both the head and the pad move towards each other during head loading. Head load time is in the range of 10 to 50 milliseconds.

Another lamp-photocell assembly in the drive detects the passage of index or sector-index holes. A simple timing circuit is required to tell the difference between the index hole and the sector hole. Some drives may require readjustment of the photocell amplifier gain in order to resolve the closely spaced holes.

All floppy disk drives employ what we call a "flux transition interface" for getting data bits in and out of the drive. For our purposes at this time a "flux transition" has exactly the same meaning as "pulse" did in describing the TCH audio cassette interface (see Vol. 1 #5). Data on the disk is represented as an isolated flux transition (pulse) for a ZERO and two closely spaced flux transitions for a ONE. Unlike the tape however, the spacing between individual bits is not allowed to vary. Using numbers, the bit spacing is 4uS and the transition-to-transition spacing for a ONE is 2uS. When writing, the controller must supply properly timed pulses to the drive which translates them into flux transitions on the disk surface. When reading, the drive picks up the flux transitions, converts them to pulses, and sends them back to the controller with the original timing nearly intact. This data coding method permits over 300,000 bytes to be recorded on a single diskette. Another method, called Miller encoding or MFM, can double the storage capacity and data rate if the drive is capable of pulse timing errors of less than 1uS. This "double density" technique will be explored more thoroughly in a future article.

Most of the drive manufacturers offer numerous extra-cost options for special applications or to simplify controller design. The features that have been discussed are present on the base models of all manufacturer's drives.

Now we come to the sticky problem of data format on the disk. Actually everyone agrees on the basic record format; some leading ZEROES for synchronization, a data ID pattern, useful data (fixed length), CRC characters, and some trailing ZEROES. The difficulty arises because the data capacity of a full track which is about 4K bytes is inconveniently large. Two fundamental methods of breaking up a track into shorter records called sectors have evolved. The differences relate to how the desired sector is located without having to read or write all of them.

The IBM method is called "soft sectoring" and utilizes the one hole pre-initialized diskettes. Each sector consists of an "ID record" followed by a data record. The ID record contains the sector number of the following data record. When a read or write is to be performed, the disk controller logic must search for the ID record that matches the desired sector number (the sectors are not necessarily in order) and then read or write the following data record. The ID and data records are distinguished by a special byte called an address mark. Address marks consist of a peculiar pulse pattern that does not conform to the usual double frequency method of data encoding. A special decoder is required that can recognize these odd pulse patterns (there are actually 4 different ones) and reliably distinguish among them. Using this sectoring scheme there is room for 26 sectors on a track. Additionally, a track is reserved for an index, two are reserved for alternates (in case of a damaged disk), and one is just plain reserved. The total number of useful data bytes is thus 73*26*128 or 242,944 bytes. It should be noted that actual data exchange with IBM equipment requires far more than adherence to the physical data format; the data itself must be formatted and indexed according to a complex set of "logical format" rules.

Use of the 33 hole diskettes is an alternative sectoring method. The individual sectors are delineated by pulses from the photodector in the disk drive. Determining when the desired sector has been reached is simply a matter of counting pulses starting at the index. A sophisticated controller can maintain a continuous count so that at any point in time it knows what sector is coming up next. Records are simply written and read between the two sector pulses that define the desired sector. With hard sectoring, 32 sectors of 128 bytes each will fit on a track for a total capacity of 77*32*128 or 315,392 bytes. A potential drawback of hard sectoring apart from non-IBM compatibility is a lack of "sector address verification" preceeding a read or write. In other words, a disk drive malfunction such as missed stepping motor pulses or mis-counting of sector pulses may cause reading or writing of incorrect sectors without an error indication. A simple solution which has been found to be effective is to initialize the CRC register to the track and sector number before reading or writing instead of initializing it to zeroes. Then if an incorrect sector is read, a CRC error will be signaled. A read before a write will at least assure that the head has been positioned at the correct track.

TCH has chosen the hard sectored approach for a floppy disk interface. Actually, anything else could hardly be called "super simple". The main problem with the IBM format in a mainly software driven interface is the limited amount of time between the ID record and the following data record. There is simply not enough time for software to decode the bit stream and decide if the desired sector is next before it arrives. Also, IBM compatability precludes consideration of double density data recording.

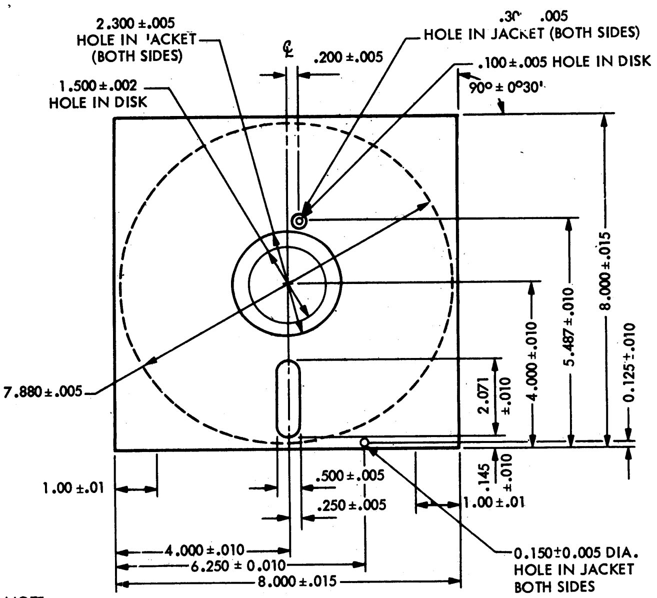
Why are commercial floppy disk controllers so expensive? The typical controller operates almost totally automatically. The CPU simply tells it what track number, what sector number, and a starting address in memory and the controller takes care of the rest via a direct memory access interface. Step counts are computed, sectors are counted (or recognized), data ID's are recognized, CRC's are computed and checked, data is serialized and deserialized, and memory addresses are counted. Additionally, the commercial jobs sometimes have elaborate error recovery logic built-in such as automatic retry. Also, multiple drives can be handled, usually with simultaneous stepping and sector counting on several drives. IBM compatible controllers often include logic for initializing diskettes as well as reading and writing and some can handle both IBM and hard sector formats. What this adds up to is a lot of IC packages, as many as 200 for an IBM compatible unit. For comparison, count up the number is IC's in an Altair or even a NOVA or LSI-11. Although some of the controller functions can be handled by a microprocessor, many of them occur at such high speed that they must be handled by hardware. Of course, these commercial controllers offer very high performance such as constant high transfer rate (reading and writing consecutive sectors) and very little load on the host processor.

How do we plan to reduce the controller complexity to only 27 packages? The obvious answer is to do everything possible in software. For example, software will issue step-in and step-out pulses to the head motor and keep track of which track the head is on. Software will wait for the index pulse and count sector pulses until the desired sector is reached. Software will serialize and deserialize the bit stream and calculate the CRC in much the same manner as was done in the audio cassette interface. Software will also control loading and unloading of the head. Hardware is still required for some functions. Encoding and decoding the double frequency pulse train, distinguishing index from sector, and starting and stopping data transfer immediately at sector boundaries are major hardware functions. The only high-speed operation is transferring the bits to and from the disk at 250,000 per second. Two 2102 memory chips will be used as a buffer between the fast disk transfer rate and the slower CPU/program transfer rate. All timing will be handled by the interface making it CPU speed independent. Believe it or not, the interface will require only one input port and one output port with 4 bits used in each!

What performance characteristics of the floppy disk are sacrificed with the super simple interface? The primary one is transfer rate. An overall average transfer rate of one sector per disk revolution can be expected which is about 750 bytes per second. A special high performance routine may be able to achieve up to 3000 bytes per second. All of the other desirable characteristics are maintained however, especially the random access and update capability. An advantage of the software driven interface is greater reliability; freedom from infrequent and often puzzling disk controller malfunctions.

How useful is a floppy disk system without a "disk operating system" monitor program? Conceptually, a floppy disk is simply a collection of 2464 individual records of 128 bytes each. Each record is addressable much like a memory location. All kinds of dedicated applications such as those described earlier can be easily written without even knowing what a disk operating system is. The user would develop storage and indexing techniques that are best suited for his application. Take for example the storage method used in our floppy based mailing list program. There are 1000 subscribers on a disk and two sectors allocated to each subscriber. To find a subscriber, the program simply computes ((Sub. No.) MOD 1000)*2+126 to get a composite sector number. The track is the quotient when the composite is divided by 32 and the sector is the remainder.

The super simple interface along with the 400 byte support software can also be used with existing disk operating systems. All that generally needs to be done is to locate the subroutine that actually handles reading and writing on the disk and replace it with calls to the support package. More extensive modification is required if the original disk system had a different sector size or different number of sectors per track.



NOTE:
ALL DIMENSIONS SHOWN IN INCHES

FIG. 1 DISKETTE PHYSICAL DIMENSIONS

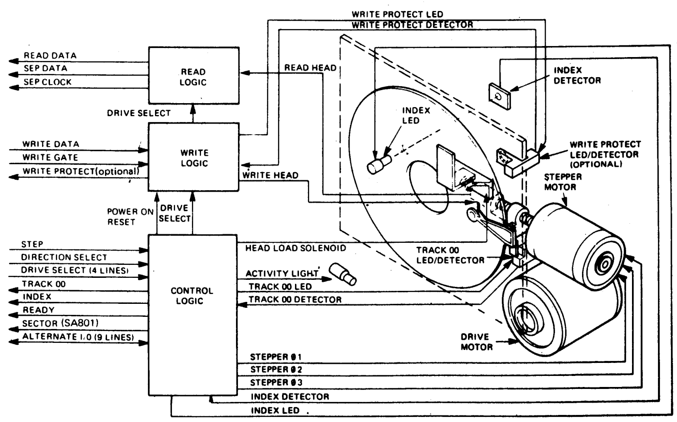


FIG. 3 TYPICAL FLOPPY DISK DRIVE

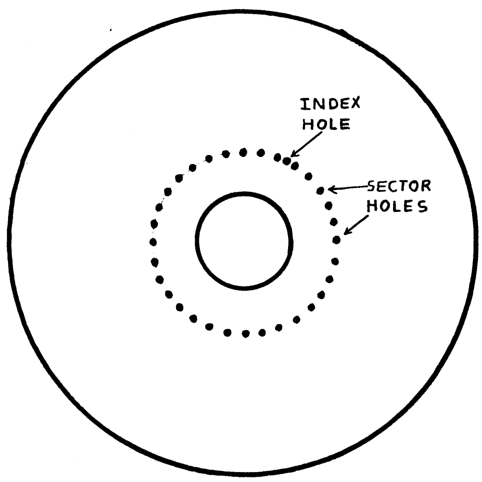


FIG. 2 HARD SECTORED DISKETTE

In issue number 10, the design, construction, operation, and programming of the floppy disk interface will be described in our usual level of detail. Also there will be a listing of the floppy disk routine necessary to operate the interface. In order to formulate support plans beyond that point, we would like to have some reader feedback. In particular, is there interest in PC boards and PROM programming for the floppy disk interface as there was for the TCH audio cassette interface? A simple postcard will do. We intend to remain the best publication for the serious computer hobbyist if not the biggest or most regular.

Following is a listing of the major floppy disk drive manufacturers. With one exception, a TCH staff member has seen a sample drive from each of the companies listed.

FLOPPY DISK DRIVE SUPPLIERS

Orbis Systems
14251 Franklin Ave.
Tustin, CA 92680

California Computer Products
2411 W. LaPalma Ave.
Anaheim, CA 92801

PERTEC, Peripheral Equipment Div.
9600 Irondale Ave.
Chatsworth, CA 91311

Control Data Corp.
P.O. Box 0
Minneapolis, MN 55440

Shugart Associates
435 Indio Way
Sunnyvale, CA 94086

SYCOR
100 Phoenix Dr.
Ann Arbor, MI 48104
(We have not seen this drive)

Remex
1733 Alton St.
Santa Ana, CA 92705
(These people resell Orbis drives, may be easy to deal with)

DISKETTE SUPPLIERS

DYSAN Corporation
2388 Walsh Ave.
Santa Clara, CA 95050
(We have gotten excellent service from these people)

AN IMP-16 MICROCOMPUTER SYSTEM PART 2 by Hal Chamberlin

Things are changing so rapidly that the first paragraph of these installments will have to be devoted to news items. Poly-Paks no longer has IMP-16 sets. We don't know if IEU still has them or not. However all surplus IMP-16 chip sets come through Godbout so perhaps some letters will persuade him to sell them directly. Of course all National distributors have some; TCH has gotten them this way for $160. The real problem is that they hit the surplus market too early. We got some more data on the "power math" CROM. Basically it provides instructions for operating on 32 bit binary fractions (mantissas) such as 32X32 add, subtract, multiply, divide, and normalize. The user need only code exponent handling and the result is a floating point package with 32 bit mantissas (10 decimal digits) and 16 bit exponents (10**10000 anybody?) with a 100 uS add time and a 600 uS multiply time. The bad news is that "power math" and the extended CROM share some op-codes so they cannot normally be used together. There is a way to enable one or the other using a status flag however (status flags can be saved during interrupt). Implementation of the scheme requires the use of a 74LS260 in place of the 74LS54. PC layout of the CPU board is planned but some readers couldn't wait and have already started to wire-wrap CPU boards. At least 3 TCH staff members will be building IMP systems and at least one of them will have a floppy disk so software support will not be lacking toward summer. Quick note: do not buy plain 2107 4K RAM's for this system! They have a different pinout, are very slow, and in a word, totally obsolete. TMS4030, TMS4060, 2107A, and 2107B are all fine as well as most gradeouts. The author has a limited supply of TMS4030-ZA0248 4K RAMS tested for operation in this system for $7.50. An error was made in the parts list for the memory board. Rather than three 7404's, it should be two 7404's and a 7440.

Now with the news out of the way, let us take a top-down approach to describing the PUNIBUS controller. The bus controller runs continuously, non-stop, from power-up to power-down crunching out 1.43 million cycles per second or one cycle every 700NS. All memories in the system likewise operate at this cycle rate. Each cycle is awarded on the basis of priority to one of 7 possible requesters. The highest priority requester is the CPU. Below the CPU are 5 direct memory access (DMA) devices. The lowest priority requester is the memory refresher which is always requesting bus cycles. Thus if the system is idle, that is, CPU halted and no DMA activity, all of the bus cycles are being awarded to the memory refresher. During operation, cycles that are unclaimed by the CPU or DMA are also awarded to the refresher. The PUNIBUS controller always generates the timing signals necessary for data transfer regardless of which requester controls the particular cycle. Thus DMA devices in the system don't have to generate any timing of their own, instead they just sit and respond to control signals issued by the PUNIBUS controller.

Any device interfaced to the bus that is not a possible DMA requester is expected to behave as if it was a memory. At the beginning of every bus cycle a 16 bit address is established. This address specifies either an actual memory location or a peripheral device register. There are only two types of bus cycles; a read cycle and a write cycle. During a read cycle, data is read from a memory or peripheral register into the CPU or DMA device. During a write cycle, data is written from the CPU or DMA device into a memory or peripheral register. The CPU or DMA device awarded the cycle determines whether a read or a write cycle is to be performed. The memory refresher, of course, always does read cycles. Undefined operations such as addressing non-existant memory or writing into a read-only peripheral register are not harmful and function as NO-OP bus cycles.

Figure 1 shows the timing relationships of the PUNIBUS. Although actual times in nanoseconds are given, it is important to note that correctly designed interfaces to the bus will work properly even with considerable variation in the timing details as long as the basic relationships are retained. This allows flexibility to change the details to accomodate other CPU's such as a bipolar IMP or a down-spec chip set without obsoleteing memory and peripheral designs.

As can be seen, a bus cycle starts with the signal BUS ADDRESS ENABLE (BAE) going high and terminates when it goes high again for the next cycle. Actually though, some preparation takes place toward the end of the previous cycle. An internal "priority strobe" is generated which causes the BUS REQUEST (BR) lines including CPU and refresh request to be examined to determine who will get the next cycle. The determination is made and the three bit grant code of the winning requestor is placed on the BUS GRANT (BG) lines immediately before the cycle commences with BAE going high. At this time the one requestor whose code is on the BG lines is expected to gate a 16 bit address onto the BUS DATA (BD) lines as long as BAE is high. Any BD lines not specifically driven will assume a ONE level because of pullup resistors. If a write cycle is to be executed, the BUS WRITE REQUEST (BWR) line should be pulled down during BAE time, otherwise a read cycle will be automatically assumed. This address phase of the cycle is identical for both read and write operation.

After the address phase we have the data transfer phase which is different for read and write cycles. In the case of a read cycle, the bus controller generates two signals, BUS DATA OUT ENABLE (BDOE) and BUS DATA OUT STROBE (BDOS) which control the data transfer from memory or peripheral register to CPU or DMA device. BDOE first goes high to cause the addressed memory or peripheral to gate its data onto the BD lines. BDOS is bracketed by BDOE and can be used to strobe data from the bus into the CPU or DMA device's data register on its trailing edge. The timing of this pair of pulses is chosen to allow memories sufficinet access time and to allow the IMP-16 chip set to grab the data directly from the bus with no intervening latches.

During the transfer phase of a write cycle, BDOE and BDOS remain inactive while BUS DATA IN ENABLE (BDIE) and BUS WRITE ENABLE (BWE) control the data transfer from CPU or DMA to memory or peripheral. BDIE becomes active first causing the CPU or DMA device to gate the data to the written onto the BD lines. BWE which is bracketed by BDIE then becomes active causing the memory or peripheral to accept and store data from the bus. The timing shown for these signals was chosen to be compatible with the 4K RAM's used in this system.

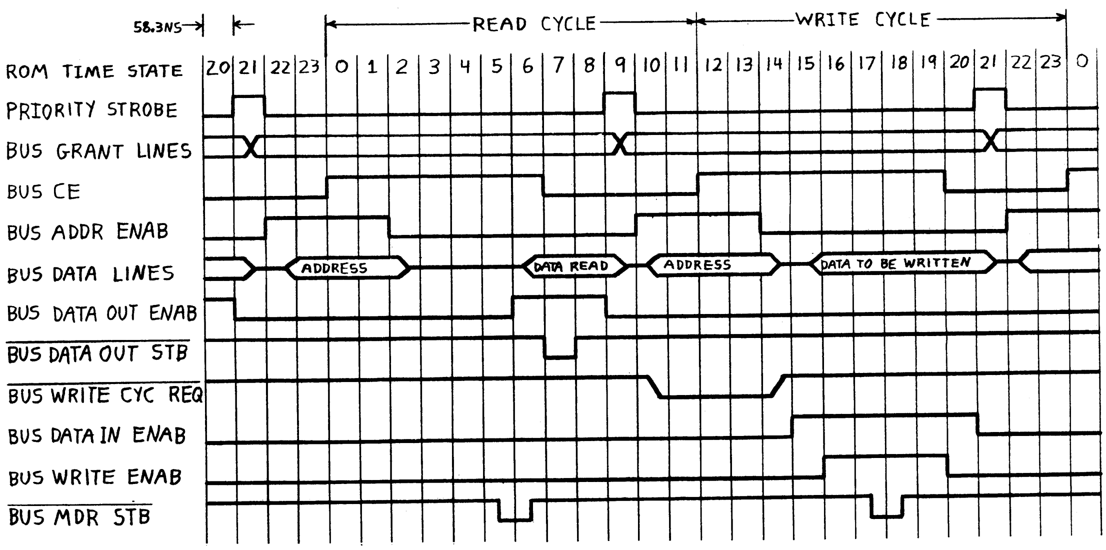


FIGURE 1. PUNIBUS TIMING RELATIONSHIPS

The responsibilities of a memory board or peripheral interface are quite simple. During the address phase of each cycle, pertinent information about the 16 bit address on the BD lines must be latched on each interface board. Generally, a memory interface using 4K RAM's need only latch a single bit since the RAM chips have built-in latches for address and chip select. The single bit needing a TTL latch simply indicates whether the board was addressed or not. Likewise, a peripheral register can decode its address directly from the BD lines and use a flip-flop to remember if it was addressed. In either case, the leading edge of BUS CE is used for address strobing since it always occurs when the address is valid. Once a memory or peripheral has latched the fact that it was addressed, it either sends its data out if it sees BDOE or accepts new data in if it sees BWE. Thus memories and peripheral registers are passive, merely responding to bus signals as they occur.

Four "convenience" signals are provided on the bus. One which has already been mentioned is BUS CE. Memory boards using 22 pin 4K RAM's can simply amplify this signal to NMOS levels and apply it to the Chip Enable clock input of the RAM chips. Its function within the RAM is to start up the memory cycle and also strobe the on-chip address and chip select latches. Another signal provided specifically for memory boards is BUS MDR STROBE. Its purpose is to strobe the data out latches on the memory board when data out from the 4K RAM's is valid. Some unfortunate timing constraints on both TMS4030 and 2107 type RAM's require latches to hold the data after it disappears from the RAM outputs. Although BDOE could have been turned on earlier with the leading edge strobing the latches, excessive noise generation would have resulted. BUS I/O ADDR is a signal that goes low whenever the binary value on the data lines is between FF00 and FF7F hexadecimal. This range of addresses is normally assigned to peripheral devices. Use of this signal in decoding I/O addresses can save a 9-input AND gate equivalent on each interface card. BUS CLOCK is provided as a convenient high frequency clock with .005% accuracy. Its frequency is such that when put through a 16 bit divider, the resulting frequency is middle C, 261.625 Hz. Additionally, 12 cycles of this clock make up one bus cycle whose length is actually 699.88NS.

Two signals are involved with power-on reset and console reset. The POWER OK bus line should be pulled low by an external circuit associated with the power supply when all supply voltages are present and stabilized. This circuit should also be connected to the console reset push button so a power on sequence can be simulated without loosing memory contents. A simple delay circuit is shown in figure 2 which functions quite well. Alternatively, a true power monitor can be built using zener diodes to sense when the supply voltages are actually present. BUS RESET is generated in response to POWER OK by the CPU board. It resets the CPU and should reset all peripheral interfaces to a safe, idle condition when it goes low. It has no effect on the bus controller or memory refresher however.

The interrupt system uses the very simple software polling technique described elsewhere in this issue. The BUS INT REQ (BIR) line is a wire-or line with pullup resistor which is pulled low by any device that wants to request an interrupt. The CPU responds, provided its master interrupt enable is on, by calling a subroutine at 0001 and simultaneously turning master interrupt enable off. After saving status, the program can look at the status register of each possible interrupting device to determine who is requesting. This search can be as fast as 9.8uS per device with proper use of the SKAZ (SKip if And is Zero, ANDs addressed memory location with a register and skips the next instruction if the result is zero) instruction. The device service routine then turns off the interrupt request for that particular device and turns master interrupt enable back on. Priority in the case of simultaneous interrupts is determined by the order of scanning. Nested interrupts can also be programmed. Thus the interrupt system essentially works like that on a PDP-8. The usual interrupting device interface also has an interrupt enable for each device making non-interrupt I/O programming possible if desired. More details on I/O interfacing and interrupts will be given in part 4.

Figures 3 and 4 show the timing generator and bus controller. Since this circuitry is on the CPU board, some CPU circuitry has encroached which will be described in part 3. See TCH #2 if any of the logic gate symbols are confusing. You will note that inputs always enter from the left of a drawing and outputs leave at the right. All signals going offpage are given a name and should mate with similarly named signals on the other pages. If an offpage signal has a number on it, it goes to the CPU board edge connector. If the number is 46 or less, it is a bus signal and is available at the same pin number of any board in the system. Some signals shown in figure 3 and 4 will not be mated until part 3.

The heartbeat of the system is the 17.145893 mHz oscillator in figure 3. Its output drives a hex latch and is buffered to drive the BUS CLOCK line. The latch and two 32 word by 8 bit bipolar PROM's make up the bulk of the timing generator. As can be seen, 6 of the 16 PROM outputs go to the 74S174 hex latch and 5 of these are fed back to the PROM address inputs. The result is that every cycle of the 17 mHz clock causes the PROM-latch combination to take one step in a programmed sequence. Using the PROM pattern in figure 5, this sequence is 24 steps long and takes 1.4uS to step through thus matching the minimum IMP-16 microcycle time. In order to avoid glitches at the PROM outputs when the address changes, the sequence of addresses has been chosen such that only one address line changes at a time. Figure 6 shows the PROM pattern in time sequence rather than address sequence. The 8 addresses not normally used all point to time state zero to avoid a possible lockup condition. The sequence of addresses was also chosen so that a decoder could be used to generate the 4-phase non-overlapping clock needed by the IMP-16 chips from 3 of the address bits.

The remaining 11 PROM bits are the various system timing signals. Those prefixed RAW require additional gating before being used; the others are ready to go. BUS MDR STROBE goes through the latch to effect an additional 30NS delay. The purpose of the flip-flop connected to the 4-phase decoder is to insure that the CPU starts up on phase 1 after a system reset. Although 8223 PROM's with pullup resistors are shown, a tri-state PROM such as an 82123 can be used without the resistors.

System reset and power up control are handled by the two 7413 Schmidt triggers and other discrete circuitry at the bottom of figure 3. The first 7413 gives a snap-action response to BUS POWER OK which may be a slowly changing signal. The R-C network and second 7413 provide a signal that tracks BUS POWER OK but with a several millisecond delay. This delayed signal, after inversion, becomes BUS RESET. The transistors apply -12 volt power to the IMP chips when bus power is OK and remove it otherwise. BUS RESET also controls application of the 4-phase clocks to the microprocessor. Thus the timing relationship between power application and removal and clock application and removal is such that the IMP is properly initialized.

The logic in the upper third of figure 4 modifies some of the timing signals from the PROM according to bus cycle type; read or write. Flip-flop 1 samples BUS WRITE REQUEST at the leading edge of BUS CE and retains the read/write decision for the remainder of the cycle. The network at the top of the page consisting of a 7432 and 7410 delays the fall of BUS CE by 50NS during write cycles. It behaves as a simple inverter during read cycles. Lengthening BUS CE during write cycles only provides improved timing margins for writing into 4K RAM's without unnecessary power dissipation during read cycles. The gates on BDOS and BDOE gate these signals on for read cycles and off for write cycles. Likewise, BDIE and BWE are gated on for writes and off for reads.

The network starting with the 74LS21's is a partial address decoder. If the address on the bus is between FF80 and FFFF, flip-flop 2 is set indicating that the on-board bootstrap ROM has been addressed. If the address is between FF00 and FF7F, BUS I/O ADDR is activated to inform peripherals that an I/O address is on the bus.

The next group of logic is the cycle request and grant priority logic. Gating for CPU cycle request and the 5 DMA request lines go into a hex latch that is strobed by PRIORITY STB near the end of each bus cycle. The latches are necessary to hold the input to the priority encoder constant throughout the next cycle. The 74148 determines the highest priority input present (active low, A is highest, H is lowest priority) and outputs a 3 bit code identifying that input. The G input is not used in this drawing but could be used for a sixth DMA request along with a latch. The H input is refresh request which is always present.

The bottom of figure 4 is the refresh logic for all dynamic memory in the system. A 74LS20 detects the coincidence of refresh grant (111) and BAE which indicates that the refresh address should be placed on the bus. The output thus enables an 8097 which gates the 6 significant refresh bits onto the bus. The other 10 bits assume a logic 1 and the bus controller assumes a read cycle. When the 8097 is gated off again, a 6 bit counter made from a 7474 and a 7493 counts up one notch in preparation for the next refresh cycle. Two 8556 tri-state counters could have replaced the 7474, 7493, and 8097 used here but they were too hard to get to justify their use.

That concludes the description of the bus controller. Everything else in the system is just a collection of bus interfaces. Although the remainder of this series will be specifically concerned with IMP-16 interfacing to the bus, the basic concepts and bus structure can be used with any microprocessor. In fact, an essentially identical bus system was used in the design of a super 8008 system over three years ago.

In the next issue a brief description of the IMP-16 chip set will be given along with the remainder of the CPU board schematic and accompanying discussion.

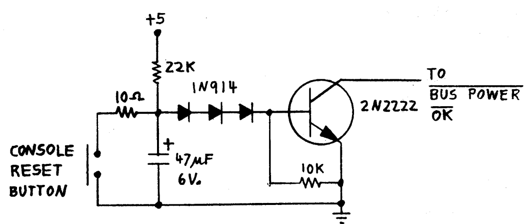


FIG. 2 SIMPLE POWER OK CIRCUIT

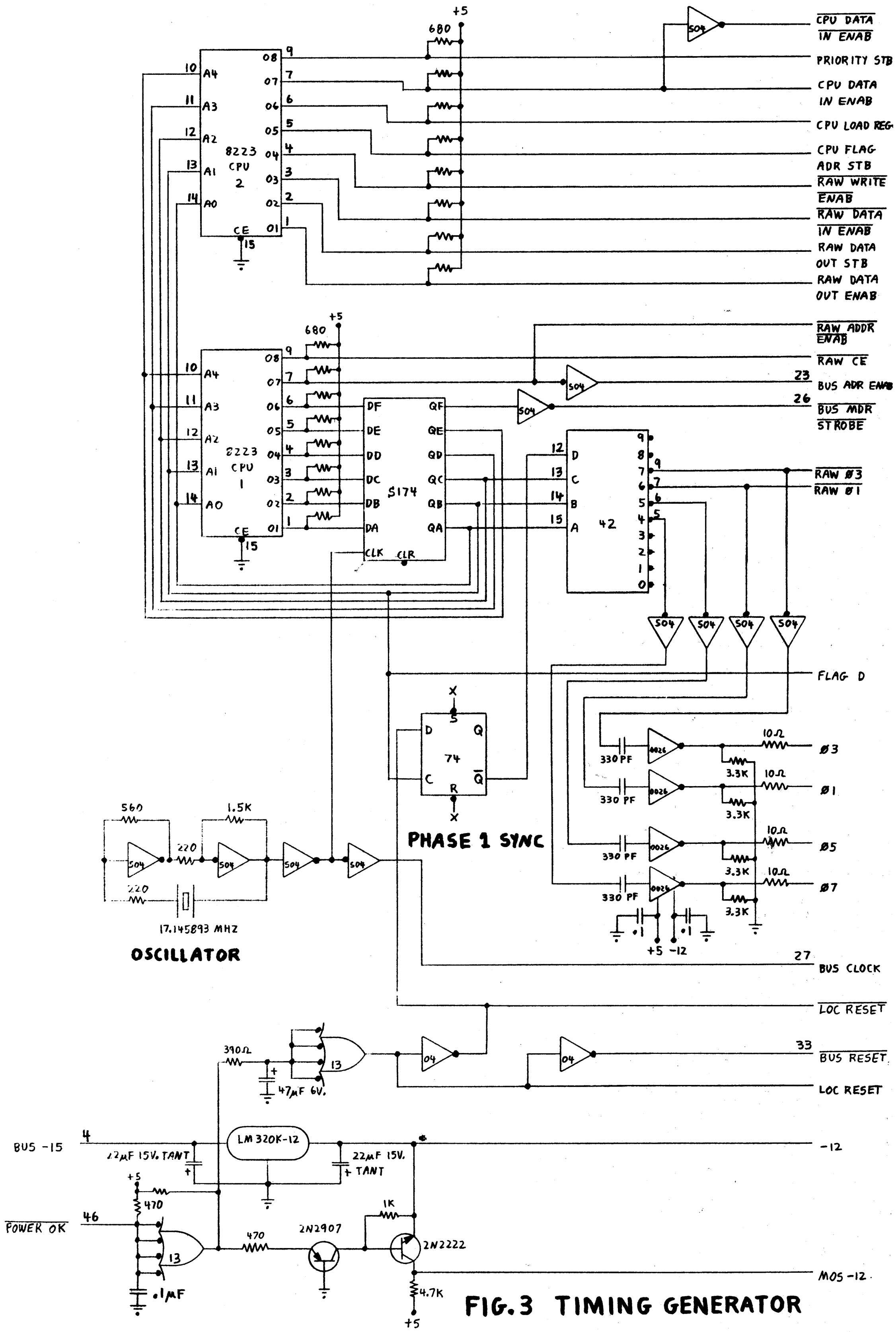


FIG. 3 TIMING GENERATOR

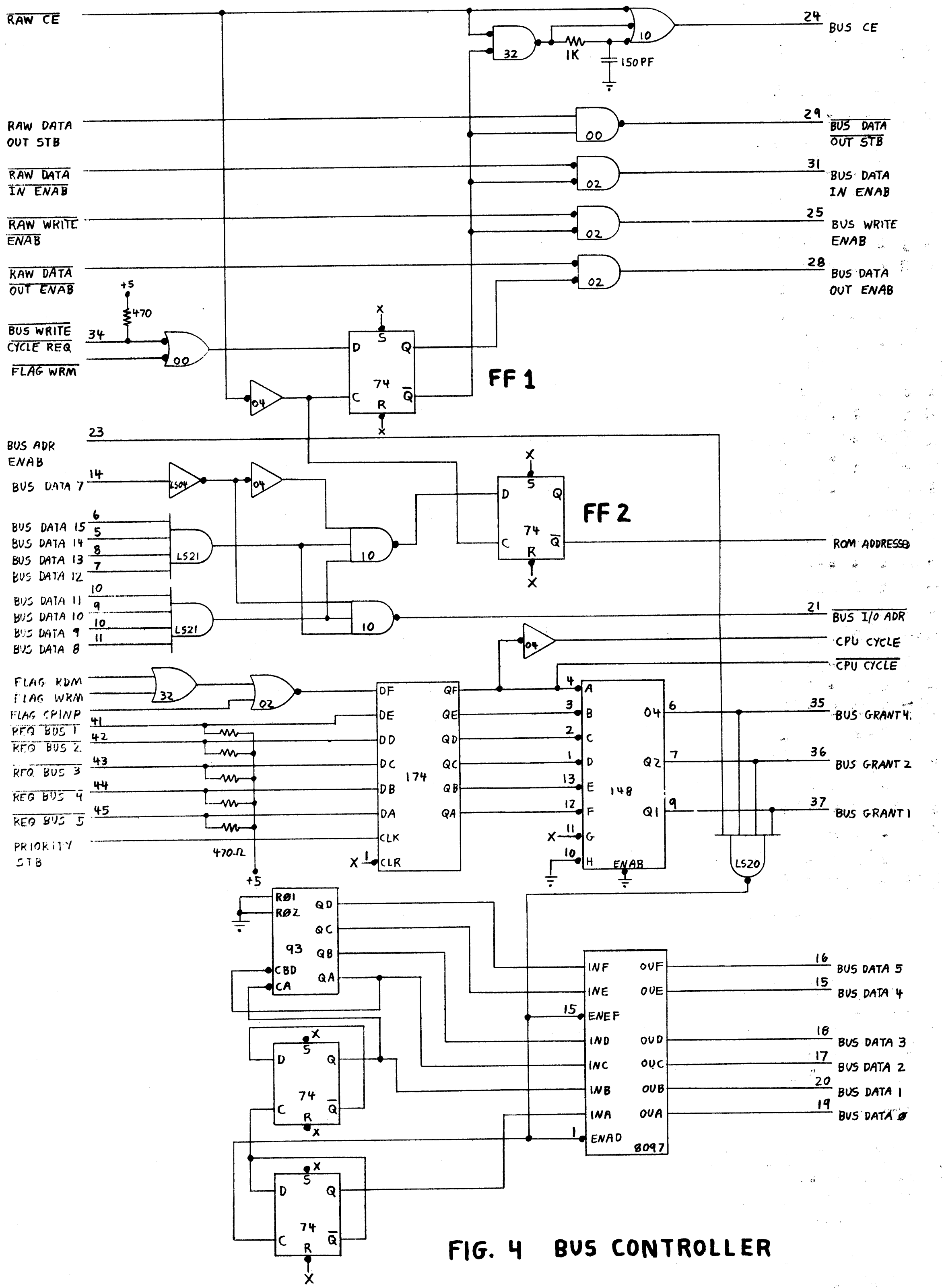


FIG. 4 BUS CONTROLLER

		CPU-1								CPU-2							
ROM ADDRESS	TIME STATE	$\overline{\text{RAW CE}}$	$\overline{\text{RAW ADDR ENAB}}$	RAW MDR STROBE	NEXT ROM ADDR 16	NEXT ROM ADDR 8	NEXT ROM ADDR 4	NEXT ROM ADDR 2	NEXT ROM ADDR 1	PRIORITY STROBE	CPU DATA IN ENAB	$\overline{\text{CPU LOAD REG}}$	CPU FLAG ADDR STB	$\overline{\text{RAW WRITE ENAB}}$	$\overline{\text{RAW DATA IN ENAB}}$	RAW DATA OUT STB	$\overline{\text{RAW DATA OUT ENAB}}$
0	16	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	1
1	15	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1
2	23	1	0	0	0	0	1	1	0	0	0	1	0	1	1	0	1
3	-	X	X	X	0	0	1	1	0	X	X	X	X	X	X	X	X
4	-	X	X	X	0	0	1	1	0	X	X	X	X	X	X	X	X
5	14	0	1	0	0	0	0	0	1	0	0	1	0	1	1	0	1
6	0	0	0	0	0	1	1	1	0	0	0	1	0	1	1	0	1
7	7	1	1	0	0	1	1	1	1	0	0	1	0	0	0	1	0
8	17	0	1	1	0	1	1	0	0	0	0	1	0	0	0	0	1
9	10	1	0	0	1	1	0	0	1	0	0	1	0	1	1	0	1
10	-	X	X	X	0	0	1	1	0	X	X	X	X	X	X	X	X
11	9	1	1	0	0	1	0	0	1	1	0	0	0	1	1	0	1
12	18	0	1	0	1	1	1	0	0	0	0	1	0	0	0	0	0
13	-	X	X	X	0	0	1	1	0	X	X	X	X	X	X	X	X
14	1	0	0	0	1	1	1	1	0	0	0	1	0	1	1	0	1
15	8	1	1	0	0	1	0	1	1	0	0	1	0	1	0	0	0
16	21	1	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1
17	-	X	X	X	0	0	1	1	0	X	X	X	X	X	X	X	X
18	22	1	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1
19	5	0	1	1	1	0	1	1	1	0	0	1	0	0	0	0	1
20	20	1	1	0	1	0	0	0	0	0	0	1	0	1	0	0	0
21	13	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1
22	-	X	X	X	0	0	1	1	0	X	X	X	X	X	X	X	X
23	6	0	1	0	0	0	1	1	1	0	0	1	0	0	0	0	0
24	-	X	X	X	0	0	1	1	0	X	X	X	X	X	X	X	X
25	11	1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	1
26	3	0	1	0	1	1	0	1	1	0	0	1	0	1	0	0	1
27	4	0	1	0	1	0	0	1	1	0	0	1	0	0	0	0	1
28	19	1	1	0	1	0	1	0	0	0	0	1	0	0	0	1	0
29	12	0	0	0	1	0	1	0	1	0	0	1	0	1	1	0	1
30	2	0	1	0	1	1	0	1	0	0	0	1	1	1	1	0	1
31	-	X	X	X	0	0	1	1	0	X	X	X	X	X	X	X	X

FIG. 5 TIMING ROMS IN ADDRESS SEQUENCE

		CPU-1								CPU-2							
TIME STATE	ROM ADDRESS	$\overline{\text{RAW CE}}$	$\overline{\text{RAW ADDR ENAB}}$	RAW MDR STROBE	NEXT ROM ADDR 16	NEXT ROM ADDR 8	NEXT ROM ADDR 4	NEXT ROM ADDR 2	NEXT ROM ADDR 1	PRIORITY STROBE	CPU DATA IN ENAB	$\overline{\text{CPU LOAD REG}}$	CPU FLAG ADDR STB	$\overline{\text{RAW WRITE ENAB}}$	$\overline{\text{RAW DATA IN ENAB}}$	RAW DATA OUT STB	$\overline{\text{RAW DATA OUT ENAB}}$
0	6	0	0	0	0	1	1	1	0	0	0	1	0	1	1	0	1
1	14	0	0	0	1	1	1	1	0	0	0	1	0	1	1	0	1
2	30	0	1	0	1	1	0	1	0	0	0	1	1	1	1	0	1
3	26	0	1	0	1	1	0	1	1	0	0	1	0	1	0	0	1
4	27	0	1	0	1	0	0	1	1	0	0	1	0	0	0	0	1
5	19	0	1	1	1	0	1	1	1	0	0	1	0	0	0	0	1
6	23	0	1	0	0	0	1	1	1	0	0	1	0	0	0	0	0
7	7	1	1	0	0	1	1	1	1	0	0	1	0	0	0	1	0
8	15	1	1	0	0	1	0	1	1	0	0	1	0	1	0	0	0
9	11	1	1	0	0	1	0	0	1	1	0	0	0	1	1	0	1
10	9	1	0	0	1	1	0	0	1	0	0	1	0	1	1	0	1
11	25	1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	1
12	29	0	0	0	1	0	1	0	1	0	0	1	0	1	1	0	1
13	21	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1
14	5	0	1	0	0	0	0	0	1	0	0	1	0	1	1	0	1
15	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1
16	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	1
17	8	0	1	1	0	1	1	0	0	0	0	1	0	0	0	0	1
18	12	0	1	0	1	1	1	0	0	0	0	1	0	0	0	0	0
19	28	1	1	0	1	0	1	0	0	0	1	1	0	0	0	1	0
20	20	1	1	0	1	0	0	0	0	0	1	1	0	1	0	0	0
21	16	1	1	0	1	0	0	1	0	1	0	1	0	1	1	0	1
22	18	1	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1
23	2	1	0	0	0	0	1	1	0	0	0	1	0	1	1	0	1

FIG. 6 TIMING ROMS IN TIME SEQUENCE

CLASSIFIED ADS

There is no charge for classified ads in TCH but they must pertain to the general area of computers or electronics, and must be submitted by a non-commercial subscriber. Feel free to use ads to buy, sell, trade, seek information, announce meetings, or for any other worthwhile purpose. Please submit ads on separate sheets of paper and include name and address and/or phone number. Please keep length down to 10 lines or less.

HELP WANTED: I have some core stacks that I wish to lash up to my Altair 8800. I have no info on them except that I believe they are from Burroughs equipment. There are no drivers or sense amps, only core frame and glass diodes galore. Would appreciate hearing from anyone that may be able to help. Stanley D. Davis, RD 1, Stittville, NY 13469

FOR SALE; MITS RS-232 serial I/O board for the Altair 8800 (88-SIOA), assembled and tested, $80. Expander motherboard (88-EC), $8. Processor Technology MB-1 full-width 16-slot heavy-duty motherboard for the 8800, $25. David Richards, 6655 Hill St., El Cerrito, CA 94530, Ph. 415/529-0759

WANTED: Floppy disk drives and matrix printer. Send information including condition and prices to Fred Holmes, 101 Brookbend Ct., Mauldin, SC 29662.

FOR SALE: Wangco model 7 tape drives, NFE model 250 cassette drives (Cybertronics sells an Altair interface board), 4Kx12 memory systems, Tally line printers, paper tape readers, paper tape punch, TMS 2501 NC's, 3002, 3003, 3113, 7491, 1414L, 710, 741. Please inquire and make offer. Also interested in corresponding with hobbyists interested in COSMAC. John L. Marshall, Box 242, Renton, WA 98055

FOR SALE: Tired of waiting for MITS memory boards? I have two MITS 4K dynamic memory boards for sale at the kit price of $195 each. These boards were carefully assembled by an electrical engineer and factory checked. All bits certified. H. S. Corbin, 11704 Isben Drive, Rockville, MD 20852, Ph. 301/881-7571

FOR SALE: BELL 103 compatible modem PC cards. These are field rejects but include documentation. $17.50 postpaid. D. Blevins, 1857 Babe Ruth Ct., San Jose, CA 95132

NEW CLUB: If you live in the metropolitan New Orleans area and are interested in computers, you are invited to join our group. Whether your interest is hardware, software, applications, or just general interest we welcome your input. For further details, please write or call: Emile Alline, 1119 Pennsylvania Ave. Slidella, LA 70458, Ph. 504/641-2360

FOR SALE: ASR 33 in good condition with stand and some extra features. Sprocket feed platten, self contained power supplies and relays for tape reader control - ready for 6 wire computer hookup. Local pickup only. $425.00. Neal Sheffield, Jr., 108 Elmwood Terrace, Greensboro, NC 27408, Ph. 919-275-7720

WANTED: Software for business applications that will run on 8080 micro computers. Will buy, trade, or distribute for costs plus royalities (where required). John Lynn, 13431 SW 79 st., Miami, FL 33183, Ph. 305/271-2805

INTERFACING THE ALTAIR 8800 PART 2

by Hal Chamberlin

Many applications of hobby computers can benefit from the use of interrupts. MITS and a couple of other alternate sources have had "vectored interrupt controller" cards announced for some time but until recently have been unable to deliver. The problem is that the vectored interrupt cards use the Intel 8214 vectored interrupt IC which was announced over a year ago but was not available in volume until now. Although these cards offer many advantages in large systems with heavy use of interrupts and stringent response time requirements, we feel that not enough attention has been given to the interrupt capability built into the basic Altair. This article will discuss the effective use of this "free" resource in the design of custom I/O interfaces. A keyboard interface with interrupt capability will be used as a model to illustrate the concepts presented.

First, let us discuss what interrupts are, what they are good for, and the three popular implementation methods that are used in minicomputers.

Often in a computer system one has a program crunching away in the CPU and an impatient I/O device that occasionally wants attention from the CPU (and, of course, a different section of the program). Many examples come to mind but we will use the case where the CPU is busy drawing on a graphics display (see TCH #1) and the impatient I/O device is a human at a keyboard typing in commands to alter the image. In computer jargon, the routine doing the drawing is called a "background task" and the routine that reads characters from the keyboard and interprets them is called a "foreground" task.

Programming this application on a system without interrupts would involve three major routines. The first is an initialization routine. Its job is to set up the initial display list, initialize the various software flags, and enable the keyboard for the first user command. Normally, the initialization routine is executed only once, just after entering the program. Following initialization, a branch is taken to the draw routine which sits in a loop drawing and refreshing the graphic image from data in the display list. Periodically, such as once per image refresh, the draw routine should test the keyboard to see if anything has been typed in. If so, a branch is taken to the third routine, a command interpreter. Its job is to read the character from the keyboard, process it, re-enable the keyboard, and finally jump back to the draw routine. Processing the character frequently amounts to storing it away in memory, a very quick operation. When a complete command has been stored, character processing would also include execution of the command such as line erase, change scale, etc.

This scheme can be made to work very well but let us look at the possible limitations. First, the keyboard test must be placed at a point in the draw routine that is executed frequently enough to satisfy the person at the keyboard. Once per refresh may not be enough if the image is complex and the typist is fast. On the other hand, testing the keyboard once per line drawn would not be good either because a large percentage of CPU time may be spent simply testing. In some applications, a suitable test point in the background routine may not exist. What is really needed is a way for the keyboard to "interrupt" the draw routine and cause a subroutine call to the keyboard routine. This is, in fact, what interrupt hardware accomplishes.

Now let us look at how a very simple (but completely effective in this example) interrupt scheme for the keyboard would work. First, the keyboard interface would have two control/status flip-flops instead of the usual one. These would be called BUSY and DONE. These flip-flops can be altered by both an OUT instruction and key action and can be read with an INP instruction. Given two flip-flops, there are four possible combinations to represent operating "states" of the keyboard. If both BUSY and DONE are off, the keyboard is in an idle state, i.e., not being used. If a program needs a character from the keyboard, it should set the BUSY flip-flop but leave DONE off. The keyboard would now be in a "busy" state waiting for the operator to press a key. TCH has found that a light-emitting diode connected to the BUSY flip-flop and mounted on the keyboard cover is very effective in informing the operator that a key may be pressed. When a key is finally pressed, BUSY is automatically turned off and DONE is turned on signifying that the keyboard register now has a valid character but the program has not yet read in the character. When the program does read the character, DONE is turned off returning the keyboard to an idle state. We have also found that a small speaker inside the keyboard case connected so that it clicks when the program reads the keyboard gives valuable feedback to the operator, reducing keying errors. Both BUSY and DONE being on simultaneously is a meaningless situation.

Now, how would these two control/status flip-flops be connected to the Altair for interrupts? First, the keyboard only requires service (reading a character) when the DONE flip-flop is on. So to implement interrupts, we would tie DONE to the PINT line (pin 73) on the Altair bus. Whenever the Altair sees the coincidence of PINT and interrupt enable (set with the EI instruction), it will execute a CALL to location 000:070 (split octal notation, see issue #8) and turn interrupt enable off. There must, of course, be an "interrupt service" routine at 000:070 or a jump to one elsewhere.

Using the same display example, we would still have three major routines in a system with keyboard interrupts. The initialization routine would first force the keyboard into an idle state (it may have been left in an undefined state by the previous program) then it would set BUSY thus enabling the keyboard. It would also execute an EI to enable interrupts on the Altair. The draw routine is the same as before but now it doesn't have to test the keyboard flags. When the operator hits a key, BUSY will be turned off and DONE turned on by the keyboard interface logic. DONE being on and the Altair interrupts being enabled will force execution of a CALL to 000:070 when the current instruction is finished. The CALL stores where it came from on the stack and disables interrupts as if a DI instruction was executed. The interrupt service routine at 000:070 should also save status and any registers it uses on the stack before doing anything else. At this point, a character is read from the keyboard, DONE is turned off thus idling the keyboard, and the character is acted upon. After processing the character, the interrupt service routine turns BUSY on to re-enable the keyboard, restores registers and status from the stack, re-enables the Altair interrupt system, and finally executes a RET instruction. The display routine is now executing again and since no status or registers were changed, it is not even aware of the interruption. Of course the display was stopped momentarily while interrupt service was in control but generally the time is so short that the interruption is not noticable.

So far we have discussed a system in which only one device had interrupt capability, namely the keyboard. Most of the fancier uses of interrupts are in systems where several peripherals can interrupt and the programmer desires simultaneous I/O, that is, more than one I/O device running at a time. In addition to the hardware and software previously described, a method must be found to identify which device caused a given interrupt, and to resolve the conflict that exists when two or more interrupts occur simultaneously. In fact, the only real difference between various interrupt systems is in how these two functions are performed.

The most obvious, best, and expensive multiple interrupt scheme is called hardware vectored interrupts. Recall the keyboard/display example in which the keyboard service routine had to be at location 000:070. With hardware vectored interrupts, other devices would cause calls to other locations. In other words, hardware takes care of identifying which device is interrupting, and automatically branches to the service routine for that device. This is in fact what the vectored interrupt cards implement.

A similar system often used in minicomputers is called software vectored interrupts. All interrupts cause the CPU to branch to the same location. A common interrupt service routine then issues an INTA instruction (INTerrupt Acknowledge) which causes the interrupting device to return its device address. Using the device address, the common interrupt service routine can set up an N-way jump to a service routine specific to that device. This method uses less hardware than the previous method but is somewhat slower because of the time taken by the common service routine. Both methods have the advantage that the time between interrupt and entry to the matching service routine (called interrupt latency time) is independent of the number of possible interrupt sources.

What about the case of two or more simultaneous interrupts? In both vector schemes each device is assigned a "priority", usually by how it is wired into the system. When two or more interrupt requests are pending, the device with the highest priority gets serviced. When the service routine turns off the interrupt request in the highest priority device and returns, another interrupt occurs immediately from one of the remaining devices. Eventually all interrupts get serviced, in order of decreasing priority. Some really sophisticated interrupt systems even allow a higher priority device to interrupt the service routine for a lower priority device! This feature is often found in process control systems and is called "nested priority interrupts".

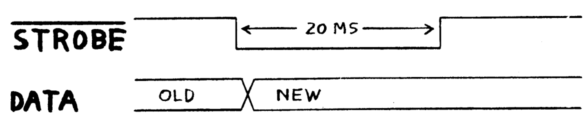


FIG. 1 CLARE-PENDAR KEYBOARD TIMING

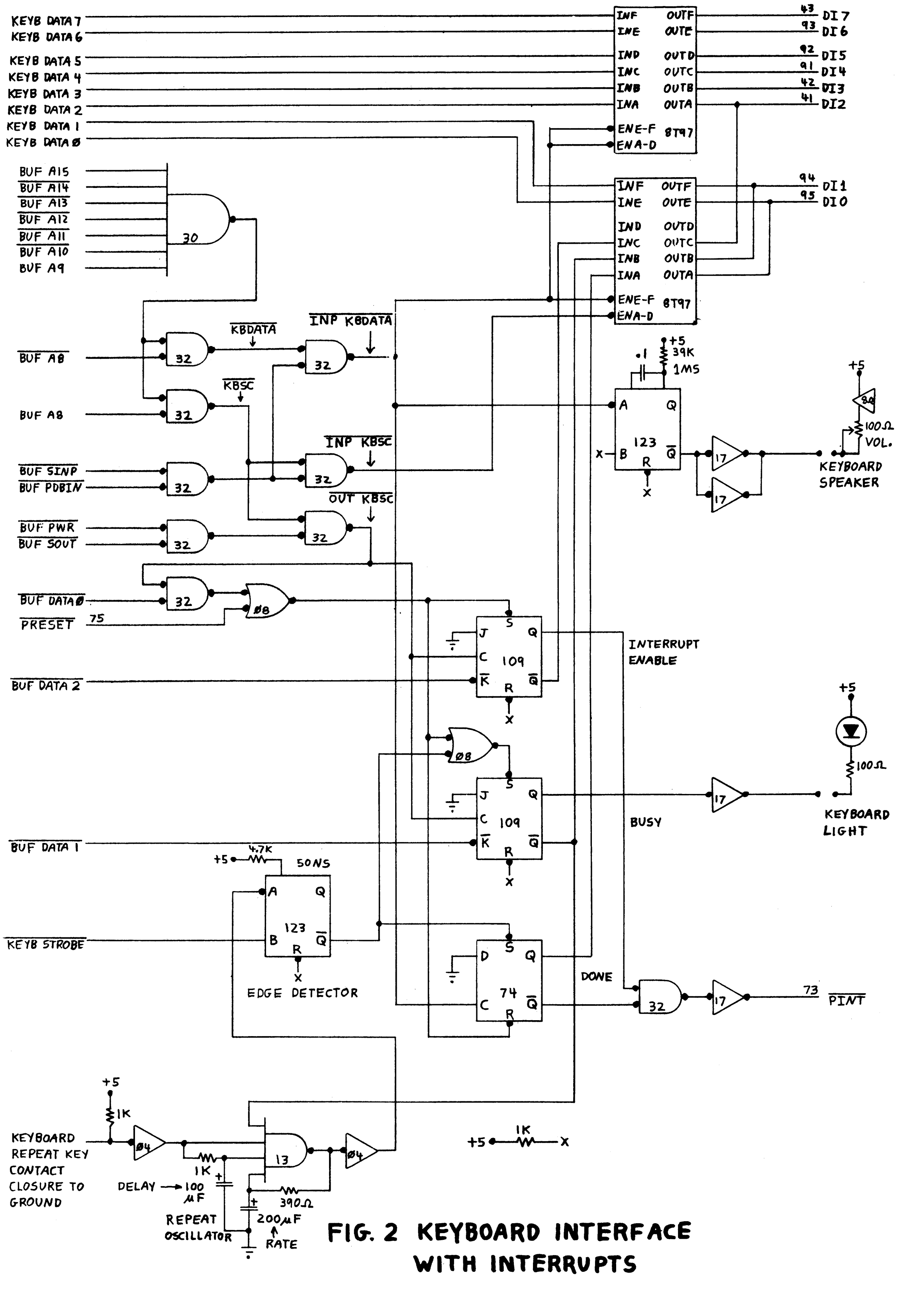


FIG. 2 KEYBOARD INTERFACE WITH INTERRUPTS

The last method is called "software polled interrupts" which is what we will be using for multiple interrupts on the Altair. It too is often found in minicomputers, most notably the PDP-8. Again, all interrupts cause the CPU to branch to the same location; 000:070 in the Altair. Here a common service routine tests (polls) the DONE status bit of every device that can cause interrupts, one device at a time. When one is found, a branch is taken to the service routine for that device. The priority is determined by the order of DONE status testing which means that priorities can be changed with software. Of course the disadvantage of polled interrupts is that interrupt latency time is longer if a lot of devices must be tested. Fortunately, the latency time is shortest for high priority devices since they are tested first. In fact, for most instruction sets, the first two or three devices in a polled interrupt system may get faster service than in a software vectored system because of the time spent in setting up the N-way branch with the latter. On the Altair, using polled interrupts and no memory waits, the worst case latency time to the top priority device is 33 microseconds. An additional 12 microseconds is added for each successively lower priority device in the polling chain. Note that these times include saving A and status on the stack. Simultaneous interrupts are disposed of in the same manner as with a vectored interrupt system. Again, they will be serviced in order of decreasing priority until all are taken care of.

So much for theory, now let us go through the design of a custom interface for a keyboard with all of the bells and whistles, including interrupt capability usable alone or in a polled interrupt system. The keyboard to be used for illustration purposes is a Clare-Pendar unit being sold by The Digital Group and by Herbach and Rademan. This, incidentally, is a very nice keyboard with a full upper and lower case ASCII character set and reasonably good "feel". A unique feature is an "upper case" latching mode key. When on, the alphabetics always come out as upper case but the numerics and specials are not affected. When off, it acts as a standard upper and lower case typewriter keyboard.

The keyboard interface consists of three status and control flip-flops. Two of these are, of course, BUSY and DONE as described earlier. The third is an interrupt enable flip-flop that applies only to the keyboard interface. If it is off, the keyboard cannot cause interrupts. If it is on, an interrupt request is generated whenever DONE is also on as described earlier. The inclusion of an interrupt enable for each device allows both interrupt and non-interrupt I/O to be mixed. It also allows interrupts from selected devices to be inhibited momentarily if necessary. Finally, it offers compatibility with old programs that were not written to handle interrupts.

Three distinct I/O instructions are used with the keyboard interface. OUT KBSC (output to KeyBoard Status and Control) is used to control the state of the three control/status flip-flops. INP KBSC is used to read the state of the same three flip-flops. INP KBDATA is used to read the content of the keyboard output register. In our system, KBSC was assigned to I/O address 202 octal and KBDATA was assigned to 203. Since these differ by only one bit, keyboard address recognition can be simplified. On input from KBSC, bit 0 is the DONE flip-flop, bit 1 is the BUSY flip-flop, and bit 2 is the state of the interrupt enable flip-flop for the keyboard. In the polled interrupt system it is advantageous to place DONE status at bit 0 since a RAR can then be used for a quick test of DONE. On output to KBSC, three bits are significant. If bit 0 is a ONE, all three status/control flip-flops are reset. If bit 1 is a ONE, BUSY is set. If bit 2 is a one, interrupt enable is set. DONE is reset automatically when the keyboard data is read (INP KBDATA executed). Striking a key, of course, will reset BUSY and set DONE.

Appendix A lists the significant portion of keyboard initialization, common interrupt service, and keyboard interrupt service routines. The initialization routine first clears the keyboard to an idle state and then sets BUSY and keyboard interrupt enable. Finally it enables the Altair interrupt system and jumps to the background routine. Any interrupt will cause entry into the common interrupt service routine. First, A and CPU status is pushed onto the stack since the polling chain will alter these. The chain itself consists of three instructions repeated as many times as there are devices that can cause interrupts. If the keyboard is the first device found with DONE on, a jump is taken to KBSRV. Within KBSRV the remaining registers are saved on the stack and the keyboard data is read in thus resetting DONE. After processing the character, keyboard BUSY is set again, the registers are restored, and a return to the interrupted program is executed.

One important consideration when writing programs that may be interrupted is that the stack pointer cannot be fooled with. This means, for example, that subroutines cannot use INX SP and DCX SP in retrieveing arguments from the stack. The reason of course is that an interrupt may strike when the stack pointer has been temporarily moved and stack data may be destroyed by the register save/restore code in the interrupt service routine. Instead, the stack pointer must be loaded into H&L and then H&L used to access arguments on the stack. Another possibility is to disable interrupts while the stack pointer is being manipulated but this may make interrupt latency quite long.

Figure 1 shows a timing diagram for the keyboard output signals. The strobe is generated whenever a key is pressed and the output register holds the key code stable until the next key is pressed. Unfortuantely, there is no ENABLE input to the keyboard so there is nothing to prevent the operator from striking another key and changing the output register contents before the previous keystroke is acted upon. Interrupt capability, of course, reduces this problem by insuring fast response at all times. Most keyboards available to the hobbyist work or can be made to work like the Clare-Pendar unit.

The actual logic to implement the keyboard interface is shown in figure 2. For the most part it is just an application of the input and output interface concepts presented in part 1. Rather than redraw all of the bus buffers that would be required if the keyboard were the only interface on a board, the prefix BUF will be used to designate a buffered bus signal. If there are already a couple of interfaces on the board, then most, if not all of the BUF prefixed signals will already be available.

Keyboard address recognition is performed by a 7430 and two sections of a 7432 connected as NAND-NOTs. One way to understand this configuration is to consider the 7430 as recognizing whether the address pertains to the keyboard, i.e., either 202 or 203 octal. The two 7432 sections then distinguish between 202 and 203 by looking at the least significant address bit.

Emerging from the mass of gates at the top of the drawing are three signals corresponding to the three possible I/O instructions for the keyboard. The topmost goes low when address 203 (KBDATA), SINP, and PDBIN coincide and causes a set of 8T97's to gate data from the keyboard data register onto the Altair DI lines. Additionally it fires a one-shot which drives a speaker in the keyboard for an audible click when the program reads data from the keyboard. Incidentally, error beeps and other audible signals can be easily generated with this setup using simple program loops. The next lower signal goes low when address 202 (KBSC), SINP, and PDBIN are coincident. It enables another section of an 8T97 to gate the state of the three control/status flip-flops onto the DI lines for input into A. The last signal goes low when an OUT KBSC is executed.

The logic around the control/status flip-flops is actually simpler than it looks. Note that interrupt enable and BUSY are upsidedown, that is, they are ONE when Q is low and ZERO when Q is high. This saved two inverters in our own system since BUF DATA 1 and BUF DATA 2 were already available. The 7432-7408 combination provides reset logic for the flip-flops. First, a system reset (called PRESET in the Altair manual) can reset all three control/status flops. However an OUT KBSC with bit 0 being a ONE can also reset all of the flops. Note that the direct inputs to the flip-flops are used which means that reset will prevail if a conflicting command is given. Interrupt enable and BUSY are clocked at the trailing edge of the strobe generated during OUT KBSC. If DATA 2 is a one, then interrupt enable will be turned on at this time. Similarly, if DATA 1 is a ONE, then DONE will be turned on. ZERO data will leave the corresponding flip-flop unchanged since it is a J-K type of flop. A strobe from the keyboard will reset BUSY and set DONE. Finally, reading the keyboard data register will reset DONE by clocking a ZERO into it.

As mentioned before, an interrupt request should be given to the Altair when keyboard DONE and interrupt enable are on simultaneously. Fortunately the Altair interrupt request line, PINT pin 73, is a "wire or" line. That means that open collector gates may be tied directly to the line in order to form the logical OR of many possible interrupt requests. Normally the line is pulled high (which means "no interrupt request" since it is an inverted signal) by a resistor to +5 on the CPU board. However a device requesting an interrupt can pull it low with an open collector gate such as a 7401. There is no practical limit to the number of interrupt requests that can be tied to PINT provided the wiring is not so extensive as to pick up noise. In figure 2 a left over portion of a 7432 and a 7417 are used to pull PINT down when interrupt enable and DONE are both on.

Since figure 1 shows that keyboard data is not valid until the trailing edge of the keyboard strobe, the other half of the 74123 one-shot is used as an edge detector. The resulting narrow width pulse directly resets BUSY and sets DONE. The 7413 and discrete components form a repeat oscillator since one is not provided on the keyboard itself. When the repeat key is pressed, there is a short delay and then the oscillator starts firing the strobe single shot thus simulating multiple depressions of the last character key struck. Note that BUSY must be on for the oscillator to run. With the components shown, the delay is about 100 MS and the repeat rate is 30 per second.

In the next issue, read/write memory interfacing will be discussed. Although some may want to build ordinary memory boards using the techniques to be described, the intent is for specialized memory systems.

APPENDIX A

```
        .
        .
*       KEYBOARD INITIALIZATION
*
        MVI  A,001Q  RESET ALL CONTROL FLOPS
        OUT  KBSC       IN THE KEYBOARD
        MVI  A,006Q  SET KEYBOARD INTERRUPT ENABLE
        OUT  KBSC       AND BUSY, TURNS KEYBOARD
        .               LIGHT ON
        .
        .
        EI           ENABLE ALTAIR INTERRUPTS
        JMP  MAIN    JUMP TO MAIN BACKGROUND PROGRAM
```

```
*       COMMON INTERRUPT SERVICE ROUTINE
*
        ORG  070Q    MUST BE AT LOCATION 000:070
        PUSH PSW     SAVE A & STATUS ON THE STACK
        INP  DEV1SC  TEST DONE IN DEVICE 1
        RAR          ROTATE DONE INTO CARRY
        JC   DEV1SV  JUMP TO DEVICE 1 SERVICE ROUTINE
*                       IF ITS DONE WAS ON
        INP  KBSC    TEST DONE IN KEYBOARD
        RAR
        JC   KBSRV   JUMP TO KEYBOARD SERVICE IF ON
        INP  DEV3SC  TEST DONE IN DEVICE 3
        .
        .
        JMP  ERROR   SPURIOUS INTERRUPT, HARDWARE FAILURE
```

```
*       KEYBOARD SERVICE ROUTINE
*
KBSRV   PUSH B       SAVE REMAINING REGISTERS
        PUSH D          ON THE STACK
        PUSH H
        INP  KBDATA  GET CHARACTER FROM KEYBOARD, ALSO
*                       RESETS DONE AND CLICKS SPEAKER
*
        .            INTERPRET CHARACTER AND ACT ON IT
        .
        MVI  A,002Q  SET BUSY ON TO ENABLE FOR NEXT
        OUT  KBSC       CHARACTER
        POP  H       RESTORE REGISTERS
        POP  D
        POP  B
        POP  PSW     RESTORE A AND STATUS
        EI           ENABLE ALTAIR INTERRUPTS
        RET          RETURN TO INTERRUPTED PROGRAM
```

SURPLUS SUMMARY

There is a new publication out which is certainly worthy of note in this column. It is ON-LINE. ON-LINE is a want-ads flyer service specifically for computer nuts. Lots of good listings. ON-LINE is issued 18 times per year and goes for $3.75/year. For a free sample issue write to:

ON-LINE
24695 Santa Cruz Highway
Los Gatos, CA 95030

Notice to you folks who wanted to build graphics systems: yoke cores for the yokes that Hal Chamberlin offered have been delivered now that Stackpole is off strike. The yokes are available for $15.00 with a 2 week delivery, see issue #3 for details.

For those who are into games or just analog inputs in general, James Electronics has a joystick for $9.95. Outputs are four 100K pots, two on each axis. Write to:

James Electronics
Box 882
Belmont, CA 94002
Ph. 415/592-8097

Want to try your hand at building a CRT monitor? If so Meshna is offering an excellent kit of parts including a 9" green phosphor (P39) tube with socket, flyback, HV rectifier, HV cap, magnetic shield, and deflection yoke for $20.

MESHNA
Box 62
East Lynn, MA 01904

ALTAIR 8800 BUS SIGNALS

TCH has had one unusual but worthwhile request since our last issue. Two readers have asked that we list all the signals used on the Altair backplane. Their purpose is simple, though they do not own or plan to own Altairs, they would like to utilize some of the plug compatible boards being offered. The bulk of the signals would eventually be explained in the Altair interfacing series but for the sake of conciseness and convenience here they are:

PIN	NAME	DESCRIPTION
1	+8V	Unregulated input to 7805 regulators
2	+16v	Unregulated input to +12 regulators
3	XRDY	Anded with PRDY and goes to 8080 RDY
4	VI0	Vectored interrupt request 0
5	VI1	Vectored interrupt request 1
6	VI2	Vectored interrupt request 2
7	VI3	Vectored interrupt request 3
8	VI4	Vectored interrupt request 4
9	VI5	Vectored interrupt request 5
10	VI6	Vectored interrupt request 6
11	VI7	Vectored interrupt request 7
18	STA DSB	Status buffer disable
19	C/C DSB	Command/control buffer disable
20	UNPROT	Input to memory protect circuitry on mem bd
21	SS	Indicates machine is in single step mode
22	ADD DSB	Address buffer disable
23	DO DSB	Data out (from CPU) buffer disable
24	02	Phase two clock TTL levels
25	01	Phase one clock TTL levels
26	PHLDA	Hold acknowledge, buffered 8080 output
27	PWAIT	Wait acknowledge, buffered 8080 output
28	PINTE	Interrupt enable, buffered 8080 output
29	A5	Buffered address line 5 (32)
30	A4	Buffered address line 4 (16)
31	A3	Buffered address line 3 (8)
32	A15	Buffered address line 15 (32768)
33	A12	Buffered address line 12 (4096)
34	A9	Buffered address line 9 (512)
35	DO1	Buffered data out line 1
36	DO0	Buffered data out line 0, least sig. bit
37	A10	Buffered address line 10 (1024)
38	DO4	Buffered data out line 4
39	DO5	Buffered data out line 5
40	DO6	Buffered data out line 6
41	DI2	Data input line 2
42	DI3	Data input line 3
43	DI7	Data input line 7, most sig. bit
44	SM1	Latched 8080 M1 status
45	SOUT	Latched 8080 OUT status
46	SINP	Latched 8080 INP status
47	SMEMR	Latched 8080 MEMR status
48	SHLTA	Latched 8080 HLTA status
49	CLOCK	2 mHz clock, crystal controlled
50	GND	Logic and power ground return
51	+8V	Unregulated input to 7805 regulators
52	-16V	Unregulated input to negative regulators
53	SSW DSB	Sense switch disable (special for console)
54	EXT CLR	Clear signal for I/O devices
68	MWRT	Write enable signal for memory
69	PS	Indicates if addressed memory is protected
70	PROT	Input to memory protect circuitry on mem bd
71	RUN	Indicates machine is in run mode
72	PRDY	Anded with XRDY and goes to 8080 RDY
73	PINT	Input to 8080 interrupt request
74	PHOLD	Input to 8080 hold request
75	PRESET	Clear signal for CPU
76	PSYNC	Buffered 8080 SYNC signal
77	PWR	Buffered 8080 write enable signal
78	PDBIN	Buffered 8080 DBIN signal
79	A0	Buffered address line 0 (1)
80	A1	Buffered address line 1 (2)
81	A2	Buffered address line 2 (4)
82	A6	Buffered address line 6 (64)
83	A7	Buffered address line 7 (128)
84	A8	Buffered address line 8 (256)
85	A13	Buffered address line 13 (8192)
86	A14	Buffered address line 14 (16384)
87	A11	Buffered address line 11 (2048)
88	DO2	Buffered data out line 2
89	DO3	Buffered data out line 3
90	DO7	Buffered data out line 7, most sig. bit
91	DI4	Data input line 4
92	DI5	Data input line 5
93	DI6	Data input line 6
94	DI1	Data input line 1
95	DI0	Data input line 0, least sig. bit
96	SINTA	Latched 8080 INTA status
97	SWO	Latched 8080 WO status
98	SSTACK	Latched 8080 STACK status
99	POC	Clear signal during power-up
100	GND	Logic and power ground return