COMPUCOLOR II DISK FORMAT.

JAN/FEB 83 UPDATE

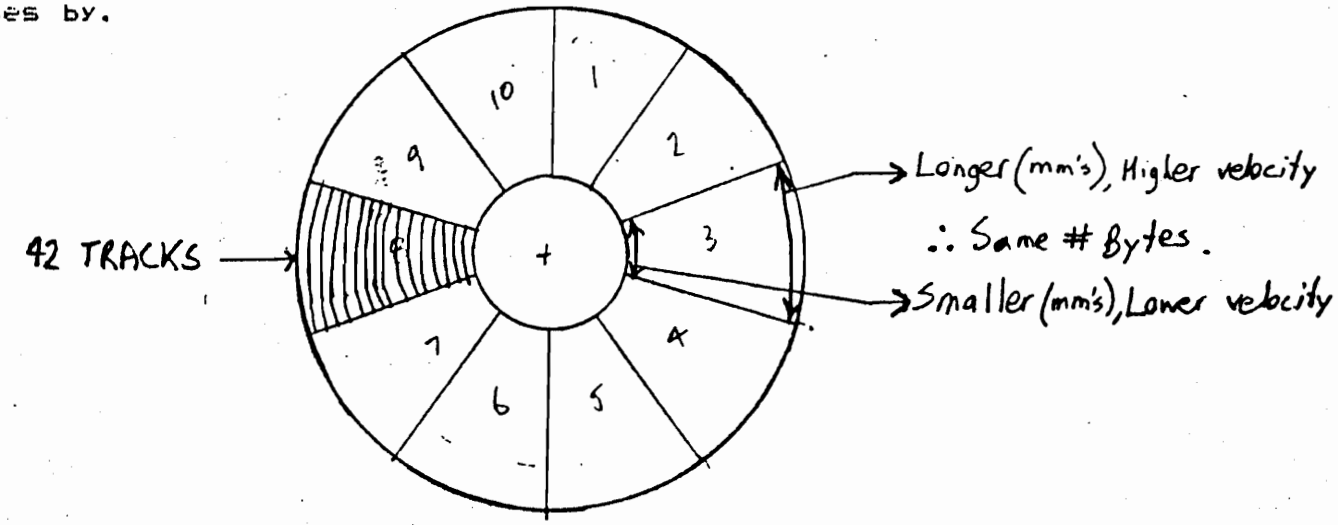
The average CCII user usually thinks of a disk as something which has about 400 'blocks' each of 128 bytes - giving him 52K total. This of course is true, but to be more correct you should say that each disk has 42 'tracks' each of 10 'sectors' each of 128 bytes - giving 52K total.

Now, a 'track' is (apart from being something you drive a train on) a ring of the magnetic material on the disk. Thus each CCII disk is divided into 42 concentric circles EACH OF THE SAME WIDTH. Just as a comparison, Apple II disks are formatted with only 35 tracks, some TRS-80 disks have 70 tracks. The increased memory is obtained by more and larger sectors.

The stepper motor which moves the read/write head is capable of stopping at any one of 84 (42*2) positions. The reason that two steps are taken for each track is that if only one is taken then the magnetic pulses written to one track will overlap with the neighbouring tracks and thus losing data. This can be used as a form of copy protection simply by writting your information on the odd steps instead of the even ones and having a program which moves the head one step off before normal reading (This is quite a popular method used with the APPLE II computer).

Now, each one of these tracks is divided into 10 identical sectors. A sector is the smallest accessable part of a disk. i.e. if you want to read a specific byte from a disk then you must read the whole sector it is contained in. Now you may or may not have realised that each track is a different physical length (as in circumference) and if you go and stick 10 equal length sectors on each then you're going to have either a large unused bit on the outer track or overlap on the inside track . . . or so some people think.

As any Physix student will tell you, the angular velocity is the same throughout the disk but the linear velocity differs track to track. So, if the period of rotation is say 1 second (actually it's 0.003 secs) that means one whole track (10 sectors) passes by the head every second, so one sector - any sector - takes 1/10 of that time which is 0.1 secs. So if the head is on the very outermost track and you sit there for 0.1 second then a lot more magnetic material will go past in that time than if the head was on the very innermost track. But as the outermost track is the longest and the inner track is the shortest then in 0.1 secs the same percentage of a full circumference passes by.



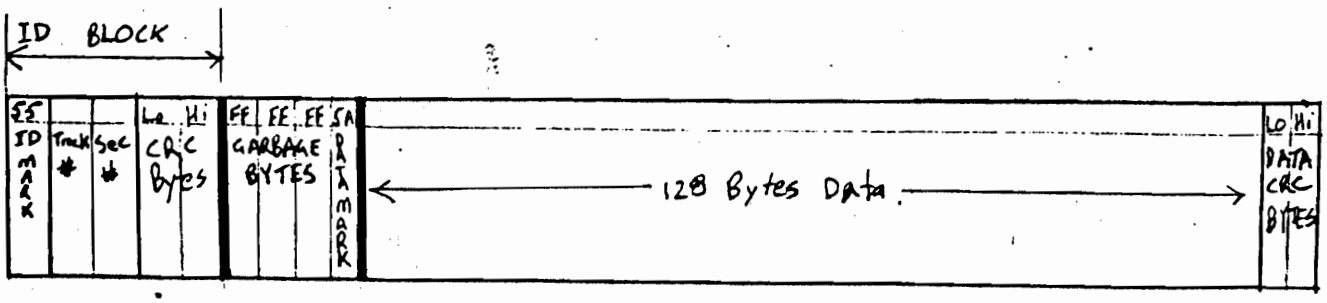
Now, if you or the computer decide that you want the formation from 'block 7A' through 'block 83' put at 6000H onwards, hen all the computer (FCS) does is to tell a very special program in the ROM called the Micro-Disk-Handler exactly that - "Read from block 7A, length of 1280 bytes, stick it at 6000H, get it from CD0: ". The MDH then figures out where block 7A is: 7A=122 decimal so it is 2 sectors into the 13'th group of 10. Put in more technical terms that means 'track 13 sector 2'. One Block represents a sector so they can be interchanged like that. So the MDH (knowing that the head is already at the outermost track 0) goes for main engine ignition on CD0: pulses the stepper motor 26 (13*2) times and then has to find the second sector of the track that we're on. This is the hard bit.

Now each track has 10 sectors on it and the beginning of each must be individually identifiable. The way this is done is that between the end of one sector and the beginning of the next there is a very small area of 'blank' which when read doesn't return a 1 or 0. This can be found quite easily by the computer. We have now found one of the 10 inter-sector areas on the track. All we do is to continue reading until we actually receive a byte i.e start receiving 1's and 0's. By definition this byte (called the ID MARK) is made 55H. If it isn't then we have an error and decrement an error count (get 3 chances) and go for it again.

Once we have found the beginning of a true sector, we have to identify which of the 10 it is. Pretty obviously they are numbered 1 to 10 so what happens is that the next 2 bytes are read. These are the address block. The first byte contains the actual track number of the track and the second contains the sector number. Using the same example as before (Track 13 Sector 2) we compare the two bytes we read with 13 and 2 respectively. If the track number doesn't match then we've got a reletively bad error as we were sure that we had the correct track so a 'pound' is executed. More on this later.

The desired sector number is now compared with the discovered sector number and as there was only a 1 in 10 chance that we got the right one straight off we will undoubtedly find that they don't match. No worries, we simply decrement an error counter (30 chances = 3 * 10 = 3 revs of disk = 3 passes over desired sector) and go back to the routine which finds the inter-sector gap.

Once the sector we want is found we continue on and read the next 2 bytes (lobyte, hibyte) which are called CRC bytes. This number is derived from a very complicated algorithm which produces its output from all the bytes written to the particular sector already. The purpose of this is for error checking. It is derived from EVERY byte written to the sector so far so if it doens't match the internally kept total that the CCII keeps as it is reading then it decrements another error counter ('read retry counter' = 10) and goes back to the set inter-sector routine. Some other computers call this a Checksum and is simply the addition of all the bytes.



Once that is over, we have successfully found the sector we want and can get the 128 bytes of data. But first, because these comparisons of track/sector and CRC just mentioned takes a small but still noticable amount of cpu-time the disk would have spun - depending on conditions - one or two bytes. Now if this was our much loved 128 bytes of data then info would be lost. What is done is that 3 FF's (technical name 'garbage bytes') and a 'data mark' (=5A) are written at this point to allow for the delay. So when we come back on the air after it we simply keep reading bytes until something other than FF is read. If this byte isn't 5A (data mark) then jump to the error routine.

Now, the next 128 bytes of data are the info we read and write to and are the only bytes accesable to the user. Once they have passed, the next 2 bytes are the data CRC bytes. These are read and compared with the internally kept running total. Error if they don't match etc.

We have now reached the end of the sector and are now on the inter-sector gap. If more sectors are required then they are read otherwise we 'home' the head by putting it back to track zero and execute MECO (Main Engine Cut Off).

Mentioned before was a routine named 'pound'. This is the routine called when we are on the wrong track or any of the error counts are exausted. This is the routine which makes the rather loud crunching noise. What it does is to - irrespective of where the head is on the disk surface - move it out 42 times. So if it was on the 10'th track it will be forced up against the stop 42-10=32 times. This doesn't damage anything so don't worry about it. The 'block retry error count' (=3) is decremented whenever another error count forces a pound. When this runs out then the whole operation is aborted and the user is told what the story is.

Whenever the Micro-disk-handler produces an error it displays it's own message. This is why you sometimes get two error messages on a disk error. e.g. if you were doing a directory and there was a read error, the handler will tell you about the read error and FCS will tell you about the directory error (EDIR). The format of Handler error messages is as follows: First, the 4 character error code is printed (EDCS=data crc wrong, ESKF=seek failure (can't find desired Trk/sec), EDSY=data sync (data mark unfound, etc.). Then the device is printed (CD0: or CD1:). Then the handler function code is printed (0=reading, 1=writing, 2=verifying (done after every write)). Then the block number that it stuffed up on is displayed. All this would not normally be of much value but there will be situations where it is important.

If you are not totally confused by here then you have a chance of getting through this next bit.

Now, if you think about it, when you go to Tricky's and buy a brand new box of disks they are not going to have any magnetic information written on them at all. So if you try to read track 13 sector 2 the head will move out there and not read a single byte because the whole track is the same as an inter-sector gap. This is also the case if you stick an Apple disk in and try to read it - there won't be any 1's or 0's written at the speed which we read at. (i.e. number of 1's and 0's written per unit time). So what is needed is a disk 'formatted' to the Compucolor format. So you get your formatter out and blast away at these disks little knowing that what it is actually doing is writting all these ID marks, track/sector numbers,

CRC bytes, garbage bytes, data marks and 128 bytes of dummy information. It does this to all 42 tracks and then goes back to read check it.

A read check simply goes through each sector on each track and checks each for correct ID marks, CRC bytes, Track/sector numbers etc. You may have noticed on the error notices that the formatted shows one of three single letter codes. 'A' means that an error in the ID block has occured, 'S' means that an error in the data mark has occured and 'D' means that an error with the data CRC has occured.

The way information is actually transmitted/received to/from the drive is (predictably) through the TMS5501 chip. The MDH tells it that we want to talk at 72K (72000) baud by an 'out' statement and starts the motor by another. When 8 1's and 0's pass the head the 5501 sets a flag to tell the CPU that it has received a byte. When writing a byte we simply out it to a port (port 0) and the 5501 shifts it out one bit at a time. By using this method the bit density is very low (bad) and accordingly only 52K per disk is available. On computers like the Apple and TRaSh the actual CPU sends straight to the disk one bit at a time and includeds clock pulses and stuff like that making reading/writing bulk fast and also lots more info per disk.

The stepper motor is controlled by the software so a special value has to be stored for the current phase it's on. V6.78 machines have a 3 phase motor and on V8.79 it is a 4 phase motor. I have no idea why it was changed so don't ask.

The recently acquired program 'SPEEDO' works on a very simple priciple. The head is moved in to track 1, and the program waits for the beginning of a sector. It then counts how long the whole sector takes to pass by. This might return a value of say 2634 units of time. This has to be scaled to RPM which involves lots of mathematics which in machine language can't be described! The program averages the times for about 10 sectors before printing the little dot on the screen. The only problem is that if the disk being read is formatted at 290 (i.e. something other than 300) RPM, but the drive is actually spinning at 300 RPM then the program will think that the drive is running faster or slower. The answer to this is to use a disk which is positively without doubt - in the sense that you're sure - FORMATTED (not written) at 300 RPM.

There is no point in giving the addresses in Rom for the particular routines relevant here as, they will only have meaning if they are accompanied by the documentation of the Rom manual.