



My Time at Acorn

From Market Hill to the Silver Building

Hugo Tyson

ROUGOL June Meeting 2025-06-16

The Duke of Sussex, nr. Waterloo

All images are copyright© whoever owns them on the web – not me!

With particular mention of the very first ARM CPUs, taken from this talk given in Cambridge:



Testing the very first ARM CPU

Writing tests for a chip that doesn't exist!

Hugo Tyson

ARM first silicon 40th Anniversary, 2025-04-26

ARM, Cambridge

My story – a timeline, with excursions into some comedy bugs, maths, and CPU testing

- Nothing to do with RISC OS – sorry folks!
- “What it was like” at Acorn, early days...
- Holiday jobs at 4a Market Hill – Atomic Age
 - Atom Invaders, Atom Snapper
- Proper job at Cherry Hinton – Beebs everywhere!
 - **ADFS**; **DFS** – bugs bugs bugs - “Disc Fault 18”
 - The 2nd processors, the “twos” bug
 - Electron ADFS *compact problems
 - **ARM! ...test suite** – FPU co-processor; RISCiX
 - The end? No! Diplomatic incident with DEC...

Context – Student Life

- 1979-1980 I was studying Mathematics Part 1A (that means the first year) at King's, Cambridge

- A very earnest young student!
Already with an interest in computers – this pic from Sixth Form.



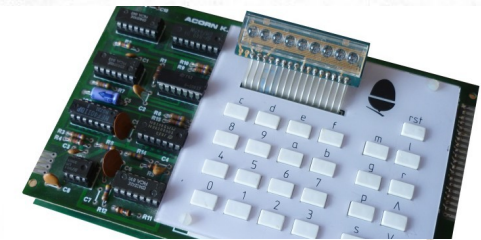
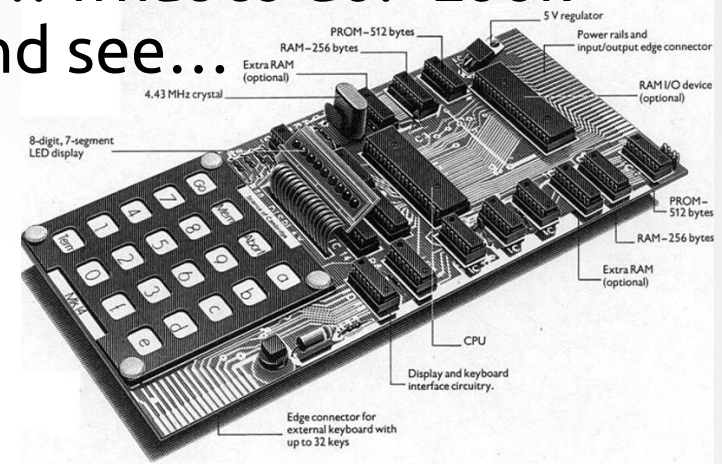
Holiday jobs - 1

Summer 1980. End of term; exams finished... what to do? Look at my mates' computer magazines and go and see...

Sinclair! With the MK14, office on King's Parade, but they had no jobs.

Next... **Acorn!** at 4a Market Hill. System 1 and this new "Atom" thing. I met Hermann and he asked "what do you know about the Atom", I said not a lot but...

So I had a job, answering tech support calls, in theory – in practice it was placating angry people who wanted to know where their Atom kit was! 'Cos of course Acorn cashed all their cheques.... £££££....!



Acorn's new baby...



Hermann and Chris (Curry) promoting the **Acorn Atom** on the front lawn at King's.

Orders were overwhelming...

Aww, didn't they look young!
The hair!
The trousers!
The ducks!

4a Market Hill



Where it all began....

- Strange building, with an electrical substation in the ground floor – was the Electricity Showroom.
- Fruit machines!
- Frozen drains!

(They used the pub yard between the Mitre and the Baron for “Micro Men”)



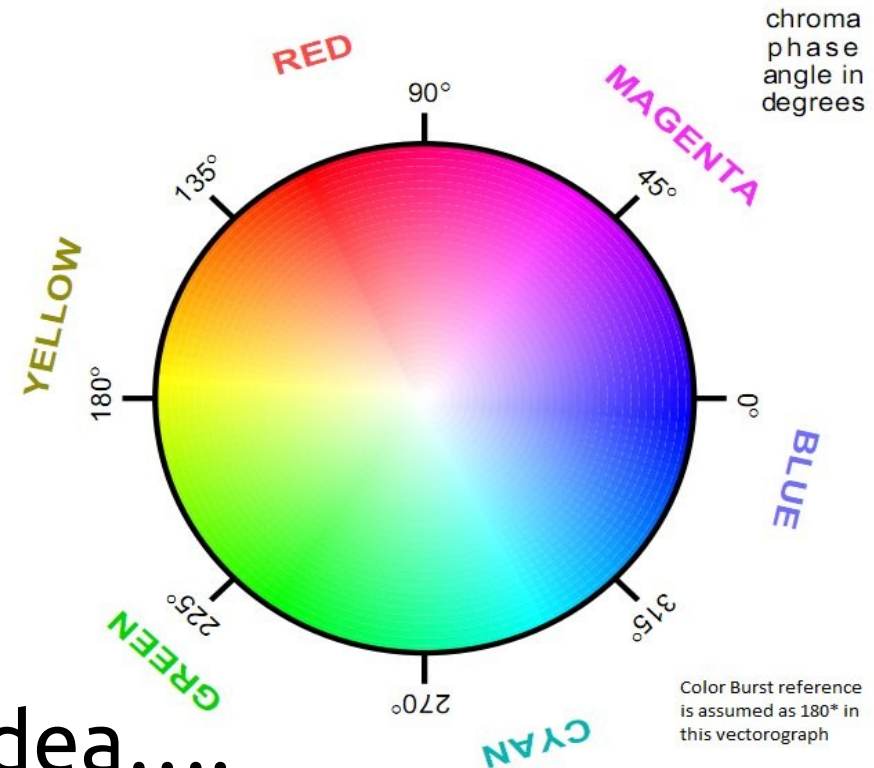
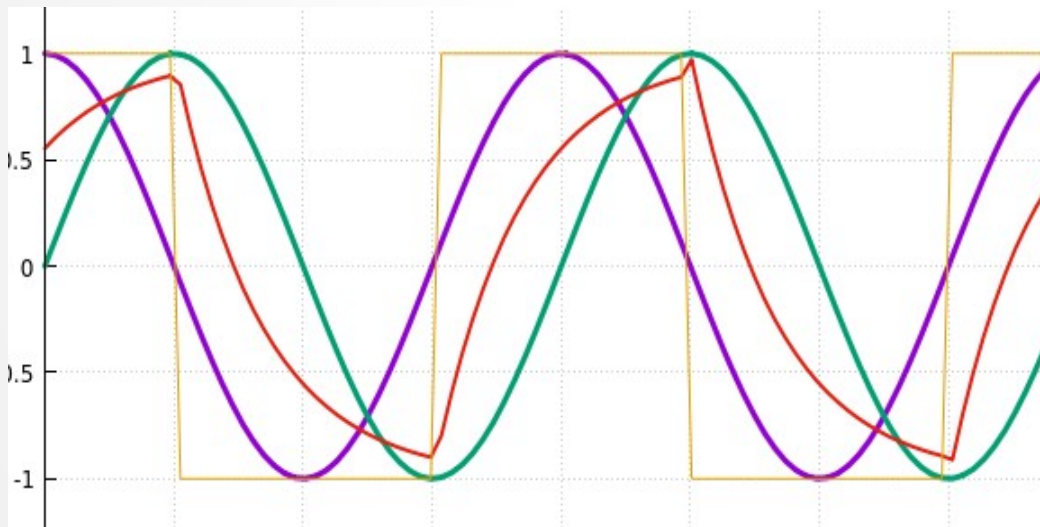


Holiday Jobs – 2

Xmas 1980, just staying up as long as poss. Cold!

The **Atom Colour Board** – add on to mix in the 4.333MHz chrominance signal to video out.

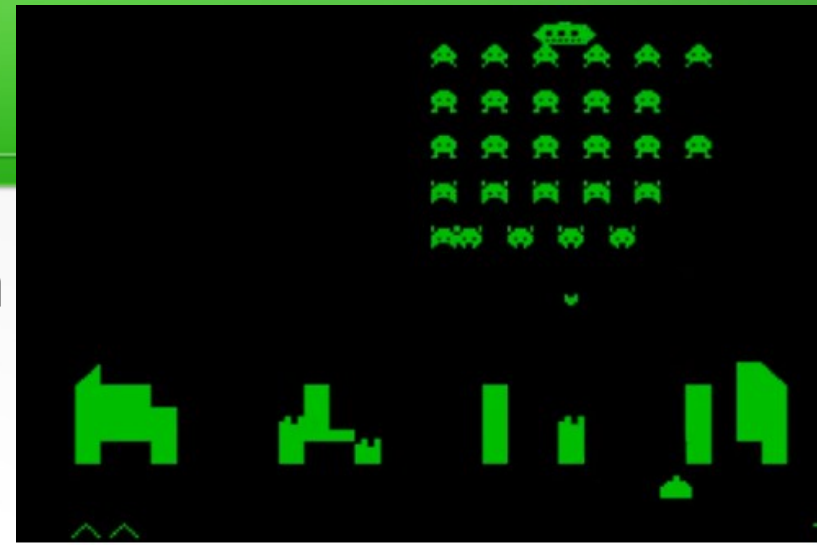
“There’s not enough red in it”



...decided hardware is a bad idea....

Atom Games

But I did buy and build myself an Atom... then wrote Invaders (instead of doing Pt 1B Maths). And sold it to DJD at AcornSoft, and on that basis worked for them over Summer 1981.



Atom Snapper! 3 CHAIN'd BASIC asm progs!



...and switched to CompSci for Part 2.

A Proper Job – Sept 1982

Acorn, John Horton: “Who are you?” ...

But anyway, a proper job with pay:
£6000 pa!

- BBC Micros *everywhere*, plus System 3/4/5...
- **ADFS** – a better filing system for the BBC Micro
- **DFS0.9** → **DFS1.2** many many bugs fixed



ADFS VS DFS 0.90

- **ADFS – aka asadDOS**
 - Bigger, heirarchical directories – 47 entries, 5 blocks
 - Type attributes for is-a-dir, plus RWXL
 - DFS: 31 files in total on the disc! Single letter “qualifier”
 - Longer filenames: 10 chars – think Twitter!
 - DFS: 7 chars + 1 “qualifier”
 - More open files: 9 at once
 - Open file block caching, 5 blocks, 9 files
 - DFS: 5 open at once
 - Double density floppies, MFM coding not FM
 - “Winchester” Hard discs – 3Mb (*sic*), 10Mb soon, 30Mb later!
 - Free space list, ‘cos you can’t see all the files at once

ADFS and its workings

- **ADFS** – aka **asadDOS** spec'd by Sophie Wilson.
 - Bigger, heirarchical directories
 - So possible to partially write ⇒ Sequence number, ID text
 - More open files: 9 at once – 5 blocks cache, 9 open files ⇒
 - LRU algorithm – chase a flag round and round the circular buffer of blocks: statistically it'll be on the one used longest ago, because it moves on whenever you use a block.
 - Great for 2 or 3 open files – deferred commit, parallelism possible.
 - Fine for regular use of, say 4 files and 2 or more idle, but...
 - Hard discs – 3Mb (*sic*), 10Mb soon, 30Mb later! ⇒
 - Free space list separate from file list – inconsistency is possible!
 - Files are single extent. So you must occasionally move an open file, if it's growing!
 - Rule of thumb: if it got to size S, chances are it'll get to $2 \times S$ so extend accordingly.
 - Free list 83 entries of 6 bytes, [Start, Length]: join &c.
 - Allocation algorithms: PFEFF selected from PFEWF, FF,WF
 - Simulated, and chosen to minimise the expected length of the free list
 - Full free list might mean it's impossible to delete a file!
 - ***COMPACT**
 - More bugs here than I can bear to think about...
 - So **lots** of consistency checking tools, in **BASIC**: block editor, **fsck**-alike.

Meanwhile, DFS 0.90 \Rightarrow 1.20

- DFS 0.90 was a terrible buggy pile of...
 - No Wonder Watford Electronics *et al* were eating Acorn's lunch with their ripped-off clones
 - One Watford product was a disassembly, re-ordered, reassembled, with the same bugs in it! Court case!
- So I was redirected (ahem) to fix it.
 - ADFS was kinda taking shape but most of the work thus far had been on Winchester drivers, and testing.
- Nightmare... but the worst of all was...

DFS: "Disc Fault 18" bug! (1)

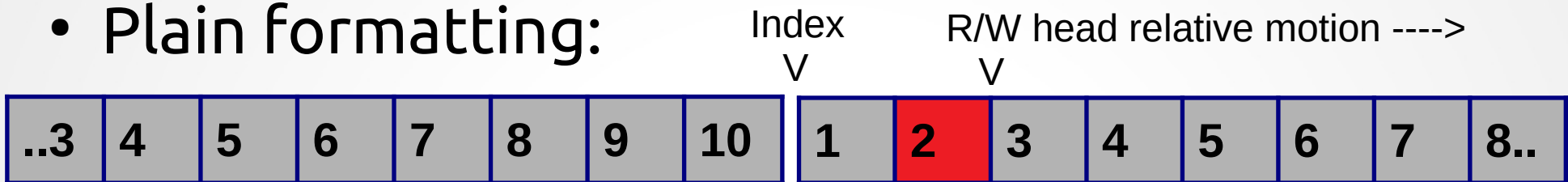
Very rarely, when writing a file to floppy disc using DFS, it would say `Disc Fault 18`, and the disc would then be unusable. Code 18 means "Sector not found" - ie. broken formatting.

- Nightmare:
 - Rarity – very difficult to reproduce
 - Writing can't *do* that, it doesn't write the headers
 - Fortunately someone in production engineering worked out that it was only one make of floppy drive that did this. Aha! Something to get hold of!
- It was always the physical sector #2 of the track – which could be different logical sector numbers...
 - So we need to talk about formatting, sorry...

"Disc Fault 18" : formats (2)

- 10 sectors per track, with an index hole to start the search.

- Plain formatting:

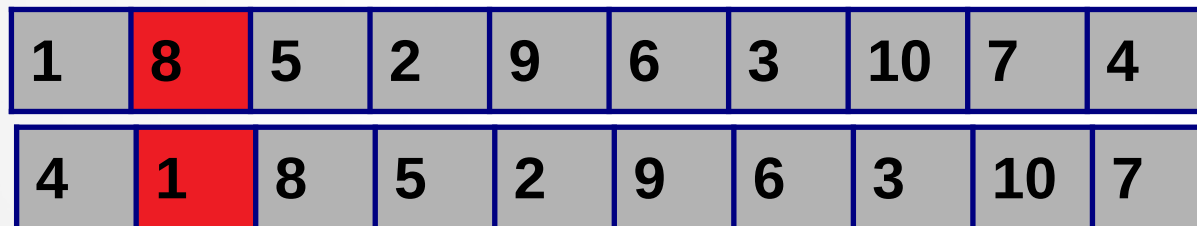


- Interleave 3:

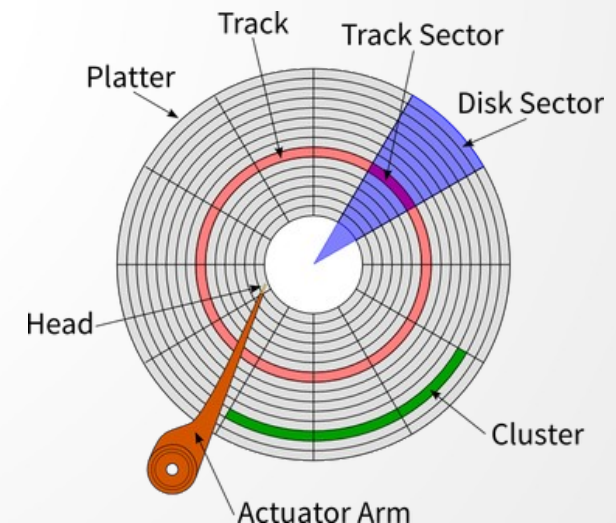
<----- Disc moves this way



- Interleave plus track interleave:



- It's all horrible...



"Disc Fault 18" – timing! (3)

- A sector is an identifying header, plus data + CRC, then a gap. Reading eg. #2 is easy:

Header1	Data1	gap	Header2	Data2	gap
Read, is it #2?	No, read, ignoring	gap	Read, is it #2?	Yes: read and send data	gap

– Time this way



- Writing #1 then #2 for example:

Header1	Data1	gap	Header2	Data2	gap
Read, is it #1?	Yes: get data and write!	gap	Read, is it #2?	Yes: get data and write!	gap

8271
chip:

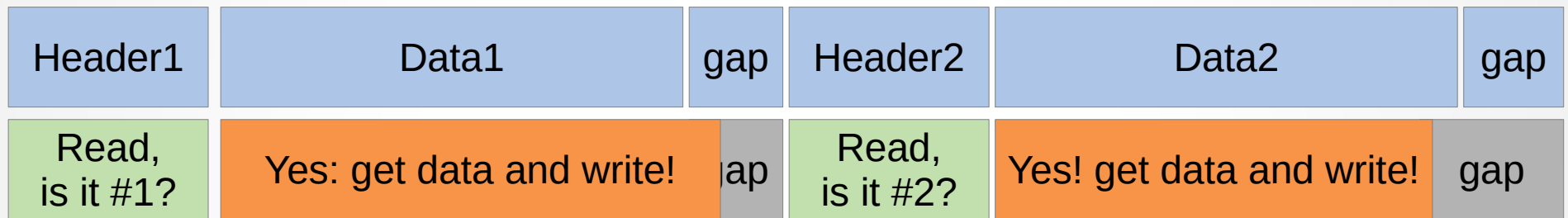
Change from
read to write

Back to
reading

Change from
read to write

"Disc Fault 18" – writes (4)

- Why's that gap there? Because rotation speed is not absolutely the same as when we formatted it – or wrote it last time....

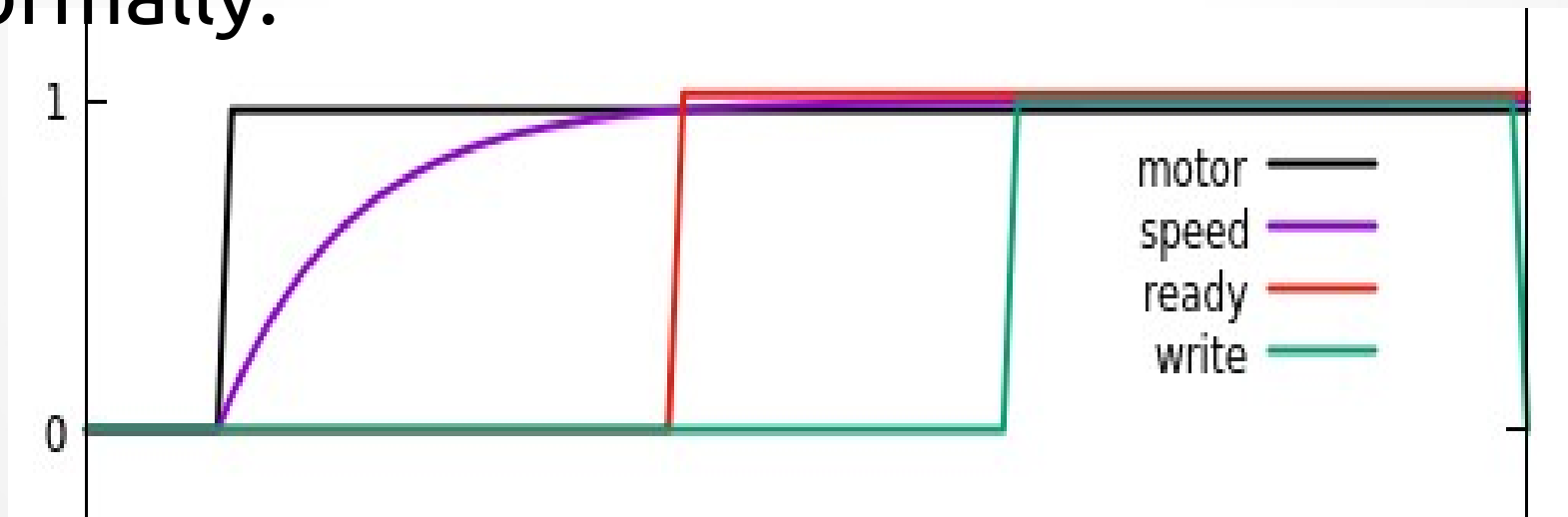


- and nor is the 8271 system clock speed!
- So what if....



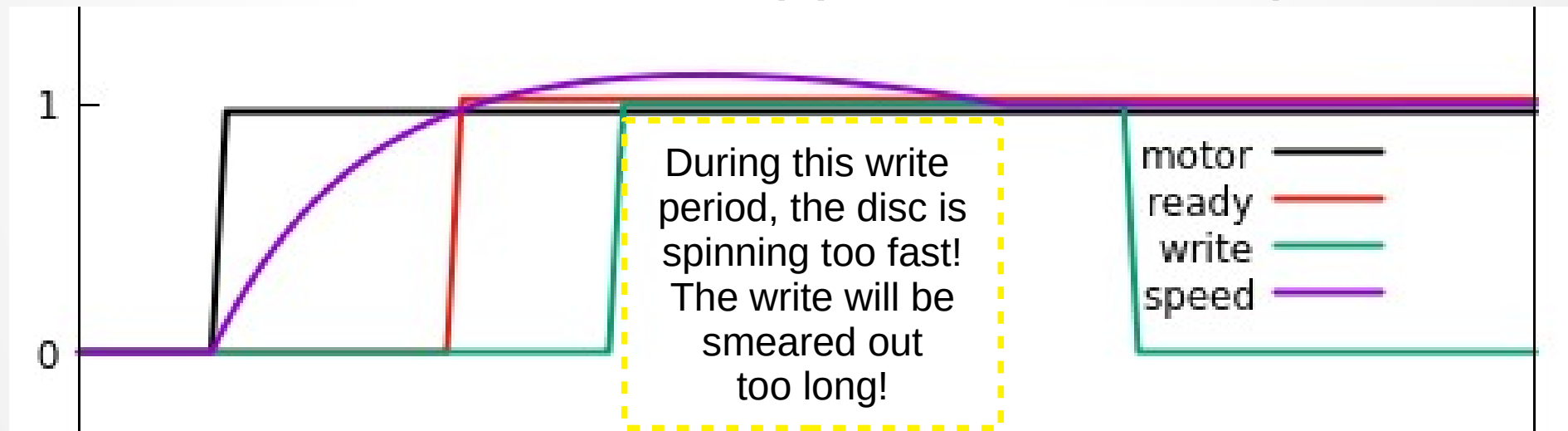
"Disc Fault 18" – a theory (5)

- So we had a possible explanation – a theory
- How do these 5¼" disc drives actually work?
 - So we set about putting oscilloscopes on the innards – there were only a few wires to the motor and the external interface to the 8271.
- Normally:



"Disc Fault 18" – Proven! (6)

- Can we catch it in the act?
- Yes! This particular brand of drives said "Ready" as soon as the motor is up to speed, including, momentarily, *faster!*
- Disc Fault 18: we saw it happen on the 'scope!



- The fix was boring: add a delay. Hardware huh.

Back to ADFS – the “twos” bug (1)

- Having released **DFS** 1.0 and then **DFS** 1.2 in the “**DNFS**” ROM, back to ADFS, with lots of work still to do. But at the same time...
- The “twos” bug. Co-processor *AKA* 2nd Processor connects to the Beeb via “The Tube”
 - Just a set of FIFOs of varying lengths; some were a couple of bytes, one 8 and one 16 bytes IIRC.
 - With flags for Full/Empty/Available &c.
 - Completely asynchronous.
- Rarely (!) the Beeb would go mad, spewing character “2” 0x22 to the screen as fast as it could.
- It was prompted by Tube traffic – like you’d do in ADFS testing. Or maybe an econet file server. I honestly can’t remember.
- So we set about it with a logic analyser....

The “twos” bug – what happens? (2)

- It would fire every few hours on a busy system
 - ...such as testing ADFS and similar, hammering away.
- The logic analyser knew 6502. Beeb was stuck in a loop, had to work back from that.
 - With a lot of logic analyser programming, I hunted down where the Beeb-side 6502 was losing its mind:
 - This was the code. It polls for anything from the Tube, fast as it can, normally spinning like mad then quitting the loop when there's something to do.

```
- BEC4  poll1: BIT  0xFEE7    ; anything ready?
- BEC7                BPL  poll1    ; nope; else:
- BEC9                LDA  0xFEE6    ; get the data
- BECC                ... ..
```

What's a logic analyser? (2a)

- A logic analyser is big **and** clever
- Records lots and lots of data in parallel, and fast
 - Literally *dozens* of wires – unheard of if you had only known an oscilloscope.
- So can disassemble 6502 ops, given the data bus, address bus, read/write lines, instruction fetch line &c.

More interestingly, they were expensive: £25,000 – same as a house (1983)
Luckily, also well made – coffee proof.

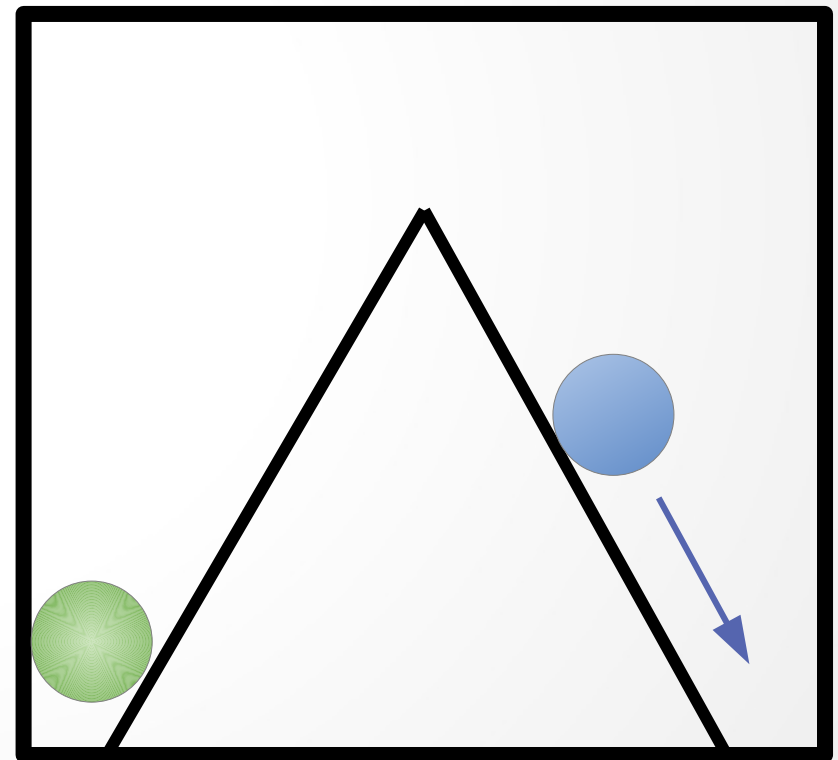


The “twos” bug – caught! (3)

- I managed to catch it, recorded on screen, performing a Branch instruction incorrectly:
 - (kind of a short version of what a logic analyser records)
 - **BEC4** **BIT 0xFEE7**
 - **BEC7** **BPL -3**
 - **BEC4** **BIT 0xFEE7**
 - **BEC7** **BPL -3**
 - **BDBE** **What the actual...!**
- It branched somewhere like **0xBDBE** – madness!
 - Not the branch target, not the next instruction!
 - Using a nasty mix of the hi-byte and the same decremented to construct a destination!

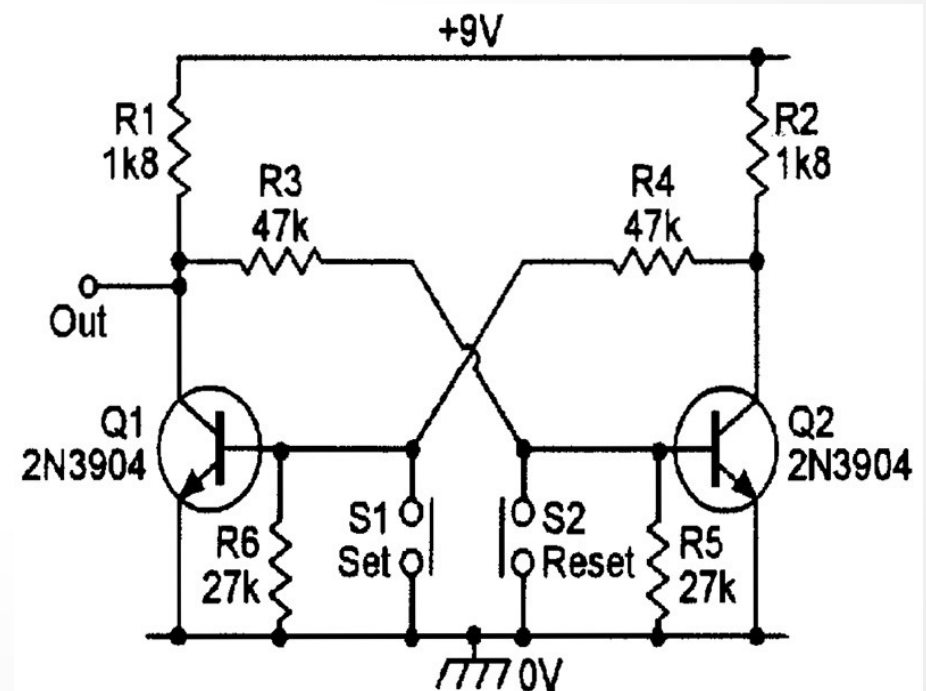
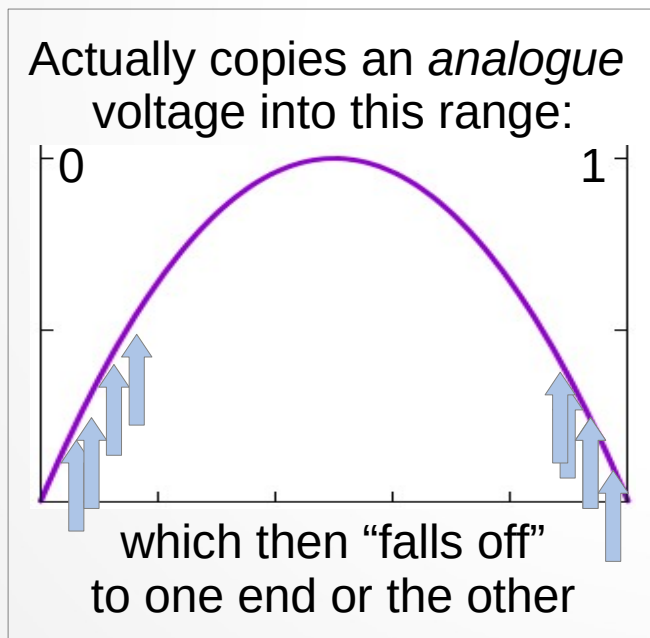
The “twos” bug – but why? (4)

- At the same time Sophie and Steve (Furber) had been thinking and talking to Western (Design?) makers of the 6502. They said:
“Yeah, we’ve seen that too. Odd, isn’t it?”
 - Hardware eh... again....
- Well. But anyway, it turns out – this was R & S’s best theory – that it’s all down to ***metastability***.
- Registers are ***bistable*** circuits, flip-flops, D-types, as we say. They remember things (“*stable*”) ‘cos the two extreme states (‘0’ or ‘1’) are lower energy states than anywhere in between.
- You need to add a lot (relatively) of energy to change it – to drive the state “over the hump”.
- Like a marble in a box with a triangular partition – it stays one side or the other, but you can change it if you raise it, uses energy



The “twos” bug – registers (5)

- This all very simplified. Anyway...
 - When it's flipped (1) or flopped (0) the energy is at one or other end of the graph.
 - Change it by hitting one of the switches to force one transistor not to conduct and so force the other to do so.

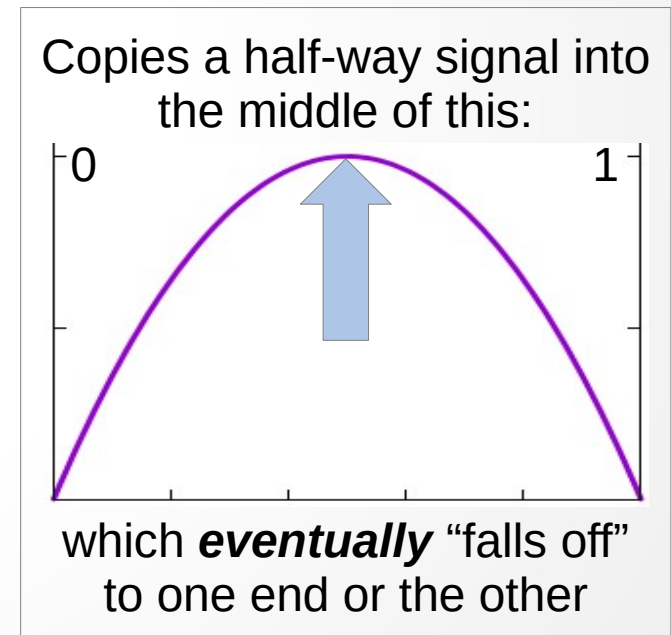


The “twos” bug – metastable (6)

Suppose a synchronous system sees an asynchronous signal, half-way through changing. It might land on the top of that curve – at the metastable point.

- Like balancing a pencil on its point.
- Thermal noise or EMI from clocks will perturb it...
 - ...so it settles into one state
 - ...or the other.

We think the Branch machine inside the 6502 saw a metastable state, and read it *one way* in the first cycle of preparation, and the *other way* in the 2nd – hence the mad, mixed-up dest address!

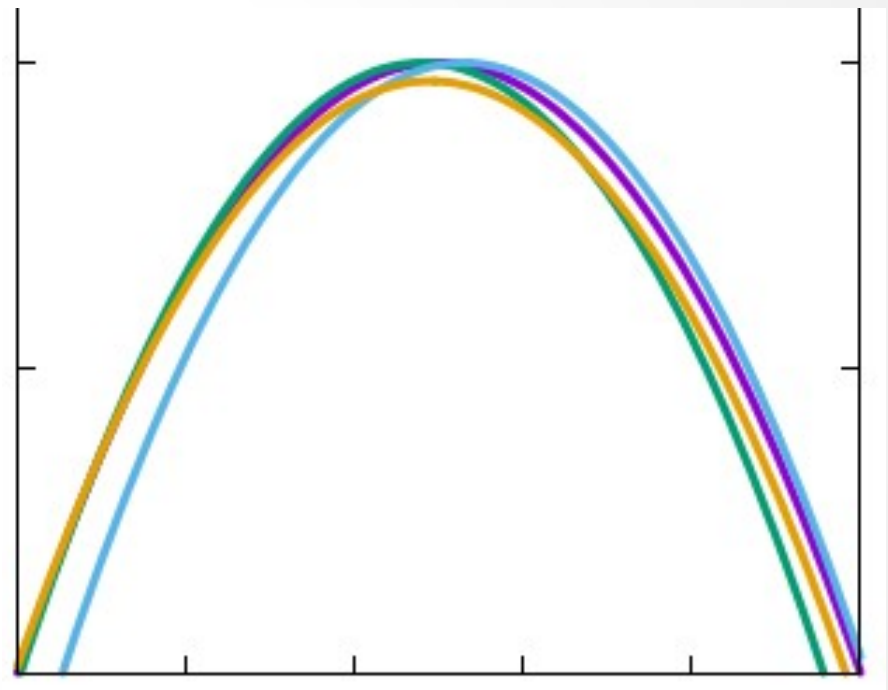


The “twos” bug – fixes (7)

Due to manufacturing variation and layout, no two flip-flops are the same. Every bit in every register is unique. So you have a set of energy curves....

By moving the data from one register to another, and back (say) it gets filtered through several sets of hardware – they won't have the same metastable point – so the value will fall off to one or other extreme.

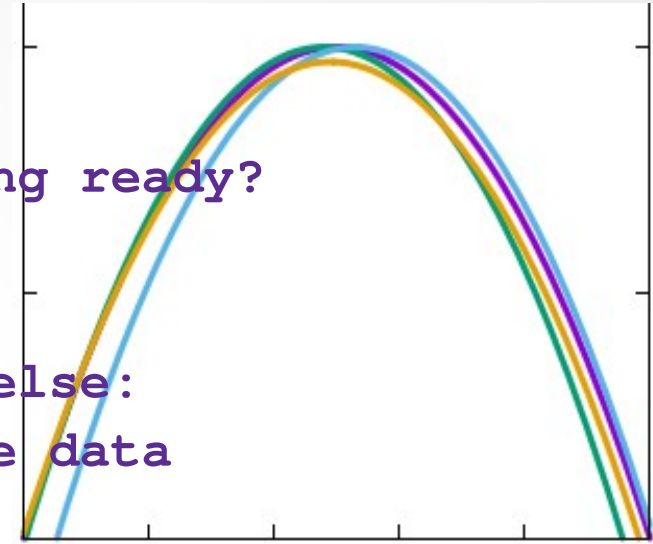
Before the mad bad Branch instruction sees it!



The “twos” bug – fixed (8)

The fix is just to add a delay *and* transfer the data around the registers a bit....

```
- BEC4 poll1: LDA 0xFEE7 ; anything ready?  
- BEC7 TAX  
- BEC8 TXA  
- BEC9 BPL poll1 ; nope; else:  
- BECB LDA 0xFEE6 ; get the data  
- BECE ... ..
```



- Original set the **N** (negative) bit in the Flags register, then immediately branched on that register bit using **BPL** (branch if positive) instruction.
- New version loads into Accumulator, then moves to X, then moves back with the side-effect of setting the **N** bit. That's 3 physically distinct places the value has been stored prior to using it in the branch. So P(fail) is *cubed*.
- Changes from O(4 hours) observed – 1 in some 10s of millions of tube transfer ops – to 1 in 10²⁰ order of magnitude – literally to O(**millions of years**)
 - At a cost of two extra 1-byte instructions, we thought that sufficient.

ADFS for the Electron



- Well, it just worked. Mostly.
- And quite right too; that was the point of all those complicated “sideways” calls and properly designed OS interfaces with indirection vectors and all that.
- Properly written software was portable!
- But... but but... ***COMPACT** would occasionally corrupt the **ADFS** file system! Badly!
 - This was bad news. We wanted it out for Xmas!
 - So, SWAT team or “Tiger team” time....
- There were other issues, wasn't just me :-)

Electron *COMPACT bug (1)

TopExpress were consulted. We set up in their offices on Bridge St – now La Margherita very lovely pizza and Italian place.



Electron *COMPACT bug (2)

- So, some all-night debugging sessions with Pizza (from downstairs perhaps) provided by legendary Commander Paul Bond.
- How could it go wrong? Nothing is different!
- But go wrong it did. Repeatably. But only when using the screen memory as is the default. Then... at about 4 in the morning... Jon T said...
 - *“You do know the cursor is in software don't you?”*
- I did then. Aargh. A timer interrupt XORs the screen memory regularly to show the flashing cursor.
 - **...so corrupting the data ADFS is moving about!**

Electron *COMPACT bug (3)

- To test this, we did the `VDU 23, 1, 0, ...` whatever it was to turn the cursor off, and re-ran the test.
- It was fine. Phew. There was space in the ROM to fix it – issue the VDU command from inside `ADFS` when using screen RAM for `*COMPACT`.
- But, the candidate ROM image at that time had been very thoroughly tested by then. It was a good candidate, and the fix for this issue was simple. There was time pressure. So:
- The ROM without the fix was shipped. Pity.
 - (a BASIC program was supplied to do `*COMPACT` safely for you)
- Sadly, because of issues with manufacturing the ULA (I think), the Elk was late for Xmas anyway...

Project “A” - Top Secret!

- It was the ARM CPU project.
 - The only thing I ever worked on which actually, arguably, bettered the lot of humankind.
- We all signed a sort of internal NDA, no talking even to colleagues unless they had also been indoctrinated into Project A. Strange, thrilling, stuff – very exciting!
 - The Museum has one of these letters.
 - (Project “B” was the Master 128 with all the sideways RAM and like HW extensions)
- ***So how do you make a completely new CPU?***

How to build an ARM - ISA

- The design was a plaintext document with much ASCII art for instruction layouts and register set description and suchlike
- <https://www.wss.co.uk/pinknoise/ARMinstrs/ARMinstrs.html> has some of the flavour of this:

Data Processing Instructions

```
xxxx000a aaaSnnnn ddddcccc ctttmmm Register form
xxxx001a aaaSnnnn ddddrrrr bbbbbb Immediate form
```

aaaa	Assembler	Meaning	P-Code
0000	AND	Boolean And	Rd = Rn AND Op2
0001	EOR	Boolean Eor	Rd = Rn EOR Op2
0010	SUB	Subtract	Rd = Rn - Op2
0011	RSB	Reverse Subtract	Rd = Op2 - Rn
0100	ADD	Addition	Rd = Rn + Op2
0101	ADC	Add with Carry	Rd = Rn + Op2 + C
0110	SBC	Subtract with carry	Rd = Rn - Op2 - (1-C)
0111	RSC	Reverse sub w/carry	Rd = Op2 - Rn - (1-C)
1000	TST	Test bit	Rn AND Op2
1001	TEQ	Test equality	Rn EOR Op2
1010	CMP	Compare	Rn - Op2
1011	CMN	Compare Negative	Rn + Op2
1100	ORR	Boolean Or	Rd = Rn OR Op2
1101	MOV	Move value	Rd = Op2
1110	BIC	Bit clear	Rd = Rn AND NOT Op2
1111	MVN	Move Not	Rd = NOT Op2

Mode	Registers available
USR	R0 - R15
FIQ	R0 - R7, R8_FIQ - R15_FIQ
IRQ	R0 - R12, R13_IRQ - R15_IRQ
SVC	R0 - R12, R13_SVC - R15_SVC

How to build an ARM – design

- Emulation! Emulators, several.
 - to express the design in different ways, with different abstractions; independent implementations verify the documentation has no ambiguities
- And tests. A whole test suite, for every nook and cranny and corner
 - to run on each and every emulator and simulator
 - to check they agree with each other
 - penultimately, to agree with the simulation of the hardware; and
 - ultimately, to validate the actual chip!
- And a chip design team of course, to make silicon

ARM Emulators

- Emulation! Emulators, several
 - Remember this is all in an environment of BBC micros!
- One in BASIC from Sophie and Steve
 - this was first, and kind-of defined what was right
- Another in BASIC that more matched the chip architecture, the internals block-by-block and clock-by-clock (I think, not that sure)
- One in 6502 Assembler from Jon T, for speed of testing the tests and other code
- The hardware simulator on the Apollo workstations, downstairs in the Silver Building with the scary chip-design people

ARM Emulators (2)

....in BASIC....

- BASIC ISA emulator
 - this was where we verified that you could do useful things with this strange new ISA – eg. memcopy(), strcmp(), lookups, lists
 - tested viability of various algorithms in ARM
 - it was slow, but that wasn't important
 - lots of sanity checking
 - evaluation of usefulness of odd instructions – RSB, CMN
- Chip Architecture emulator in BASIC
 - this echoed the chip architecture, the internals block-by-block and clock-by-clock (I think, not that sure) [very slow, 100s of Hz]
 - it validated the block design of the silicon
- Also for considering feasibility of new instructions that weren't originally decided: RSB and CMN for example, I do remember.
 - RSB in particular implied a new data path in the ALU – feasible? Yes.
 - **dest := K - rn** as opposed to the usual **dest := rn - K**

ARM Emulators (3)

- 6502 Assembler, ran on Tube® 6502 co-processors
 - Written by Jon T, for speed of testing the tests and other code
 - Written according to the ISA Spec – another test for ambiguity in that
 - Usefully fast - 10's of kHz +
- The hardware simulator on the Apollo workstations where the Silicon was laid out
 - sloooow.... 1/10 Hz?
 - Validated the actual on-chip layout of polygons and transitors – really the most important thing ever!

ARM Emulators - runtime

- All these emulators offered the most trivial run-time services:
- Start at address zero
- Jump to exception vectors on exceptions
 - arguably this is part of implementing the ISA, but anyway...
- One Software Interrupt is handled:
 - **SWI 0, "Hello World\n"**
 - somehow shows you the text message
- Later, external "hardware" to cause IRQ and NMI
 - after variable time delays
 - controlled by co-processor instructions

Writing for ARM

- All test, indeed all software for ARM was written in assembler language
- Acorn previous had several assemblers for 6502, themselves written in 6502 assembly language, such as UASM, MASM
 - MASM in particular had good Macro ASseMbly features

```
MOV      Rd, #0
ADDEQS  Rd, Rn, Rm, ASL Rc
ANDEQ   Rd, Rn, Rm
TEQP    Pn, #&80000000
CMP     Rn, Rm
```

- So Jon T made ARMASM (IIRC) from MASM, keeping the same good macro assembly features
 - Macros are like #define macros in C (well, the C preprocessor)
- I don't recall how much I macro-ized test stuff tho

The ARM test suite – how?

- How to test something where you know *nothing* about what already works?
- Build from nothing!
- ...because you have to assume *something!*
- The fundamental something is that instructions are executed in the right order and that the SWI 0, .. does indeed show you a message
- But you can test this! Just make the most fundamental test be “Hello World!” sitting right there at address zero, the reset vector
- I *think* the initial tests actually sat at zero, but they maybe depended on one branch working, to jump over the other vectors...

The ARM test suite - bootstrap

- Build from almost nothing:
 - Load immediate zero; check the Z flag is set
 - Load immediate 1; check the Z flag is **not** set
 - Compare a zero with zero; is Z set?
 - Compare a zero with 1; is Z **not** set?
 - Ditto with a selection of other numbers.
 - Fixed ADDs, compare with immediate answer
 - and wrong answer
 - Fixed ADC, SUB, SBC, RSB....
 - All written out in full, because you don't know that other instructions work enough (yet) to use a table, and a loop, and memory – all this sort of thing:
 - `MOV R1, #17; ADD R2, R1, #20; CMP R2, #37; BEQ OK238; ...`
 - So lots of little programs that just use `SWI 0; ="Bad Subtract", 0;` (just like BRK on 6502) for error messages/printout.
 - Don't remember how we tested SWI itself! Probably `SWI <nonzero>`.



The ARM test suite - corners

- These ALU tests are all framed also to test the branch instruction, taken (and sometimes not-taken)
 - if BEQ does nothing, get a fail message, so style:
 - `MOV R7, #77; CMPS R7, #77; BNE fail77;`
 - `BEQ aok77;`
 - `fail77: SWI 0, "FAIL77\n"`
 - `aok77: ...next test`
- Also, explicit, tedious longhand tests for things like, are registers distinct? Is the PC doing what it should?
 - interleave a lot of similar other tests
 - “interesting” constants: 2^n+k , 2^n-1 ...&c &c
 - An observation about RAM: you can connect the data and address lines in any order you like! The register bank is just RAM. So be aware.... only PC=R15 and LR=R14 are special.

The ARM test suite - longhand

- Once you think you have a working ALU and branching ability, you can do the same sorts of things with load and store, again slowly building on fundamentals, treating some RAM a little like you treated the register bank in the earlier tests.
 - Check for distinction of locations
 - Try patterns that would spot, for example, address lines being used or OR'd or AND'd &c with data
 - After simple LDR, STR, with all the addressing modes – and checking writeback or not, move onto LDM and STM, mix the two ways

One example in detail - Shifter

```
T2-32 $Revision: 3.2 $
***** Testing ASL *****
MOV, C=1, data=1
MOV, C=0, data=1
MOVS, C=1, data=1
MOVS, C=0, data=1
MOV, C=1, data = &7FFFFFFF
MOV, C=0, data = &7FFFFFFF
MOVS, C=1, data = &7FFFFFFF
MOVS, C=0, data = &7FFFFFFF
***** Testing LSR *****
MOV, C=1, data=1
MOV, C=0, data=1
MOVS, C=1, data=1
MOVS, C=0, data=1
MOV, C=1, data = &7FFFFFFF
MOV, C=0, data = &7FFFFFFF
MOVS, C=1, data = &7FFFFFFF
MOVS, C=0, data = &7FFFFFFF
***** Testing ASR *****
MOV, C=1, data=1
MOV, C=0, data=1
MOVS, C=1, data=1
MOVS, C=0, data=1
MOV, C=1, data = &7FFFFFFF
MOV, C=0, data = &7FFFFFFF
MOVS, C=1, data = &7FFFFFFF
MOVS, C=0, data = &7FFFFFFF
***** Testing ROR *****
MOV, C=1, data=1
MOV, C=0, data=1
MOVS, C=1, data=1
MOVS, C=0, data=1
MOV, C=1, data=&7FFFFFFF
MOV, C=0, data=&7FFFFFFF
MOVS, C=1, data=&7FFFFFFF
MOVS, C=0, data=&7FFFFFFF
***** Testing RRX *****
```

```
***** Testing RRX *****
MOV, C=0, data=1
MOV, C=1, data=1
MOVS, C=0, data=1
MOVS, C=1, data=1
MOV, C=0, data=&FFFFFFFF
MOV, C=1, data=&7FFFFFFF
MOVS, C=0, data=&FFFFFFFF
MOVS, C=1, data=&7FFFFFFF
Test large register shifts
***** ROR *****
***** ASL *****
***** LSR *****
***** ASR *****
Special cases of immediate shifts
Testing all imm. ASLs, data=1
Testing all imm. ASLs, data=&FFFFFFFF
Testing all imm. LSRs, data=1
Testing all imm. LSRs, data=&FFFFFFFF
Testing all imm. ASRs, data=1
Testing all imm. ASRs, data=&7FFFFFFF
Testing all imm. RORs, data=1
Testing all imm. RORs, data=&7FFFFFFF
*** Const Shifter ***
Sucessfully completed 00000006 of 6 Test
All OK $Revision: 3.2 $

** TEST PASSED OK **
```

The ARM test suite - coverage

- Thus, in small steps, you bootstrap understanding of what works – has been shown to work – so you can run loops with table-driven data for the tests, and so on.
- Ultimately, the most complex tests featured a memory-mapped timer that could make interrupts, after some number of ticks.
 - Simple enough to add a software simulation of *that* to all the CPU simulators
 - after all they already had RAM, and a lack of it, for testing traps of bad addresses &c.

The ARM test suite - interrupts

- So the ultimate tests were a selection of the previous “simple” tests, run 100s of times, with
 - an IRQ happening at 1,2,3...N ticks into the test
 - a FIQ happening at 1,2,3...N ticks into the test
 - two nested for-loops, so taking N-**cubed** time, *wrt* the length of the underlying test.
- whilst verifying the the IRQ and FIQ were all taken correctly, **and** that the original test passed in every case
- So proving that every instruction was safely interruptable **and** restartable
 - Harder to run the exact same binary test on the actual silicon though – the timer peripheral was not the same⁴⁹

The ARM test suite - application

- So we had lots of tests; the software simulators verified the tests for mutual consistency
 - Fast sim would run the simple ones in a couple of minutes; the FIQ/IRQ versions in half an hour or so
 - Block sim would take hours for the simple cases, overnight for the FIQ/IRQ
 - Actual chip layout sim took days for even the simple cases; running all the FIQ/IRQ ones turned out not to be feasible before tape-out
 - We did definitely once run the whole set of simple tests on the layout sim, plus a few of the FIQ/IRQ ones to boost confidence in the design
 - We may have run the lot in the time awaiting silicon back from the fab – I don't remember

The ARM test suite – in use

- The tests did occasionally catch differences between the various emulators, but I don't remember any details
 - Then it was over to Sophie and Steve to work out which was right, and occasionally to Jon T if the assembler hadn't quite done what was intended
- We ran as many tests as we could, as often as we could, as new versions of the emulators were made – especially the one reflecting the internal block-level layout of the chip
 - I think there were choices available as to which parts were emulated in hw detail versus per ISA definition, ie. writing an “add” in BASIC vs in bits.

Silicon! Actual ARM!



- And astonishingly, they worked!
- And the rest is history!
- Well not really, there was one minor bug that I *think* my tests found, it was something like the barrel-shifter OR'd together bit 0 and bit 31 when setting the C (carry) bit in the extended rotate instruction
 - (RRX, 33-bit rotate by one)
- But anyway, the shifter was tricky...
 - They fixed it for the next silicon, v2 easily.

After I left Acorn...



- You'd think that's the end of my involvement, but No! I got the last laugh...
- By the mid-1990s ARM had spun out and was busily being ARM and licencing chip designs to customers. ARM tested all the chip designs in-house with my test suite, still – very likely without changing the binaries at all, because they were “known good” or “Golden”
- But one customer had to be different...

• DEC!



PDP-11

VAX 11/780

DEC and StrongARM/Xscale

- Mid-1990s, DEC – Digital Equipment Corp - licenced the ARM Architecture (ISA) from ARM, a very unusual arrangement.
 - ARM wanted to grow the ecosystem
- DEC implemented their own ARM-compatible CPUs “**StrongARM**” (later “**Xscale**”) entirely from scratch
 - to use their own expertise in fast chips and fast memory-access hardware. Fair enough.
- But how to test them? ARM released the test suite, in binary form, for DEC to check that their chips were actually doing the instructions right.
 - ARM may have done this for many other customers – but other customers’ chips, being designed by ARM, always worked right.
- Problem was.... they weren’t. It was the shifter. Again.
- This was a problem in more than one way...

Language!

- When I wrote the ARM CPU test suite, it was for my mates, in a team of about 10 of us. Very informal.
- The message reporting a some problems was none too polite, a bit swearsy, and
- DEC, American suits all, were very strait-laced and serious.... and not at all happy with the swears.
 - None the less, their shifter **was** fucked. The test was right!
- In due course, an all-hands email went around ARM, lamenting this disaster, "*how could ARM be so unprofessional? This looks so bad! Who's to blame?*"
- Older hands at ARM laughed a lot, and pointed out who wrote it, and that it hadn't been modified in 10 years because other customers' shifters all worked...
- ...I laughed a lot too; I never even worked for ARM. Ever!

After we had ARMs

- I worked on a variety of projects in Acorn:
- ARX with Acorn Research Centre in Palo Alto
 - Interesting, weird people, ex-PARC folks
 - Didn't get the idea of making a product though...
 - So another team did Arthur then RISC OS *pdq* just like Jon T, Jez, Brian and I (and others) had tried to make happen a year or so before... we could have had a BBC-MOS like environment going in 18 months had we started early. (admittedly without the GUI)
 - So ARX was a waste of time, sadly – a bridge too far
- Then the floating-point co-processor – the “floating poo” (FPU) – it worked, but not well enough, but that was quite fun.
 - Visited Olivetti and got told off for importing “typewriter parts”
- Then RISCiX – port of BSD 4.2 Unix, then 4.3
 - that worked and was a lot of fun... TCP/IP over EcoNet!
- And then I left Acorn in 1990 to go do other pointless stuff for other pointless companies.

That's all – Questions?

- Questions?
- ***Relevant*** questions?



My Time at Acorn

From Market Hill to the Silver Building

Hugo Tyson

ROUGOL June Meeting 2025-06-16

The Duke of Sussex, nr. Waterloo

All images are copyright© whoever owns them on the web – not me!