A Career in Research: Mike Gordon and Hardware Verification

Lawrence C Paulson

Automated Reasoning Workshop 2018

The world of computing in 1975

16MB a *colossal* amount of memory

minicomputers,
e.g. 16KB memory

mainframe disk, about 5MB

The world of theory, 1975–80

C. A. R. Hoare. An axiomatic basis for computer programming
Communications of the ACM **12** (10), Oct. 1969

Dana Scott. Outline of a mathematical theory of computation.
Technical Report PRG-2, University of Oxford, Nov. 1970.

denotational semantics and fixed-point theory

operational semantics just emerging

R. Milner. *A Calculus of Communicating Systems*. Springer, 1980.

type theory emerging

process algebras

Edinburgh LCF (1975–1979)

M. J. C. Gordon, R. Milner, and C. P. Wadsworth.
Edinburgh LCF: A Mechanised Logic of Computation.
LNCS 78. Springer, 1979.

- The first real *proof assistant* (for computation theory)
- Introducing ML (the first *polymorphic* functional language)

With so many nascent fields, what did Mike decide to do?

Software is being solved, so let's verify *hardware*.

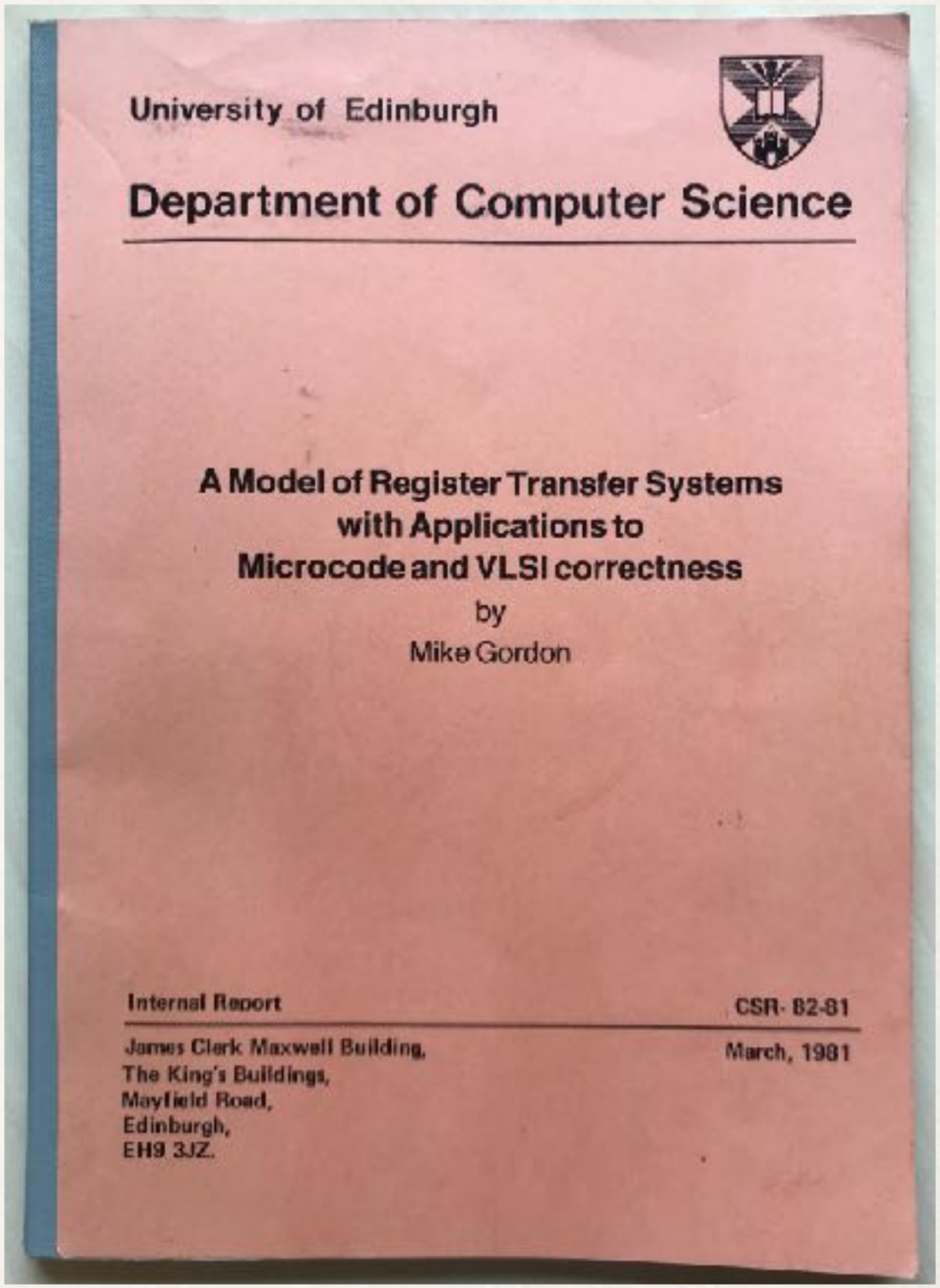
He talked to the hardware experts at Edinburgh

and sketched out some theory

… and designed his own computer!

A model of register transfer systems with applications to microcode and VLSI correctness.

Technical Report CSR-82-81, University of Edinburgh, Mar. 1981.



the circuitry

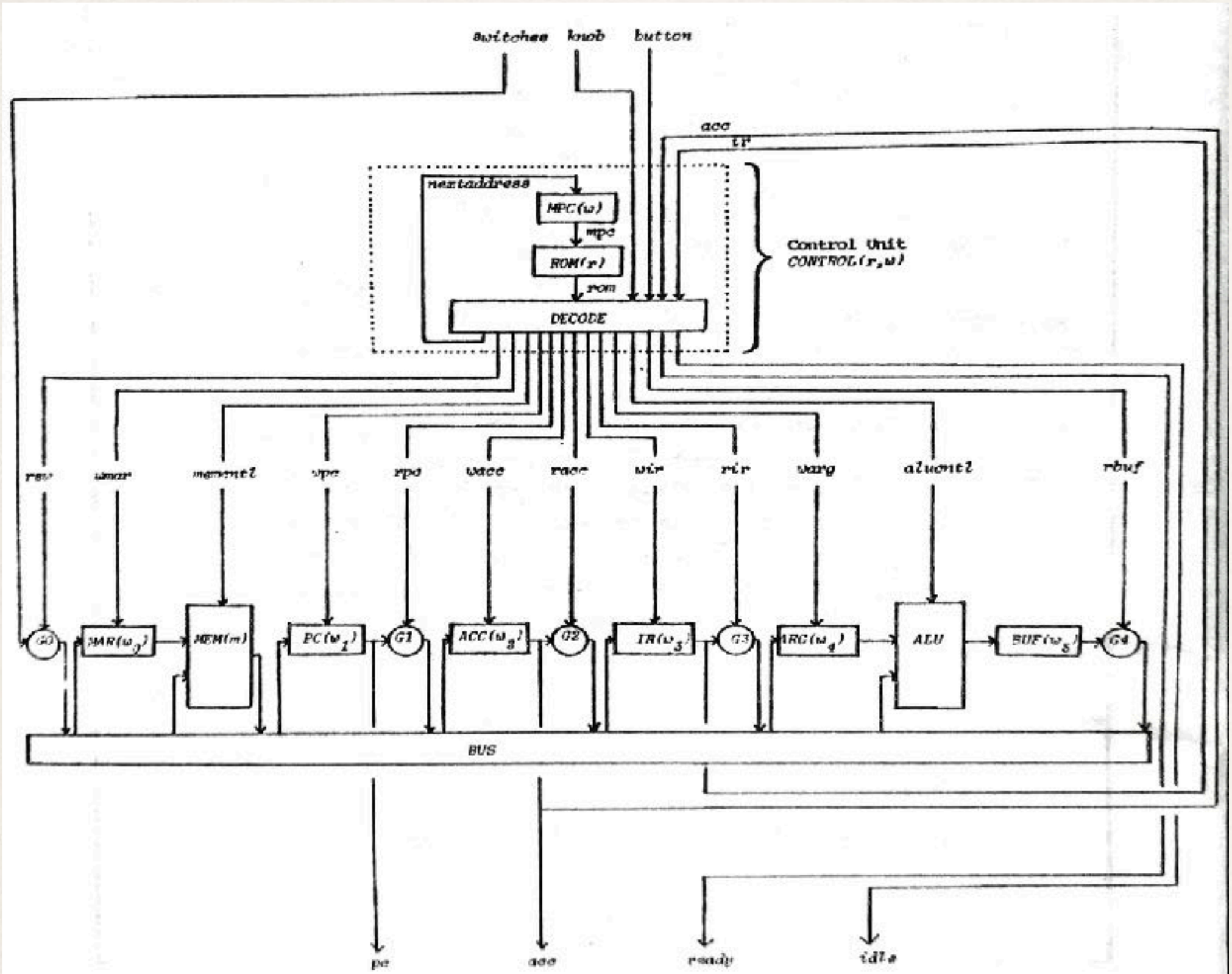


Fig. XIV: The host machine for implementing the computer

a microprogram

ROM address	Microinstruction		Comments	
0	ready,idle;	button→1,0	branch to 1 if button pressed, otherwise loop	
1	;	knob→1	decode knob position	
2	rsw,wpc ;	0	switches→PC	
3	rsw,wacc ;	0	switches→ACC	
4	rpc,wmar ;	7	PC→MAR	
5	ready ;	button→0,6	begin fetch-decode-execute cycle	
6	rpc,wmar ;	8	PC→MAR	fetch
7	racc,write;	0	ACC→MEM(MAR)	fetch
8	read,wir ;	9	MEM(MAR)→IR	fetch
9	;	opcode+10	decode	
10	;	0	halt	
11	rir,wpc ;	6	JMP: IR→PC	
12	;	acc=0→11,17	JZRO:	
13	rdoc,warg ;	19	ADD:ACC→ARG	
14	racc,warg ;	22	SUB:ACC→ARG	
15	rir,wmar ;	24	LD:IR→MAR	
16	rir,wmar ;	25	ST:IR→MAR	
17	rpc,inc ;	18	PC+1→BUF	increment program counter
18	rbuf,wpc ;	5	BUF→PC	increment program counter
19	rir,wmar ;	20	IR→MAR	
20	read,add ;	21	ARG + MEM(MAR)→BUF	
21	rbuf, wacc ;	17	BUF→ACC	
22	rir,wmar ;	23	IR→MAR	
23	read,sub ;	21	ARG-MEM(ARG)→BUF	
24	read,wacc ;	17	MEM(MAR)→ACC	
25	racc,write;	17	ACC→MEM(MAR)	

Fig. XVI Microinstruction in control units' *ROM*: the microprogram *mucode*

internal specifications

$HOSTMACHINE = (S,\ out,\ next)$

$S = Word[5] \times Mem \times Word[13] \times Word[13] \times Word[16] \times Word[16] \times Word[16] \times Word[16]$

MPC, memory, MAR, PC, ACC, IR, ARG, BUF

$out((knob = k,\ button = b,\ switches = sw),\ (w, m, w_0, w_1, w_2, w_3, w_4, w_5))$

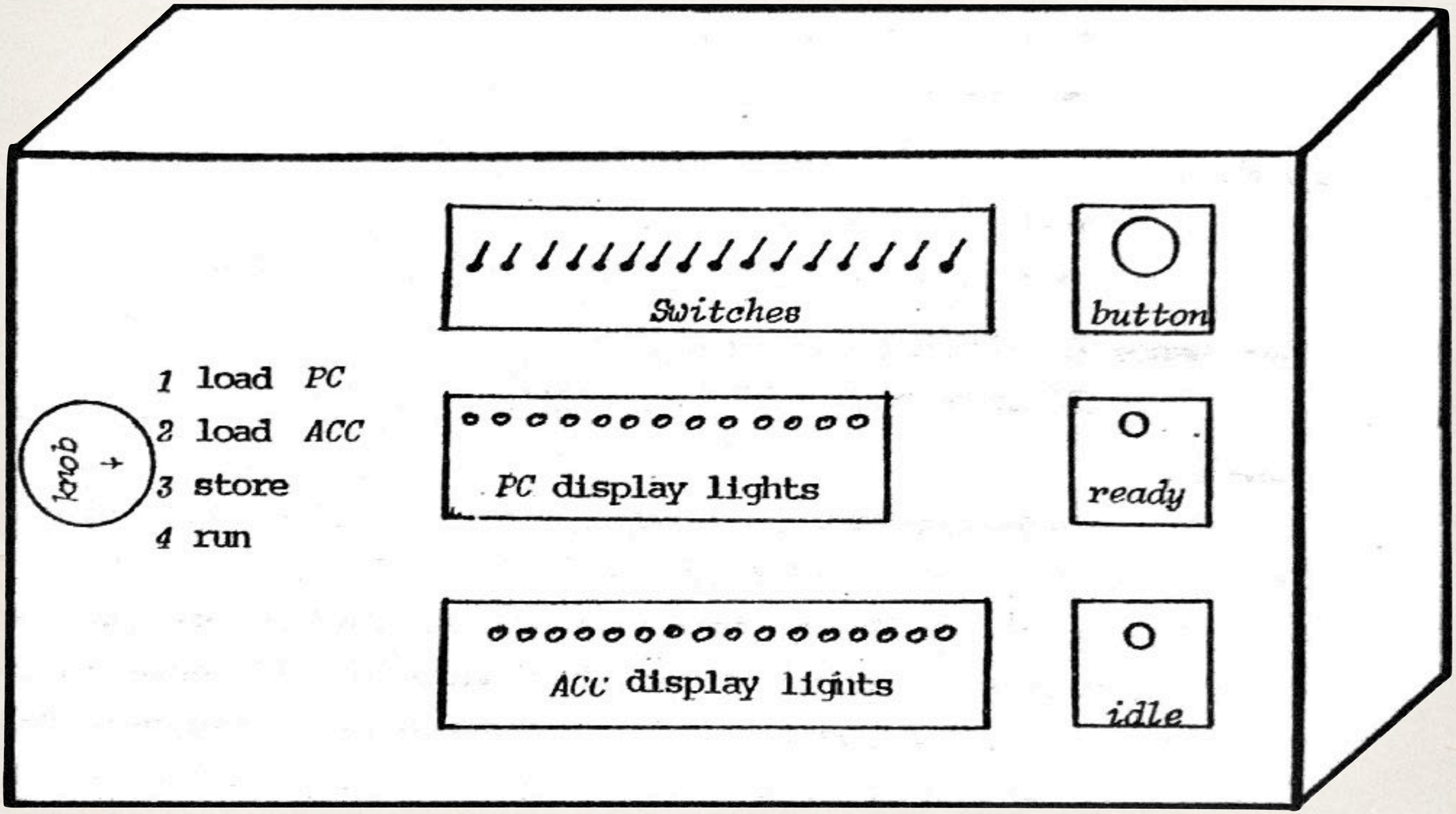
$= (pc = w_1, acc = w_2,\ ready = (w = 0 \vee w = 5 \to 1, 0),\ idle = (w = 0 \to 1, 0))$

$next((knob = k,\ button = b,\ switches = sw), (w, m, w_0, w_1, w_2, w_3, w_4, w_5))$

$=$

$w = 0 \to ($	$b \to 1, 0)$	m	w_0	w_1	w_2	w_3	w_4	w_5),
$w = 1 \to ($	0	m	w_0	w_1	w_2	w_3	w_4	w_5),
$w = 2 \to ($	0	m	w_0	$sw\langle 0:12\rangle$	w_2	w_3	w_4	w_5),
$w = 3 \to ($	0	m	w_0	w_1	sw	w_3	w_4	w_5),
$w = 4 \to ($	7	m	w_1	w_1	w_2	w_3	w_4	w_5),
$w = 5 \to ($	$b \to 0, 6)$	m	w_0	w_1	w_2	w_3	w_4	w_5),
$w = 6 \to ($	8	m	w_1	w_1	w_2	w_3	w_4	w_5),
$w = 7 \to ($	0	$m[w_2/w_1]$,	w_0	w_1	w_2	w_3	w_4	w_5),
$w = 8 \to ($	8	m	w_0	w_1	w_2	$m(w_0)$,	w_4	w_5),
$w = 9 \to (w_3\langle 13:15\rangle + 10$		m	w_0	w_1	w_2	w_3	w_4	w_5),
$w = 10 \to ($	0	m	w_0	w_1	w_2	w_3	w_4	w_5),
$w = 11 \to ($	5	m	w_0	$w_3\langle 0:12\rangle$,	w_2	w_3	w_4	w_5),
$w = 12 \to ((w_2 = 0 \to 11, 17)$,		m	w_0	w_1	w_2	w_3	w_4	w_5),
$w = 13 \to ($	19	m	w_0	w_1	w_2	w_3	w_2	w_5),
$w = 14 \to ($	22	m	w_0	w_1	w_2	w_3	w_2	w_5),
$w = 15 \to ($	24	m	$w_3\langle 0:12\rangle$,	w_1	w_2	w_3	w_4	w_5),
$w = 16 \to ($	25	m	$w_3\langle 0:12\rangle$,	w_1	w_2	w_3	w_4	w_5),
$w = 17 \to ($	18	m	w_0	w_1	w_2	w_3	w_4	$w_1\langle 0:12\rangle + 1)$,
$w = 18 \to ($	5	m	w_0	$w_5\langle 0:12\rangle$,	w_2	w_3	w_4	w_5),
$w = 19 \to ($	20	m	$w_3\langle 0:12\rangle$,	w_1	w_2	w_3	w_4	w_5),
$w = 20 \to ($	21	m	w_0	w_1	w_2	w_3	w_4	$w_4 + m(w_0)$),
$w = 21 \to ($	17	m	w_0	w_1	w_5	w_3	w_4	w_5),
$w = 22 \to ($	23	m	$w_3\langle 0:12\rangle$,	w_1	w_2	w_3	w_4	w_5),
$w = 23 \to ($	21	m	w_0	w_1	w_2	w_3	w_4	$w_4 - m(w_0)$),
$w = 24 \to ($	17	m	w_0	w_1	$m(w_0)$,	w_3	w_4	w_5),
$w = 25 \to ($	17	$m[w_2/w_0]$,	w_0	w_1	w_2	w_3	w_4	w_5))

even the industrial design!



- Mike wrote *21 pages* on the computer alone
- 75 pages on a theory of hardware verification, from gates to computers
- Already many key insights by 1981, e.g. to treat *combinational* (stateless) devices like *sequential* ones.

Three ideas for modelling devices

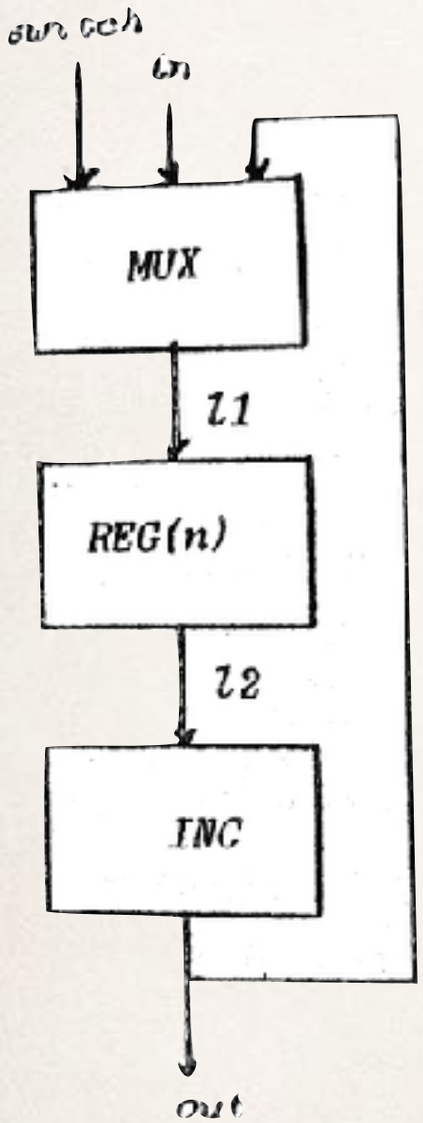
- 1981: recursive domain equations (*too complicated*)

- 1983: Logic for Sequential Machines, or LSM: Mike's hardware formalism, loosely based on CCS (*too ad-hoc*)

- 1985: higher-order logic! And all devices as **relations**.

A concerted effort to *minimise* reliance on theory!

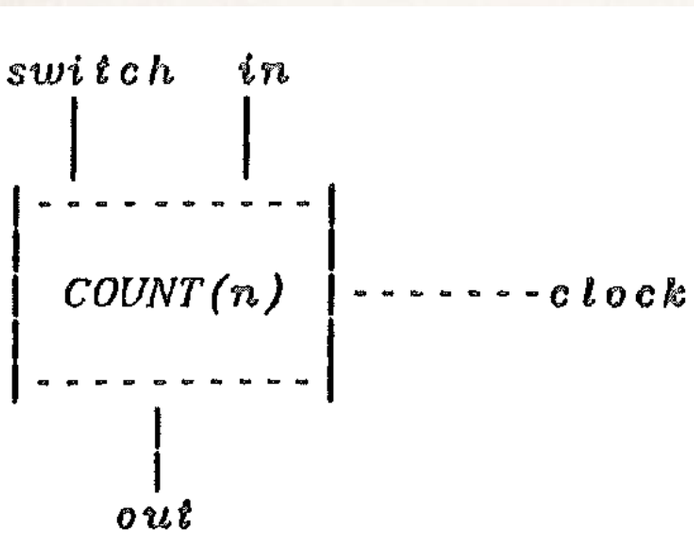
A counter using domains (1981)



$$COUNT(n) = \lambda(switch, in).(out = n + 1), COUNT(switch \rightarrow in,\ n + 1)$$

Fig. II. Behaviour of *COUNT(n)*

A counter in LSM (1983)



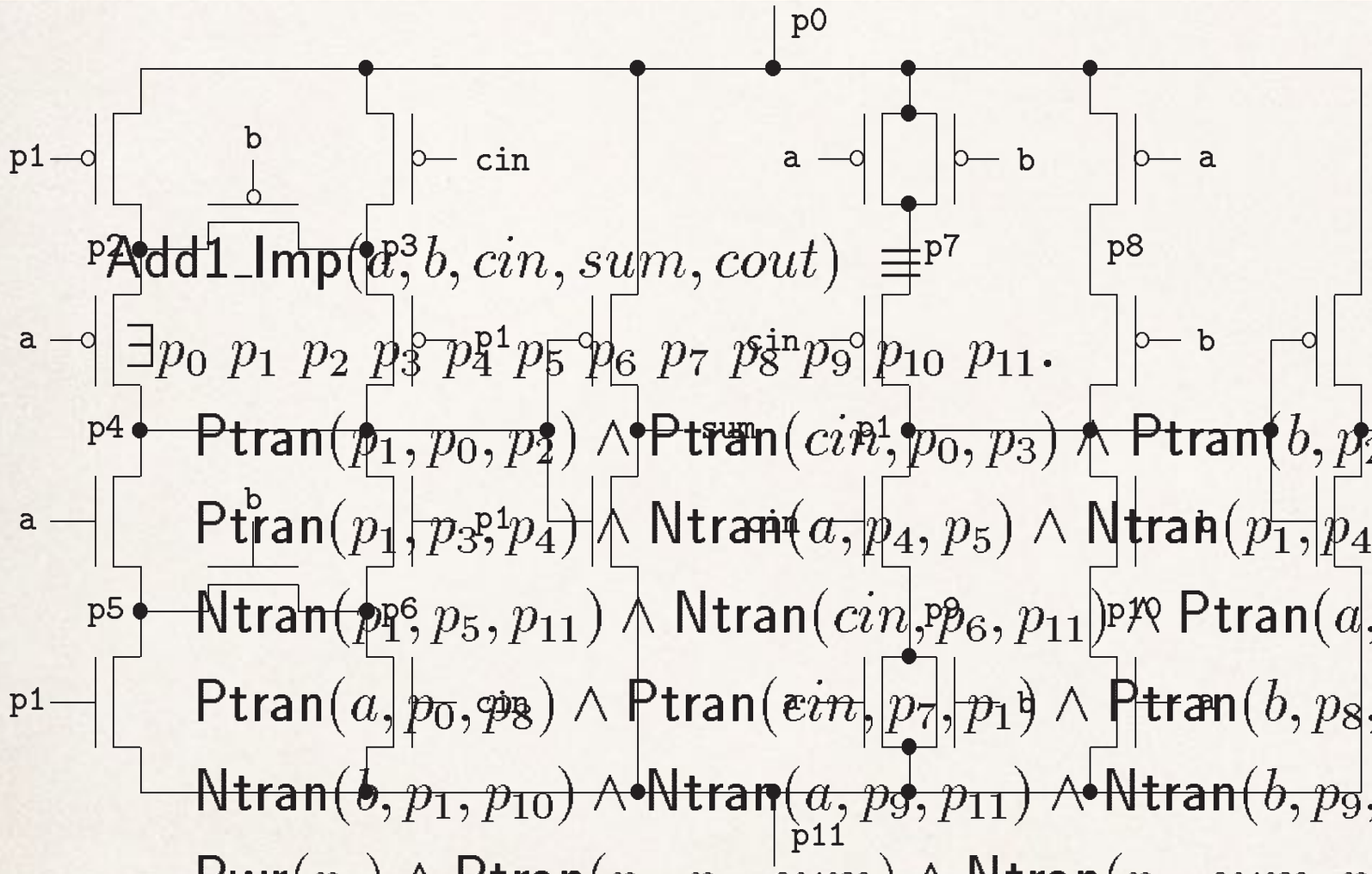
When the counter is clocked, the new value of the state variable *n* becomes *n+1* (i.e. the old value plus one) if *false* is being input on line *switch*, otherwise it becomes the value input on line *in*. We can express this by:

```
CLOCK(n) --> CLOCK(switch -> in | n+1)
```

In LSM the behaviour of the counter would be specified by the formula

```
COUNT(n) == dev{switch, in, out}. {out=n}; COUNT(switch -> in | n+1)
```

A CMOS full adder in HOL (1985)



$$\mathsf{Add1\_Imp}(a, b, cin, sum, cout)$$

$$\exists p_0\ p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ p_8\ p_9\ p_{10}\ p_{11}\cdot$$

$$\mathsf{Ptran}(p_1, p_0, p_2) \wedge \mathsf{Ptran}(cin, p_0, p_3) \wedge \mathsf{Ptran}(b, p_2, p_3) \wedge \mathsf{Ptran}(a, p_2, p_4) \wedge$$

$$\mathsf{Ptran}(p_1, p_3, p_4) \wedge \mathsf{Ntran}(a, p_4, p_5) \wedge \mathsf{Ntran}(p_1, p_4, p_6) \wedge \mathsf{Ntran}(b, p_5, p_6) \wedge$$

$$\mathsf{Ntran}(p_1, p_5, p_{11}) \wedge \mathsf{Ntran}(cin, p_6, p_{11}) \wedge \mathsf{Ptran}(a, p_0, p_7) \wedge \mathsf{Ptran}(b, p_0, p_7) \wedge$$

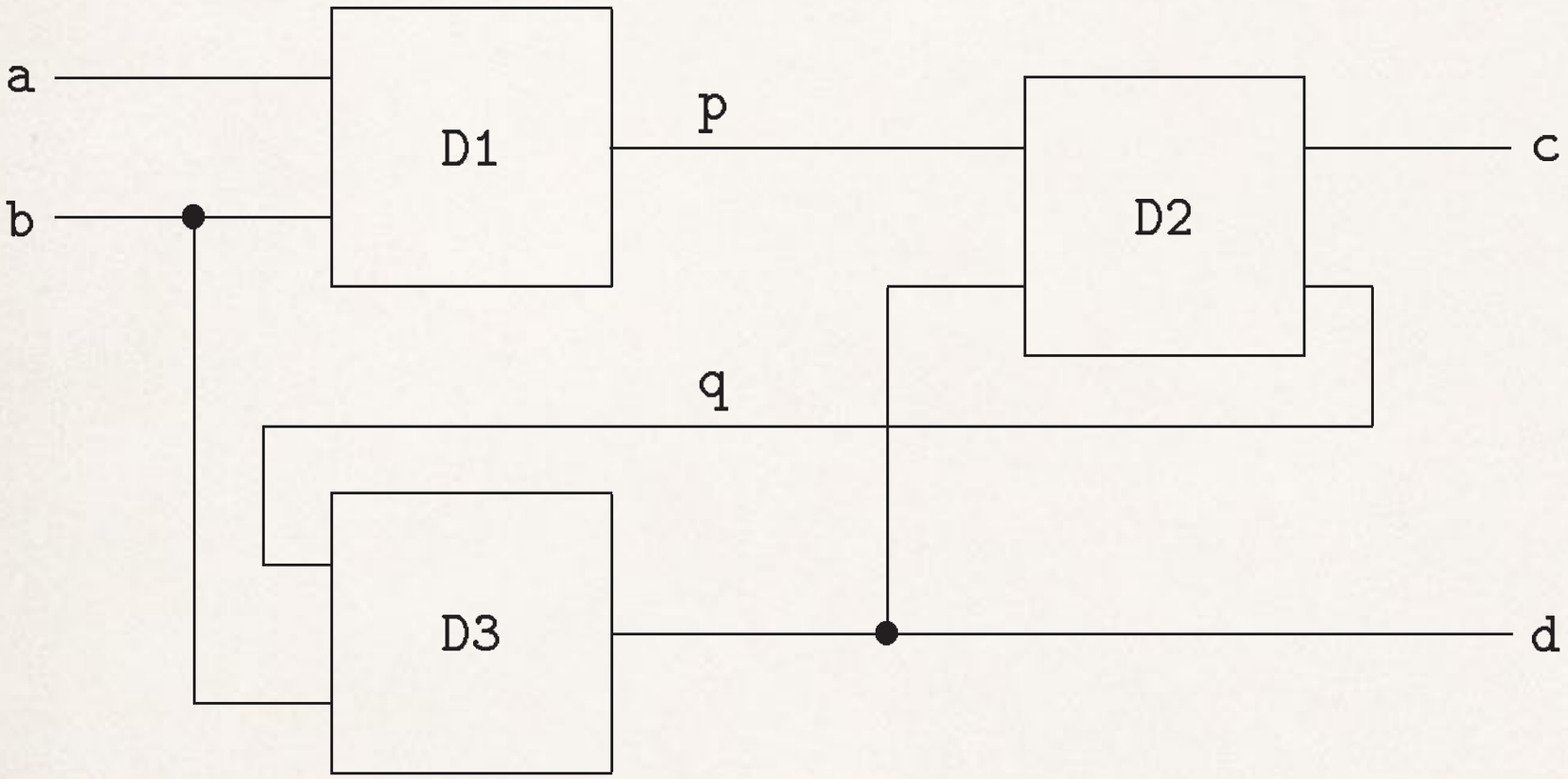
$$\mathsf{Ptran}(a, p_0, p_8) \wedge \mathsf{Ptran}(cin, p_7, p_1) \wedge \mathsf{Ptran}(b, p_8, p_1) \wedge \mathsf{Ntran}(cin, p_1, p_9) \wedge$$

$$\mathsf{Ntran}(b, p_1, p_{10}) \wedge \mathsf{Ntran}(a, p_9, p_{11}) \wedge \mathsf{Ntran}(b, p_9, p_{11}) \wedge \mathsf{Ntran}(a, p_{10}, p_{11}) \wedge$$

$$\mathsf{Pwr}(p_0) \wedge \mathsf{Ptran}(p_4, p_0, sum) \wedge \mathsf{Ntran}(p_4, sum, p_{11}) \wedge$$

$$\mathsf{Gnd}(p_{11}) \wedge \mathsf{Ptran}(p_1, p_0, cout) \wedge \mathsf{Ntran}(p_1, cout, p_{11})$$

The insight that devices are *relations*



$$\mathsf{D}(a, b, c, d) \equiv \exists p\ q.\ \mathsf{D}_1(a, b, p) \wedge \mathsf{D}_2(p, d, c) \wedge \mathsf{D}_3(q, b, d)$$

HO Logic was a radical choice!

"Unlike first-order logic and some of its less baroque extensions, second and higher-order logic have no coherent well-established theory; the existent material consisting merely of scattered remarks quite diverse with respect to character and origin."
(Van Benthem and Kees Doets, 1983.)

And we need a type of *n*-bit words, so we need dependent types, right?

WRONG.

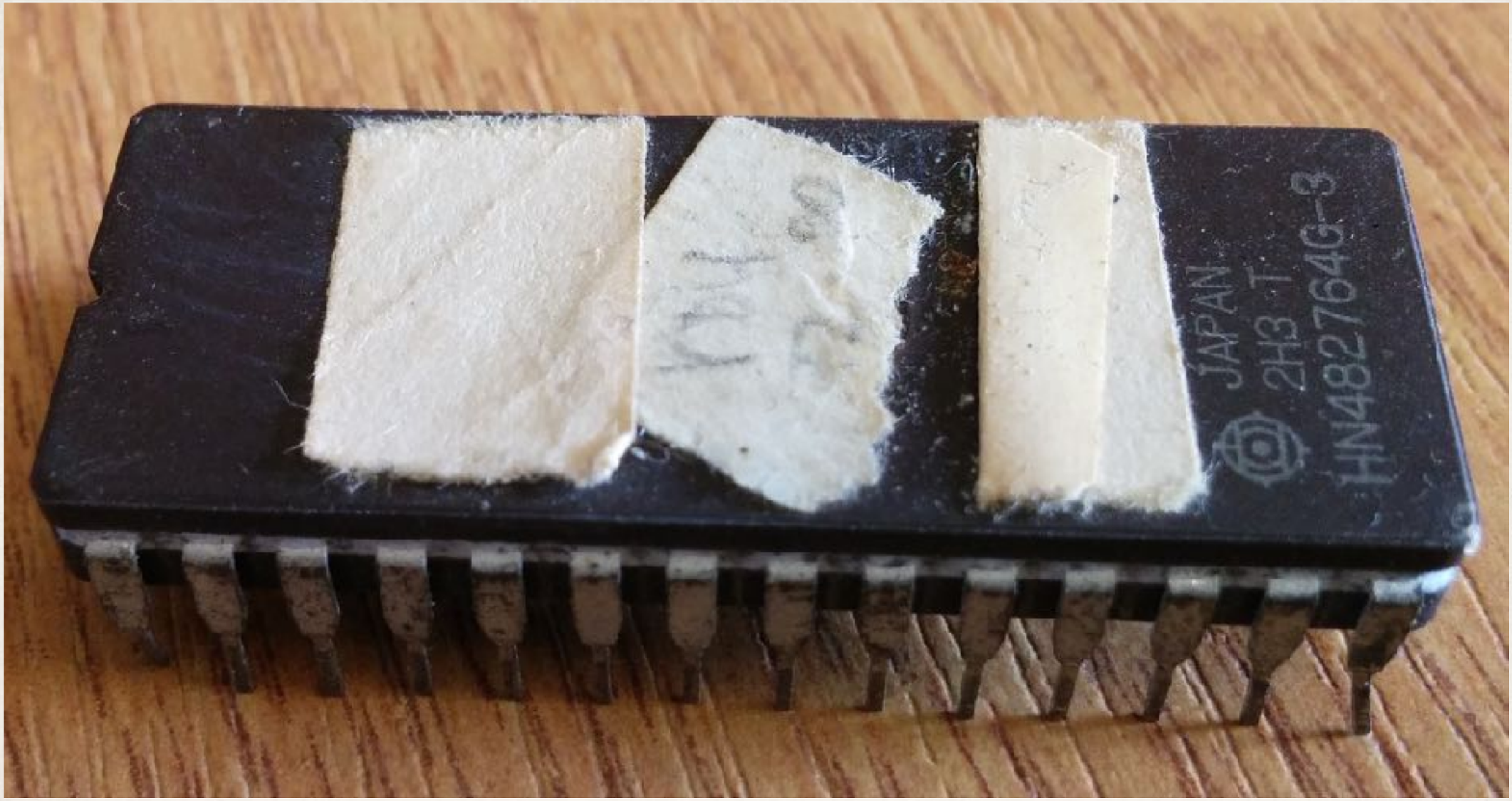
Verifying Mike's computer

- In 1983, using LCF_LSM

  "The entire specification and verification described here took several months, but this includes some extending and debugging of LCF_LSM … it would take me two to four weeks to do another similar exercise now. The complete proof requires several hours CPU time on a 2 megabyte Vax 750."

- In 1986, with Jeff Joyce, Graham Birtwistle, using HOL

- And — under the name *Tamarack* — by many others!



It was even fabricated!

Verifying a *real* computer: VIPER

- designed by a UK defence lab for military purposes
- specified by a series of abstract layers
  - equivalence of the *top three* proved by Avra Cohn
  - controversy due to exaggerated claims made by the chip's marketers

Hardware verification went from plan (1981) to realisation (1989) in eight years!

Some papers from that era

Mike Gordon. Proving a computer correct with the LCF LSM hardware verification system (1983).

Avra Cohn. A proof of correctness of the VIPER microprocessor: The first level (1987).

Avra Cohn. Correctness properties of the Viper block model: The second level (1988).

Brian T. Graham. *The SECD Microprocessor, A Verification Case Study*. Kluwer, 1992.

"Verification involves a pair of models that bear an uncheckable and possibly imperfect relation to the *intended design* and to the *actual device*."

–Avra Cohn, 1989

So what next?

"a long term project on verifying combined hardware/software systems by mechanized formal proof"

Mike Gordon, "Mechanizing programming logics in higher order logic", 1989

- Mike *derived* Hoare logic in HOL from the operational semantics of a programming language
- supporting the illusion by pretty-printing
- the first **shallow embedding** of one formalism within another

Whooshing to the year 2000...

Lots of PhDs on floating point arithmetic,
process calculi, BDDs, etc., etc….

Verifying the ARM6 processor

- a “commercial off-the-shelf” design
  - joint project with Graham Birtwistle at Leeds
  - verification by Anthony Fox at Cambridge
- *the ARM6 microarchitecture implements its ISA*
- And we have a *complete formal spec* of this machine

Verification and assembly language

(Magnus Myreen, working with Mike)

- Hoare-style logics for assembly languages
- decompilation of assembler to HOL (for 3 machines!)
- proof-producing translation from HOL to assembler

"verifying combined hardware/software systems"

Culminating in CakeML

A dialect of Standard ML, with semantics formalised in HOL

The compiler backend can generate code for 5 different architectures

Source code can be written directly or translated from HOL

Bootstrapped via compilation within HOL4, yielding a
"verified binary that provably implements the compiler itself"

The work of Ramana Kumar, Magnus Myreen, Scott Owens, etc.

How Mike accomplished so much

- he ignored "hot topics" to pursue an original plan
- … and talked to "the enemy" across the Department
- … to learn another subject *really well*
- while using his own knowledge of theory (denotational semantics and CCS)

Learn your application *thoroughly*

E.g. *cryptographic protocols*: such a **simple** field

… so much flawed research

… leading to more bad research

Because people weren't learning from *security* experts!

Rely on robust tools and theory

- LCF provided a good verification engine
- Higher-order logic was old, solid theory (1940!)
- [Standard ML was emerging; Mike didn't use it!]

Type theory would take many years to settle down and would have been a distraction back in 1985.

Where are we today?

- LCF architectures dominate the landscape
- … using higher-order logic or its extensions
- hardware verification is done extensively in industry
- academic research continues to push forward

But modern processors are *still*
too complex to verify in full!

Mike Gordon

1948–2017