

Digital Equipment PDP-8

The Minicomputer That Anticipated Microprocessors

• PDP-8/e • PDP-8/m •

Giovanni Becattini



References

The following references are used in this book:

- [TEO] Tektronix Epic Oscilloscopes – Elektor Books
- [7KS] Tektronix 7000 Series – Mainframes – Elektor Books
- [7KS] Tektronix 7000 Series – Plug-ins – Quacktech
- [TREG] Tektronix Oscilloscopes Restoration Guide – Elektor Books
- [HP1] Hewlett-Packard and its Voltmeters and Oscilloscopes – Quacktech
- [HP2] Hewlett-Packard Signal Sources – Quacktech
- [HP3] Hewlett-Packard Counters and Spectrum Analyzers – Quacktech
- [HP4] Hewlett-Packard Calculators, Computers and HP-IB – Quacktech
- [VRE] Vintage Radio Equipment – Elektor Book
- [MAC] Apple Macintosh – History, Engineering, and Restoration – Elektor Books
- [QT602] The Good Giant – Tektronix DSA 602A Oscilloscope – Quacktech
- [QT11KP] The Last Plug-ins – Tektronix 11000-Series Plug-ins – Quacktech
- [SCLI] Strumentazione Vintage – Edizioni C&C
- [QTCOL] Collins Classics – KWM-2, 30L-1 and 51S-1: History, Engineering, and Restoration – Quacktech Editions
- [QT324] Two Giants and a Shorty – Tektronix 323/324 – Quacktech
- [QTBYP] Become Your Own Publisher – Quacktech
- [QTHPSS] Hewlett-Packard Signal Sources – Quacktech
- [DAB] Digital but Analog – Quacktech
- [TG7] When Machines Learned to Write – Quacktech
- [PDP-8] DEC PDP-8 – The Minicomputer That Anticipated Microprocessors – Quacktech
- [OTTO] Build Your Own Educational PDP-8 Clone Using an FPGA – Quacktech
- [PRWI], Franklin P. Prosser and David E. Winkel, 1986, *The Art of Digital Design – An Introduction to Top-Down Design*, Prentice-Hall, ISBN 0-13-046780-4

- All rights reserved. No part of this publication may be reproduced or stored in any form or by any means, electronic or mechanical, without the prior written permission of the Author.
- The Author has made every reasonable effort to ensure the accuracy of the information contained herein but assumes no responsibility for errors or omissions, or for any loss or damage resulting from the use of this material.
- Most photographs were taken by the Author; others derive from commercially licensed or Creative Commons sources, clearly indicated where applicable.
- AI-assisted content disclosure – This book contains AI-assisted content used for language revision and for the creation of original illustrative material. All ideas, technical content, structure, and final editorial decisions remain entirely the Author's own.
- Some illustrations in this book were generated with the assistance of artificial intelligence tools, using original prompts and concepts developed by the Author. These images serve a purely illustrative and metaphorical purpose and are not intended to depict or reference any existing copyrighted characters, brands, or proprietary visual styles.
- Apple, Hewlett-Packard, Tektronix, Teletype, and other trademarks or logos mentioned in this book are acknowledged and remain the property of their respective owners.
- © 2025, 2026 Giovanni Becattini, Florence, Italy.

Table of Content

Historical Notes	13	Freeing from Mental Schemes	57
From Dinosaurs to Miniskirts	14	What's Next	57
Information Technology in the 60s	14	Sources	59
Electronic Brains	14	A Different Approach	59
The Smell of Modernity	17	The Small Computer Handbook	59
Machines	19	Other Booklets	60
Prisoners of Their Own	19	Education as a Strategy	60
Technological Improvement	20	Specific Documentation	60
The Logical Leap	20	Programming Model	63
Microprocessors	21	Logical Architecture Overview	63
PDP-8, Our Hero	23	<i>Registers</i>	63
Time to Introduce It	23	<i>Instruction Set Summary</i>	64
Digital Equipment Corporation	25	<i>Addressing Modes</i>	64
Programmable Data Processors	26	Accumulator and Link	65
Expensive but Possible	26	Other Registers	65
DECUS	29	Self-modifying Code	66
A Different Business Model	29	Machine Code Instructions	68
Mini-Mania	30	More About Memory	68
PDP-5 And Something In Between	33	<i>7-bit Memory Addresses</i>	68
Twelve Bits Were Enough	33	<i>Indirect Addressing</i>	69
The Right Machine At The Right Time	35	<i>Autoindexing</i>	69
Model T Of Computing	35	<i>Breaking the 4K Barrier</i>	69
A DEC Brain Inside Machines	36	Classes	69
Not a Computer	36	Memory Reference Instructions	70
The Miniprocessor	41	Operate Instructions	72
Before Programmability Became Invisible	41	Input/Output Transfer Instructions	79
Evolution And Grandeur	42	<i>IOT Processor instructions</i>	79
The End: Evolution Or Death	44	<i>Augmented Instructions</i>	81
Ken Olsen: The Man Behind DEC	47	Efficient After All	81
<i>Early Life</i>	47	12 Bits, Some Extra Reflections	81
<i>Digital Equipment Corporation</i>	48	Console, an Important Peripheral	84
<i>Retirement and Death</i>	51	Mechanicals Teleprinters	84
<i>Ken's Legacy</i>	51	The Only One	84
Functional Architecture	53	Dedicated Instructions	84
Programmer's View	55	A Sample Program	86
It Could Do almost Anything	56	Programmed Interrupts	89
A Small but Self-Consistent Machine	56	Shocking Simple	89
		A Software Responsibility	90
		The Role of Flags	90
		Enabling and Returning	91

4 Table of Content (cont'd)

Minimal, But Not Careless	93	Datapath	132
A Very PDP-8 Solution	93	Arithmetic Logic Unit (ALU)	132
Memory Extension	94	Other Building Blocks	133
When Memory Was a Scarce Commodity	94	General Organization	133
The PDP-8 Memory Extension Mechanism	94	<i>Datapath</i>	133
Instruction Field and Data Field	97	<i>Control</i>	134
Why Changing IF Is Delayed	98	<i>Terminology</i>	134
CDF, CIF, and CDF CIF	99	Accumulators and Incrementing Them	134
<i>CDF n</i>	99	Adding Two Values	135
<i>CIF n</i>	99	Where Are We Heading?	136
<i>CDF CIF n</i>	100	Memory	136
Reading the Field Registers: RDF and RIF	100	Completing the General Picture	138
<i>RDF</i>	100	<i>An Example</i>	138
<i>RIF</i>	100	Final Implementation	140
Interrupts and the Save Field Register	101	FSM or Microcode?	142
RIB: Reading the Saved Fields	102	Finite-State Machine Control	142
RMF: Restoring the Saved Fields	102	Control Unit	143
Interrupt Inhibit	104	Microprogrammed Control	144
A Simple Summary	104	Macroinstructions and Microinstructions	144
Programming Practices	106	Why Microcode?	145
Making Do	106	OTTO Microcode	145
Loading the Accumulator	106	Clock	145
Storing the Accumulator	106	Microprogram Instructions	147
Immediate Constants	107	The Jump Part	148
Subroutines	108	The Load Part	148
Call and Skip	108	Notation	149
Far Calls	110	Example: Instruction Fetch	149
Pseudo-Instructions and Assembler Tricks	111	Example: AND Instruction	150
Using Assemblers	112	Example: AND Instruction	150
Designing a PDP-8	117	Other Examples	150
An Inspiring Design	118	Physical Architecture	153
PDP-8 Clones	118	Introduction	155
The OTTO Project	122	The True PDP-8	156
My Own Attempt	122	From Logical to Physical	156
Why "OTTO"?	122	My Own PDP-8/m	158
Why Talk About OTTO Here?	124	Another Revolution	159
FPGAs	125	Dinosaurs Computers	159
Why an FPGA?	125	PDP-8 Physical Architecture	160
A Wonderfully Useless Project	127	Board-level Block Diagram	162
Learning Verilog	127	The Omnibus	165
From Sequential to Parallel	127	A Carrier for Everybody	165
From 2013 to Today	129	The OMNIBUS Revolution	165
Results	129	Open Architecture	167
Designing a PDP-8 from Scratch	131	Modular and Expandable	168
Books that Change Your Life	131		

More Precisely	168	Engineering Drawings	218
OMNIBUS Pin Numbering	170	FAULT: AC Not Displayed	218
Beware of Names	172	<i>The Problem</i>	218
PDP-8 Boards	175	<i>Circuit Analysis</i>	219
M8330 Timing Generator	177	<i>Diagnosis and Solution</i>	220
Like a Metronome	177	FAULT: Accumulator Bit 10	222
Cycles, Time States, and Pulses	178	<i>The Problem</i>	222
Slow and Fast Cycles	178	<i>Circuit Analysis</i>	222
The Walking Zero	180	<i>Circuit Analysis: Dynamic Signals</i>	222
Signals Generation	180	<i>Circuit Analysis: The "Gap Filler"</i>	223
Misleading Connection	181	<i>Diagnosis and Solution</i>	226
M8300 Major Register	183	M8320 Bus Load	229
The Datapath	183	Getting Bus Signals	229
Board Contents	184	M8650 Async Data Control	231
Manual Operation	184	Description	231
FAULT: ADDR LOAD	186	Repairing My Own M8650	232
<i>The Problem</i>	186	Transmit Section	232
<i>Circuit Analysis</i>	186	FAULT: Transmit Flag	234
<i>Diagnosis</i>	188	<i>The Problem</i>	234
<i>Solution</i>	188	<i>Circuit Analysis</i>	234
EXAM/DEPOSIT Flow Digression	190	<i>Diagnosis & Solution</i>	236
Manual Start	190	Sample Tx Test Program	236
Cycle Begins	192	Interfacing the M8650	238
Read Operation	192	Connection Cable	238
Actual Operation	192	Communicating at 110 Baud	238
Final Operations	194	Receiver Section	240
FAULT: EXAM and DEPOSIT	197	Some Peculiarities	240
<i>The Problem</i>	197	Mini Test Program	242
<i>Circuit Analysis</i>	197	FAULT: General Erratic Behavior	242
<i>Diagnosis</i>	198	<i>The Problem</i>	242
<i>Solution</i>	198	<i>Circuit Analysis (IOT System)</i>	242
M8310 Major Register Control board	201	<i>Diagnosis</i>	244
Official Description	201	<i>Solution</i>	246
An Attempt at a Better Description	201	FAULT: Interrupt Enable FF	246
FAULT: Solution of EXAM/DEPOSIT Fault	204	<i>The (non) Problem</i>	246
FAULT: Solution of ADDR LOAD Fault	205	The Reader Enable Flag	246
FAULT: Non-incrementing ISZ Fault	208	<i>What It Is</i>	246
How Does ISZ Work	208	<i>Why It Is So Important</i>	247
Fault Analysis	209	Reader Enable Flag Circuit	248
Diagnosis	210	Making It Work	249
Solution	210	A Sample Test Program	250
FAULT: AND with 7777 Failed	212	Need for Speed	250
KC8-ML Programmer's Console	215	M8350 Positive Bus Interface	253
Personal Memories	216	Core RAM Memory	255
Save, Save, Save!	216	Magnetic Memories	256
Models	217	Complex, Power Hungry and Delicate	256
Lamps and LEDs	217		
Selector Switch	218		

6 Table of Content (cont'd)

Magnetic Cores	258	<i>Slicing</i>	304
Reading Cores	260	<i>Two Ways to Reject Delta Noise</i>	304
Data General	261	Amplifier Input Network	306
PDP-8 Memory	263	Use This Method Instead	308
From General to Particular	263	FAULT: Memory Erratic Operation	308
Documentation	263	<i>The Problem</i>	308
Row and Column Selection	264	<i>Fault Analysis</i>	309
X-Y Selection	264	<i>Solution and Lesson Learned</i>	310
The H-Bridge	266	Power Supply	313
The PDP-8 Memory Assembly	269	For those Hungry TTL Chips	314
The Memory Register	269	Linear Circuit	314
The Memory Cycle	270	Overhaul	314
Signals and Timing	270	Last Restoration Notes	319
But Do They Wear?	272	Not The Easiest	320
What's Next	272	Half a Century	320
G227 X-Y Board	274	One of the Most Complex	320
Description	274	Original Schematics	320
FAULT: Nothing Works	276	Last Notes	321
<i>Problem</i>	276	They Make Our Work Simpler	324
<i>Circuit Analysis: Address selection</i>	276	Coeval Instrumentation	324
<i>Circuit Analysis: Currents Management</i>	277	To Be Honest...	324
<i>Circuit Analysis: Bias Generator</i>	279	Logic Analysis	327
<i>Diagnosis and Solution</i>	282	Desoldering Station	331
FAULT: Memory Erratic Behavior	282	Current Probe	331
<i>The Problem</i>	282	Chip Tester	332
<i>Circuit Analysis: Diode Selection Matrices</i>	282	Final Balance	334
<i>Diagnosis</i>	285	Replacing the DEC8881	334
<i>Solution</i>	285	In-Depth Summary	339
Reference Waveforms	286	Ending from the Beginning	341
A Precious Help for Repairs	286	Machine Cycles	342
Waveforms	286	What I Wish I Had Known	342
Diagnostic Value	288	FETCH Cycles	342
Current Probe and Waveforms	288	<i>What Happens During the Time Slots?</i>	343
	288	<i>Memory</i>	346
A Moment of Panic	290	<i>IOT Transfers</i>	346
G104 Sense/Inhibit Board	292	EXECUTE Cycles	346
G104 Sense/Inhibit Board	294	<i>What ISZ Does</i>	347
Reading and Writing	294	<i>Time Slot 1</i>	347
G104 Board Overview	297	<i>Time Slot 2</i>	347
Twelve Slices	298	<i>Time Slot 3</i>	349
Sense Amplifiers	298	<i>Time Slot 4</i>	350
How to Make Blunders	299	IOT Data Transfers	352
"Seeing" the Currents	300	Input/Output System	352
A Memory Cycle	300		
What Did We Read?	302		
Correctly Reading a Pulse	302		
<i>Delta Noise</i>	303		

• New "Instructions"	352
Setting Addresses	353
Returning Data	353
Interrupt System	356
The Simplest One	356
Basic Interrupt Operation	356
What Happens During an Interrupt	357
A Very Small Hardware Mechanism	358
Interrupt Specific Instructions	358
ION — Interrupt Turn On (6001)	358
IOF — Interrupt Turn Off (6002)	359
SKON — Skip if Interrupt System On (6000)	359
SRQ — Skip on Interrupt Request (6003)	359
GTF — Get Flags (6004)	359
RTF — Restore Flags (6005)	360
Peripheral Interrupt Enables	360
Clearing the Device Flag	361
A Minimal Test Program	362
Why the Return Address Is in Location 0000	363
No Vectors, No Priorities	363
A Practical Debugging Note	364
A ROM-Unfriendly Mechanism	364
Conclusion	365
Will It Come Back to Life?	367

Software 369

Peculiarities of the First Seventies	370
PDP-8 Software	371
PDP-8 System Software	372
A Family of Tools and Systems	372
Paper Tape	372
Loaders and Assemblers	372
Programming Languages	375
Operating Systems	376
Specialize Systems	376
Diagnostics	376
DECUS	376
A Little Universe	377
\$\$\$ Prices \$\$\$	379
Hardware Prices	379
Software Prices	381
Books	384

Hands On 387

Most Common System Software	389
Introduction	389
Typical Software Development	389
Development Cycle	390

Code Loaders

391

Binary Loader	391
What it Is	391
General Logic	391
Using It	392
The RUBOUT Function	392
That Small "But"	392
RUBOUT Troubles	393
Conclusion	393
RIM loader	394
Symbolic Editor	396
Infernal Editors	397
Concepts Behind It	397
PDP Symbolic Editor	398
A Practical Example	399
Assemblers	400
Not Just One	403
MACRO-8 Assembler	403
FOCAL Programming Language	406
Feeling Part of It	407
A FOCAL Examples	407
A Damped Sinusoid	407
Temperature Conversion	409
Editing on a Teletype	410
Self-test Programs	412
Software for Servicing	412
Not as a Whole	412
Down to Gates	412
Diagnostic Programs	412
Tested Tests	413
Instruction Test #1	413
Instruction Test #2	414
Adder Test	414
Teletype Test	415
PRG0 — Receiving Section	415
PRG1 — Transmitting Section	416
Basic JMP-JMS Test	417
Memory Test	417
Random AND	418
Some Simple Programs	419
Minimal Testing	419
Indirect Addressing	419
JMS Jump to Subroutine	420
Its First Words	420
Book Conclusions	422



Although every effort has been made to ensure accuracy, occasional errors may still be present. If you discover any mistake, however small, I would be grateful if you let me know. Your feedback helps ensure that future readers receive an improved edition.

You can contact us on www.quacktech.it. Thank you for contributing to the growth of this work.

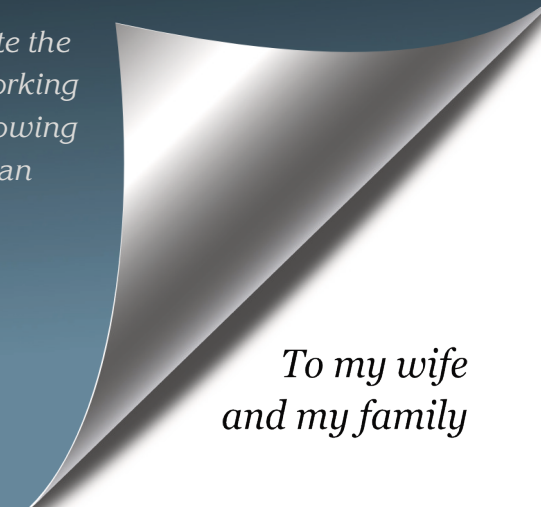
Preface

The 1960s are remembered above all for the cultural revolution of which the miniskirt and the Mini motor car became unforgettable symbols. Yet another, quieter revolution marked those years: the transition from dinosaur-like computers to the minicomputer. In that revolution, Digital Equipment Corporation—if not strictly its inventor—was its most important standard-bearer.

This book is not simply the story of Digital Equipment Corporation, nor only of the PDP-8. Its purpose is broader: to retrace the climate that made such a machine necessary, the company that created it, and the transformation that made programmability one of the decisive ideas of the modern age. With the minicomputer, computation left data centers and entered laboratories, factories, offices, instruments, machines, and eventually almost every appliance.

This is also a personal story: the story of my own PDP-8/M, how it came into my hands, and the work required to understand, preserve, and bring it back to life. I hope the reader will find it worth following, because old machines are rarely interesting only for what they are. More often, they matter because of the paths by which they reach us, the questions they force us to ask, and the effort they demand before revealing their secrets.

Finally, although PDP-8 implementations exist, I wanted to recreate the machine in programmable logic myself: not merely to obtain a working replica, but to offer a first-hand account of entering its design, following its logic from within, and discovering the elegance hidden inside an apparently modest architecture.



*To my wife
and my family*

PDP-8

*The Minicomputer That
Anticipated Microprocessors*



Historical Notes

Functional Architecture and
FPGA Reimplementation

Physical Architecture and
Restoration

Software





This cash-sales approach also created space for third-party leasing companies around the DEC ecosystem. Once again, DEC was operating outside the normal assumptions of the established computer industry. It was not merely selling smaller machines. It was helping to create a different relationship between the user and the computer.

Mini-Mania

Today the PDP-8 is routinely described as one of the first truly successful **minicomputers**. This is correct, but it can be slightly misleading if read with modern eyes.

When the PDP-8 was introduced in 1965, the word minicomputer was not yet the name of a clearly defined market category. The PDP-8 was not born inside an already established class of machines. It was a computer that crossed a boundary.

It became small enough, inexpensive enough, and practical enough to leave the computer room and move close to laboratories, instruments, industrial processes, and machines.

Only afterward did the word *minicomputer* become the natural label for this new reality.

According to a well-known historical account, the term seems to have entered DEC's internal language through John Leng, head of DEC's operations in England. In a sales report, he reportedly wrote: *"Here is the latest minicomputer activity in the land of miniskirts as I drive around in my [Austin] Mini Minor."*

The expression quickly became part of DEC jargon and then spread more widely. Curiously, the first computer explicitly advertised as a minicomputer appears to have been not a DEC machine, but an Interdata system, in a 1968 advertisement in Computers and Automation.

This detail is more than a curiosity. It shows that the PDP-8 did not merely belong to the minicomputer era. It helped make that era visible.

The PDP-8 was a computer that became "**mini**" before the world had fully understood what mini meant in computing.

Swinging London, mid-1960s. The Mini Morris, the miniskirt, pop music, new design, and a new sense of youth freedom became the symbols of a city that seemed to be inventing the future in real time. In that same cultural atmosphere, even the computer was about to become "mini." The word minicomputer captured not only a technical reduction in size, but also the spirit of an age in which small, modern, accessible things promised to overturn old hierarchies.

larger systems made DEC much less accessible than it should have been.

Their failure was not technical weakness. That is the saddest part. DEC had immense technical strength. It had brilliant engineers, excellent architectures, serious operating systems, and a tradition that had helped invent the very idea of interactive and accessible computing.

But evolution does not reward past greatness by itself.

The company that had helped move computing out of the computer room did not fully understand the next movement: computing leaving the minicomputer rack and entering the desk, the workshop, the school, the small office, and eventually the home.

DEC had taught machines to become programmable before microprocessors existed. But when microprocessors made programmability cheap, personal, and everywhere, DEC was no longer ready to follow the revolution it had helped prepare.

Their tragedy was not that they lacked technology. Their tragedy was that they failed to recognize the new form that their own idea had taken.

PDPs' Essential Data					
Model	Date	Price	Bits	Produced Units	Remarks
PDP-1	1960	\$120,000	18	50	DEC's first computer
PDP-2	NA	NA	24	-	Never built? Prototype only?
PDP-3	NA	NA	36		One built by a customer*, not by DEC.
PDP-4	1962	\$60,000	18	45	Predecessor of the PDP-7.
PDP-5	1963	\$27,000	12	1,000	The ancestor of the PDP-8.
PDP-6	1964	\$300,000	36	23	A big computer; 23 built, most for MIT.
PDP-7	1965	\$72,000	18	120	Widely used for real-time control.
PDP-8	1965	\$18,500	12	~50,000	The smallest and least expensive PDP.
PDP-9	1966	\$35,000	18	445	An upgrade of the PDP-7.
PDP-10	1967	\$110,000	36	**~700	A PDP-6 followup, great for timesharing.
PDP-11	1970	\$10,800	16	>600,000	DEC's first and only 16 bit computer.
PDP-12	1969	\$27,900	12	725	A PDP-8 relative.
PDP-13		NA		-	Bad luck, there was no such machine.
PDP-14				***	A ROM-based programmable controller.
PDP-15	1970	\$16,500	18	790	A TTL upgrade of the PDP-9.
PDP-16	1972	NA	8/16	?	A register-transfer module system.

60 Other Booklets

In the same practical format, DEC published other important books, such as Introduction to Programming, a true bible for the assembly-language programmer, filled with dozens of those tricks we have already mentioned, and Programming Languages, the book in which I studied FOCAL and BASIC for the first time, being struck by new truths I had never even imagined.

Education as a Strategy

These books were not just accessories to products already sold. They were instruments of education. They explained the machines, but also the way of thinking behind them. A reader could learn the basic architecture of a PDP-8, understand its instruction set, discover how input/output worked, and begin to imagine real applications. The tone was technical, but not closed; precise, but not impenetrable.

This was more than good documentation. It was a **strategy**. DEC understood that the small-computer market could grow only if a new population of users grew with it. The company needed programmers, technicians, laboratory people, students, and engineers who could see a computer not as a remote institutional machine, but as a tool they could personally understand and use.

For this reason, the Small Computer Handbook was not only a handbook. It was a **cultural object**. It helped transform the PDP-8 from a piece of hardware into an idea: the idea that a computer could be small, accessible, programmable, and close to the people who used it.

DEC did not merely sell machines. Through these books, it helped form the generation that would know what to do with them.

Specific Documentation

More technically, for my PDP-8 restoration project, I was able to collect the following documents, which proved to be an essential part of the game. With them, the restoration required effort, patience, and a good deal of study, but it was possible. Without them, it would have been totally impossible. All of them can be easily found online:

- The Small Computer Handbook, as mentioned, 1973 edition;
- DEC-8E-HMM1A-D-D Maintenance Manual, Volume 1, Sep. 1973
- PDP-8/e Engineering Drawings, December 1972
- PDP-8E_Engineering Drawings, November 1973

When bit 3 is 1, the sense of the test is reversed. The processor skips only if none of the selected normal conditions is true. This is equivalent, by De Morgan's law, to a logical AND of the complementary conditions. For example:

SZA SNL / SKIP IF AC = 0 OR L = 1

but:

SNA SZL / SKIP IF AC ≠ 0 AND L = 0

This second case is the complement of the first. In logical terms:

NOT (AC = 0 OR L = 1)

which is equivalent to:

AC ≠ 0 AND L = 0

For this reason, one should not freely mix the “normal” skip mnemonics, such as SMA, SZA, and SNL, with the reversed mnemonics, such as SPA, SNA, and SZL, unless one is deliberately constructing the resulting bit pattern. The assembler merely combines bits; the processor then interprets the final word according to the Group 2 rules.

Some examples follow:

1 00000	7460	SZA SNL	/ SKIP IF AC = 0 OR L = 1, OR BOTH
2 00001	7470	SNA SZL	/ SKIP IF AC ≠ 0 AND L = 0
3 00002	7520	SMA SNL	/ SKIP IF AC < 0 OR L = 1, OR BOTH
4 00003	7530	SPA SZL	/ SKIP IF AC ≥ 0 AND L = 0
5 00004	7540	SMA SZA	/ SKIP IF AC ≤ 0
6 00005	7550	SPA SNA	/ SKIP IF AC > 0
7 00006	7560	SMA SZA SNL	/ SKIP IF AC ≤ 0 OR L = 1
8 00007	7570	SPA SNA SZL	/ SKIP IF AC > 0 AND L = 0
9 00010	7402	HLT	
10		\$	

A small detail is worth noting. In PDP-8 terminology, “positive” means simply that the accumulator's most significant bit is zero. Therefore SPA alone also skips when AC is zero. To test for a strictly positive value, one must combine it with SNA, as in:

SPA SNA / SKIP IF AC > 0

Likewise:

SMA SZA / SKIP IF AC < 0 OR AC = 0, I.E. AC ≤ 0

This is another good example of the PDP-8 philosophy: the machine offers very little hardware, but the few available bits can be combined in surprisingly expressive ways.

90 A Software Responsibility

The most important thing to understand is that the PDP-8 does not automatically identify the source of the interrupt.

All interrupting devices share the same general interrupt entry point. If several peripherals are present, the interrupt service routine must poll them, usually by testing their flags with the appropriate IOT skip instructions. The order in which the devices are tested determines the effective software priority.

A typical interrupt routine therefore has the following logical structure:

- Save AC and Link, if necessary
- Test keyboard flag
 - If set, service keyboard
- Test teleprinter flag
 - If set, service teleprinter
- Test another device flag
 - If set, service that device
- Restore AC and Link, if necessary
- Re-enable interrupts
- Return through JMP I 0

This is very different from a vectored interrupt system, where the hardware automatically directs execution to a different service routine for each interrupt source. On the PDP-8, the hardware only says: something needs attention. The program must discover what it is.

The Role of Flags

Most PDP-8 peripherals are built around the concept of a flag. A device sets its flag when it needs attention or when it is ready for the next operation.

For example, a keyboard interface may set its flag when a character has been received, while a teleprinter interface may set its flag when it is ready to accept another character.

- IB holds the next instruction field until a JMP or JMS occurs.
- SF saves IF and DF during interrupt entry.

The essential operations are:

- CDF n – immediately changes DF
- CIF n – loads IB; IF changes only at the next JMP/JMS
- RDF – reads DF into AC bits 5:3
- RIF – reads IF into AC bits 5:3
- RIB – reads the saved IF/DF value after interrupt entry
- RMF – restores DF and IB from the saved interrupt fields

The most important rule is:

- Data field changes are immediate.
- Instruction field changes are delayed until a control transfer.

This may seem strange at first, but it is actually a very effective solution. It preserves the small and simple 12-bit PDP-8 instruction format, while allowing the machine to address up to 32K words of memory. It also keeps program execution well controlled: a program changes instruction field only when it deliberately performs a jump or subroutine call.

In this sense, the memory extension system is typical of the PDP-8 philosophy. It does not make the processor much larger or more complicated. Instead, it adds a small amount of clever external logic that extends the machine while preserving its original simplicity.

Note: To avoid confusion, throughout this book and unless otherwise noted I use the modern convention: bit 0 is the least significant bit. With this convention, RDF and RIF place the field number in accumulator bits 5 through 3, producing values such as 0030 for field 3 and 0040 for field 4

A Word about Bit Numbering

One of the most surprising aspects of encountering the PDP-8 for the first time is its bit numbering convention: for DEC, bit 0 is the most significant bit, exactly the opposite of what we have long been used to.

I tried to resist this at first, and in some drawings I even used double numbering, but it soon became clear that this was completely impractical. I therefore kept the DEC numbering system for almost everything related to the PDP-8, or used the terms “MSB” and “LSB” when clarity was needed. When discussing the Verilog code, however, I retained the modern numbering convention.

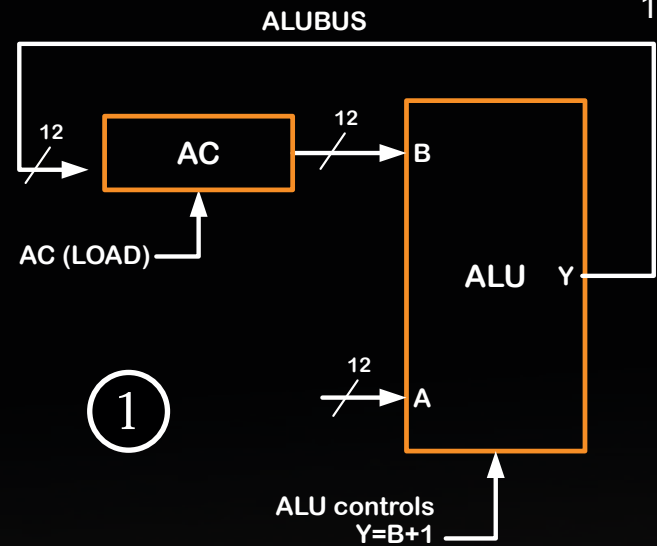


In figure ❶ only two elements are present:

- a 12-bit register, called the accumulator (AC), destined to become one of the central registers of the computer (nothing to do with rechargeable batteries;)
- a 12-bit ALU, provided with two inputs, A and B. The former is temporarily left open, while the latter is connected to the accumulator.

Let us suppose that the ALU is set to perform the operation $\text{INC } A$, that is, to place at its output the value appearing at input B, increased by one.

In this arrangement, the input of the accumulator receives the value $\text{AC} + 1$. A single clock pulse applied to the AC register is then enough to replace its previous contents with the incremented value. Thus, even with this extremely simple configuration, the machine is already capable of incrementing the accumulator.

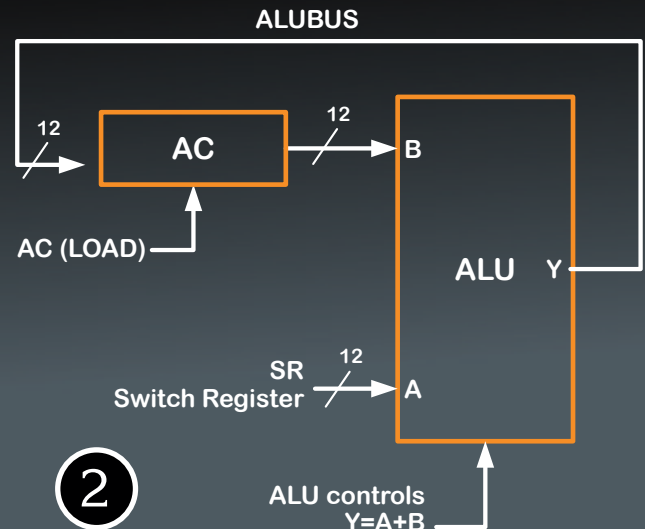


Adding Two Values

Let us now connect the 12 lines of the A input of the ALU to 12 switches (figure ❷), each able to select either a HIGH or a LOW level. In this way, the switches define a binary number. For instance, they may be set to D'16, corresponding to B'000000010000.

If the ALU is instructed to perform the operation $A + B$, its output will then be equal to the current contents of the accumulator plus the number set by the switches. In the present example, the value produced will be $\text{AC} + \text{D}'16$.

If a clock pulse is now applied to the accumulator, that result will be loaded into it. The contents of the accumulator will therefore be updated by adding the selected constant value.



Example: AND Instruction

The fetch part of the microcode continues with some instruction which analyze the addressing mode, in case of an instruction which references the memory. Let's skip them and just assume that the MA register has been properly set to point the memory data we need, and see the AND instruction which "ands" it with the accumulator:

```
// AND INSTRUCTION
046 UJPT QFALSE XXX, LD SMB OAANDB ->DAC    // NO JMP, LD SMB AND SAC ->DAC
047 UJPT QTRUE 001,  LD SMB OTO ->DXX        // JMP TO NEXT FETCH
```

Example: AND Instruction

The fetch part of the microcode continues with some instruction which analyze the addressing mode, in case of an instruction which references the memory. Let's skip them and just assume that the MA register has been properly set to point the memory data we need, and see the AND instruction which "ands" it with the accumulator:

```
// AND INSTRUCTION
046 UJPT QFALSE,      LD SMB OAANDB ->DAC    // NO JMP, LD SMB OAND SAC ->AC
047 UJPT QTRUE 001,   LD SMB OTO ->DXX        // JMP TO NEXT FETCH
```

Other Examples

Deposit and Clear Accumulator (DCA) – stores the data contained in accumulator

```
// DCA INSTRUCTION
048 UJPT QFALSE XXX, LD SAC OTO ->DMB // LD SAC OTO ->DMB READY TO BE STORED
049 UJPT QFALSE XXX, LD SAC OZERO DAC // CLEAR ACCUMULATOR (SOURCE IGNORED)
050 UJPT,QTRUE 001, LD SMB OTO DMEM // LD SMB --EM<-MB AND JUMP TO 001
```

It's evident the benefit of microprogramming.

The Omnibus

A Carrier for Everybody

The Latin word *omnibus* means “for all.” In English it later acquired other meanings, including that of a work or collection containing many different items. As a noun, however, its modern history goes back to the nineteenth century, when the first public transport services were introduced: vehicles available, precisely, to all. Its shortened form survives today in the familiar word bus.

In computers, the term returns quite naturally to this original idea. A bus is a shared set of lines and signals available to the various modules of a system, allowing them to exchange addresses, data, and control information through a common path.

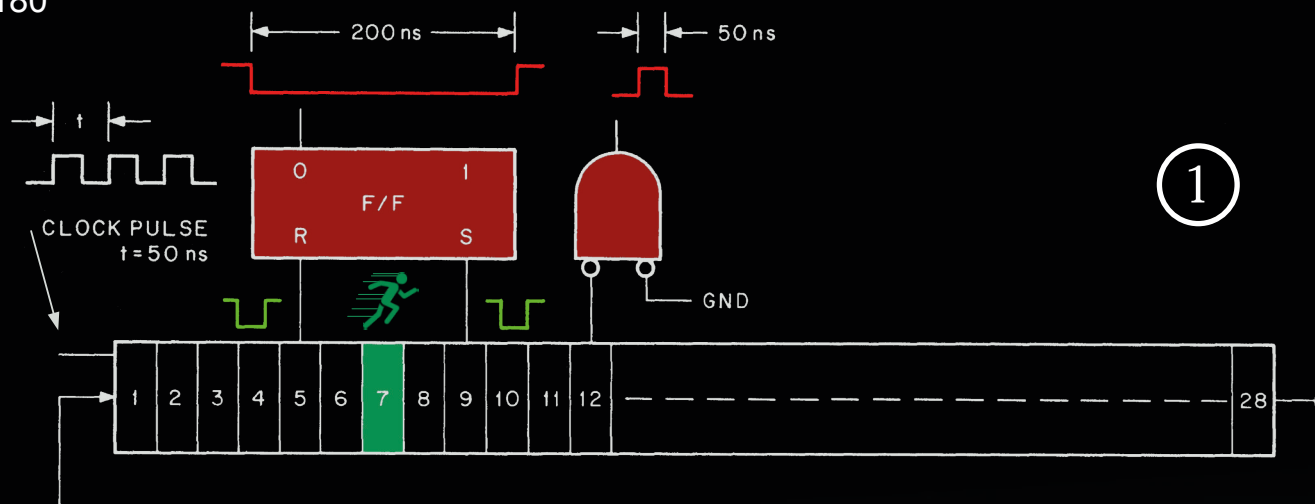
The OMNIBUS Revolution

One of the important conceptual advances of the PDP-8/e was precisely the adoption of a much more systematic internal structure: the OMNIBUS. Rather than wiring each functional block to the others through a maze of dedicated connections, DEC organized the machine around a common backplane that carried the signals needed by the processor, memory, and peripherals.

This did not make the PDP-8/e a simple machine, of course, but it made it a far more modular and serviceable one. Boards could be inserted, removed, expanded, or diagnosed in a much more orderly way. In this sense, the OMNIBUS was not only an electrical structure, but also a very practical architectural idea: a carrier for everybody.

This was not merely a convenient backplane, but a new way of thinking about how a small computer could be assembled. Instead of treating the machine as a collection of boards joined by a large number of dedicated interconnections, the PDP-8/e used a passive, horizontal bus. The corresponding

Image: The original “omnibus” was one of the first forms of public transport open to everyone. The name was well chosen: the PDP-8/e OMNIBUS was also conceived as a shared pathway, replacing a large number of dedicated board-to-board interconnections with a passive, horizontal bus.



It is important to note that the duration of each timing signal remains constant in both cycles, with the exception of TS2. The 200 ns difference between the slow and fast cycle is obtained by varying the duration of TS2 alone. In this way, the amount of time between the read and write portions of the memory cycle can be adapted to the operation being performed.

The Walking Zero

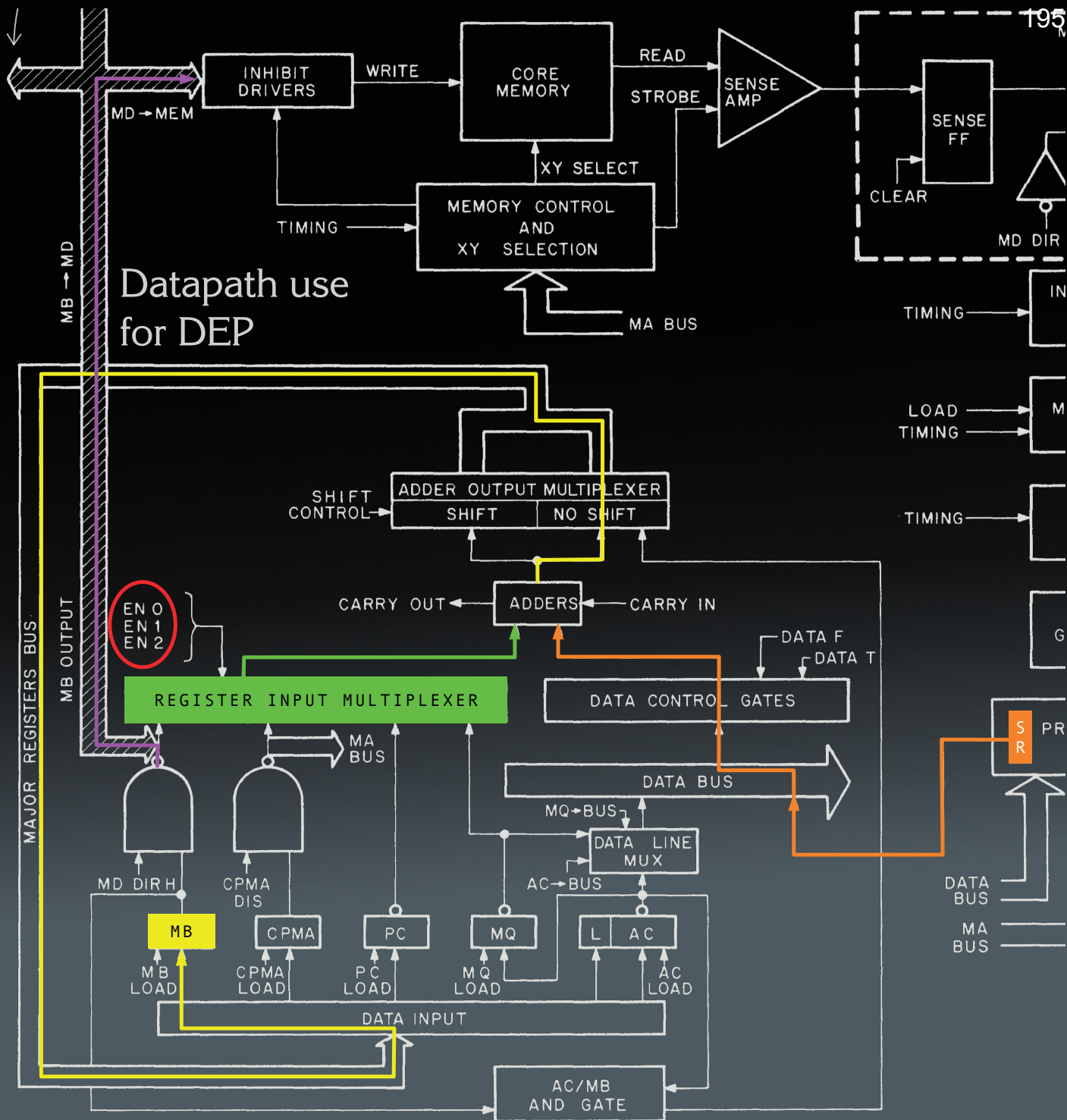
The timing pulses are generated by a walking-zero circuit, implemented with a set of shift-register stages connected in series.

Strictly speaking, in the PDP-8/e Timing Generator the useful traveling condition is often a zero moving through a chain of normally high states. Conceptually, however, the idea is the same: a single active condition walks from one position to the next.

Once this traveling bit exists, it becomes relatively easy to generate timing signals of different durations. A signal may be taken directly from one shift-register output, or it may be produced by setting a flip-flop from one output and resetting it from another.

Signals Generation

I found particularly interesting the simplified schematic of figure (2) showings where, and thus when, the most important signal are generated. This makes less abstract the purpose of the M8330 board.



210 DEC8251, which had already caused so many problems in the memory subsystem, as we will see in the next section (not to be confused with the later Intel USART of the same number.)

So now we must also investigate how the ISZ • E • TS2 signal is generated.

With reference to the schematic on the right, this signal is produced by combining:

- **TS2 • E**, that is, Time Slot 2 of an EXECUTE cycle. I will not discuss this part in detail, because it is fairly simple and, in any case, the signal was correct; it is therefore highlighted in green;
- the **ISZ** signal, which is asserted when, during the previous FETCH cycle, an ISZ instruction has been fetched. This suspect signal path is shown in red.

The ISZ signal is derived from bits 11, 10, and 9 of the instruction register, shown here as the brown flip-flops. These bits are placed on the OMNIBUS through the green gates. However, the same IR bits are also sent to **E46**, shown in red, the DEC8251 decoder used to recognize the ISZ opcode, 2XXX.

Diagnosis

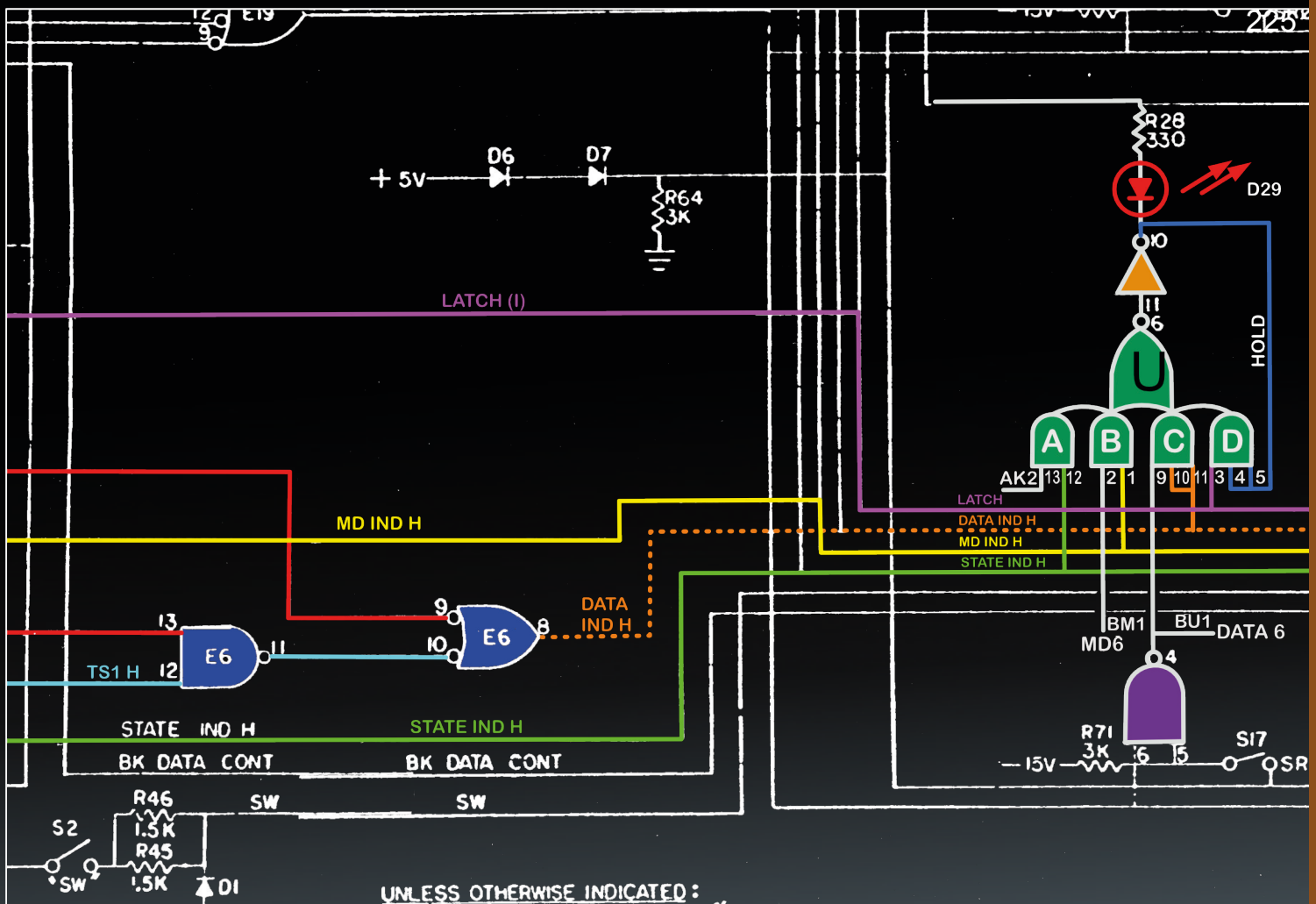
There is one small but important detail: the data produced by the green gates is complemented. Therefore, the decoder must recognize 5, not 2, and this is why the output corresponding to number 5 is used. In my case, this output was defective and remained permanently high, so the ISZ signal was never generated.

Solution

At this point, I resorted to a small superhero that we will meet again later: a tiny adapter PCB which allows a 7442 to be used in place of the now unobtainable 8251.

Once the new chip was in place, I finally had the satisfaction of seeing the ISZ signal appear. Unfortunately, I did not yet have the satisfaction of seeing the ISZ instruction execute correctly. There was a second member in that gang of criminals, but at this point finding it was almost straightforward: it was E8, shown in the figure on the previous page.

I replaced this 7405 with a 7406, which in this position works perfectly, and finally the test mini-program ran as expected, giving me the wonderful sight of the accumulator incrementing on the front-panel display.



signals BUS, MQ, AC, and STATUS are combined on the DATA IND H line, shown as a dotted orange trace. On the right-hand side of the figure is the circuit for LED 6; the other LED circuits are similar. Note the typical DEC method used to create a LOW level: a resistor to -15 V and a diode clamp. The resistor provides a strong current pull-down, improving noise immunity, while the diode prevents the node from going beyond a safe negative L level.

The first sections select the ordinary display sources:

- Gate A handles the STATE signals;
- Gate B handles the MD bus;
- Gate C handles the information coming from the OMNIBUS DATA lines, namely the selected dynamic indication: BUS, AC, MQ, or STATUS.

The fourth section, Gate D, is the ingenious part. It receives a feedback signal derived from the LED side itself and is enabled by LATCH, which is roughly complementary to the valid-data interval pro-

240 Receiver Section

The receiver section has a logic similar to that of the transmitter section.

- KCF 6030 — Clear Keyboard Flag
- KSF 6031 — Skip on Keyboard Flag
- KCC 6032 — Clear Keyboard Flag and AC
- KRS 6034 — Read Keyboard Buffer Static, ORing the data into AC
- KIE 6035 — Set or clear Interrupt Enable, as defined by AC11
- KRB 6036 — KCC + KRS

Keyboard Read Static, KRS, simply places the received character on the I/O bus and ORs it into the accumulator, without first clearing AC. For this reason, it is normally preceded by a CLA instruction if the character is to be read cleanly.

Keyboard Read Dynamic, KRB, performs the complete input operation: it clears AC, reads the keyboard buffer into it, and clears the keyboard flag, preparing the interface to receive the next character.

The meaning of the flag is simple: a character has arrived and is ready to be read.

Some Peculiarities

It is interesting to note that the hardware receiver is always active, without requiring any preliminary setting. If you switch on the computer and then send a character to it through the serial line, the character is received and stored in the receiver buffer. You can read it later.

If, in the meantime, another character arrives before the previous one has been read, there is no sophisticated overrun management. The M8650 has only a single-character buffer, so the previous character may simply be lost or replaced.

Many of the features that one could expect from a UART are simply absent. There is no FIFO buffer, no framing-error indication, no parity handling, and no selectable number of stop bits. In the case of my M8650-0, even the communication speed cannot be changed: it is fixed at 110 baud.

Right: Some of the circuits forming the heart of the M8650 receiver section. In the upper left corner is E48, the receiver flag, cleared by the 6030 instruction and set when a “one” reaches the last position of the receiver shift register, generating the NOT YD signal. The lower part of this shift register, E10, is also visible. In the lower left part is E43, the 8251 decoder responsible for decoding the IOT instruction codes. Note also the gates generating the C0 and C1 signals, which during an IOT cycle request the transfer of the receiver buffer to the AC. The truth table on the left explains whether this transfer is performed as a direct load into AC or as a bitwise OR with its previous contents. The color coding makes this redrawn schematic much easier to read than the original one, where following the various signal paths is considerably more difficult.

Section:

Physical Architecture

Subsection:

Core RAM Memory

A precious relic from another era, and now a rare collector's item: a magnetic-core memory module. In the 1960s, core memory was the dominant technology for computer main memory. Its physical construction remains fascinating even today: the module described here contains 49,152 tiny ferrite cores, each less than a millimeter in diameter.

270 The Memory Cycle

Each processor cycle described above — TS1 through TS4 — includes a memory cycle. Fortunately, this is not the most complicated part of the PDP-8. On the contrary, once the general principle of magnetic core memory is understood, the memory cycle is logically quite straightforward.

As expected, it consists essentially of two phases:

- a **read** phase;
- a **write**, or more often rewrite, phase.

This is necessary because, as already explained, reading a magnetic core memory is destructive. When a word is read, its original contents are lost and must be restored immediately afterwards if the information is to be preserved. It looks as cumbersome but it works.

Unlike modern RAMs, the memory cycle itself is not fundamentally different for a read or a write operation; what changes is only the source of the data used during the write phase.

For example, when the front-panel **EXAM** or **DEPOSIT** keys are used, the memory cycle is almost identical in both cases. It is always composed of a read phase followed by a write phase, but:

- in the case of EXAM, the data written back is the original content of the selected memory location, supplied by the Memory Register;
- in the case of DEPOSIT, the data written into memory is taken from the front-panel switch register, through the OMNIBUS MD bus.

All the rest is essentially identical.

Signals and Timing

Figure 3.32 of the Processor Maintenance Manual describes the timing of the memory cycle, but it is not especially friendly. It is rather crowded, and it also appears to contain two evident errors: WRITE L should actually be understood as WRITE H, and similarly INHIBIT L should be read as INHIBIT H.

For our purposes, it is better to start from the simplified timing diagram shown in Fig. 1. The five signals shown at the bottom of the diagram may at first look rather mysterious, but they are in fact quite easy to understand. Moreover, like most timing signals in the PDP-8, they are essentially fixed: they repeat in the same way during every memory cycle.

In other words, the G227 board and the diode selection matrices 285 must be understood together. The G227 provides the active switching elements of the bridge; the diode matrices provide the passive steering network that makes only one selected memory line conduct.

Once this is understood, the whole circuit becomes much less mysterious. It is not a collection of unrelated transistors and diodes, but a distributed H-bridge and decoder system: part of it is active, on the G227 board, and part of it is passive, in the diode selection matrices.

And this also answers the initial question. Those two diodes cannot simply be moved onto the G227 board, because they are not just two isolated components belonging to one driver circuit: they are part of the diode selection matrix, and their function depends on their position within that matrix.

Diagnosis

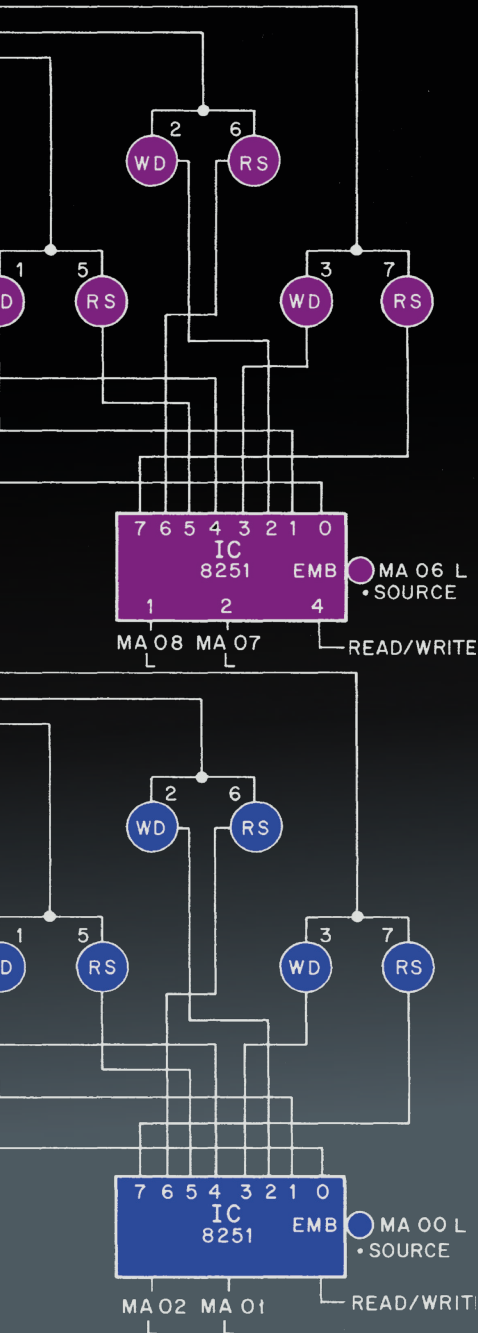
You can easily understand how the missing operation of row and columns can create an erratic and unpredictable behavior, and in my case several defections in the 8251 decoder ranks where the other big problem in addition to the BIAS generator. Considered the number of crossing flaws these decoders can produce together, I finally decided to:

- remove all of them from the board; a desoldering pump was mandatory;
- replace them with sockets;
- build a simple dedicated tester with a breadboard.

This way, I could find as many as three faulty 8251s.

Solution

You can buy spare 8251 on eBay, but at high price, shipment and custom. I chose a different solution building an adapter for the 7442, created by Roland Huisman (you can find it on github (DEC8251_replacement.)). See below "A Moment of Panic" for further details



300 “Seeing” the Currents

The MC1540G sense amplifier is normally used in a way that makes it difficult to observe what it actually *sees*. Its output is enabled only when the surrounding logic asserts the **TIME STROBE H** input. This strobe pulse is very short — about 30 ns — and therefore, under normal operating conditions, the output only tells us the final sampled result. It does not easily reveal what is happening before, after, or around that precise instant.

Fortunately, in machines of this generation, where signals are still physically accessible, there is a simple way to make the amplifier output continuously visible. In my case I removed **E9** (74H00), installed a 14-pin socket in its place, and reinserted the IC with pin 3 left outside the socket. A 1 kohm resistor was then connected between socket pin 3 and Vcc — for example, using E9 pin 14.

In this way, TIME STROBE H is held **permanently active**, and the output of the sense amplifier can be examined with an oscilloscope.

This modification is not dangerous in itself, but in my case it proved intellectually dangerous. It allowed me to see too much, and this led me into several rather elaborate lucubrations that kept me away from the real fault for a couple of days. Still, I do not regret the detour too much, because it helped me understand details that I would probably never have noticed otherwise.

Needless to say, with this modification in place the memory **cannot be expected** to operate normally. The purpose here is not to make the PDP-8 read correct data, but to observe the analog behavior of the sense amplifier during a memory cycle.

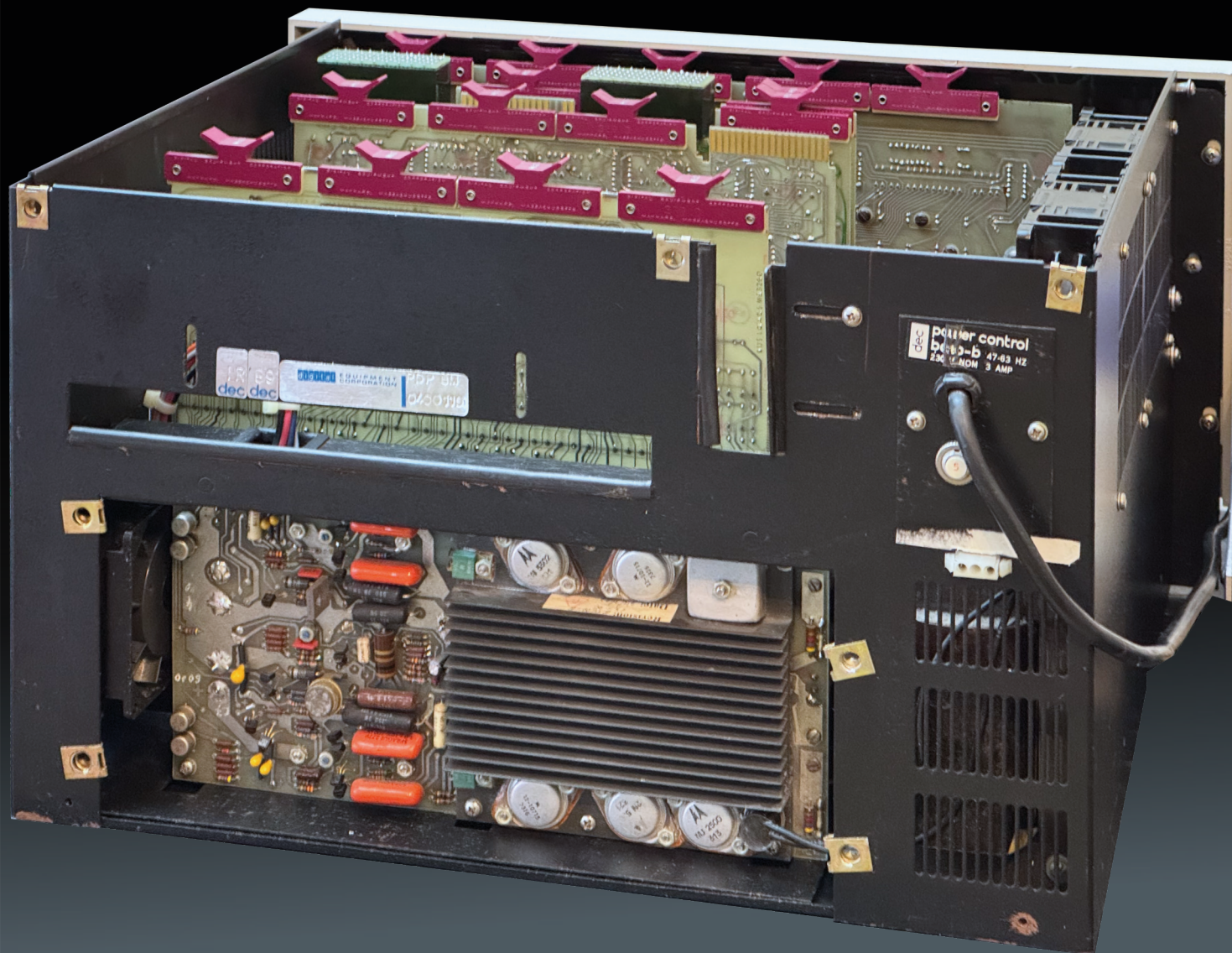
A Memory Cycle

The oscillogram on the right shows the output of the sense amplifier on the upper trace, with TIME STROBE H forced active as described above.

The **first pulse**, on the left, corresponds to the READ portion of the memory cycle, when all selected cores are driven **toward 0**. At first sight, one might be tempted to conclude that the selected core was previously in the 1 state, and that the pulse was produced by its change of magnetic state.

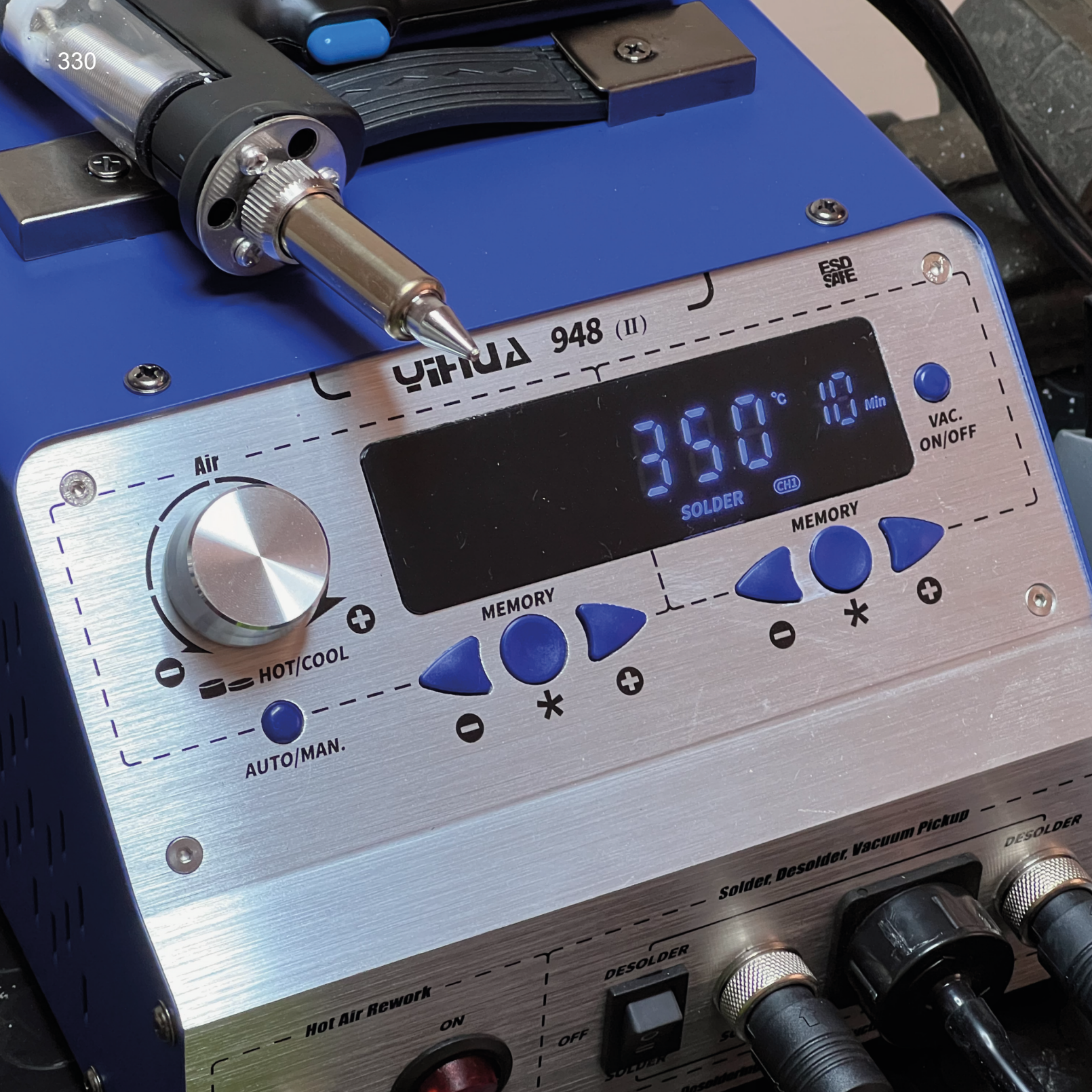
This interpretation is tempting — but, as we shall see shortly, **it is not necessarily correct**.

The **second pulse** corresponds to the WRITE portion of the cycle. The lower trace was taken with a current probe on the inhibit wire, and clearly shows that an inhibit pulse was applied.

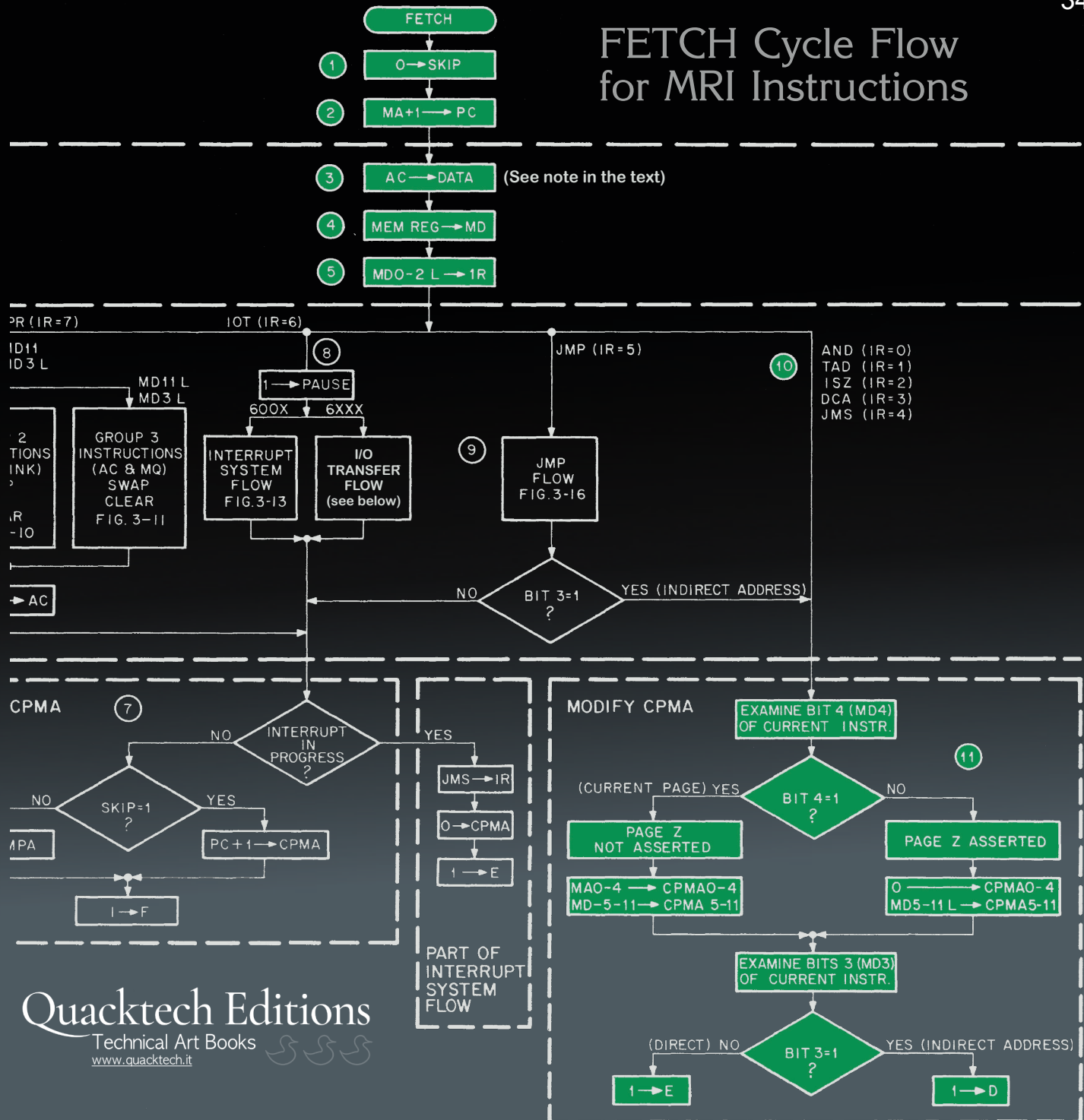


Above: Rear view of the PDP-8/m cabinet. The “/m” was the compact version and, surprisingly, its overall volume is not much greater than that of a modern gaming computer. Its construction is extremely robust, which also means that the machine is quite heavy—not least because of the linear power supply and its large transformer.

The cover, removed here, gives access to the power-supply circuitry. In practice, however, serious work on this section is easier if the entire power supply is removed from the frame. The PDP-8/m should not be operated for more than 15 minutes without the cover, since the cover is part of the designed airflow path. Note the power supply’s own fan, barely visible on the left side.



FETCH Cycle Flow for MRI Instructions



360 • AC3 Interrupt Inhibit Flip-Flop

- AC4 Interrupt Enable Flip-Flop
- AC5 User Flag
- AC6-11 Save Field Register

The exact meaning of some of these bits becomes more important when the memory extension and user mode mechanisms are involved. In a minimal system, the most immediately interesting bits are the Link, the interrupt request state, and the interrupt enable state.

RTF — Restore Flags (6005)

RTF performs the complementary operation to GTF. It restores several processor flags from the accumulator. In particular:

- AC0 is loaded into the Link
- AC1 is loaded into the Greater Than Flag
- AC5 is loaded into the User Flag
- AC6-11 are loaded into the Save Field Register

In addition, the interrupt system is enabled in the same delayed manner as with ION, and the interrupt inhibit flip-flop is set.

For simple interrupt experiments, ION and IOF are usually enough. For more complete systems, especially those using memory extension, GTF and RTF become much more important.

Peripheral Interrupt Enables

There is another important point: enabling the processor interrupt system is not always enough.

Many peripherals have their own local interrupt enable flip-flop. This local enable determines whether the device is allowed to assert an interrupt request when its flag becomes set.

This gives two levels of control:

- processor interrupt enable
- peripheral local interrupt enable

Both must be in the appropriate state for an interrupt to be generated and accepted.

For example, on the M8650 asynchronous data control board, the keyboard/prINTER interface has its

There was, however, a considerable difference between having an assembler available and using it comfortably. On a paper-tape-based system, the complete edit-assemble-load cycle could be painfully slow. In theory, the user had to load an editor, prepare or modify the source program, punch a new source tape, load the assembler, produce the binary object tape, and finally load that tape into the computer. With some bad luck, even a very small program could require far more time in loading, punching, and reloading than in actual programming.

For this reason, especially on small installations, many users worked very close to the machine. Short programs could be written on paper, manually assembled into octal words, and then entered directly from the front panel or through a small loader. This was not elegant, but it was often faster.

The consequence was familiar to anyone who later programmed the first microcomputers. When a correction or addition had to be made and there was no room in the existing code, moving all the following instructions was an unattractive prospect. The usual solution was to place the new code somewhere else and reach it with a JMP, followed by another JMP to return to the original flow. After a few modifications, a once-linear program could become a small maze of jumps, patches, and detours.

Another useful tool in this context was **ODT-8, the Octal Debugging Technique**. Unlike a symbolic debugger, ODT worked directly in the octal representation of the program. It allowed the user to examine and modify memory locations, test small portions of code, patch a program, and generally communicate with the running machine at a very low level. In a system where programs were often assembled by hand, loaded from paper tape, or corrected directly in memory, such a tool was extremely valuable. It did not hide the PDP-8 behind a comfortable abstraction; rather, it gave the programmer a practical way to work with the machine in its own native language: addresses and 12-bit octal words, but more conveniently than from the front panel alone.

I remember that we used to refer to tools of this kind as “software panels.” In the first microcomputers I produced, I could not resist the temptation to provide them with a front panel inspired by the PDP-8, although with hexadecimal displays instead of rows of binary LEDs (see photo). This was especially useful for those users who had neither a Teletype nor a video terminal, as was often the case at the time. It gave the microcomputer a splendid, almost professional appearance, and I must admit that it was a little painful when we eventually removed it completely in favor of a ROM-based debugger.

Programming Languages

Programming languages were also an important part of the offering. FOCAL, DEC’s conversational language for formula calculation and numerical work, was especially well suited to small interactive systems. It gave the user a relatively friendly way to perform calculations, experiments, and demonstrations without writing full assembly-language programs. FORTRAN was also available, in forms adapted to the limited resources of the PDP-8. BASIC would become important as well, espe-

390 Development Cycle

So let us look at the typical development cycle of a 1970s programmer:

- editor: normally the Symbolic Editor
- assembler: with two main choices, PAL III or MACRO-8
- debugger: ODT or DDT
- loader: the Binary Loader
- linker: let us not even talk about it.

If you had only a paper-tape system, development was very painful and slow. It was possible, nevertheless, provided you had the patience of a saint.





If you also want the listing file, perform a third pass: press ADDR LOAD again, raise SW03, then press CONT and send the file once more. If you are using GBT, press the F8 key instead. 405

The listing file will then be downloaded. The sample run follows.

```

      /MACRO-8 TEST PROGRAM: PRINT MESSAGE, THEN ECHO KEYBOARD
0200  7300  START,  CLA CLL
0201  1225          TAD MSGP
0202  3224          DCA PTR
0203  1624  NEXT,   TAD I  PTR
0204  7450          SNA
0205  5211          JMP ECHO
0206  4216          JMS OUT
0207  2224          ISZ PTR
0210  5203          JMP NEXT
0211  6031  ECHO,   KSF
0212  5211          JMP ECHO
0213  6036          KRB
0214  4216          JMS OUT
0215  5211          JMP ECHO
0216  0000  OUT,    0
0217  6046          TLS
0220  6041          TSF
0221  5220          JMP .-1
0222  7200          CLA
0223  5616          JMP I  OUT
0224  0000  PTR,    0
0225  0226  MSGP,   MSG
0226  0310  MSG,    310
0227  0305          305
0230  0314          314
0231  0314          314
0232  0317          317
0233  0215          215
0234  0212          212
0235  0000          0
```

Yes, the example is minimal, but it gave me great satisfaction. After all, it represents the arrival point of several converging efforts: restoring the PDP-8 hardware, bringing the original DEC software back to life, and creating the GBT terminal emulator needed to make the whole experiment practical.

Left: DEC's "Programming Languages" was one of the favorite books of my professional youth. It was the second volume of a small documentation set that also included "Introduction to Programming". The book covered not only BASIC, FORTRAN, and FOCAL, but also the assemblers — although I would probably not have classified them in that way.

It seems almost impossible that, for such a widely distributed book, no decent photograph appears to have survived. The image shown here was therefore recreated by me with the aid of Artificial Intelligence — ChatGPT.

420 JMS Jump to Subroutine

```
/ JMS JUMP TO SUBROUTINE
0100    7200    CLA          /CLEAR ACCUMULATOR
0101    4310    JMS 110
0102    5301    JMP 101     /LOOP

0110    XXXX                /LEAVE IT FREE FOR RETURN ADDRESS
0111    7001    IAC          /INCREMENT ACCUMULATOR
0112    5710    JMP I 110   /RETURN
```

Its First Words

```
/ HELLO←ECHO←FULL←LISTING.TXT
/ PDP-8 LOW-SPEED CONSOLE TEST
/ LOAD WITH STANDARD RIM LOADER AT 7756
/ START AT 0200
```

ADDR	CODE	LABEL	INSTRUCTION	COMMENT
----	----	-----	-----	-----
0200	7300	START,	CLA CLL	CLEAR AC AND LINK
0201	1225		TAD MSGP	LOAD ADDRESS OF MESSAGE
0202	3224		DCA PTR	STORE IT INTO AUTO-USED POINTER
0203	1624	NEXT,	TAD I PTR	GET NEXT CHARACTER FROM MESSAGE
0204	7450		SNA	SKIP IF CHARACTER IS NON-ZERO
0205	5211		JMP ECHO	ZERO TERMINATOR: ENTER ECHO LOOP
0206	4216		JMS OUT	PRINT CHARACTER IN AC
0207	2224		ISZ PTR	INCREMENT POINTER
0210	5203		JMP NEXT	CONTINUE WITH NEXT CHARACTER
0211	6031	ECHO,	KSF	KEYBOARD FLAG SET?
0212	5211		JMP ECHO	NO: WAIT
0213	6036		KRB	READ CHARACTER AND CLEAR KEYBOARD FLAG
0214	4216		JMS OUT	ECHO IT TO TELEPRINTER
0215	5211		JMP ECHO	CONTINUE FOREVER
0216	0000	OUT,	0	JMS RETURN ADDRESS STORAGE
0217	6046		TLS	LOAD TELEPRINTER BUFFER AND START TX
0220	6041		TSF	TELEPRINTER FLAG SET?
0221	5220		JMP .-1	NO: WAIT FOR TX COMPLETE
0222	7200		CLA	CLEAR AC BEFORE RETURNING
0223	5616		JMP I OUT	RETURN FROM SUBROUTINE
0224	0000	PTR,	0	MESSAGE POINTER