PlasMa

aka Phil's lights and switches Machine

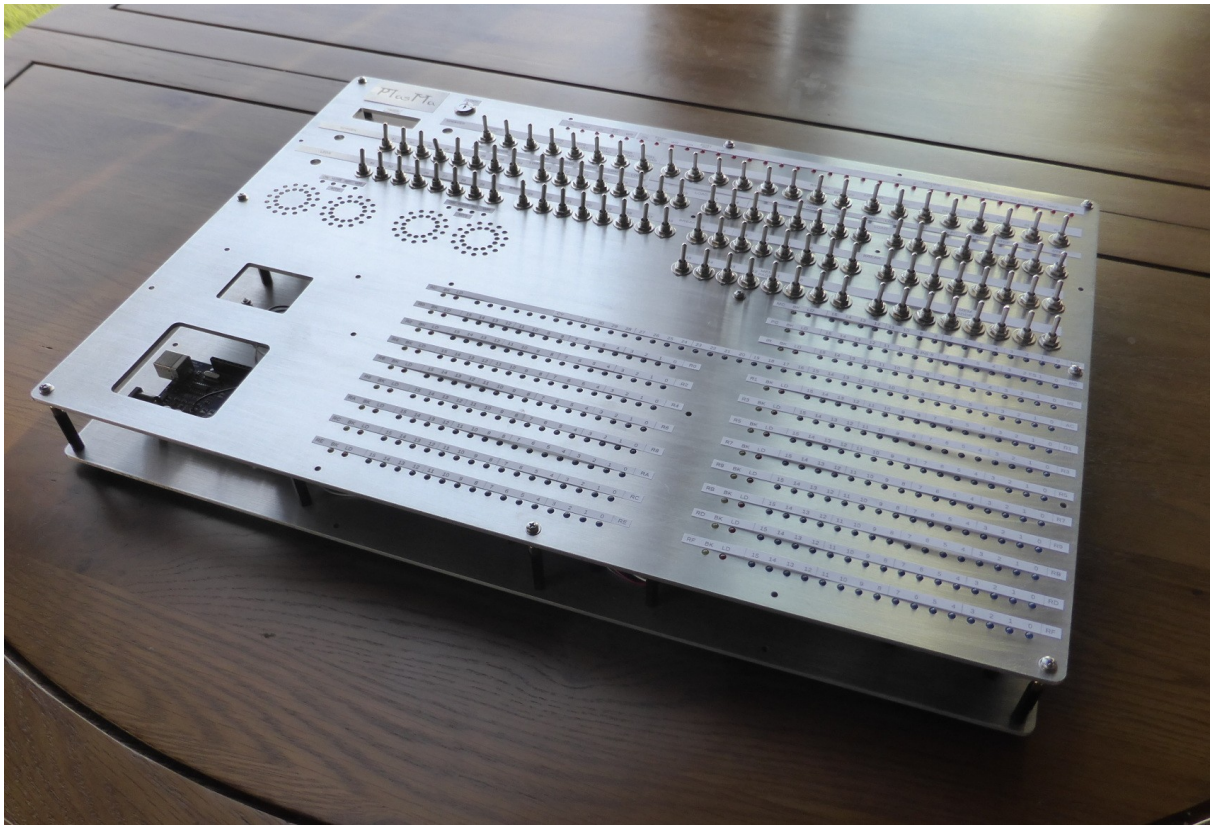
Mini-Mainframe

Simulator Project

Machine Manual

Phil Tipping

www.philizound.co.uk



Machine under construction (progress to date)

Table of Contents

1. Introduction

The PlasMa machine is a simple homage to mainframe computers from my early 1970's career at International Computers Ltd (ICL) in the UK. It makes no pretence to simulate any specific computer, either in design or performance; its aim is purely to rekindle the hands-on ‘lights and switches magic’ from that era, and explore the challenges of programming a machine from scratch.

It simulates a complete mini/mainframe system in a self-contained desktop-sized box containing real lights and switches. The system comprises a 'mini-like' processor with a small amount of main memory running a relatively simple instruction set, and assorted 'mainframe-like' peripherals such as a paper tape reader and punch, mag tapes, exchangeable disks and an operators console.

The machine is currently a ‘work in progress’ so some functions are not yet implemented; the cover picture shows a partially constructed machine. Progress is being documented in a series of YouTube videos and on the philizound.co.uk website.

The project is just a personal indulgence re-exploring the 'good old days', but if you are interested in building a similar machine for developing programs and ideas, details are in the Construction section. The design caters for novices and experts alike, so just skip any parts you are already familiar with.

2. Acknowledgements

The Toy instruction set is used by kind permission of Robert Sedgewick and Kevin Wayne at Princeton university, and is described in their book 'Computer Science'. More details at:

https://introcs.cs.princeton.edu/java/home and https://introcs.cs.princeton.edu/java/60machine

Coursera course site: https://www.coursera.org/learn/cs-algorithms-theory-machines

Thanks also to Adrian Rawson for suggestions and support.

3. Documentation

These following documents are downloadable from the philizound.co.uk website or by contacting me directly (email address is at the bottom of the website page).

- PlasMa Machine Manual
- PlasMa Instruction Set - Toy-A
- PlasMa Instruction Set - Toy-B
- PlasMa Instruction Set - Advanced
- PlasMaSim Simulator Manual
- Plasm Assembler Manual

Terminology and conventions

3.1 Instructions and opcodes

In this manual, **opcode** is the functional part of an instruction, such as add, subtract, jump etc. An **instruction** contains the opcode plus its operands, such as register numbers, memory addresses, literal numbers etc.

3.2 Binary and hex

This manual is all about computers and binary, so wouldn’t be complete without a brief discussion on hex numbers.

Our everyday numbers use the **decimal** system, otherwise known as base 10, which means any whole number can be represented by a combination of 10 unique characters: 0, 1 ... 8, 9.

Conventional computers use the **binary** system, base 2, which means numbers are represented with a combination of 2 unique characters, 0 and 1.

The downside is that binary numbers are long and unwieldy, e.g. decimal 7623 is binary 1110111000111, and there is no obvious correlation between characters in the two bases, making it difficult to convert mentally between the two.

One compromise commonly used is the hexadecimal system, or **hex**, which uses base 16. The 16 unique characters (**hex digits**) are the 10 numbers 0 to 9 and the 6 letters A to F (lower-case can also be used), e.g. 7623 is hex 1DC7 or 1dc7.

As with the decimal system, the left end is the most significant end, and the right is the least significant.

Mentally converting between decimal and hex is still tricky unless you know your 16 times table, but it's now much easier between hex and binary. If you split the binary representation into sets of 4 bits, starting from the right-hand end and padding the left with zeros as required, you can see that each hex digit equates to a set.

0001	1101	1100	0111
1	D	C	7

Sets of 8 bits are **bytes**, and sets of 4 bits are **nibbles**, so each hex digit equates to a nibble. This means you only have to learn/recognise 16 combinations of 4 bits.

The first 2 sections in the Hex conversion table show the mapping for all 16 values (0 to 15) which can be stored in one nibble.

The shared use of characters 0 to 9 in decimal and hex numbers can cause confusion. It's fairly obvious a number is hex if it contains one or more letters, but if it only contains numbers, such as 123, it could be decimal or hex; both completely different.

The convention used in the PlasMa project is to prefix hex numbers with '**$**' if there is any ambiguity.

Another convention used is to number bits from the right-hand, least significant (ls) end, starting at 0. This may seem counter-intuitive, but it helps when converting binary to decimal as the bit number is then the 'power of 2' which can be added together for all bits which are '1'.

Nibbles are also numbered the same way for consistency; from right to left, starting at 0.

3.3 Negative numbers

There is no special sign character in binary or hex, unlike the '-' prefix used with decimal numbers. PlasMa uses the standard 2's complement system, where the most significant bit is 0 for positive numbers, and 1 for negative numbers.

To convert a positive number into a negative one, invert all the bits (1's complement), then add 1. The Advanced instruction set contains functions for both of these complement methods.

How a set of binary bits is interpreted depends on context, e.g. some PlasMa instructions use 4 bits to represent a signed number, others use 4 bits to represent unsigned numbers. Similarly, some instructions use 8 bits, both signed and unsigned, others 16 bits, etc.

Numbers within your own programs can be interpreted however you like.

The advantage with the complement system is that operations such as add and subtract work the same way on both signed and unsigned numbers[1]. The disadvantage is that signed numbers have a smaller maximum positive value, about half that of unsigned numbers, as they lose a bit for the sign; see the Hex conversion table for examples.

1 On the other hand, *shifting* bits left and right will cause problems if the sign bit is not managed properly; some instructions have two variants to cater for both signed and unsigned numbers.

3.4 Hex conversion table

hex	bin	dec	hex	bin	decimal unsigned	dec signed
0	0000	0	8	1000	8	-8
1	0001	1	9	1001	9	-7
2	0010	2	A	1010	10	-6
3	0011	3	B	1011	11	-5
4	0100	4	C	1100	12	-4
5	0101	5	D	1101	13	-3
6	0110	6	E	1110	14	-2
7	0111	7	F	1111	15	-1

hex	decimal unsigned	decimal signed
7F	127	+127
80	128	-128
FF	255	-1
3FF	1023	+1023
7FFF	32767	+32767
8000	32768	-32768
FFFF	65535	-1
7FFFFFFF	2147483647	+2147483647
80000000	2147483648	-2147483648
FFFFFFFF	4294967295	-1

4. Overview

The machine is programmed using toggle switches to load binary values into registers and memory. The program can then be run or single-stepped, and break-points set for debugging. The internal workings are visible at all times in binary format, and include the program counter, instruction register, accumulator and work registers.

PlasMa simulates a microcoded computer so, by loading different microcodes, is able to execute different instruction sets[2]. Three microcodes are provided, so there are 3 instruction sets, all emulating fictitious computer architectures using the same lights and switches on the front panel.

Full details of these instruction sets and I/O functions are in separate manuals; see Documentation.

4.1 Microcodes 1 and 2 (Toy-A and Toy-B)

These are for educational purposes and emulate two variants of the ‘Toy’ computer used as a teaching aid at Princeton university, USA.

There are 16 functions and 16 registers, all represented as 4-bit nibbles within a 16-bit instruction, making it very easy to mentally translate to and from binary.

The university uses variants of the 16 functions to demonstrate different programming techniques, so PlasMa simulates two of the most common, which I've named Toy-A and Toy-B. In both emulations, Princeton’s terminology is retained where possible so existing tutorials on the internet can be followed (see the links in Acknowledgements). All examples and existing programs should run with little or no change.

Toy-B has been enhanced with extra system and I/O functions from the Advanced (microcode 3) emulation, such as timers, delays and an emulated paper-tape reader and punch, but these changes should be backwards compatible with the Princeton design.

4.2 Microcode 3 (Advanced)

This is for the more adventurous, and emulates an advanced computer. It moves on from the Toy architecture by adding a 32-bit accumulator and many more instructions, including I/O for accessing emulated peripherals based on removable sd-cards (currently TBA), such as a paper tape reader and punch, magnetic tape decks and exchangeable disk drives.

The first few words of memory are non-volatile to simulate a small ‘fixed/core store’; this can be programmed with a bootstrap routine for loading larger programs from the peripherals.

2 Microcodes made it cheaper to fix hardware bugs. They also allowed customers to upgrade their hardware without having to convert their old software straight away. In ICL’s case, their ‘new range’ VME computers could also run old George-based applications.

5. Operation

5.1 Startup/Loading a microcode

The machine needs to be started up in a specific way otherwise nothing will happen; step one is to load a microcode.

The microcode interprets an instruction set, so by loading different microcodes, different instruction sets can be obeyed so different computers can be emulated.

At switch-on, no microcode is loaded so the machine will not know how to interpret *any* instructions.

Some controls are disabled until a microcode is loaded.

To load a microcode[3]:

- Set the required microcode number (1..3) in binary on the 4 most significant LOD VAL switches.
- Press the RESET switch to clear all registers.
- Use the SET/RESET FMP switch to turn on the FMP light; this enables 'force microprogram load'.
- Press the RUN switch; PlasMa will load the required microcode[4] then halt. The HALT light will turn on and the MCODE lights will show the microcode number in binary.
- Press the RESET switch to clear the halt state. The HALT light will turn off unless the Instruction register (IR) contains a halt instruction.
- Use the SET/RESET FMP switch to turn off the FMP light; all controls are now enabled.

5.2 Shutdown

There is no close-down procedure. The OFF switch stops all processing immediately, so for safety the switch has to pressed twice within 1 second otherwise it is ignored.

5.3 Loading registers and memory

This can only be done when PlasMa is stopped.

- Set the value to be loaded on the LOD VAL switches.
- Use the SEL LOD UP/DN switch to select the register you want to load; the Target LOD light indicates the target register. Only one register can be the target at any one time.
- Use the LOD/CLR switch to load or clear the register. If the register is less than 32 bits wide, the high order bits on the LOD VAL switches are ignored.

If the target is the Instruction register (IR), the memory location at the address in the Program counter (PC) is also loaded or cleared. In addition, if you use the LOD/CLR INC switch instead of the LOD/CLR switch, PC is automatically incremented after the operation. This makes it easier to load or clear consecutive locations.

If the target is the Memory contents (MC), the memory location at the address on the MEM ADDR switches is loaded or cleared. You can change these switches at any time to inspect any memory location.

3 The startup sequence is included out of nostalgia for ICL's P3/2970 machine. The details may be fuzzy after all these decades, but I do remember the sequencing of 'reset' and 'set/reset fmp' confusing the chief engineer many times, much to the amusement of us young apprentices.

4 The microcode would normally be loaded from a specified peripheral; PlasMa uses built-in microcodes.

5.4 Breakpoints

To set a breakpoint:

- Set the required breakpoint match value on the BRK VAL switches.
- Use the SEL BRK UP/DN switch to select the register you want to check; the Target BRK light indicates the target register. Only one register can be the target at any one time.
- Use the BRK ON/OFF switch to turn the BRK light on.

To turn breakpoints off:

- Use the BRK ON/OFF switch to turn the BRK light off.

When breakpoint checking is on and a program is running, PlasMa compares the target register's contents with the value on the BRK VAL switches after every instruction. If the target register is less than 32 bits, the high order bits on the BRK VAL switches are ignored.

If a match occurs, PlasMa stops and turns on the MATCH light.

You can run or single-step from this point without having to turn off break-point checking (assuming the instruction in IR is valid and not a halt instruction). If you choose run, PlasMa will stop again when another match is found.

All the BRK switches can be used at any time.

5.5 Writing programs

Programming/coding is outside the scope of this manual, but basically you break down your problem into a list of instructions using the ones available in your chosen instruction set.

The 3 instruction sets provided vary in complexity, so your choice depends on the problem's requirements and your programming proficiency. Instruction tables are in the corresponding Instruction Set manual; see Documentation.

You *could* write your program directly using the hex values shown in the table, but this 'machine code' would be difficult to understand, especially if you need to debug or modify it later, or if you want to share it with others.

A clearer way is to use a slightly higher 'assembler code' which uses mnemonics or names for each instruction. Converting the assembly level program into machine code can be a manual process, where you look up each mnemonic in the table and write the 4 hex digits alongside each instruction or data value, or you can automate the process using an assembler program.[5]

More advanced techniques use even higher level instructions which require a compiler program to convert to machine code.

Whichever method you pick, use plenty of notes or comments to remind you (and possibly others) of the logic behind your thinking[6].

Once the machine code has been created, the values can be loaded into PlasMa's memory, either manually using the switches (see Loading programs), or loaded from a paper tape, in which case you will need a 'loader' program running on PlasMa to read the tape.

5 The Plasm assembler is one such program for offline assembly on a PC. An assembler which runs on PlasMa itself would make an interesting challenge.

6 For example, commenting 'ld r3 5' with 'load reg 3 with 5' adds no useful information; it doesn't explain the underlying logic or *why* you are loading 5.

5.6 Loading programs

5.6.1 Loading consecutive locations

- Load PC with the first address; see Loading registers and memory.
- Set IR to be the target load register.
- Set the value on the LOD VAL switches.
- Use the LOD/CLR INC switch to load IR and memory, and automatically increment PC.
- Repeat the previous 2 steps for each value.

5.6.2 Loading non-consecutive locations

Either:

- Load PC with the address; see Loading registers and memory.
- Set IR to be the target load register.
- Set the value on the LOD VAL switches.
- Use the LOD/CLR switch to load IR and memory.
- Repeat all the above steps for each non-consecutive location.

Or:

- Set MC to be the target load register.
- Set the address directly on the MEM ADDR switches.
- Set the value on the LOD VAL switches.
- Use the LOD/CLR switch to load memory.
- Repeat the previous 3 steps for each non-consecutive location.

5.7 Saving programs

All memory contents are lost when PlasMa is switched off. If you want to retain your programs and avoid entering them via the switches after every switch on, they need to be written to non-volatile storage.

Toy-A has no such storage, although you could simplify entering programs by writing a small program to read subsequent values from the keypad instead of the LOD VAL switches, but this will still be lost when you switch off.

Toy-B and the Advanced emulations contain non-volatile peripherals. Both have a paper tape reader and punch, and the Advanced also has mag tape decks and disc drives. Whichever you use, you will still need to enter a basic ‘loader’ program first to read data from them.

The Advanced emulation has a ‘fixed store’ which occupies the first few words of memory. Anything written to these addresses will be present every time you switch on, so a basic loader program could be saved here. To run this after switch on, load microcode 3, set PC to zero and press the RUN switch.

The fixed store can only hold 16 words of instructions and/or data, but this is enough to read a paper tape[7]. This tape could contain a larger program, which in turn could load even larger programs from the mag tapes or disk drives[8].

7 It *may* also be enough to access the mag tapes and discs but this has not been confirmed at the time of writing.
8 This process is called bootstrapping.

5.8 Running programs

You can run a program at various speeds or single-step it one instruction at a time. Either way, you need to set PC to the start address of your program:

- Make sure you have the correct microcode loaded; see Startup/Loading a microcode.
- Load PC with the address of your program's first instruction. You can either use the INC/DEC PC switch, or use the LOD VAL switches; see Loading registers and memory. If the required address is zero, you could just use the RESET switch as a quick way of setting PC to zero, but note this will also clear *all other* registers.

To start the program running, use the RUN switch for full-speed operation, or the SLOW/MED switch for slow or medium speeds. You can use any of these 3 switches to alter the speed on-the-fly; you don't need to stop the program first.

Use the STEP switch to obey your program one instruction at a time. This is useful for new programs as you can inspect the registers at each step to confirm it is working as intended. Once you are satisfied things are ok, you can press one of the 3 run switches and the program will run from that point.

You could also set a breakpoint to cause your program to stop when the breakpoint condition is met, regardless of the run speed; see Breakpoints. This means you can either:

- Stop at a specific address (set PC as the target break register).
- Stop when a specific instruction is about to be executed (set IR as the target break register).
- Stop when a work register contains a specific value (set R0-RF as the target break register).
- Stop when the accumulator contains a specific value (set ACC as the target break register).
- Stop when a specific memory location contains a specific value (set the address on the MEM ADDR switches and set MC as the target break register).

6. Lights

6.1 Status lights

6.1.1 SLOW/MED/FAST lights

These indicate the current run speed. They are only lit when PlasMa is running.

6.1.2 ILLEGAL light

This is lit if an illegal instruction is encountered, in which case PlasMa will stop running if it is not already stopped. The instruction cannot be obeyed (by definition), so the run and step switches have no effect. If you want to continue the program, alter the instruction in IR or the address in PC.

6.1.3 HALT light

This is lit if a halt instruction is encountered, in which case PlasMa will stop running if it is not already stopped. The instruction prevents the next instruction from being fetched, so the run and step switches have no effect. If you want to continue the program, alter the instruction in IR or the address in PC.

The light is also lit when a microcode has just been loaded, in which case press the RESET switch to clear it.

6.1.4 KPAD light

This is lit in the Toy emulations when an I/O instruction is waiting for TTY input from the hex keypad. The program is halted and any hex digits entered on the keypad are written to the destination register. Press the RUN switch or STEP switch to resume the program.

6.1.5 MATCH light

This is lit if a break-point match is encountered, in which case PlasMa will stop running if it is not already stopped. If the instruction in IR is valid (and is not a halt instruction), you can continue from this point by pressing the run or step switches.

6.1.6 BRK light

This is lit if break-point checking is on. It can be turned on and off at any time via the BRK ON/OFF switch.

6.1.7 MCODE lights

These 3 lights indicate the current microcode number in binary. Numbering starts at 1, so if they are not lit, there is no microprogram loaded; see Startup/Loading a microcode.

6.1.8 FMP light

This is lit if the FMP (force microprogram load) flag is set. It can be turned on and off via the SET/ RESET FMP switch. When the light is on, only controls related to loading a microcode are enabled.

6.1.9 PTR 7/8 lights

These 2 lights indicate the paper tape reader device format: 7 or 8-hole. Toggling between the 2 formats is simulated by pressing the PTR Unload switch *before* loading a tape.

It is up to you to load tapes which have been written in the correct format; I/O functions trying to read a tape in the wrong format will return incorrect data or a read-error response.

6.1.10 PTR LD light

This is lit to simulate a paper tape has been loaded into the tape reader. It does not necessarily mean the corresponding sd-card is ready or that it contains valid data; this can only be determined by your PlasMa program; see Peripherals and SD-Cards. Do not remove or insert the sd-card if this light is on; use the corresponding Unload switch to turn it off.

6.1.11 PTR RD light

This is lit if a tape read I/O function is in progress.

6.1.12 PTP 7/8 lights

These 2 lights indicate the paper tape punch device format: 7 or 8-hole. Toggling between the 2 formats is simulated by pressing the PTP Unload switch *before* loading a tape.

6.1.13 PTP LD light

This is lit to simulate a blank paper tape has been loaded into the tape punch. Do not remove or insert the sd-card if this light is on; use the corresponding Unload switch to turn it off.

6.1.14 EDS LD light

This is lit to simulate a disk has been loaded onto the EDS (exchangeable disk system) drive. There are 2 drives: EDS0 and EDS1. Do not remove or insert the sd-card if this light is on; use the corresponding Unload switch to turn it off.

6.1.15 EDS RD/WR lights

These are lit if a disk read or write I/O function is in progress.

6.2 MT status lights

These 4 lights are located above each of the simulated mag tape deck lights. There are 2 decks: MT0 and MT1.

6.2.1 MT WPR light

This is lit if the loaded tape spool is fitted with a write-permit ring. Fitting and removing the WPR is simulated by pressing the relevant MT Unload switch *before* loading a tape.

Note WPR means write-*permit* ring, and not write-*protect* ring, so the tape can only be written if the light is on.

6.2.2 MT LD light

This is lit to simulate a mag tape has been loaded and threaded onto the tape deck. Do not remove or insert the sd-card if this light is on; use the corresponding Unload switch to turn it off.

6.2.3 MT RD/WR lights

These are lit if a tape read or write I/O function is in progress.

6.3 Register lights

6.3.1 Memory contents (MC)

This shows the 16-bit contents of the memory location whose address is on the MEM ADDR switches. The switches can be changed at any time, allowing you to inspect any location in memory. The location can also be treated as a 'target register' for loading (see Loading registers and memory) and break-point checking; see Breakpoints.

6.3.2 Program counter (PC)

This shows the memory address (only 8 bits are used in the Toy emulations) of the instruction about to be executed (obeyed). The instruction itself is shown in the Instruction register (IR).

PC automatically increments after each instruction is executed, unless that instruction loads a specific value into it, such as with a jump instruction.

6.3.3 Instruction register (IR)

This shows the 16-bit contents of the memory location whose address is in the Program counter (PC) register. It is the instruction which will be executed next when the program is run or single-stepped.

Writing to IR using the LOD/CLR switch or LOD/CLR INC switch will also write to the corresponding memory location.

6.3.4 Accumulator (ACC)

This shows the contents of the 32-bit accumulator. Not used in the Toy emulations.

6.3.5 Carry flag (CY)

This shows the state of the 1-bit carry flag. Not used in the Toy emulations.

6.3.6 Work registers (R0-RF)

These show the contents of the 16 x 16-bit work registers. In the Toy-A emulation, register R0 is read-only and always contains zero.

6.4 Target LOD light

This is to the left of each register and is lit when that register is the current target for load operations via the LOD/CLR switch. It can be changed using the SEL LOD UP/DN switch; see Loading registers and memory.

There is no safety warning or 'undo' option, so do not press the LOD/CLR switch before confirming the target register is the one you want to overwrite.

6.5 Target BRK light

This is next to the Target LOD light for each register and is lit when that register is the current target for break-point checking. It can be changed using the SEL BRK UP/DN switch; see Breakpoints.

7. Switches

7.1 Control switches

These 2-way momentary switches can be moved up or down depending on the function required. Some switches have no effect if PlasMa is running or if there is no microcode loaded. In the following descriptions, assume the switch has no effect if the condition is not listed.

7.1.1 RUN switch

If stopped, start PlasMa running (executing instructions) at full speed from the current address in PC. The speed is shown on the SLOW/MED/FAST lights.

If running, change speed to full and continue running.

If no microcode is loaded, and PlasMa is stopped, and FMP is reset, and PC is zero, load the microcode as specified on the ms nibble of the LOD VAL switches, then halt.

7.1.2 STEP switch

If stopped, execute a single instruction from the current address in PC, increment PC, then remain stopped.

7.1.3 SLOW/MED switch

If stopped, start PlasMa running at slow or medium speed from the current address in PC.

The speed is shown on the SLOW/MED/FAST lights.

If running, change speed to slow or medium and continue running.

7.1.4 RESET switch

If stopped, clear all registers to zero, reset the MATCH light and KPAD light, and reset the System Timer.

7.1.5 STOP switch

If running, stop PlasMa. The SLOW/MED/FAST lights will turn off.

7.1.6 SET/RESET FMP switch

If stopped, set or clear the FMP (force microprogram load) flag. The FMP light shows the state.

7.1.7 INC/DEC PC switch

If stopped, increment or decrement the address held in PC. IR always shows the memory contents at this address, so you can use this switch to inspect consecutive memory locations.

7.1.8 SEL LOD UP/DN switch

Select the target register for a subsequent load operation via the LOD/CLR switch. The Target LOD light will move accordingly.

7.1.9 SEL BRK UP/DN switch

Select the target register for break-point checking when the BRK light is lit. The Target BRK light will move accordingly.

7.1.10 LOD/CLR switch

If stopped, load or clear the target load register. LOD writes the value on the LOD VAL switches to the register, CLR resets the register to zero. If the target is IR, the memory contents at the address held in PC are also loaded or cleared.

There is no safety warning or 'undo' option, so do not operate this switch before confirming the target register is the one you want to overwrite. See Loading registers and memory.

7.1.11 LOD/CLR INC switch

Same as the LOD/CLR switch but only if IR is the target register. PC is automatically incremented after the load/clear operation.

7.1.12 BRK ON/OFF switch

Turn break-point checking on or off. The BRK light shows the state. The switch can be operated at any time.

If a program is running and break-point checking is on, the target break register is checked with the contents of the BRK VAL switches after each instruction is executed. The program will stop if they are equal, and the MATCH light will turn on. See Breakpoints.

7.1.13 PTR LOAD switch

Simulate loading a paper tape onto the tape reader. A valid sd-card must already be inserted into the ptr card socket.. If the card is valid and ready, the PTR LD light will turn on. Do not remove the sd-scard while this light is on. See Punched Paper Tapes (PT).

7.1.14 PTR UNLOAD switch

Simulate unloading a paper tape from the tape reader. Only remove the sd-card from the ptr card socket when the PTR LD light has turned off.

7.1.15 PTP LOAD switch

Simulate loading a paper tape onto the tape punch. A valid sd-card must already be inserted into the ptp card socket.. If the card is valid and ready, the PTP LD light will turn on. Do not remove the sd-scard while this light is on. See Punched Paper Tapes (PT).

7.1.16 PTP UNLOAD switch

Simulate unloading a paper tape from the tape reader. Only remove the sd-card from the ptp card socket when the PTP LD light has turned off.

7.1.17 MT LOAD switch

Simulate loading a mag tape onto tape deck mt0 or mt1. A valid sd-card must already be inserted into the relevant mt card socket. If the card is valid and ready, the MT LD light will turn on. Do not remove the sd-scard while this light is on. See Magnetic Tapes (MT).

7.1.18 MT UNLOAD switch

Simulate unloading a mag tape from tape deck mt0 or mt1. Only remove the sd-card from the mt card socket when the corresponding MT LD light has turned off.

7.1.19 EDS LOAD switch

Simulate loading an exchangeable disk onto drive eds0 or eds1. A valid sd-card must already be inserted into the relevant eds card socket. If the card is valid and ready, the EDS LD light will turn on. Do not remove the sd-scard while this light is on. See Exchangeable Disk System (EDS).

7.1.20 EDS UNLOAD switch

Simulate unloading a mag tape from tape deck mt0 or mt1. Only remove the sd-card from the eds card socket when the corresponding EDS LD light has turned off.

7.1.21 OFF switch

Turn off the machine. To avoid accidents, the switch must be pressed twice within 1 second.

Note the circuitry is still powered so switch off or disconnect the power supply when not in use.

7.1.22 ON/LAMPS switch

If the machine is off, turn on the machine.

If the machine is already on, perform a 'lamp test' by turning all lights on for a brief period.

7.2 LOD VAL switches

The value on these 32 latching switches is loaded into the target load register and/or memory when the LOD/CLR switch or LOD/CLR INC switch is pressed; see Loading registers and memory.

The LOD VAL switches can be changed at any time, but the load/clear switches only work if PlasMa is stopped.

If the target register has less than 32 bits, the high order bits are ignored.

The ms 4 bits are also used to define the required microprogram to load; see Startup/Loading a microcode.

The Toy-B and Advanced instruction sets have an I/O function to read these switches.

7.3 BRK VAL switches

The value on these 32 latching switches is checked with the target break register when break-point checking is on and PlasMa is running; see Breakpoints. The switches can be changed at any time.

If the target register has less than 32 bits, the high order bits are ignored.

The Toy-B and Advanced instruction sets have an I/O function to read these switches.

7.4 MEM ADDR switches

These latching switches are used to display the contents of a single memory location on the Memory contents (MC) lights. The switches can be changed at any time.

The Toy-B and Advanced instruction sets have an I/O function to read these switches.

8. Pots

Short for potentiometers. These are rotary controls.

8.1 SPKR VOL pot

This controls the volume level of the built-in speaker. Whenever a jump instruction is executed, the speaker will click. This provides audible feedback of a program's activity, and can be a useful diagnostic aid[9].

8.2 LED/OPER BRT pots

These control the brightness of the lights and oper screen.

9 Writing programs to play tunes was a popular challenge.

9. Peripherals and SD-Cards

SD-cards are used to simulate peripheral media: paper tapes, mag tapes and exchangeable disks. SD-Card sockets represent peripheral devices.

PlasMa has a separate socket for each of the following devices:

- PTR - paper tape reader.
- PTP - paper tape punch.
- MT0 - mag tape deck 0.
- MT1 - mag tape deck 1.
- EDS0 - exchangeable disk drive 0.
- EDS1 - exchangeable disk drive 1.

9.1 SD-Card format

PlasMa has no understanding of file systems, so all sd-cards must be prepared using another computer as follows:

- If the card is larger than 2GB, create a single 2GB partition on it.
- Format the card/partition as FAT16.
- If the card is for writing, create a single empty file on it with the exact uppercase name as shown below to indicate the media type:
  - PLASMA.PT - paper tape
  - PLASMA.MT - mag tape
  - PLASMA.EDS - disk
- If the card is for reading, *first* create the file with the above name on your computer and fill it with the data required, and only *then* copy the file to a freshly formatted sd-card.
  Do not edit the file directly on the card as this may affect the physical location of the data. If PlasMa cannot find the file, the media will fail to load.
  Data formats are described in Punched Paper Tapes (PT), Magnetic Tapes (MT) and Exchangeable Disk System (EDS).

This may seem an inefficient use of sd-cards (which can be much bigger than 2GB), but the advantages are:

- It simulates real-life scenarios where media is physically mounted and loaded before use.
- It provides basic computer compatibility (e.g. for exchanging data with others or developing PlasMa programs), without the overhead of conventional file-system code.
- The added susceptibility to data errors will add to the realism and encourage writing your own checking and correcting algorithms, such as parity and cyclic redundancy checks etc. If a card becomes completely unusable by PlasMa, it will need replacing as with real-life media.

10. Peripherals and Devices

10.1 System Timer

The system timer is displayed on the front panel every second in hh:mm:ss format. It runs at a resolution of 1/100 second and can be used for timing experiments or delays within your program.

It can be controlled manually via the RESET switch, the RUN switch and the STOP switch, so the display normally represents elapsed run-time for your program.

It can also be controlled via I/O instructions, so programs can use it for their own purposes, for example timing specific sections of code, or as a delay mechanism for program loops.

10.2 Keypad

This contains 16 buttons for entering data into a program. It can be used in different ways depending on the emulation.

10.2.1 Toy-A

Toy-A uses the 'special' memory address $FF for simple I/O. When this address is read, either by a 'load' or 'load indirect' instruction, the following occurs:

- The program stops.
- The KPAD light turns on to indicate the program is waiting for user input.
- The destination register R[d] specified in the instruction is reset to zero.

While the program is waiting, pressing a keypad button shifts the register left by one nibble (discarding the ms nibble), and inserts its hex digit value (0 to F) into the ls nibble. This can be repeated as often as you like until the register holds the required value.

There is no undo or backspace, so if you enter the wrong value, just keep pressing buttons and it will scroll out of the register.

When ready, press any of the run or step switches to resume the program and turn off the KPAD light.

10.2.2 Toy-B

Toy-B contains two dedicated I/O functions for reading the keypad. One is the 'TTY Read' function as used by Princeton in the original Toy-B design, the other is a new function called 'KPad Read'.

The TTY Read function works in a similar way to Toy-A in that the program stops when the I/O instruction is executed and waits for user input; see above description for Toy-A.

The 'KPad Read' function is the same as the 'Oper Read' polling function in the Advanced emulation, and is described in the following section.

10.2.3 Advanced

The I/O function 'Oper Read' reads the keypad and writes a single value to the destination register without stopping the program. This means you could, for example, poll the keypad at regular intervals and still do useful work in between polls.

The value written to the register is $00 to $0F if one of the 16 buttons is pressed, or $FF if no buttons are pressed.

10.3 Punched Paper Tapes (PT)

Punched tapes are simulated as a single file on an sd-card; every tape needs a separate sd-card; see Peripherals and SD-Cards.

To simulate loading a paper tape for reading, insert the required sd-card into the ptr socket, then press the PTR LOAD switch. If the sd-card contains a valid file as defined in SD-Card format, the PTR LD light will turn on. If there is a problem with the card, the light will remain off. Do not remove or insert the sd-card while this light is on.

To unload a tape, press the PTR UNLOAD switch and wait for the PTR LD light to turn off. The sd-card can then be removed if required.

Similarly for writing to a paper tape, insert the sd-card into the ptp socket, then press the PTP LOAD switch, etc.

For authenticity, the file contents are lines of plain text, formatted to look like a physical tape. Each line represents a row of punched holes in character positions 1 to 9^{10}.

Two tape formats are supported, both using 8 holes per line, but with different interpretations.

The simulated tape devices can be toggled between the 2 formats by pressing the PTR UNLOAD switch or PTP UNLOAD switch just before loading a tape. The relevant PTR 7/8 lights and PTP 7/8 lights indicate the current format.

The reader and punch do not have to be the same format, so you can copy tapes from one format to another.

10.3.1 8-bit format

The 8 holes represent 8 data bits with no parity, ms data bit on the left.

No indentation or comments are allowed.

Each line can only contain the following characters:-

'O' (upper-case letter O) represents a 'hole', meaning logical '1'.

'-' (hyphen) represents a logical '0'.

'.' (full-stop) represents the sprocket and every line must contain one of these.

```
e.g.
OO-OO.-OO  = 11011.011 = $DB
123456789
```

A tape-read error will occur if:-

A line does not contain a sprocket character at position 6.

A line contains a character not in the above list, although spaces and tabs are allowed (and ignored) after position 9.

10.3.2 7-bit format

The 8 holes represent 7-bit ASCII, plus parity, with the parity and ms data bit on the left. The parity is such that, on every line, the total no. of holes (including parity) is even.

No indentation or comments are allowed, although $00 is ignored so blank lines (apart from the sprocket char) can be used to separate sections.

$7F is also ignored to allow editing where chars can be nullified by punching all 8 holes on that line.

Each line can only contain the following characters:-

'O' (upper-case letter O) represents a 'hole', meaning logical '1'.

'-' (hyphen) represents a logical '0'.

'.' (full-stop) represents the sprocket and every line must contain one of these.

10 You can replicate the old party trick of using the hole patterns as a font. The first time I saw a real mainframe was on a school trip to Leeds university where the KDF9 operators tried to impress us by typing our names into a teletype... a few seconds later, out came a paper tape 'banner' with our names on it!

```
e.g.
OO-OO.-OO  = 1011.011 = $5B
P23456789
```

A tape-read error will occur if:-

- A line does not contain a sprocket character at position 6.
- A line contains a character not in the above list, although spaces and tabs are allowed (and ignored) after position 9.
- A line contains an odd no. of holes (parity fail).

10.4 Magnetic Tapes (MT)

Mag tapes are simulated as a single file on an sd-card; every tape needs a separate sd-card; see Peripherals and SD-Cards.

To simulate loading a tape, insert the required sd-card into the mt0 or mt1 socket, then press the relevant MT LOAD switch. If the sd-card contains a valid file as defined in SD-Card format, the spools will turn as the tape is 'threaded' and the MT LD light will turn on. If there is a problem with the card, the light will remain off. Do not remove or insert the sd-card while this light is on.

To simulate unloading a tape, press the MT UNLOAD switch and wait for the MT LD light to turn off. The sd-card can then be removed if required.

10.5 Exchangeable Disk System (EDS)

Disks are simulated as a single file on an sd-card; every disk needs a separate sd-card; see Peripherals and SD-Cards.

To simulate loading a disk, insert the required sd-card into the eds0 or eds1 socket, then press the relevant EDS LOAD switch. If the sd-card contains a valid file as defined in SD-Card format, the EDS LD light will turn on. If there is a problem with the card, the light will remain off. Do not remove or insert the sd-card while this light is on.

To simulate unloading a disk, press the EDS UNLOAD switch and wait for the EDS LD light to turn off. The sd-card can then be removed if required.

11. Construction

The physical design is fairly simple; the hardware is just a set of lights and switches driven by a standard micro-controller unit (MCU). The MCU software handles all the complexity.

The hard part was deciding exactly how to interface and physically support 400+ lights and 100+ switches in a box which would fit onto the drilling machine, and still leave room for maintenance.

Numerous circuits and mechanical layouts were tried and discarded[11] before settling on a modular design, and this is now in electronic form so should be a lot easier to replicate.

The printed circuit boards (PCBs) use standard interface chips so they are not tied to any particular MCU. The current software runs on a single atmega2560 MCU executing the opcodes and driving the lights and switches, but future versions may split this over multiple MCUs for performance.

If you are interested in building one, the following items may be useful; see philizound.co.uk for details and prices:

- Simulator program PlasMaSim for evaluating the project before committing. A user manual and limited version of the program is downloadable from the website.
- Assembler program Plasm for assembling source code offline. This generates hex and paper-tape files suitable for loading into the machine and simulator. A user manual and limited version of the program is available on the website.
- Blank PCBs, etched and drilled. The current design uses 8 unique boards types, 20 boards in total. You don’t need to use all of these if you want to simplify the mechanical layout. The boards use standard interface chips so may be useful for other projects.
- MCU software in compiled .hex format. This is for the current design using 20 boards, but customised versions may be available by negotiation. The source code will not be made available.
- Front and rear panel hole positions in .dxf or .svg format (to be confirmed) corresponding to the lights and switches on the PCBs.
- (To be confirmed) Aluminium panels, cut and drilled.

11 I was ready to give up many times, but kept thinking of a pep talk by my first manager at ICL. We were in our tiny ex-cloakroom 'office'; he was sat on the floor, leaning back against the wall and said: "We've *got* to finish this... ... even if we make a pig's ear of it"!