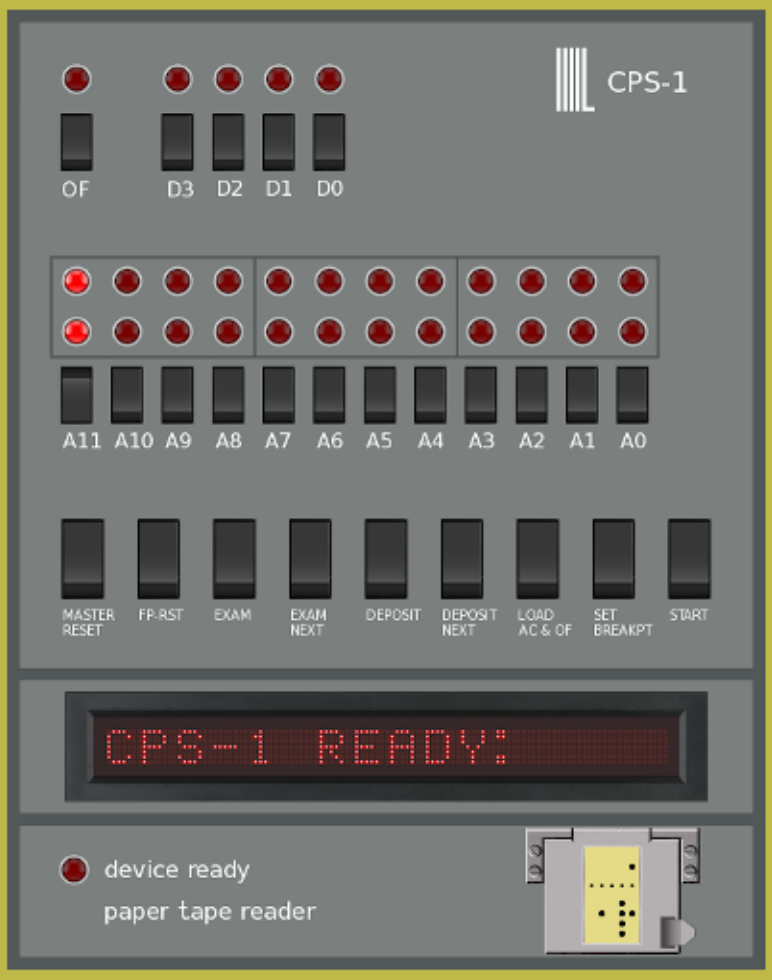
Emulator Series

MIL CPS-1 Emulator Design Notes and Programmer's Manual



Version 2.1
January 2019

York University Computer Museum
Toronto, Canada

MIL CPS-1 Emulator Design Notes and Programmer's Guide

Version 2.1
2019

Zbigniew Stachniak

York University Computer Museum
York University, Toronto, Ontario, Canada

CONTENTS

Introduction

The CPS-1 computer was the first Canadian microprocessor-based computer and one of the world's first such computers. It was designed at Microsystems International Ltd. (MIL), Ottawa, between 1972 and 1973. The computer was built around the 4-bit MF7114 microprocessor also designed by MIL engineers between 1971 and 1972 (see [1]).

The simplicity and flexibility of the MF7114's design and interfacing allowed for diverse hardware solutions concerning, for instance, the implementation of the front panel or the interfacing of peripherals. However, the underlying architecture of all CPS-1 computers was the same. The computers used the MIL MP-1 chip set consisting of the MF7114 CPU communicating with ROM (e.g. MIL MF1601 or MIL 1701A chips) and RAM (e.g. MIL MF7115 chips) memories over a 12-bit address bus and 4-bit data bus.

It is not known how many CPS-1 computers were built and sold and in what configurations. Furthermore, none of the CPS-1 computers have survived. However, a rich CPS-1 documentation collected by the York University Computer Museum (YUCoM) allowed to design and implement an emulator of the computer in a basic hardware configuration consisting of the CPU module interfaced with its front panel, paper tape reader, and a plasma display.

This manual describes the architecture, installation, and programming of the CPS-1 Emulator developed at YUCoM between 2011 and 2015.

1 The CPS-1 Architecture

The main component of a CPS-1 systems is the MF7114 microprocessor (4-bit data, 12-bit addressing). The microprocessor communicates with memory and external I/O devices using 12-bit address bus and 5 control lines (COMBUS), as well as 4-bit data bus (see [1, 2] for details). The MF7114 was designed to work with standard memory devices.

The MF7114 CPU has only four internal registers:

- 4-bit accumulator AC,
- 1-bit overflow register OF,
- 12-bit data pointer register DP,
- 12-bit program counter register PC.

Additional 16 (virtual) registers are implemented using the first 32 nibbles (or 4-bit words) of memory (more on virtual registers below).

1.1 The MF7114 instruction set

According to L.R. Schweizer (one of the main engineers behind the CPS-1 design), the MF7114's instruction set "is equivalent to that of the PDP/8 series of computers from DEC." [5]. The instruction set, given in Table 1, was first published in [2]. For each instruction, the table provides the instruction's mnemonic, binary code, and short description.

```
       AC load and store instructions

LAD    00011000          Load AC with memory[DP].
LAM    00011xxx          Load AC with memory[DP'], where DP' is DP in which bits 6-4 are
                         replaced with xxx from the instruction; DP is not modified.
LAR    00010DDD          Load AC with data register 0DDD
LAI    0111xxxx          Load AC with nibble xxxx from the instruction.
SAD    00101000          Load memory[DP] with AC.
SAM    00101xxx          Load memory[DP'] with AC, where DP' is DP in which bits 6-4 are
                         replaced with xxx from the instruction; DP is not modified.
SAR    00100DDD          Load data register 0DDD with AC

         DP load and store instructions

LDI    1101xxxxxxxxxxx   Load DP with xxxxxxxxxxxx from the instruction.
LDR    10010AAA          Load DP with address register 0AAA.
SDR    10000AAA          Load address register 0AAA with DP.
IDP    00001000          Increment DP.
DDP    10100000          Decrement DP.
SIDR   10001AAA          IDP followed by SDR.
XPD    00001111          Exchange DP and PC.
LDID   10011000          DP is loaded with three consecutive nibbles starting at address DP.
                         The least significant address nibble should be  of the form xx01.
ISZD   01101000          Increment memory[DP]; if the increment results in 0, PC is
                         incremented by 4; OF is not affected.
ISZM   01101xxx          Increment memory[DP'], where DP' is DP in which bits 6-4
                         are replaced with xxx from the instruction;
                         if the increment results in 0, PC is incremented by 4
                         (one 4-nibble instruction of rwo 2-nibble instructions).
                         DP and OF are not affected.
ISZR   01100DDD          Increment data register 0DDD; if the increment results
                         in 0, PC is incremented by 4.
```

```
         skip operations

NOP2   00000000          Increment PC by two.
NOP4   11100000          Increment PC by four.

       jump instructions can modify the least significant 8
       bits of the program counter (hence, the range of a jump
       instruction is restricted to blocks of 256 nibbles);
       let PC=aaaayyyyyyyy

JG      11100001xxxxxxxx  If AC>9, then PC=aaaaxxxxxxx.
JZ      11100010xxxxxxxx  If AC=0, then PC=aaaaxxxxxxx.
JGZ     11100011xxxxxxxx  If AC>9 or AC=0, then PC=aaaaxxxxxxx.
JT      11100100xxxxxxxx  If OF=1, then PC=aaaaxxxxxxx.
JTG     11100101xxxxxxxx  If OF=1 or AC>9, then PC=aaaaxxxxxxx.
JTZ     11100110xxxxxxxx  If OF=1 or AC=0, then PC=aaaaxxxxxxx.
JTGZ    11100111xxxxxxxx  If OF=1, or AC>9, or AC=0, then PC=aaaaxxxxxxx.
JMP     11101000xxxxxxxx  PC=aaaaxxxxxxx jump unconditionally.
JL      11101001xxxxxxxx  If AC < 10, then PC=aaaaxxxxxxx.
JN      11101010xxxxxxxx  If AC > 0, then PC=aaaaxxxxxxx.
JLN     11101011xxxxxxxx  If 0 < AC < 10, then PC=aaaaxxxxxxx.
JF      11101100xxxxxxxx  If OF=0, then PC=aaaaxxxxxxx.
JFL     11101101xxxxxxxx  If OF=0 and AC < 10, then PC=aaaaxxxxxxx.
JFN     11101110xxxxxxxx  If OF=0 and AC > 0, then PC=aaaaxxxxxxx.
JFLN    11101111xxxxxxxx  If OF=0 and 0 < AC < 10, then PC=aaaaxxxxxxx.

         more operations on AC

ADD    01011000          Load AC with AC+memory[DP]; OF affected.
ADM    01011xxx          DP is temporarily modified by replacing bits 6-4 with
                         xxx from the instruction giving DP'; load AC with
                         AC+memory[DP']; OF affected; DP is not affected.
ADR    01010DDD          Add data register 0DDD to AC
NAD    01001000          Load AC with (AC NAND memory[DP]).
NAM    01001xxx          DP is temporarily modified by replacing bits 6-4 with
                         xxx from the instruction giving DP'; load AC with
                         (AC NAND memory[DP']); DP is not affected.
NAR    01000DDD          Load AC with (AC NAND data register 0DDD).

         arithmetic operations

COM    00110000          Load AC with complement of AC.
RAR    00110001          Rotate OF and AC right one bit.
RAL    00110010          Rotate OF and AC left one bit.
IAC    00110011          Increment AC by 1; set OF to 1 if the increment results in overflow.
CLA    00110100          Set AC to 0.
CLARR  00110101          Do CLA followed by RAR.
CLARL  00110110          Do CLA followed by RAL.
STA    00110111          Set AC to 0001.
CLF    00111000          Set OF to 0.
```

```
CLFRR  00111001          Do CLF followed by RAR.
CLFRL  00111010          Do CLF followed by RAL.
IACCF  00111011          Do IAC followed by CLF.
STF    00111100          Set OF to 1.
STFRR  00111101          Do STF followed by RAR.
STFRL  00111110          Do STF followed by RAL.
IACS   00111111          Do IAC followed by STF.
```

Table 1. The MF7114 instruction set; memory[DP] denotes the nibble stored in memory at address DP.

1.2 The CPS-1 memory organization

The 12-bit address bus of the MF7114 allows to directly address 4096 nibbles of memory (1 nibble = 4 bits). This memory space can be populated with ROM and RAM devices (such as the MF7115 64-nibble RAM and MF7102 EPROM) provided that the first 32 nibbles are in RAM (cf. Fig. 1).

Figure 1. CPS-1 memory organization adopted in the emulator.

The first 32 nibbles are reserved for working (or virtual) registers that the CPU can access directly (these registers are discussed in the next section).

Typically, a segment of memory starting from address 1024 is reserved for ROM memory containing a front panel program as well as some applications programs. The MF7114 is designed in such a way that its reset (through the CPU's reset pin) always results in the program counter PC set to the address 1024 (leaving all other registers undefined). One of the early implementations of the CPS-1 computer was equipped with 1024 nibbles of RAM followed by 2048 nibbles of ROM. The remaining memory space was left unpopulated.

The CPS-1 emulator's memory map is depicted in Fig. 1. The emulator's memory consists of 1024 nibbles of RAM followed by 1024 nibbles of ROM and another 2048 nibbles of RAM. The 64-nibble memory block that starts at address 4032 is unpopulated and is reserved for interfacing with I/O devices. We shall call this memory block – the I/O memory space and discuss the CPS-1 interfacing techniques later on in this document.

1.3 Virtual registers

The CPS-1 supports eight 1-nibble data registers ($D_0, \ldots D_7$) and eight 3-nibble address registers ($Adr_0, \ldots, Adr_7$) located in the first 32 nibbles of memory. In this emulator, a 12-bit address is stored in an address register Adr_i using three consecutive nibbles in big-endian order, [1] as shown in Fig. 2:

Figure 2. Address storing convention for virtual address registers with most (resp. least) significant nibble first (res. last).

The memory address of the D_i register is $4i, i \leq 7$. The address of the first nibble of the Adr_i register is $4i + 1, i \leq 7$. In other words, the data and address registers are interwoven in the following way:

$$D_0 \quad Adr_0 \quad D_1 \quad Adr_1 \ldots D_7 \quad Adr_7$$

2 CPS-1 Interfacing Techniques

The CPS-1 used the I/0 memory space to interface I/O devices such a paper tape reader, a Self-Scan plasma display, and even the front panel. (Fig. 3 depicts the CPS-1 emulator's three I/O devices. The top module is the front panel. The middle and the bottom modules depict a Self-Scan display and a paper tape reader/punch, respectively.)

Each I/O device was connected to the computer's address and data buses and was assigned one or more (but unique) addresses from the I/O memory space. The CPS-1 communicated with an I/O device by reading from or writing to memory at assigned addresses using instructions such as LAD and LDID (reading) and SAD (writing). For instance, each time the CPU placed an address *adr* assigned to a device and executed LAD, that device responded by placing the content of mem[*adr*] on the data bus which was then placed in the accumulator. Hence, the CPU interacted with an I/O device in the same way as with a memory device.

[1] Some MIL documents concerning the CPS-1 architecture indicate the little-endian order for storing 12-bit data (e.g. numbers) and addresses in virtual registers.

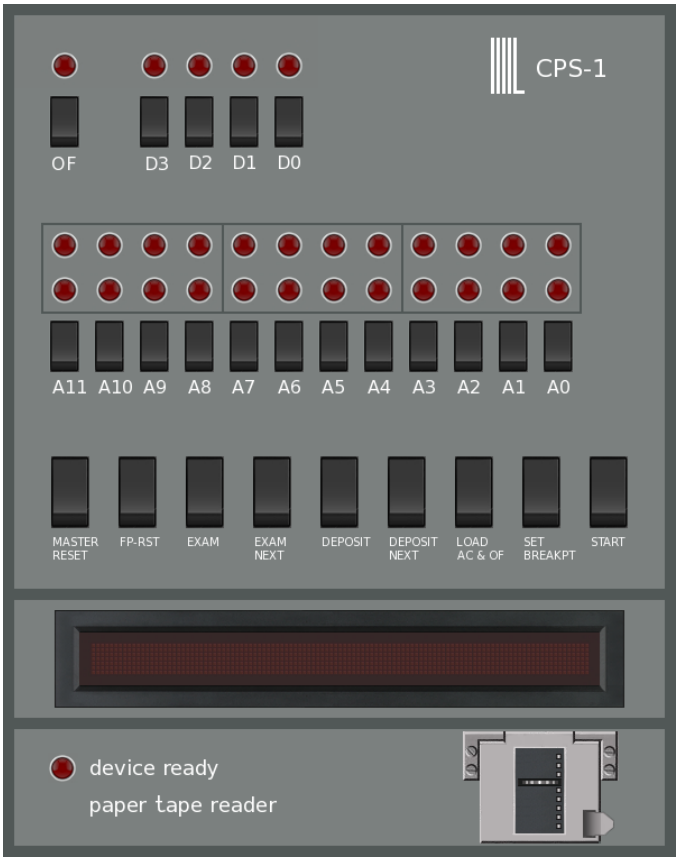


Figure 3. The CPS-1 Emulator's front panel.

2.1 Front panel as an I/O device

John Heckman, a former MIL's employee and a member of the CPS-1 design group, indicated that the functionality of the CPS-1's front panel was implemented in software and was modeled after similar techniques implemented for minicomputers of that era (specifically, minicomputers manufactured by Data General Corp. such as the Nova computer introduced in 1969), [6]. In other words, the front panel was an I/O device. The software approach to the emulation of the front panel was also considered in MIL's document *Techniques for the CPS/1* [3].

CPS-1 documentation in the YUCoM's MIL collection does not provide detailed information concerning the design and functionality of the front panel. However, one of the documents contains a drawing of a front panel from which most functionality can be deduced. The CPS-1 Emulator's front panel (depicted in Fig. 3) is implemented using that information as well as the fact that some of that functionality was adopted from the minicomputer designs. Let us note that the CPS-1's front panel is just an user-defined I/O device and, hence, much of

its functionality depends on software written to interact with it.

The CPS-1 Emulator's front panel contains 26 switches and 31 LEDs. All the switches can be actuated by left clicking on their images. The meanings of all these devices are defined below.

- D3,..., D0 – these data switches set the four bits of the accumulator;
- OF - this switch sets the overflow buffer;
- A0,..., A11 – these switches set a 12-bit address.

The remaining nine function switches are defined as follows:

- MASTER RESET - sets PC to address 1024; sets all the LEDs to correspond to the positions of switches OF, D0,..., D3, and A0,..., A11.
- FP-RST - resets the front panel; Heckman recollected that "FP-RST was a reset that forced a different reset address to enter the console program." In the CPS-1 Emulator, FP-RST sets PC to 1028 which, in the supplied boot3.rom program, starts the paper tape loader.
- EXAM - displays the contents of memory at the address set by A0,..., A11 (as indicated by the bottom row of 12 LEDs); the contents is displayed using LEDs above D0,...,D3.
- EXAM NEXT - increases the address displayed by the top row of 12 LEDs by 1 and displays memory contents at this new address using LEDs above D0,..., D3.
- DEPOSIT - saves data set by D0, ..., D4 in memory at the address specified by the switches A0,...,A11 and displayed by the bottom row of 12 LEDs;
- DEPOSIT NEXT - increases the address displayed by the top row of 12 LEDs by 1 and saves data set by D0, ..., D4 in memory at this new addressed;
- LOAD AC & OF - loads the accumulator and OF registers with data set by switches D0,...,D3 and OF;
- SET BREAKPT - sets breakpoint – currently not implemented;
- START - sets PC to the address indicated by the switches A0,...,A11 and displayed by the bottom row of 12 LEDs.

Note that the MASTER RESET switch can only reset PC to 1024. To execute an instruction stored at a different address, one has to set such an address using switches A0,..., A11 and, then toggle the START switch. Alternatively, one can use the FP-RST to start the emulator's execution at the reset address.

The front panel of the CPS-1 Emulator is an I/O device that communicates with CPU using the following six uniquely assigned addresses to store:

- 4040 - panel's status;
- 4041 - the most significant nibble of an address;
- 4042 - the second nibble of an address;
- 4043 - the least significant nibble of an address;
- 4044 - 1 nibble of data corresponding to switches D0, ..., D3;
- 4045 - 1 nibble of data corresponding to OF.

It is plausible that in the physical implementation of the panel, the bottom row of 12 address LEDs always displayed the address stored in mem[4041], mem[4042], and mem[4043]. Furthermore, the 4 LEDs above D0, ..., D3 always displayed the contents of mem[4044]. Finally, the LED above the OF switch always displayed the contents of mem[4045].

To get information from the panel, the front panel software sends the "4040 request" first. The panel responds by writing a unique integer to mem[4040] depending on the switch that has been currently actuated. This integer to switches assignment is shown below:

- 0 – no switch pressed;
- 8 – MASTER RESET pressed;
- 3, 4, 5, 6, 7 – unassigned;
- 9 – FP-RST is pressed;
- 10 – EXAM is pressed;
- 11 – EXAM NEXT is pressed;
- 12 – DEPOSIT is pressed;
- 13 – DEPOSIT NEXT is pressed;
- 14 – LOAD AC & OF is pressed;
- 2 – SET BREAKPT is pressed (currently not implemented);
- 15 – START is pressed.

Depending on the response, the CPS-1 may request a nibble of data (via memory locations 4044 and 4045 when, for instance, LOAD AC & OF is pressed) or an address (via memory locations 4041 to 4043 when, for instance, EXAM is pressed). It can write a nibble of data to mem[4044] (when EXAM or EXAM NEXT is pressed). The following example explains the way the START switch is implemented.

Example. Suppose that the CPS-1 front panel software sends the panel status request (LDI 4040) and that START switch is pressed. The front panel responds by placing 15 on the data bus which triggers the 4041 request (LDI 4041) to which the panel responds by writing the address currently set by the switches A0,...,A11 in memory at locations 4041, 4042, 4043. Finally, the emulator collects the address using the LDID instruction and does something with it (e.g. stores it in the address register 0). Here is a sample code:

```
LOOP: LDI 4040        ; request panel status
      LAD             ; store status in AC
      JZ LOOP         ; try again, if status=0
      LDI 4041        ; send address request
      LDID            ; store address in DP
      SDR 0           ; store address in address register 0
```

All other function switches (with the exception of SET BREAKPT) are implemented in a similar way.

The CPS-1 Emulator front panel features 26 LEDs to visually support the functionality of the panel switches. The bottom row of 12 LEDs always displays the positions of address switches A0,..., A11. The top row of 12 LEDs has a different purpose. In the emulator, it is used to display the address at which data is to be stored or examined when either DEPOSIT NEXT or EXAM NEXT switches are pressed during the execution of the front panel software. Initially, these LEDs display the same address as the bottom row of LEDs. However, when either EXAM NEXT or DEPOSIT NEXT is pressed, the LEDs display the next memory address whose contents is to be examined or rewritten. Pressing the MASTER RESET switch will reset all these LEDs to correspond to the settings of the corresponding address switches. The top raw of LEDs could be assigned other functionality as well. According to John Heckman, "one bank of 12 leds was intended to also display contents of the 12 bit registers... The bottom row would probably display the current program counter and breakpoint address." [6].

The top five LEDs are used to support the functionality of LOAD AC&OF, EXAM, and EXAM NEXT switches. The LEDs above the OF, D0,..., D3 switches visualize either the bits destined for OF and AC registers or the contents of a specific memory location. When EXAM or EXAM NEXT switch is pressed, the LEDs above D0,..., D3 display the nibble stored at the address indicated by the top row of 12 LEDs above the switches A0,...,A11.

Anytime a switch OF, D0,...,D3 changes its position, the LED corresponding to that switch displays the switch's new position. The MASTER RESET switch resets all the LEDs to correctly display the positions of the OF, D0,...,D3 switches.

2.2 Interfacing a paper tape reader

Paper tape reader/punch is interfaced to a CPS-1 as if it were an 8-bit device. The MIL document *CPS/1 Software Notes* specifies that the paper tape interface uses ASCII format (with or without parity) and that only 7 bits (out of 8) will be internally stored or tested. This implies that, in general, a byte of data YZ (in hex) can only be transmitted as two consecutive nibbles Y followed by Z.

According to the CPS-1 input/output conventions for operating with 8-bit data, upon receiving a byte YZ (in hex) of data, 4 least significant bits, Z, are stored in the accumulator and the most significant, Y, in the data register D0. In addition, the address register A0 may, optionally, contain the address of an I/O device.

In view of this information, the CPS-1 emulator implements a paper tape reader (PTR) which transmits binary data using 7-hole paper tape (no parity). Four least significant bits are used to encode data (in hex) while 3 most significant bits may be used as 'control bits' to transmit control information such as the end of tape indicator, the beginning of character data, etc. An 8-hole paper PTR could be emulated in exactly the same way.

In the CPS-1 Emulator, PTR is assigned the following four addresses:

- 4032 – PTR's status request,
- 4033 – data ready request,
- 4034 – 4 bits of data (most significant)
- 4035 – 4 bits of data (least significant)

Before we can get any data from PTR, the reader has to be initialized. This is accomplished by sending the request address 4032

```
    LDI 4032
```

Upon receiving it, PTR places 0 on the data bus, goes through its initialization loop and, when done, places "done" message–a non-zero value–on the data bus. Since the initialization of a physical PTR may take some time, the CPS-1 PTR loader software initializes the device by executing this code:

```
loop: LDI 4032      ; paper tape reader's status request
      LAD           ; get status from memory[4032]
      JZ loop       ; jump to loop if AC=0
```

At this point, the device address (4032) can be placed in A0

```
    SDR 0       ; load A0 with DP=4032
```

Next, to get a single byte from PTR, the CPS-1 inquiries PTR about its 'data ready' status. PTR responds either with 0–not ready–or 1 – ready. The CPS-1 executes the following loop until a byte is ready for transmission:

```
loop1: LDI 4033     ; data ready status request
       LAD          ; get data ready status from memory[4033]
       JZ loop1     ; jump to loop1 if AC=0
```

Finally, since a byte of data is ready for transmission, the CPS-1 emulator gets it into the accumulator and D0 by executing

```
    LDI 4034
    LAD         ; load AC with memory[4034]
    SAR 0       ; load D0 with memory[4034]
    LDI 4035
    LAD         ; load AC with memory[4035]
```

2.3 Interfacing Burroughs Self-Scan

The Self-Scan displays were introduced by Burroughs in the early 1970s. The MCM/70 microcomputer designed by a Canadian company Micro Computer Machines and announced in 1973 used such a device for its built-in display. Hence, the CPS-1 could, in principle, operate with such a display as well. (In fact, one of an early drawings made by a member of the MIL CPS-1 group in April 1972, depicts a CPS-1 system as a calculator using a LED display and a 16-key numeric keypad.)

The CPS-1 Emulator includes the emulation of the Burroughs SSD 1000-0030 Self-Scan display. The display consists of 111 columns of display cells, each column consisting of 7 cells. It is assumed that the SelfScan display is controlled by an interface card connected to the 12-bit address bus and 4-bit data bus of the CPS-1. Two display modes are emulated: the text and graphics mode. We describe both modes separately.

Text Mode: In the text mode (originally offered for SSD 1000-0030), the device displays 16 patterns (characters) in a single-row; each character occupying a 7-by-5 matrix of display cells, with two columns of space between characters. The display has a repertoire of 64 characters, each character being defined by a six-bit code. The following pattern codes are adopted from [4]:

A	0	@	36	;	44	$	52	>	60
B	1	∼	37	?	45	”	53	[	61
..	..	{	38	/	46	+	54	]	62
Z	25	}	39	%	47	-	55	empty	63
1	26	(	40	:	48	!	56		
..	..	)	41	’	49	#	57		
9	34	.	42	,	50	&	58		
0	35	=	43	*	51	<	59		

(For more information on the SSD 1000-0030 consult [4]).

In the text mode, the interface card of SelfScan can store 16 patterns and their locations in the so-called pattern memory. During the Self-Scan's refresh cycle, this pattern memory area is scanned, position after position, and the corresponding patterns presented to the Self-Scan for display. The CPS-1 Emulator can update the pattern memory of the Self-Scan's control card using standard CPS-1 interfacing techniques and the following addresses:

- 4050: to send the 1st digit (tens) of a pattern code;
- 4051: to send the 2nd digit (units) of the code;
- 4052: to send 1 nibble representing pattern's position on Self-Scan.

As soon as the address 4052 is placed on the address bus, the Self-Scan's control card computes the pattern number using data stored at memory locations 4050 and 4051:

$$\text{pattern=memory[4050]}\times 10+ \text{ memory[4051]}$$

Then, it updates its pattern memory at the position stored at the address 4052. The following sequence of instructions will display the message "CPS" on the Self-Scan, starting at position 0:

```
LAI 0
LDI 4050
SAD
LAI 2
LDI 4051
SAD                    ; pattern 02 (C) transmitted
LAI 0
LDI 4052
SAD                   ; C displayed
LAI 1
LDI 4050
SAD
LAI 5
LDI 4051
SAD                    ; pattern 15 (P) transmitted
LAI 1
LDI 4052
SAD                    ; P displayed
LAI 1
LDI 4050
SAD
LAI 8
LDI 4051
SAD                    ; pattern 18 (S) transmitted
LAI 2
LDI 4052
SAD                    ; S displayed
```

Graphics Mode: In the graphics mode, (model SSD 1000-0039), each of 111 columns of display cells can be actuated individually. The interface card of Self-Scan accepts two nibbles of information defining a column and two nibbles of x-coordinate ($x < 112$) and actuates the information. It does that using standard CPS-1 interfacing techniques and the following addresses:

- 4055: to send the most significant nibble of x-coordinate;
- 4056: to send the least significant nibble of x-coordinate;

– 4057: to send the nibble defining cells 4, 5, and 6 of a column (1 represents "cell on" while 0 represents "cell off");
– 4058: to send the nibble defining cells 0, 1, 2, and 3 of a column.

Here is an example of displaying a column of 7 cells in which the top and the bottom cells are on while the remaining cells are off. The column is to be displayed at x=97.

```
LAI 6                ; AC=most significant nibble of 97 in binary
LDI 4055
SAD                  ; most significant nibble of 97 transmitted
LAI 1                ; AC=least significant nibble of 97 in binary
LDI 4056
SAD                  ; least significant nibble of 97 transmitted
LAI 4                ; turn the 6th cell on
LDI 4057
SAD                  ; information re cells 4, 5, and 6 transmitted
LAI 1                ; turn the 0th cell on
LDI 4058
SAD                  ; information re cells 0,...,3 transmitted
```

In both modes, SelfScan can be cleared (blanked) by storing 8 at address 4050 as shown here:

```
LAI 8
LDI 4050
SAD                  ; blank screen
```

2.4 Interfacing numeric keypad

The CPS-1 Emulator is interfaced with a 16-key numeric keypad (of a standard keyboard) that can send the following characters:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, /, *, -, +, ., ⟨ENTER⟩

using the following codes:

character	code	character	code
/	11	7	7
*	12	6	6
-	13	5	5
+	14	4	4
⟨ENTER⟩	15	3	3
.	10	0	2
9	9	1	1
8	8	0	0

To interface the keypad with the CPS-1 emulator, the emulator is using the following addresses:

4060 – data ready request,
4061 – data request.

When the CPS-1 puts the address 4060 on the bus, the keypad responds by consulting its data-ready port and places one of the following values on the data bus:

0 - key has not been pressed;
1 - key has been pressed and character is ready.

When the CPS-1 puts the address 4061 on the bus, the keypad responds by consulting its data port and places its contents (character code) on the data bus.

EXAMPLE: The following list of instructions is a fragment of a program that waits for a keypad input and executes subroutine **process0** when the "0" key is pressed (**process0** could, for instance, display "0" on the Self-Scan).

```
        LDI 4060
loop2: LAD
        JZ loop2                ; wait for char ready

; char ready, so get it.
        LDI 4061
        LAD

; if char is 0, then process it by jumping to process0
        JZ process0
```

3 Operating the CPS-1 Emulator

The CPS-1 Emulator should be used with a ROM program that implements at least the front panel's functionality (a bootstrap program). The CPS-1 Emulator is supplied with such a program named boot4.rom (or similar). The supplied boot4.rom also contains a paper tape loader. The loader allows a user to load data and applications programs stored on paper tapes, i.e. in paper tape files prepared by the programmer. The format of paper tape files is discussed below.

3.1 Representation of ROMs and paper tapes in the CPS-1 Emulator

One of the MIL documents specifies that PTR transmits ASCII characters (with or without parity) and that the end of paper tape is indicated by the sequence of three characters: $ (CR) (LF) (i.e., $ followed by carriage return (CR), and line feed (LF)).

In the current implementation of the emulator both ROM files and paper tape files are just sequential files of positive integers ≤ 127, each integer representing either a nibble of binary data (integers ≤ 15) or a control signal. An application program will interpret these integers as code or data. For instance, the 4 nibble instruction

LDI 4060

is represented by 4 nibbles 1101 1111 1101 1100 which are stored in a paper tape file as four consecutive integers

13 15 13 12

The emulator, in turn, interprets these numbers as 8-bit bytes

00001101 00001111 00001101 00001100

On the other hand, the end of the tape, represented by $ (CR) (LF) will be transmitted as three bytes 00100010 00001101 000010010, the first of which contains the 'control bits' 0010 which are sufficient (in the current implementation of PTR) to terminate the paper tape reading process by the paper tape loading program.

Data can be represented in a variety of ways. One of the MIL documents recommends the following representation for binary and decimal integers and for characters:

binary integers: 16 consecutive nibbles;

decimal integers: sixteen digits of 4 bit BCD format;

characters: eight bit USASCII.

To load program and data files into specific memory locations, the CPS-1 Emulator expects that the first three nibbles of a program (or a data set) file constitute the loading address (MS nibble first, see Fig. 2). Furthermore, the emulator expects that each paper tape is terminated with $ (as discussed above, (CR) (LF) are optional). Tape organization is shown on Fig. 4.

Figure 4. Paper tape format.

If the paper tape file is produced using the CPS-1 assembler (supplied with the emulator), then the tape ending nibbles are placed by the assembler and there is no need to add them explicitly when creating a program.

3.2 Executing a CPS-1 application

Both ROM and paper tape files should be placed in the ROMs sub-directory. All these files are accessible via "right click" menus as shown in Figure 5.

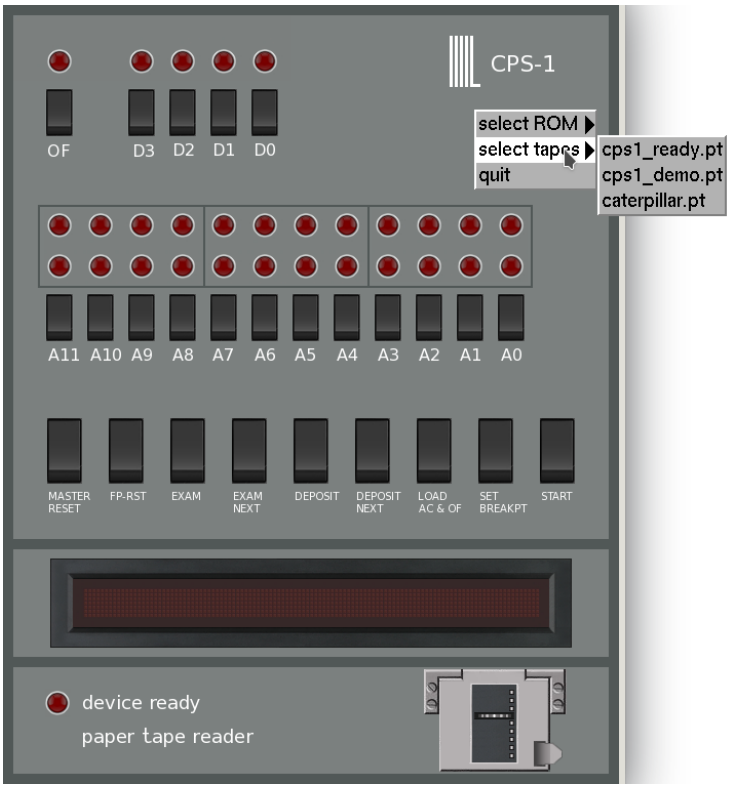


Figure 5. ROM and paper tapes selection menus.

To explain how to operate the CPS-1, let us assume that, in addition to the bootstrap program, the ROM directory contains a program called cps1_demo.pt which, when executed, will display the message "CPS-1 READY:" on the Self-Scan. To execute cps1_demo.pt, the following steps should be taken:

Step 1. Select the ROM program boot4.rom from the ROM menu "select ROM" (see Fig. 5). When the selection is completed, the CPS-1 emulator begins the execution of the front panel loop.

Step 2. Select the cps1_demo.pt tape from tapes menu "select tape" as shown in Fig. 5. Note that when the selection is completed, cps1_demo.pt is not yet loaded into the computer's memory – it's just selected for reading.

Step 3. Set the address switches A11,..., A0 to the paper tape loader address 1028 (decimal) (or 0100 0000 0100 (binary)) as shown in Fig. 6

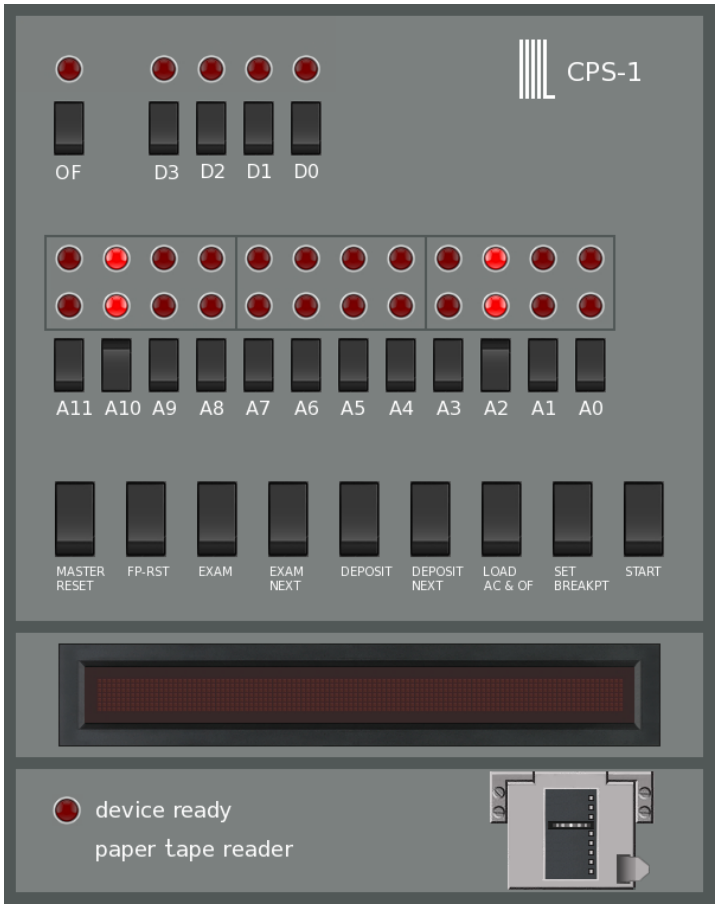


Figure 6. Setting the address 0100 0000 0100.

Step 4. Click on START switch; the emulator initializes PTR and, then, loads the contents of the paper tape cps1_demo.tp to RAM starting at the address indicated by the first three nibbles on the tape (in program cps1_demo.pt, this address is 2048 (decimal)).

Alternatively, Steps 3 and 4 can be replaced with just a single click on the FP-RST switch which sets PC to 1028 and begins the execution of the PTR loader. After the successful loading of cps1_demo.pt (indicated by PTR stopping its operations) the paper tape loader program jumps back to 1024 and resumes the execution of the front panel loop.

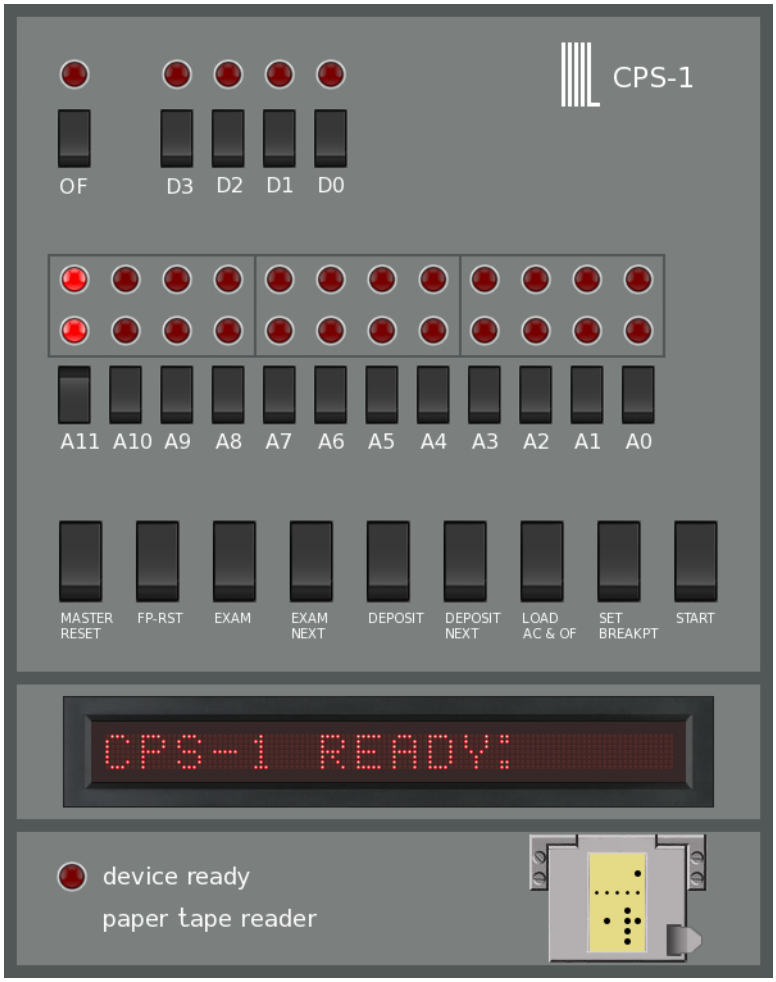


Figure 7. The execution of the cps1_demo.pt program results in "CPS-1 READY:" message displayed on the Self-Scan.

Step 5. Set the address to 2048 (the loading address of cps1_demo.pt) and click on START switch to execute the program (see Fig. 7).

If more than one tape is needed (e.g. one containing data and one an application program), then Steps 2–4 have to be repeated for each tape.

It is advisable that every application program terminates with either the jump to 1024 (to allow other apps to be loaded) or an infinite loop

```
loop: JMP loop
```

3.3 The debug mode

The emulator can be executed in the debug mode. At any time of the emulator's execution, pressing the F1 key will cause the emulator to enter the single step mode during which the contents of all the registers will be displayed, as shown on Fig. 8. Continue to press F1 for single-stepping. To end the debug mode, press F2.

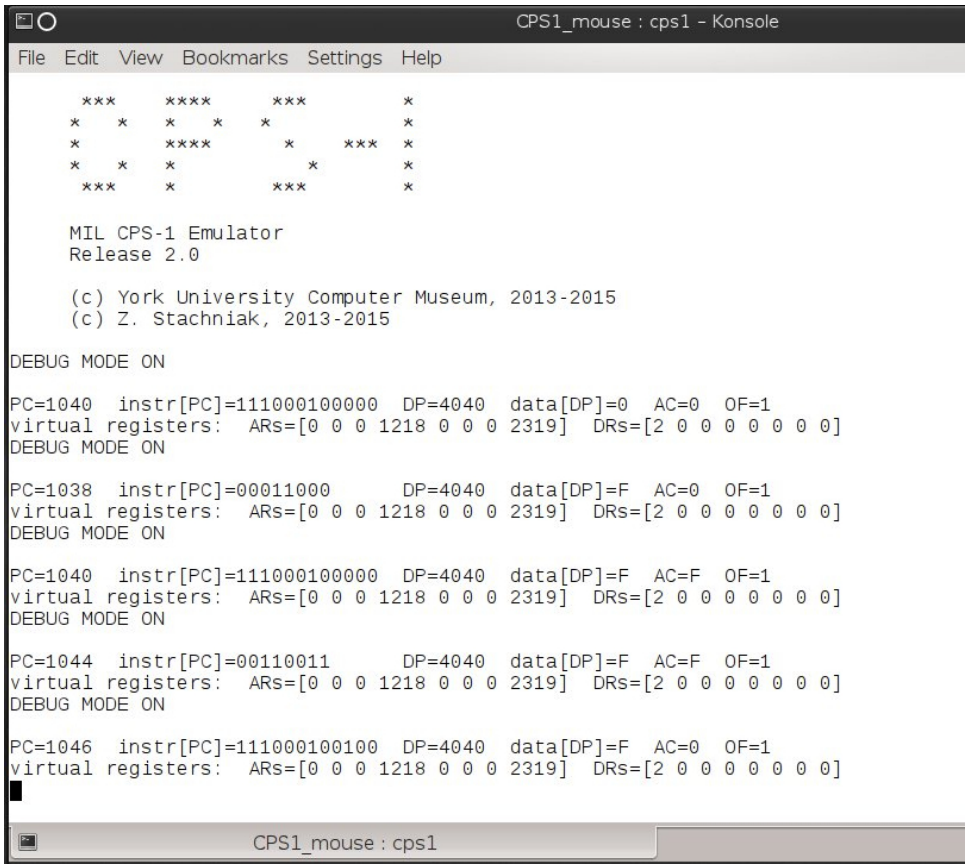


Figure 8. The emulator in the debug mode.

4 Installing the CPS-1 Emulator

The tarball cps.tar.gz contains all the necessary directories and files. The emulator is written in C and requires OpenGL and Glut (or FreeGlut) libraries. Before compiling the emulator's code make sure that these libraries are available; they are typically located in /usr/include. Executing:

```
gcc cps1.c -lGL -lglut -o cps1
```

should create executable file cps1.

5 CPS-1 Assembler

The tarball cps.tar.gz also contains a CPS-1 assembler–cps1asm.c–for the MIL MF7114 microprocessor. The program is written in C. The assembler translates a CPS-1 program into an MF7114 binary code ready for loading into and execution by the CPS-1 emulator.

5.1 CPS-1 Programs

A syntactically correct CPS-1 program consists of a loading address, a sequence of MF714 instructions, comments, labels, and data. These components are discussed below.

Loading address: A program is loaded into a continuous block of memory. Therefore it should contain information about such a block in the form of the address of the first nibble of the block. Such an address (an integer) is placed at the beginning of the program and must be proceeded by at least one blank. It can also be proceeded by any number of comment lines as shown in this example:

```
;-----------------------------------
; sample program, July 2019
;-----------------------------------
;
; loading address
        2048
```

where lines beginning with ";" are comments which are defined next.

A ROM code should always be loaded at memory address 1024. A programmer does not need to remember this address – writing 0 as the loading address instructs the assembler to set the loading address to 1024.

Comments: A comment is a string of characters on a program line that begins with ";" and extends up until the end of the line. In other words ";" and every character on the line to the right of ";" is treated as a comment character

and will be ignored by the assembler. So, the code:

```
;-------------------------------------
; sample program
;-------------------------------------
;
        2048        ; loading address
        LAM 13      ; 13 is sufficient
        LAD         ; don't delete this line
```

is equivalent to:

```
        2048
        LAM 13
        LAD
```

MF714 instructions: are written as mnemonics (as illustrated in the above examples). Each instruction must be proceeded by at least one blank.

Labels: A program may contain up to 1000 distinct labels which define memory addresses. Each label is a string of up to 9 digits and capital letters and must begin with a letter. A label has no leading blanks and must end with ":". A label may appear by itself on a line, as shown in this code:

```
LABEL1:
          LAM 13
```

or may be followed by an instruction or data, as shown in this code:

```
LABEL1:   LAM 13
```

Data statements: A program may include integer data with values between 0 and 15. This is achieved using the assembler code of the form:

<LABEL> DEFN <INT LIST>

where:

<LABEL> is a program label (as defined above),
DEFN is the "define nibble" assembler directive,
<LIST> is a comma separated list of integers (data).

Here is an example of including 8 nibbles of data:

```
SCORE:
          DEFN 7, 0, 4, 8, 0, 0, 0, 0
```

To store integer data with values greater than 15, one can use multiple nibbles. For example, one can store the hex value 0xACE1 in the following way:

```
DEFN 10,12,14,1          ; seed value 0xACE1, msn first and lsn last
```

5.2 No Arithmetic (yet)

The cps1asm.c code is a rudimentary assembler with plenty of room for improvement. In particular, the assembler does not perform any evaluation of arithmetic expressions. So,

```
LDI 1024+48
```

is not a legal instruction.

5.3 How to Use the Assembler

Compile the assembler with

gcc cps1asm.c -o cps1 cps1asm

The MF7114 code to be assembled has to be in the same directory as the assembler. Execute the assembler cps1asm code and follow the instructions.

References

[1] Z. Stachniak, The MIL MF7114 Microprocessor, *IEEE Annals of the History of Computing*, October-December 2010 (vol. 32 no. 4) pp. 48-59.

[2] *How To Use The CPS/1 Micro-Computer System*, Bulletin 50001, Microsystems International Ltd., 1972.

[3] *Techniques for the CPS/1*, MIL preliminary document, no author and no date specified (York University Computer Museum has a copy of this document).

[4] *Burroughs Specifying Guide: Electronic components and systems*, Bulletin 1061K, Burroughs, 1970.

[5] L/R. Schweizer, *CPS/1 Concepts and Facilities*. MIL Report, May 1972.

[6] Author's correspondence with John Heckman, 2011.