

PERSEUS-8 homemade 6502 computer

<https://hackaday.io/project/179196-perseus-8-homemade-6502-computer>

The PERSEUS-8 is the enhanced version of the minimum configuration type the PERSEUS-7 6502 computer.

Mitsuru Yamada

Description

In 2020, I developed the PERSEUS-7[3], a minimum configuration version standalone computer with the 6502 CPU. On the PERSEUS-7, It can run software up to 1 kB, but I need more memory space to implement simplified floating point interpreters and so on. Therefore, I started to develop PERSEUS-8 as an extended version of the PERSEUS-7. The features of the PERSEUS-8 are as follows.

- (1) The DMA address setting, which was 10 bits in the PERSEUS-7, is now 16 bits for the entire the 6502 CPU space.
- (2) Implemented 16 kB of battery-backed RAM so that the software is always saved. 8 kB of it can be switched with PROM.
- (3) The single step control circuit uses the same circuit as the proven PERSEUS-7.
- (4) The enclosure has almost the same outline as the PERSEUS-3[4], but both sides of the main board are accessible.

Details

1. Concept

Since this computer uses a CPU that is about 45 years old, it cannot compete at all with today's computers in terms of performance. However, I wanted the PERSEUS-8 to have functions and appearance that would remind us of concepts that today's computers have lost. Figure 1 shows the appearance of PERSEUS-8.



Fig. 1 Appearance of PERSEUS-8.

2. Enclosure

The enclosure of the PERSEUS-8 is assembled cut out of aluminum materials. I used 2 mm thick aluminum sheets and 15 mm and 10 mm aluminum angles. Only the bottom plate is made of 1mm thick aluminum. M3 screws and nuts were used to fix each part. This method is quite tedious, but it allows me to design my own free will. The outside dimensions without handles and rubber legs are WHD: 300 x 88 x 310 mm. The front panel was sandpapered for a hairline finish, and the side and top panels were painted with gray lacquer spray. Figure 2 shows the enclosure components before assembly.

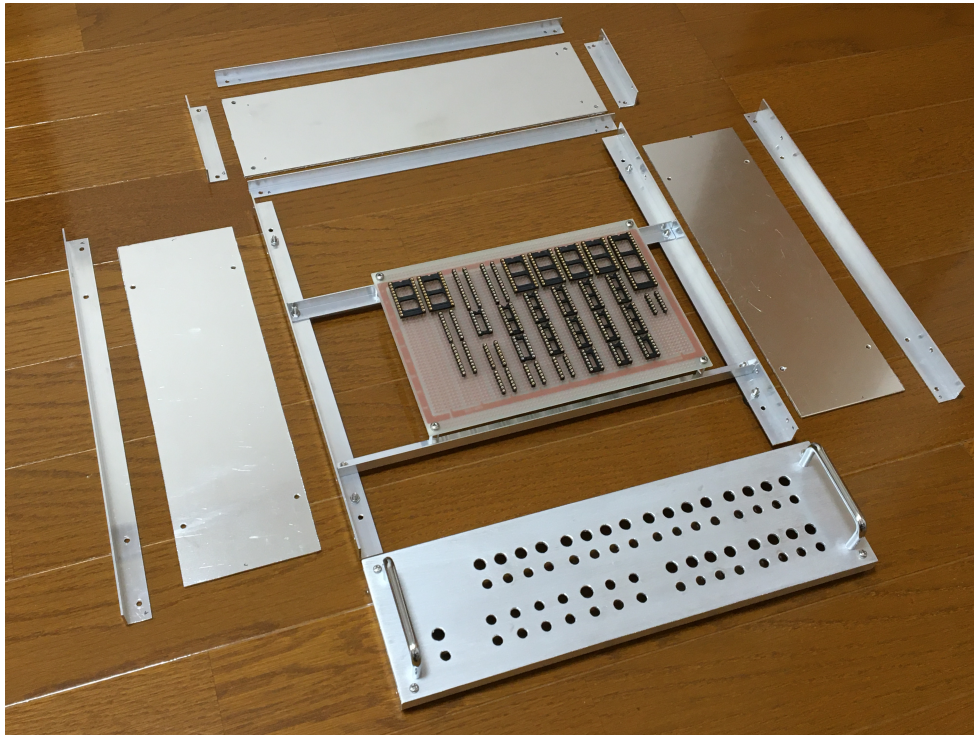


Fig. 2 Enclosure components of PERSEUS-8 before assembling.

The following video shows the assembly of the enclosure.



<https://www.youtube.com/watch?v=ihWKIHSgX4Q>

3. Hardware configuration

Figure 3 shows the hardware block diagram and Fig.4 shows the main board of PERSEUS-8. When the CPU is set to the halt state by the front panel switch, the memory address and data can be set with Direct Memory Access (DMA) by the toggle switches on the front panel for programming. In this state, single-step operation is

possible. The single-step control circuit performs these controls. The schematics PERSEUS-8_schematics_01.pdf[5] are on the attached files.

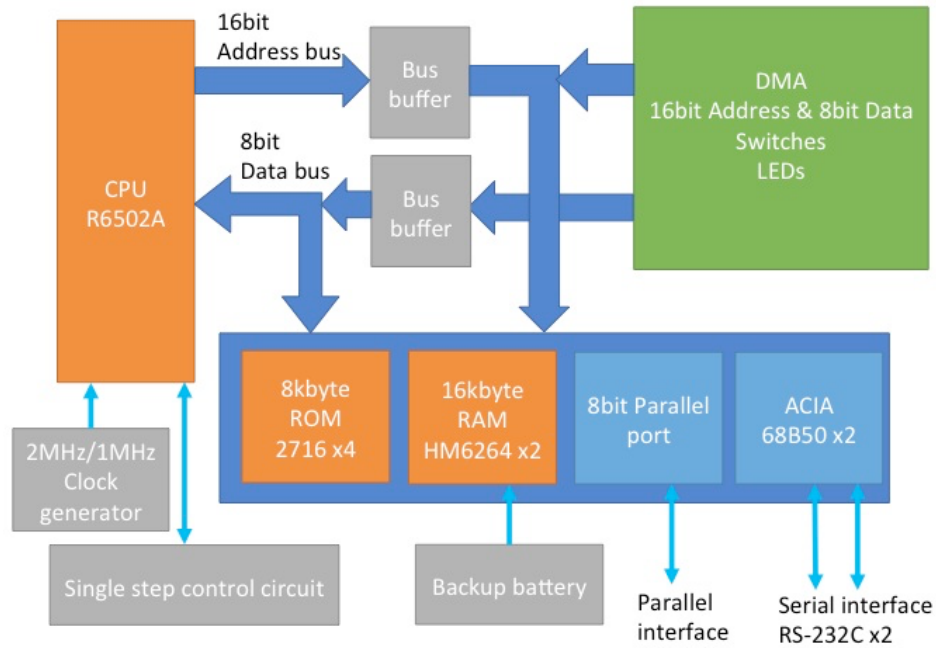


Fig.3 Block diagram of PERSEUS-8

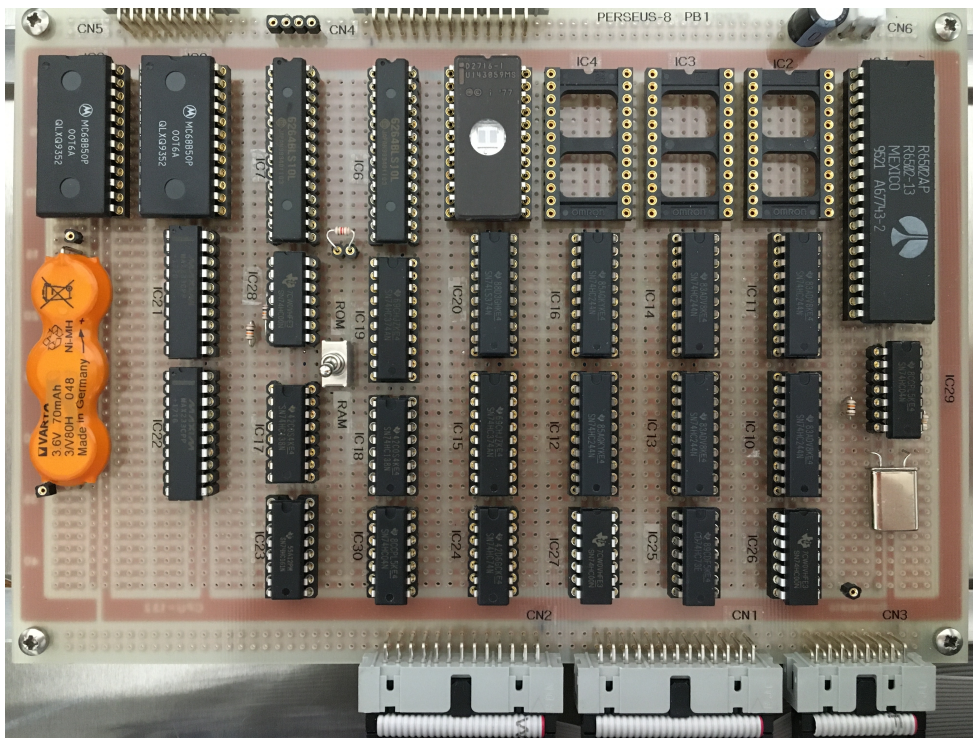


Fig. 4 Main board of PERSEUS-8.

4. CPU

The CPU is the R6502A, the same as the PERSEUS-7, but driven at the highest clock speed of 2 MHz.

5. Memory

The memory devices are two 64 k bit SRAMs (HM6264BLSP-10L). The total memory capacity is 16 k bytes. Of these, 8 k bytes were mapped to the zero page area side and 8 k bytes to the vector area side. The SRAMs were backed up by a 3.6 V, 70 mAh Ni-MH battery, the same method used in the PERSEUS-3. The supply current of the memory IC in the backup state is only 0.56 μ A. It has a proven track record of retaining memory contents stably for 10 years. The memory protection switch was installed on the front panel this time, as it was difficult to use on the main board of PERSEUS-3. For the ROM, four 16 kbit type 2716 PROMs that can be programmed with a homemade PROM programmer can be mounted. The ROMs can be selected between the RAM on the vector space side by a switch on the main board.

6. Panel switches and LEDs

Figure 5 shows the front panel switches and LEDs.

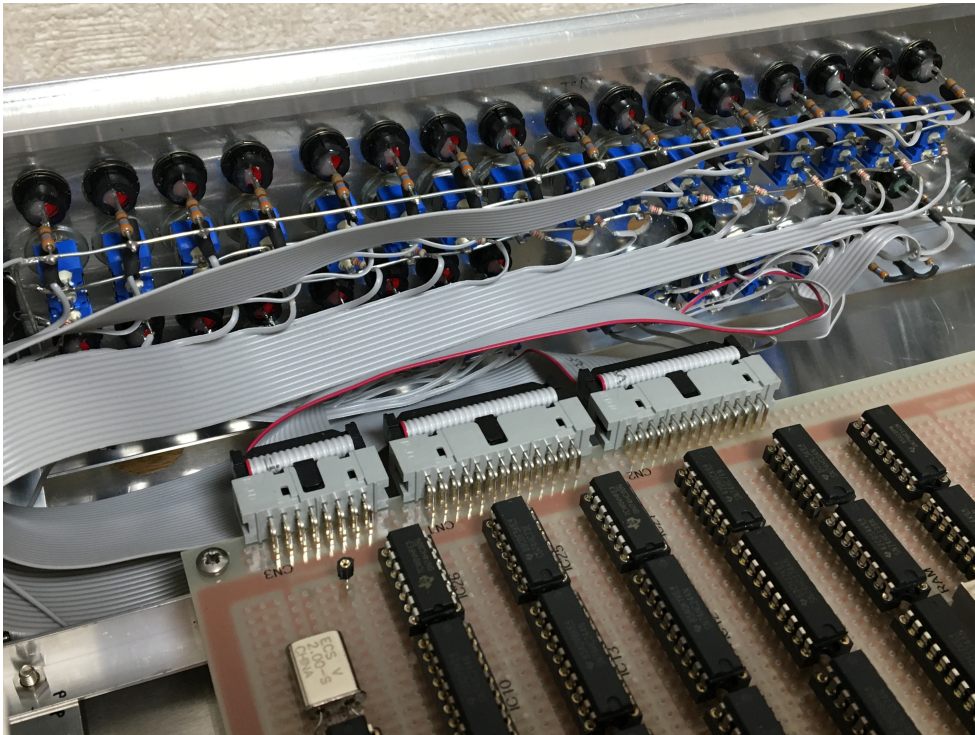


Fig. 5 Panel switches and LEDs of PERSEUS-8.

Sixteen address switches are connected to address bus through 3-state buffers (74HC244). Eight data switches are also connected to data bus through 3-state buffers (74HC244). The address LED displays the execution address of the CPU, not the address set by the DMA. This is almost the same circuit as PERSEUS-7 except for the address bus width. In the serial interface, there are LEDs on the transmit/receive signals so that you can directly recognize the communication status.

7. Single step execution

Figure 6 shows the single step control circuit and Fig.7 shows the corresponding timing chart. The single-step circuit was designed to be as simple as possible by referring to the references [1],[2]. In R6502, there is no halt signal, but since the CPU sends out a SYNC signal in the first machine cycle M1 of each instruction, this can be used to disable the RDY signal given to the CPU to create a single step execution. The following is a description of how it works.

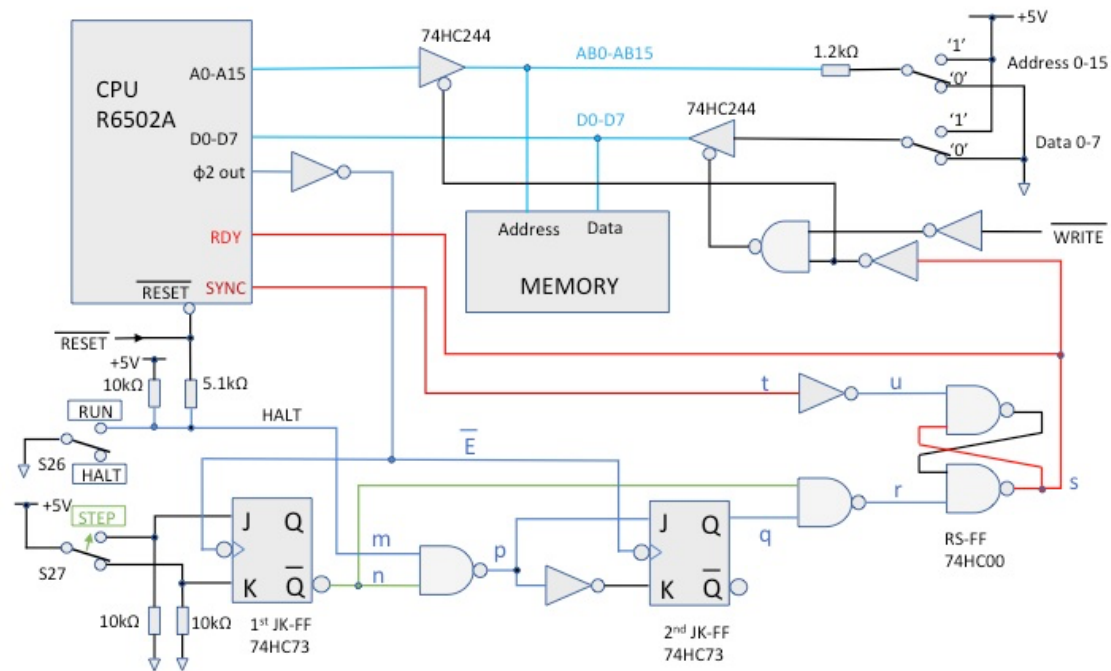


Fig.6 Single step control circuit of PERSEUS-8

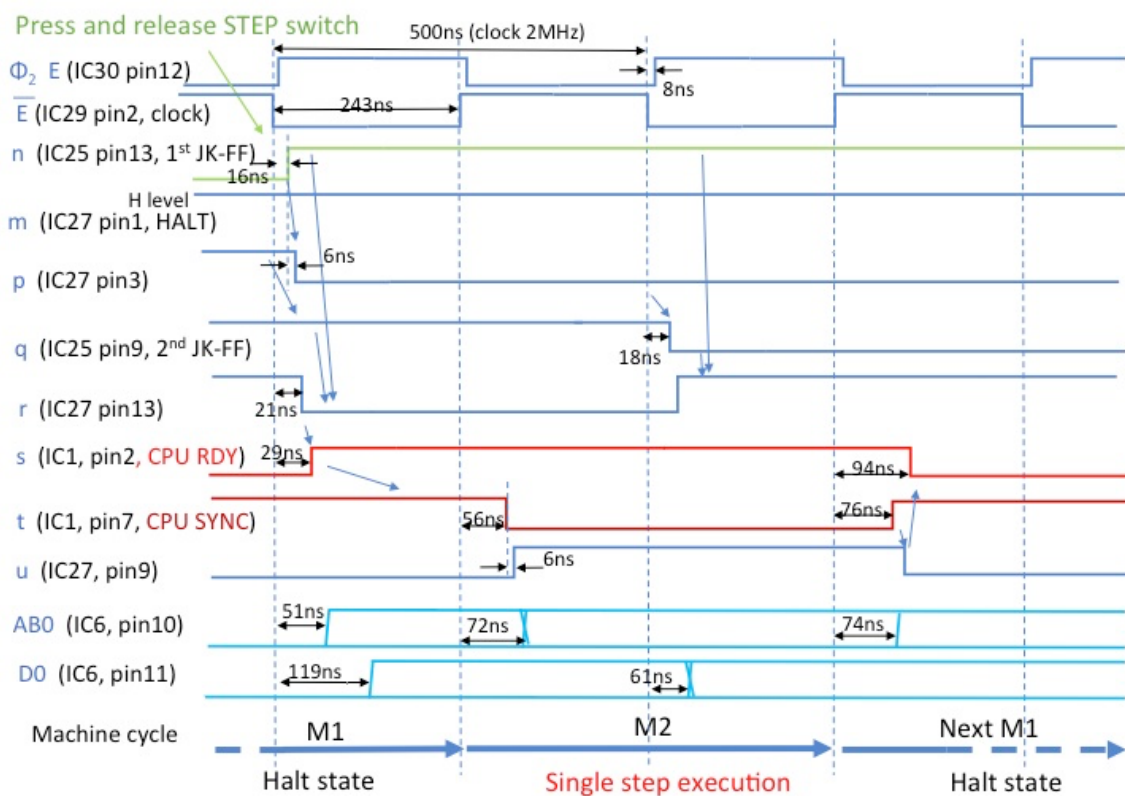


Fig.7 Single step timing of PERSEUS-8

Figure 7 is the case of a single-step operation of an instruction that takes two clock cycles to complete the shortest machine cycles. The timing values in Fig.7 are based on actual measurements using an oscilloscope.

When the RUN/HALT switch on the panel is set to the HALT side, the signal 'm' of Fig.6 and Fig.7 becomes logic 1. Now, when the STEP switch is pressed and released

one time, the first JK-flip-flop (JK-FF) of Fig.6 samples it on the falling edge of the clock, and the signal 'n' becomes logic 1. Any switch chatter is removed by this first JK-FF. Since the signal 'm' = 1, the signal 'n' passes through the next NAND gate and the signal 'p' becomes logic 0. The second JK-FF of Fig.6 is making the D-FF work, and generates a signal 'q' that delays the signal 'p' by one clock cycle. A negative logic pulse signal 'r' with a length of one clock cycle is generated from signals 'n' and 'q' by NAND operation. This signal 'r' sets the RS-FF of Fig.6 and transitions the signal 's' from 0 to 1. This signal 's' is given to the CPU as an RDY signal, the CPU resumes execution of the instruction.

After that the CPU completes its instruction execution as shown in machine cycle M2 of Fig.7. The CPU then sets the SYNC signal 't' to 1 in the first machine cycle M1 of the next instruction. By resetting the RS-FF with the signal 'u', which is the inversion of this signal 't', the signal 's' becomes RDY = 0 and the CPU stops in the M1 cycle of this instruction. This state continues until the STEP switch is pressed and released again. Following video shows an example of single step execution test. Please turn on subtitles to watch.



https://www.youtube.com/watch?v=6ES_qLPkIdw&t=4s

Here, RS-FF by NAND gates is normally prohibited to set both r and u to '0', but in this circuit, there is the transition from 'r' = 1, 'u' = 0, 's' = 0, to 'r' = 0, 'u' = 0, 's' = 1, to 'r' = 0, 'u' = 1, 's' = 1 in that order, and it does not in indefinite. The example in Fig. 7 is for the shortest machine cycle of two clocks, so if the instruction has a large number of machine cycles, the length of the RDY signal will be correspondingly longer.

When the RUN/HALT switch on the panel is set to the RUN side, the signal 'm' of Fig.6 and Fig.7 becomes logic 0. The SYNC signal changes 0 to 1 but there is the transition from 'r' = 0, 'u' = 1 's' = 1, to 'r' = 0, 'u' = 0, 's' = 1, to 'r' = 0, 'u' = 1, 's' = 1. Therefore, 's' = RDY = 1 is sustained and the CPU continues to execute instructions continuously.

The 5.1 kΩ resistor R4 between the CPU reset signal and signal 'm' in Fig. 6 is to pull down signal 'm' to release the halt state when the CPU is reset. The RDY control input is forced to a high level during this period. This is due to a countermeasure for a reset sequence defect that was identified during the development of my computer PERSEUS-7. My experimental results at that time had obtained that, with a certain probability, if the RDY control input is a low level during the reset period after power-on, resulting in incorrect execution addresses in subsequent single-step operations. This was not necessary in the case of R65C02.

8. Peripherals

The computer has two channels RS-232C serial interface and an 8bit parallel input/output interface. As a serial interface, Asynchronous Communication Interface Adapter (ACIA) MC68B50 is used. For the baud rate clock, I used 76.92 kHz, which is

1/13th of clock 1 MHz. 1/16th of this value is an approximation of 4800 bps for ACIA. The parallel interface is configured with standard logic 74LS374 and 74HC374. Figure 8 shows an example of the use of this parallel interface with a homemade PROM programmer[6] connected.

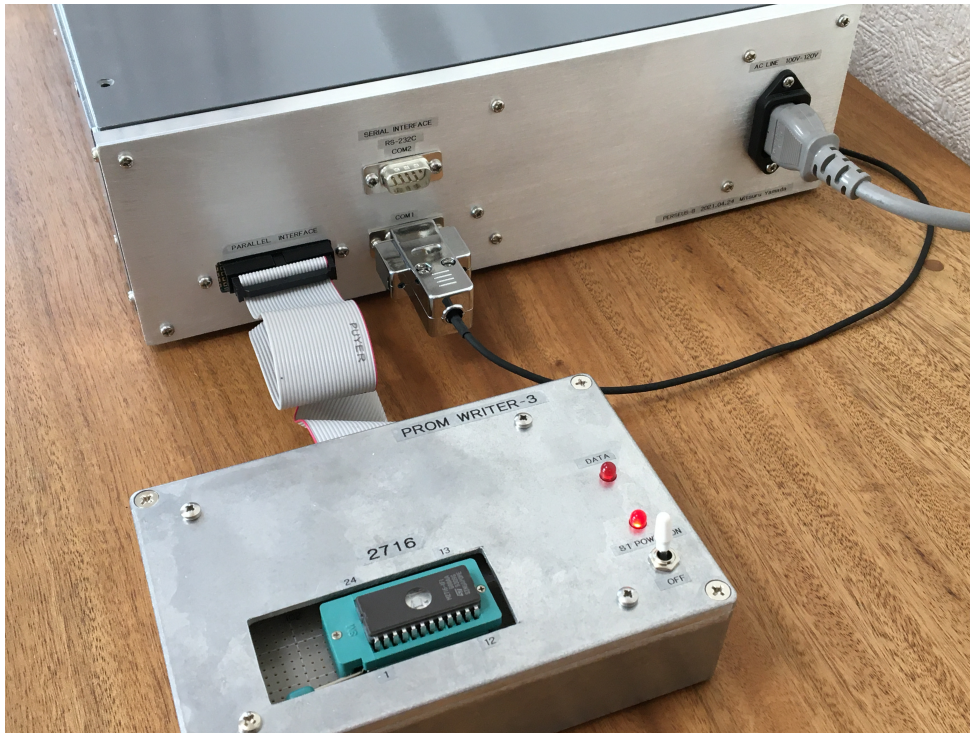


Fig. 8 Homemade PROM programmer connected to the parallel interface.

9. Build

The wiring on the main board was done with the same wire wrapping method as the PERSEUS-3 and PERSEUS-7 as shown in Fig.9.

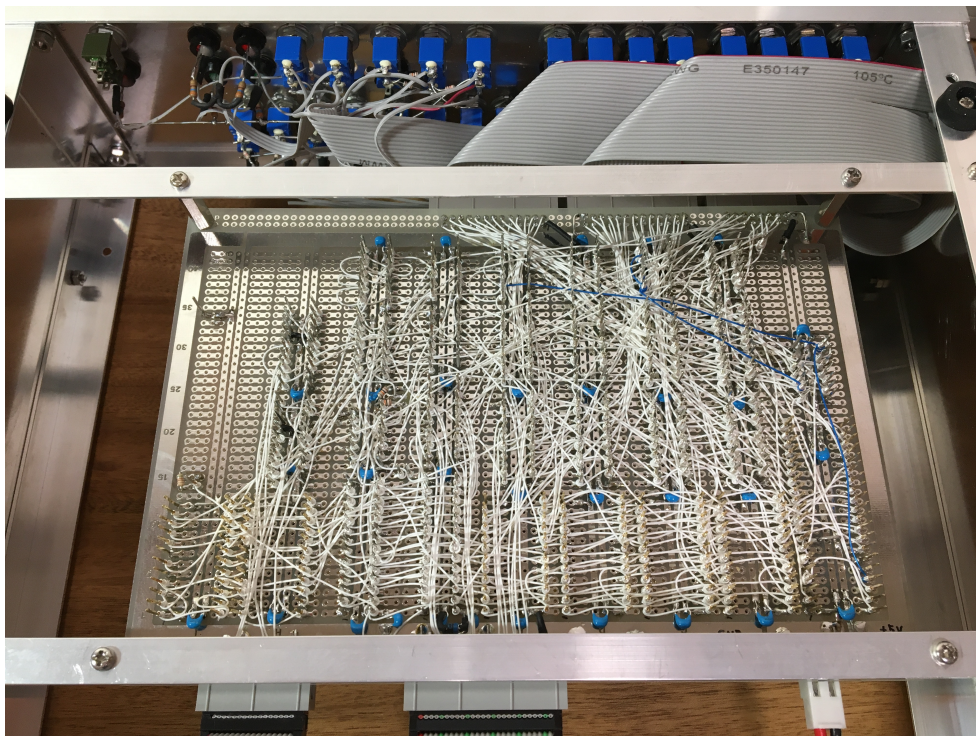


Fig. 9 Wiring side of the main board of PERSEUS-8.

The following video shows the wire wrapping process. Please turn on subtitles to watch.



<https://www.youtube.com/watch?v=OujuzAUD9pc&t=2s>

The back panel and the power supply section are shown in Fig.10. The back panel has an AC inlet, two RS-232C serial interfaces, and an 8-bit parallel interface connector. The power supply section is fixed to the back panel, and is a linear regulator using a transformer, a bridge rectifier, and a 3-terminal regulator IC (NJM7805FA) as shown schematic PERSEUS-8_Power_supply_schematic_01.jpg[7] on the attached files. The regulator IC is fixed to the back panel for heat dissipation. The power supply is capable of supplying 1.5 A at +5 V DC.

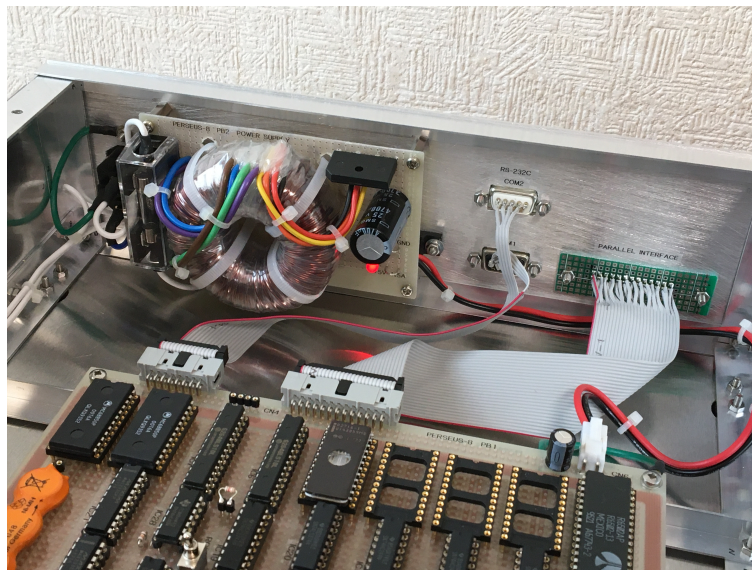


Fig. 10 The back panel and the power supply section of PERSEUS-8.

10. Operation verification

I used an oscilloscope to measure the timing of memory read-write and single-step operations, and examined whether there were any operations that were close to timing violations. The circuit was modified in about 10 places from the initial state before normal operation was achieved. The schematic is the one after the modifications. The program produced by the PERSEUS-7 was executed as simple operation verification. For example, after loading 1 k byte RPN calculator software including parallel port read/write from the ROM module[8] with a boot loader, it was able to run normally.

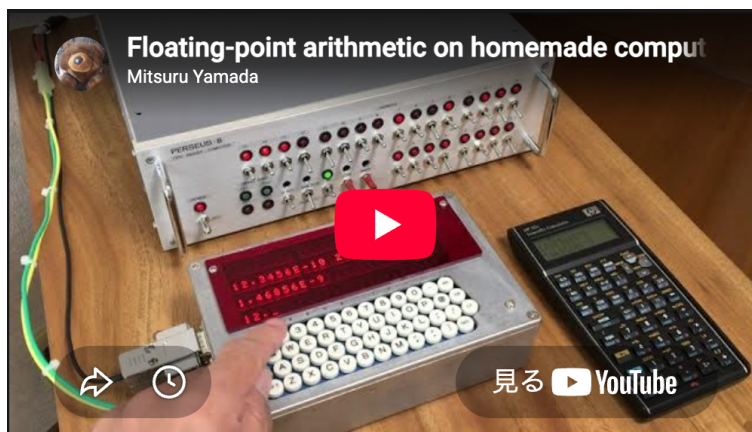
I am planning to develop a formal test program and implement it on the ROM as utility software. As an example of the test program, a program to execute a marching test for this computer is shown in the attachment Source_code_MEMORY_TEST-

1_1_0.pdf[9]. The marching test is one of the most common memory test methods. This program would be useful for self-diagnosis of normal operation.

11. Floating point interpreter for PERSEUS-8

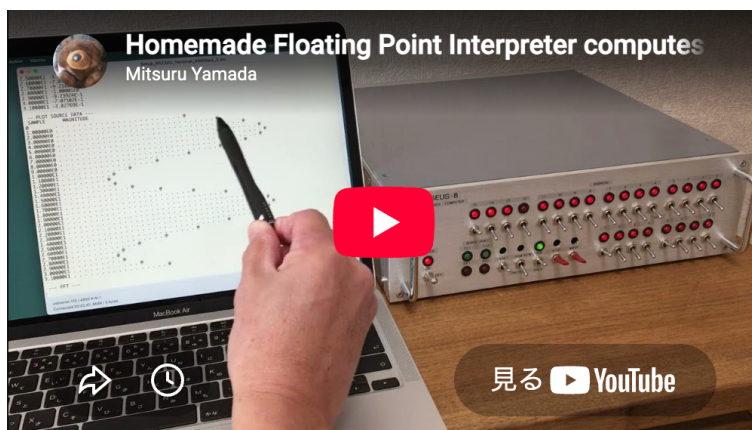
The hardware of the PERSEUS-8 has almost completed with the power supply and the back panel. Next, I challenged to make an interpreter system. I built an integer interpreter on my other computer PERSEUS-3 about 10 years ago. This time, in order to make this computer PERSEUS-8 practical for numerical calculations, I challenged myself to create a floating-point interpreter with elementary functions such as trigonometric and exponential functions. This interpreter is described in detail in another project Homemade Floating Point Interpreter for 6502[10].

As a first attempt, the following video shows the floating point arithmetic section being tested. This machine language program does not conform to the IEEE standard to simplify the study. The specification is 24-bit BCD for the mantissa part, 1-bit for the sign, and 7-bit BCD for the exponent part, and the numerical range is from $-9.99999\text{E-}39$ to $9.99999\text{E}39$. The code size is about 1.5 k bytes. The only development environment is the toggle switches and LEDs of this computer with hand-assembled code. Please turn on subtitles to watch this video.



<https://www.youtube.com/watch?v=XF88nxAeItk>

The following video shows an example of this floating-point interpreter program performing a Fast Fourier Transform (FFT). There is no narration in this video, please turn on subtitles to watch. The Fourier transform is a basic signal processing technique that is now widely used in many computers and portable devices for similar processing.



<https://www.youtube.com/watch?v=ZZ5HwPYRB9w&t=7s>

The interpreter is about 6.3 k bytes in size. Elementary functions such as sine, cosine, and exponential are implemented by machine codes. And it has program control instructions, pointer variables, array variables, and a line editor. I spent about three months hand coding and debugging with step operation by front panel switches for this interpreter.

12. PROMs

The interpreter is running on the computer's battery-backed RAMs, but it can also run on ROMs. The programming to the ROMs was done using the programmer presented in my other project Fully manual PROM programmer[11], which uses four D2716 type 2 kB EPROMs to store the entire interpreter in an 8 kB capacity. The entire programming process took 20 hours manually.

In March 2023, a PROM programmer shown in Fig. 8 was successful made that can be automatically programmed by the interpreter's user program process[12], so PROMs can be revised in this way from version V.1.5.0 onward. The following video shows how this PROM programmer is used to update the interpreter. There is no audio explanation, so please turn on the subtitles to watch.



<https://www.youtube.com/watch?v=3ZnFJ-Ky71o&t=7s>

Figure 11 shows how the programmed ROMs are mounted on the computer PERSEUS-8.

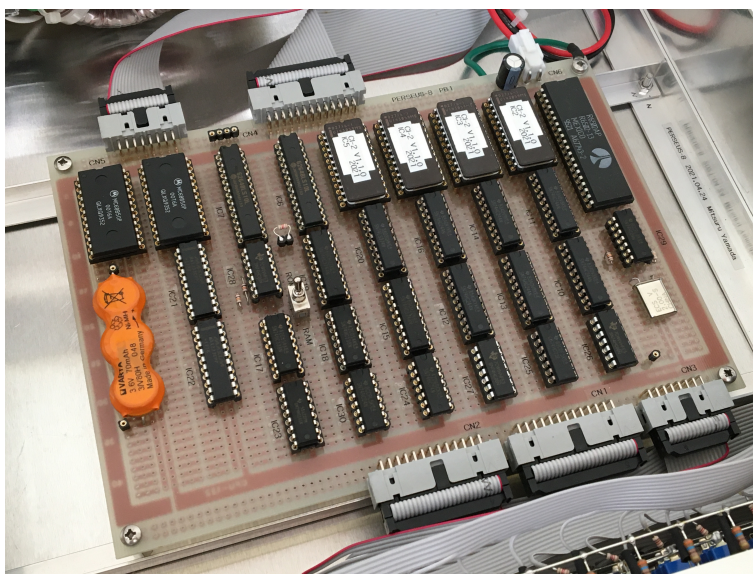


Fig. 11 ROMs of the floating point interpreter mounted on the PERSEUS-8 main board.

Whether the interpreter is executed in ROMs or RAMs is switched by a toggle switch on the board. When running in ROMs, the single step operation mode of PERSEUS-8 with a system clock of 2 MHz will not work. This is because the access time of this PROM is slow at 450 ns compared to 100 ns for SRAM.

References

- [1] R6500 MICROCOMPUTER SYSTEM HARDWARE MANUAL, Rockwell international
- [2] Yoshihiko Matsumoto, Watashi dakeno maikon sekkei & seisaku, CQ Publishing Co.,Ltd. 1977
- [3] 6502 standalone computer, <https://hackaday.io/project/175866-6502-standalone-computer>
- [4] PERSEUS-3 6802 homebrew computer, <https://hackaday.io/project/176137-perseus-3-6802-homebrew-computer>
- [5] https://cdn.hackaday.io/files/1791967666721664/PERSEUS-8_schematics_01.pdf
- [6] [https://cdn.hackaday.io/files/1791967666721664/PROM WRITER-3_schematic_01.pdf](https://cdn.hackaday.io/files/1791967666721664/PROM_WRITER-3_schematic_01.pdf)
- [7] [https://cdn.hackaday.io/files/1791967666721664/PERSEUS-8 Power supply schematic 01.JPG](https://cdn.hackaday.io/files/1791967666721664/PERSEUS-8_Power_supply_schematic_01.JPG)
- [8] ROM module for 6502 standalone computer, <https://hackaday.io/project/175923-rom-module-for-6502-standalone-computer>
- [9] [https://cdn.hackaday.io/files/1791967666721664/Source code MEMORY TEST-1_1_0.pdf](https://cdn.hackaday.io/files/1791967666721664/Source_code_MEMORY_TEST-1_1_0.pdf)
- [10] Homemade Floating Point Interpreter for 6502, <https://hackaday.io/project/181098-homemade-floating-point-interpreter-for-6502>
- [11] Fully manual PROM programmer, <https://hackaday.io/project/175924-fully-manual-prom-programmer>
- [12] [https://cdn.hackaday.io/files/1791967666721664/PROM WRITER 05_2.pdf](https://cdn.hackaday.io/files/1791967666721664/PROM_WRITER_05_2.pdf)

(Posted on Apr. 17, 2021)

(Latest revision on Aug. 06, 2026)