The Initial Input Routine of the Parametron Computer PC-1

Eiiti Wada*

Abstract

Forty years ago, the PC-1, parametron computer 1, was born at Professor Hidetosi Takahasi's Laboratory. The logical elements of the PC-1 were parametrons, which supported majority logic. The memory system operated in a two frequency read/write scheme. The word selection mechanism applied error correcting code to decrease the number of elements. Most of the hardware technologies were created by Eiichi Goto. We studied the EDSAC computer precisely, however we developed our own architecture and programming system based upon our own philosophy. The machine instruction set was chosen to ease programming. The normal teletype on the market was employed, leaving the burden of code conversion tasks to software, which seemed to us to have had almost infinite abilities.
However, the real memory capacity was indeed very small, which forced us to invent a clever way to implement things. In this paper, after introducing the functions of the initial input routine R0, examples of (i) code conversion table parasitic on the program body and (ii) the magic number method to control the number of multiplications, both used in the initial input routine, are described.
The PC-1 is one of the first computers which implemented interruption. That is, the peripheral devices would interrupt the running program by saving the address of the next instruction to be executed and jumping to a fixed location in the memory. As a simple experiment of multiple programming, cooperation of the binary to decimal conversion program and the printer control program by means of the circular buffer was performed.
At the end of this paper, the program lists of the selected routines are appended.

0 Introduction

The PC-1 (Parametron Computer 1)[0] was a binary, single-address computer developed at Professor Hidetosi Takahasi's Laboratory of the Department of Physics, University of Tokyo, and one of the first general purpose computers using parametron logic and two frequency magnetic core memory. Its construction was started in September 1957 and completed on March 26, 1958. The PC-1 closed its operation in May 1964.
The PC-1 was used for research works both in hardware and software at Takahasi's Laboratory and for scientific computation by the researchers of the Faculty of Science.
The EDSAC computer was the most influential in designing and implementing in all aspects because, in those days, [1] was the only available textbook.
The arithmetic and control circuits were made of 4200 parametrons; numbers were 2's complement binary: short number word was 18 bit and long one was 36 bit; instruction word was 18 bit single-address, some 20 different instructions; memory: 512 short words; clock was 15 KHz; addition and subtraction 4 clock times, multiplication 26 clock for short multiplier or 44 clock for long multiplier, division 161 clock, store 8 clock; power consumption: 3kva; floor area: 8 square meters. Input: photoelectric paper tape reader; output: teletype.
The PC-1 seemed to be the first computer that installed the interruption mechanism, which enabled us to experiment with multiple programming in 1959. The research on modular computations referred to in [2] was conducted on this machine.
The present paper reviews the parametron circuits and memory structure in sections 1 and 2, then the structure of memory and registers in section 3. In section 4, the teletype code used by the PC-1 is given so that the readers can understand the details of the input/output routines. Section 5 is the summary of the machine instructions. A few other instructions implemented later for experimental use are not included here. In section 6, the mechanism of interruption is described briefly. The program used in the multiple programming is listed in Appendix A. Section 7 sketches the initial input routine R0. A copy of the user's guide reproduced from the program library and the program list will be found in Appendices B and C. In section 8, I conclude. Appendices not mentioned above, D and E, are typical examples of the PC-1 input and output routines.

*Fujitsu Laboratories Ltd, e-mail: wada@u-tokyo.ac.jp

1 Parametron

The parametron was invented by Eiichi Goto, when he was a graduate student, in 1954. It was a resonant circuit of frequency f energized by a parametric excitation. When the circuit parameter was changed repeatedly with the twice frequency of the resonance, the circuit got energized. With respect to the excitation frequency of $2f$ (gray line), the circuit might be energized in one of the two possible phases, 0 and π (broken lines). (see Figure 0)

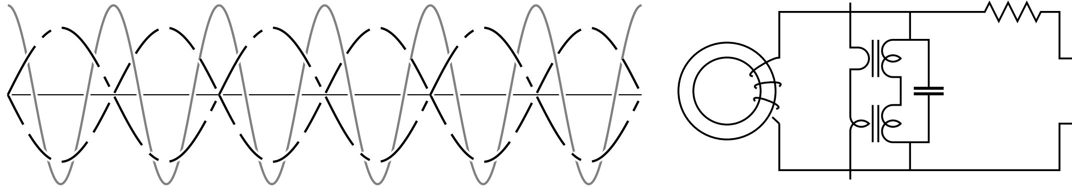


Figure 0 Parametron

One of the two phases was considered to represent 0 and the other 1, thus it was possible to represent one bit of information.

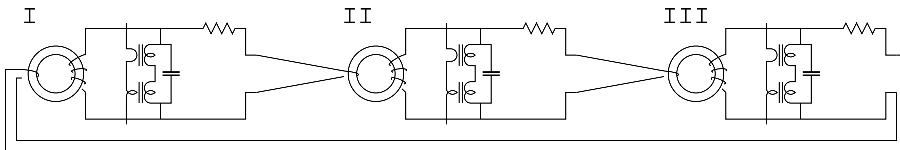


Figure 1 Parametron connection

The parametron had a circuit shown in figure 0 right. All the parametrons were grouped in one of the three sets, I, II and III. Each set was excited in turn and the output of set I was fed to II, II to III, III to I by connecting through the input transformers (Figure 1). The input signals were oscillation phases of the previous set and the oscillation of the new set was determined by the majority of input signals (phases). So, the parametrons realized majority logic (Figure 2). The negation was achieved by reverse coupling.

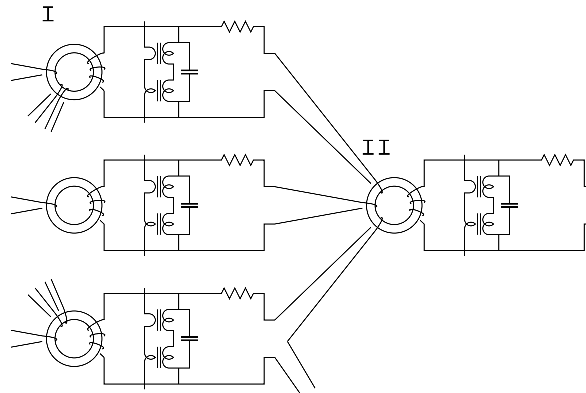


Figure 2 Parametron connection for majority logic

Figure 3 shows the typical operations with the possible constant input 0 or 1. (0 is represented as $-$, 1 as $+$ in the circle. The small cross on the input line indicates the negation.) In Figure 3, the rightmost circuit is the full adder. ([x,y,z] means majority of x, y, z.) One of the interesting circuits was a carry assimilator which assimilated n digit carries in $\log_2 n$ clock times.

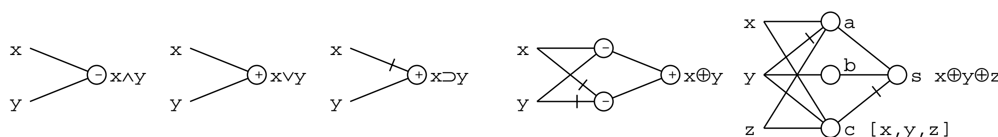


Figure 3 Parametron logic

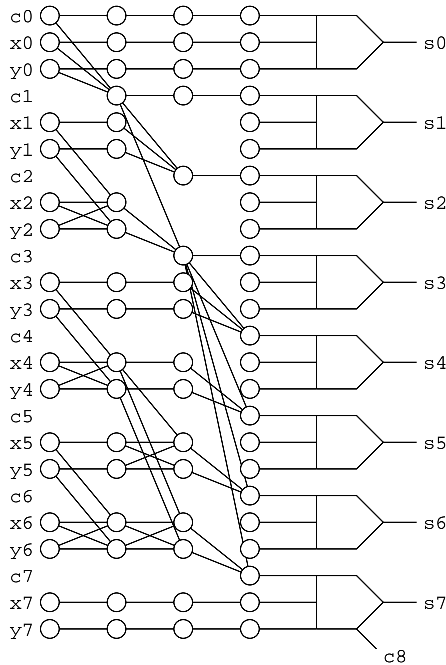


Figure 4 Carry assimilator

The carry assimilator works as follows: The parametrons in the figure are subscripted in each horizontal line 0, 1, etc from the left. The carry input to the highest full adder, c7, is 0 if both x6 and y6 are 0 and 1 if both x6 and y6 are 1. If $x6_0$ and $y6_0$ are 1 then $x6_1$ and $y6_1$ become to 1 and $x6_2$ and $y6_2$ also become to 1, thus $c6_3$ is 1. In case both x6 and y6 are 0, similarly c7 is 0. However, if x6 and y6 are 0 and 1, then two inputs to $x6_1$ and $y6_1$ cancel out and x5 and y5 will have the casting vote. If x5 and y5 are again 0 and 1, then c6 is determined by x4 and y4 at $x6_2$ and $y6_2$. In this fashion, carry inputs c4 to c7 are determined at the third parametron column. The carry inputs to c2 and c3 are determined at the second parametron column. Thus, in general, the steps to assimilate carries are $\log_2 n$.

Although not shown in Figure 4, the usual carry paths from one full adder to the next exist. They are used in the repeated additions during multiplication. At the final stage of multiplication, carries are assimilated using this circuit.

2 Memory

Most of the memory technologies of the PC-1 were also invented by Goto. Magnetic core memory of the PC-1 used sinusoidal waves rather than pulses for write/read operation. The core matrix consisted of a 36 $\times$ 256 rectangular wire net. In each writing operation, a sinusoidal wave of frequency $f/2$ was put through the selected one of the 256 row wires, and the 36 information bits were applied to the 36 column wires in the form of the sinusoidal wave of frequency f, where the phase of the latter wave represented each information bit. The cores on the cross points of both wires were subjected to the magnetizing force of the form $\mathrm{I}_0 \cos \pi f t \pm \mathrm{I}_1 \cos 2\pi f t$, and asymmetry of this wave form caused magnetization of the core in one or the other direction. (Figure 5)

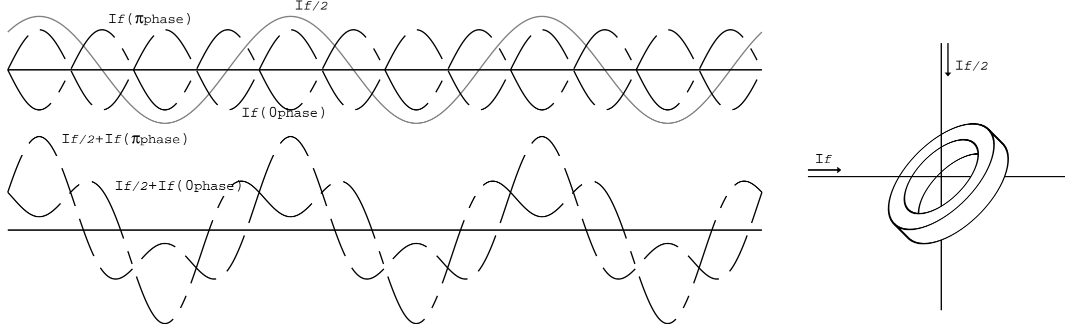


Figure 5 Memory write in

The reading out signals were obtained from the 2nd harmonic waves from the column wires. (Figure 6)

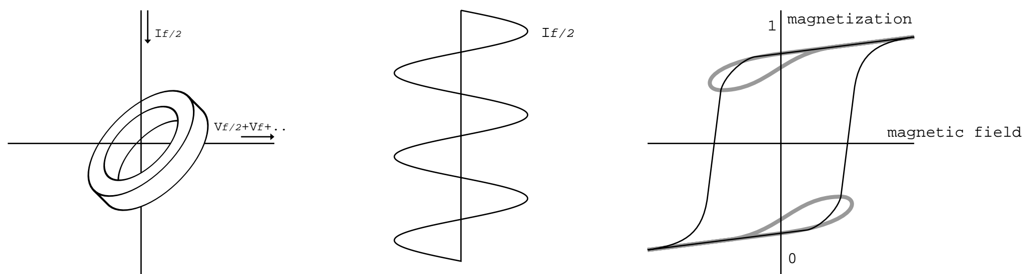


Figure 6 Memory read out

The address selection mechanism was based on the error correcting code[3].

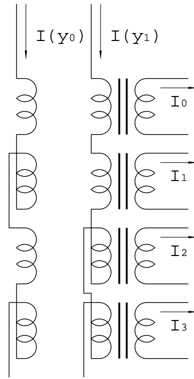


Figure 7 Word selector

Table 0 Selection logic

input		output			
$I(y_0)$	$I(y_1)$	I_0	I_1	I_2	I_3
−I	−I	−2I	0	0	+2I
+I	−I	0	−2I	+2I	0
−I	+I	0	+2I	−2I	0
+I	+I	+2I	0	0	−2I

The transformers shown in Figure 7, when driven by input currents $I(y_0)$ and $I(y_1)$ of the same frequency and the same amplitudes but in either phase, 0 or π, which are indicated by +I and −I, will produce the output current shown in the Table 0. The parametrons will oscillate when driven with full amplitude, but will not oscillate by weaker current. So, in this case, of the word selection parametrons driven by the output current, only those that are fed by ±2I oscillate, others remaining inactive. Now the difference between the maximal amplitude and the next one is only 2I and the discrimination power is not enough. However, by employing, for example, the 7 bit Hamming error correcting code for 4 input lines, discrimination power is 7I vs ±I, which is enough for the purpose of selection. The PC-1 used 18 bit input lines and excited only one word selection parametron out of 256.

3 Structure of Memory and Registers

Figure 8 left is the structure of the PC-1 memory. It consists of 512 short words. Like the EDSAC, two short words in $2n$ and $2n+1$ are used as one long word. Instructions are in the short words, but numeric data may be in 18 bits or 36 bit. The instructions which refer to the long word operand must have even number address and 1 in l/s bit. Some instructions, however, used the l/s bit for other purposes. Of three arithmetic registers prepared in the arithmetic unit, the accumulator and the R register were used for programming. The memory register was used to hold the multiplier and divisor. The contents of the arithmetic register were assumed to be fractional, i.e. numbers represented are in the range of $-1 \leq n < 1$.

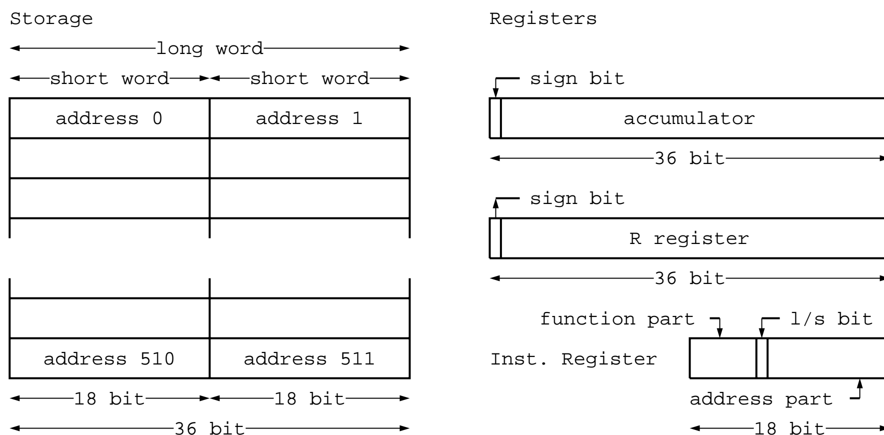


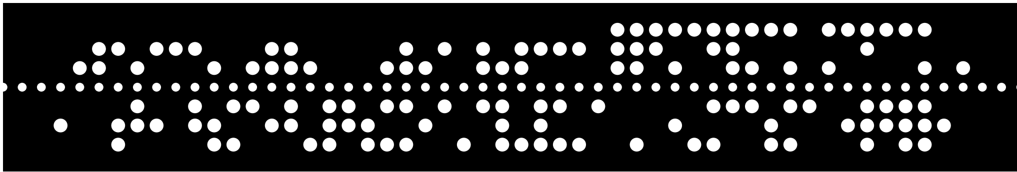
Figure 8 Storage and registers

4 Teletype Code

The PC-1 used the normal teletype for input/output without modifying the code. In those days, the teletype code used in Japan was of 6 bits. The lower 5 bits are similar to the 5 bits international teletype code. Number digits and special characters have 1 in the most significant bit. However, since the codes of the number digits are based on the character codes of the third row of the teletype, the code patterns and the values of numerical digits were independent. Therefore code conversion tables were needed by input and output routines.

Table 1 Teletype code

Code	Char	Code	Char	Code	Char	Code	Char
000000	Blank	010000	e	100000	Signal	110000	2
000001	t	010001	z	100001	4	110001	
000010	CR	010010	d	100010	-	110010	
000011	o	010011	b	100011	8	110011	
000100	SP	010100	s	100100		110100	
000101	h	010101	y	100101		110101	5
000110	n	010110	f	100110	,	110110	
000111	m	010111	x	100111	.	110111	=
001000	LF	011000	a	101000	+	111000	0
001001	l	011001	w	101001		111001	1
001010	r	011010	j	101010	3	111010	
001011	g	011011	FGRS	101011		111011	
001100	i	011100	u	101100	7	111100	6
001101	p	011101	q	101101	9	111101	
001110	c	011110	k	101110		111110	
001111	v	011111	LTRS	101111	:	111111	Erase



abcdefghijklmnopqrstuvwxyz 0123456789 +-=,.:

Figure 9 Tape image

5 Machine Instructions

`a` n, `al` n — Add the number in storage location n, nL into the accumulator.

`b` n, `bl` n — Replace the accumulator with the bitwise exclusive or of the numbers in storage location n, nL and the accumulator.

`c` n, `cl` n — Replace the accumulator with the bitwise logical and of the numbers in storage location n, nL and the accumulator.

`d` n, `dl` n — Divide the number in the accumulator and R register by the number in storage location n, nL and place the quotient in the accumulator, remainder in the R register. The remainder is always positive.

`i` n — If the tape reader is ready, read a character and place it into a_0 !A a_5 and clear a_6 !A a_{35}; if not ready, jump to n.

`jl` n — Jump to n.

`k` n — If the number in the accumulator is <0, jump to n.

`kl` n — If the number in the accumulator is >=0, jump to n.

`l` n — If $n < 1024$, shift the accumulator n places to the left; if $n >= 1024$, shift the accumulator logically $2048 - n$ places to the right.

`ll` n — Same as `l` n except to shift the accumulator and the R register.

`n` n, `nl` n — Clear the accumulator and subtract the number in storage location n, nL from the accumulator.

`o` n — If the teletype is ready, place a_0 !A a_5 to the teletype; if not ready, jump to n.

`p` n, `pl` n — Clear the accumulator and add the number in storage location n, nL into the accumulator.

`q` n, `ql` n — Place the number in the R register and load the R register with the number in storage location n, nL.

`r` n — If $n < 1024$, shift the number in the accumulator n places to the right; if $n >= 1024$, shift the number in the accumulator $2048 - n$ places to the left.

`rl` n — Same as `r` n except to shift the accumulator and the R register.

`s` n, `sl` n — Subtract the number in storage location n, nL from the accumulator.

`t` n, `tl` n — Store the contents of the accumulator to storage location n, nL.

`v` n, `vl` n — Multiply the number of the accumulator and the number in storage location n, nL and place the product in the accumulator and the R register.

`w` n, `wl` n — Do the same as vn vl$n$ and add the original contents of the R register multiplied by 2^{-17} if `w` n, 2^{-35} if `wl` n to the product.

`x` n — Store a_7 !A a_{17} to the address part of storage location n.

`z` n — Jump to n if a_7 !A a_{17} is 0.

`zl` n — Jump to n if the content of the accumulator is 0.

6 Interruption

From the very beginning, the input/output instructions were designed to have busy jump facilities, by which, when the devices were not ready, instead of waiting for the completion of the operation, the programmer could choose another path by jumping to that program context. However, one year's experience concluded that the busy jump facilities were hard to use effectively.

So in the summer of 1959, another approach was undertaken. That was interruption. i.e., the devices were designed to interrupt the program whenever their state became ready for use.

The interruption designed at that time worked as follows:

0. When the device operations completed, the program counter which held the location of the next instruction was stored in the address part of location 510 and control was sent to 511 (last storage location).
1. At the same time, further interruption was prohibited by setting a flipflop, because, otherwise, the return location would be overwritten by the subsequent interruption.
2. The jump instruction in 511 could lead the control to the interruption process program.
3. At the end of the interruption process program, after resetting the interruption prohibit flipflop, the program returned to the former routine with the information in 510.

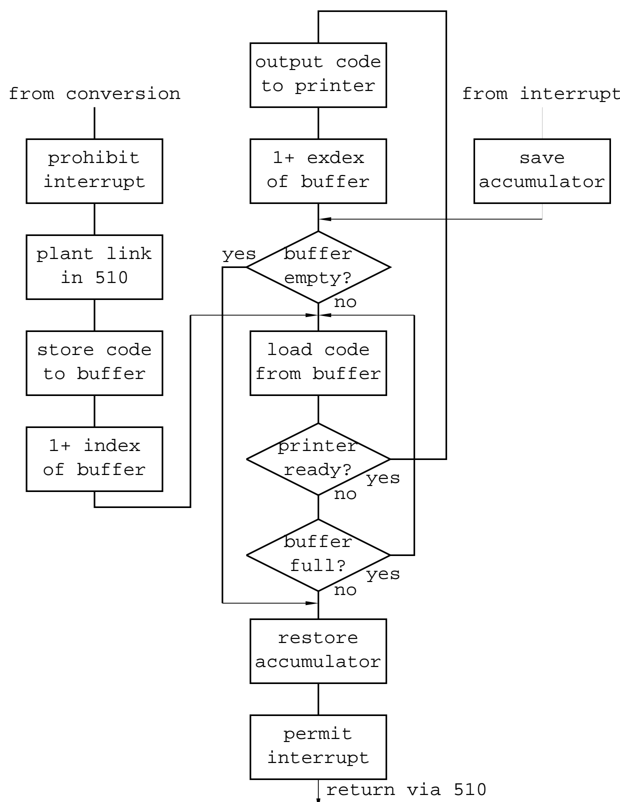


Figure 10 Interruption process program

In the print routine like P2 (Appendix E), digits were printed after binary-decimal conversion and code conversion to the teletype codes. However, one digit conversion was quicker than one digit printing by the teletype. Accordingly, the output conversion program placed the teletype code in the circular buffer without waiting for the completion of the previous printing. Later, when the teletype finished one digit printing, it interrupted a running program, and the interruption process program, taking up the control, started the teletype again with the next code taken from the circular buffer.

The program list is shown in Appendix A.

7 The Initial Input Routine R0

R0, the initial input routine of the PC-1, is similar to the initial orders of the EDSAC or the DOI (decimal order input) of the ILLIAC I, in that the function codes are denoted by the mnemonic code, the address numbers are denoted by decimal numbers etc. In the microscopic view, there are several differences.

0. Because of the use of 6 bit teletype code, there were more characters available for programming. Without shifting to the upper case, we could use all the small alphabetic characters, decimal digits, and 6 special characters, (comma, period, colon, equal, plus and minus)
1. Instructions and directives were terminated by the special characters, thus it looked more user friendly.
   (a) instructions and short numbers were delimited by commas.
   (b) directives were delimited by periods.
   (c) start addresses of the program segments were indicated by colons.
   (d) code letters were defined by equal signs.
   (e) long numbers were delimited by plus or minus characters.
2. Teletype codes for number digits had no relation to the number values, a code conversion table had to be used and it was allocated in the initial input routine. The address numbers of some instructions of the initial input routine served as the code table.
3. A digit counter for reading fractional numbers was implemented using the magic numbers, that is the ten odd numbered decimal fraction consisted of the most significant ten bits.

The user's guide of R0 is given in Appendix B. Program code is in Appendix C.
A short example of the usage of the initial routine follows:

```
0p=116,                              {code letter p is defined to 116.}
0p: [program of print routine]       {print routine stored from 116}
...tlh, it, jlp,...                  {tlh, a number to be printed is stored in 0hL,
                                      it, loads the address in the accumulator (Wheeler linkage),
                                      jlp, is an unconditional jump to print routine}
```

7.0 Code Table Hidden in the Program

As aforementioned, the codes of number digits and number values were independent. So, we have to have the code conversion table somewhere. In the first version of the initial input routine, ten consecutive short words were used to accommodate the code table, just like the character table in the subroutine P2 shown in Appendix E, and each input character had to be compared to determine its numerical value. The conversion using the table of other direction, i.e. a table from code to numerical value was considered to occupy a larger area since the smallest code was -31 (4) and the largest -4 (6); the table size could be 28 short words, which seemed too large for the 512 word computer. So, the accepted solution was to hide (or distribute) the code table in the program body of the initial input routine. This really worked fine.
Table 2 is a list of digits, the corresponding code and the value of the codes. Each of the values + 56 was the address where an instruction with the number in its address part had to be stored. The instructions of these locations were shown in the last column of the table. The total program is in Appendix C.
Similarly, other terminating characters were treated by assigning the pseudo instructions (in the program of R0, this is referred to as the base orders) in the appropriate locations. The result are shown in Table 3. The star marks in the column of content indicate 1 in the sign bit. To obtain the instructions (most of them are jump instructions), the content was added to `*nl 12` (instruction in 14 of the initial input routine).
This was possible because the initial input routine was stored in a fixed place and the addresses were unchanged.

Table 2 Conversion table for digits

digit	code	value	address	content
0	111000	-8	48	r 0
1	111001	-7	49	l 1
2	110000	-16	40	p 2
3	101010	-22	34	x 3
4	100001	-31	25	jl 4
5	110101	-11	45	kl 5
6	111100	-4	52	jl 6
7	101100	-20	36	rl 7
8	100011	-29	27	Space 8
9	101101	-19	37	ll 9

Table 3 Base orders for special symbols

symbol	code	value	address	content	directive
,	100110	-26	30	*s 45	jl 57
.	100111	-25	31	*s 3	jl 15
:	101111	-17	39	*el 54	x 66
=	110111	-9	47	*nl 12	p 24
+	101000	-24	32	*s 6	jl 18
-	100010	-30	26	*s 6	jl 18

7.1 The Magic Number

In the days of the PC-1, memory was the most crucial resource. One of the techniques used in the input routine was counting the number of characters by means of the strobe, the magic numbers, in the course of reading in a fractional number up to 10 decimal digits.
Reading in the fractional numbers, for instance, 0.25, the fractional part was first read as an integer. So, 0.25 appeared in the accumulator simply as 25. Now, the number of characters already read in was 2. Accordingly, the content of the accumulator was multiplied by 10^8 and then divided by 10^{10}.
When the decimal point was read, the magic number was loaded into the accumulator where the decimal to binary conversion was performed . (The period was identified as the decimal point if the working area was cleared; otherwise, it served as the directive terminator.)
The magic number was 1.193359375 in decimal and 1.001100011 in binary representation. Since all digits were odd, in the course of multiplication by 10 during the decimal to binary conversion, the sign bit always remained as 1, i.e. the number seemed negative. And the least significant bit of the magic number was being shifted to the left 1 bit each time. At the same time, the result of decimal to binary conversion creeped up from the right.
The input of a long fractional number is terminated by + or -. When the input was terminated, the dummy multiplication by 10 was repeated until the accumulator became positive, which meant the multiplication by 10 was executed exactly 10 times. (The program list of R1 is in Appendix D.)
In the diagram below, the magic number for 6 digits is shown and the process of multiplication is attached.

The 6 digits magic number is calculated like this. Write down the numbers $2^0, 2^{-1}, 2^{-2}, \dots, 2^{-5}$. Then summing up proceeds from the bottom seeing if the result of addition brings the odd digit in the last position of that number. If the odd digit is produced, add that number; or else skip the addtion of the number by crossing it out. This brings up the number 1.59375 as shown below.

The magic number for 10 digits

1.0
~~0.5~~
~~0.25~~
0.125
0.0625
~~0.03125~~
~~0.015625~~
~~0.0078125~~
0.00390625
0.001953125 +
1.193359375
1.001100011

The magic number for 6 digits

1.0
0.5
~~0.25~~
~~0.125~~
0.0625
0.03125 +
1.59375
1.10011

```
   1.10011 =  1.59375
*      1010  (1st)
------------
   11.0011
+ 1100.11
------------
 1111.1111  = 15.9375
    1.1111  =  1.9375
*     1010   (2nd)
------------
   11.111
+ 1111.1
------------
 10011.011  = 19.375
    1.011   =  1.375
*    1 010   (3rd)
------------
   10.11
+ 1011
------------
 1101.11    = 13.75
```

to upper right↗

↗from lower left

```
    1.11    =  1.75
*   10 10    (4th)
------------
   11.1
+ 111
------------
 10001.1    = 17.5
    1.1     =  1.5
*  101 0     (5th)
------------
   11
+ 11
------------
  1111      = 15
     1      =  1
* 1010       (6th)
------------
   1
+ 1
------------
  1010      = 10
     0      =  0
```

8 Conclusion

The PC-1 played a remarkable role in the research of computer architecture and program library within our group and helped a lot the research in computational physics and chemistry in the neighbour laboratories. The preparation of the system programs was quite essential in all activities in and out of the group. In this paper, some typical techniques used in our system programs were explained.

In retrospect of the research activity of that period of time, the members of the laboratory benefited very much by having our own computer. In these relatively short years, we could learn the whole life cycle of computer development from the hardware elements to the program library. The home made computer enabled us to make any experiments in hardware or software that came to mind promptly. The scarceness of computers around us resulted in the original research in this field.

After the fundamental system programs were prepared, more advanced programming systems had been developed, though I have to say, the PC-1 is too small for such ambitious projects. For instance, introduction of the symbolic addresses in the initial input routine was tried. This idea was implemented by people of the chemical department. They used a list mechanism to remember the unsolved symbols until symbol/location correspondence is settled. The modular arithmetic system came from the idea of Takahasi. In this implementation, he used `wl` n operation to obtain the remainder of division by a large prime number. Let A and R be the contents of the accumulator and the R register respectively and n be the address of the location which holds h such that $2^{35} - h$ is the prime number with which the division of $2^{35}A + R$ will be made. His method is to repeat the multiplication with `wl` n until the accumulator becomes zero. One day, a filp flop of the PC-1 was connected to a speaker cone and program could push or pull the cone generating a sound. The sound pitch was controlled by adjusting the shift number of the shift instruction and the sound duration was controlled by a busy jump facilities of output instruction. It is easy to implement this sort of program hacking.

A few years later, the design of the next parametron computer was started. But, for this new machine, PC-2, the initiatives of construction was in the hands of Fujitsu Ltd., it never gave us a big impact as the PC-1 did. The PC-2 was installed in the common computer room of the Faculty of Science, University of Tokyo and was used for a while by the community.

Every five years, the members of the group who built the machine and maintained the library meet on March 26 to celebrate the birthday of our lovely PC-1.

A The Interruption Process Program

The interruption process program is stored in the location 470∼511. During the binary to decimal conversion in the print routine, each time one character code is obtained, instead of output the code directly to the printer, the code should be placed in the upper 6 bits position of location 470. Then this routine is called by jumping to 474 using the normal linkage. The code is stored once in a cyclic buffer and then, when the printer completed the printing action of the previous character and interrupted, the code will be output to the printer by this program. The cyclic buffer must be specified by the preset parameters:
`0h` = the first address of the buffer,
`0n` = the size of the buffer.
Note:
The instructions `yl 30` and `y 30` sets and resets flip flop 30 respectively. This flip flop is used to mask the ready signals from the teletype. When the signal is not masked, it interrupts immediately; if it is masked, it waits until the mask is turned off and then interrupts the program.

```
        470  (     )
        471  (     )
        472        1  Constant
        473        2  Constant
        474  yl   30  From code conversion, Prohibit Interruption
        475  a   473  }Plant link
        476  x   510  }
        477  p   470  }Write the code in
        478  t ( h)   }the cyclic buffer
        479  p   478  }Advancd the input address
        480  a   472  }of the cyclic buffer
   486→481  x   478
        482  s   490
        483  z   485
        484  jl 495
             ------
   483→485  p   487  }Reset the address of
        486  jl 481  }the cyclic buffer
             ------
        487       h   The first address of the cyclic buffer
        488  (     )  }The location to save the
        489  (     )  }accumulator
        490     n+h   The last address of the cyclic buffer + 1
   511→491  tl 488   Save the accumulator
   502→492  p   495  }
        493  s   478  }Jump to 508 if buffer is empty
        494  z   508  }
   507→495  p ( h)   Read the code to print from the buffer
        496  o   505  Print the code if printer is ready
        497  p   495  }
        498  a   472  }Advance the output address of
   504→499  x   495  }the cyclic buffer
        500  s   490  }
        501  z   503  }
        502  jl 492  }
             ------
   501→503  p   487  }Reset the address of the
        504  jl 499  }cyclic buffer
             ------
   496→505  p   495
        506  s   478  }If the buffer is full, jump to 495
        507  z   495  }
        508  pl 488   Restore the accumulator
        509  y    30  Reset the interruption prohibit
        510  jl ( )   Return to the mail routine
             ------
        511  jl 491   Entrance to the interruption program
             ------
```

B Instruction for use of the basic input routine R0

B.0 Introduction

The PC-1 is a stored program computer and uses the binary system within the machine for the representation of numbers and addresses. In using the PC-1, the program as well as the numerical data should first be stored in the machine's memory, before computation starts. This may be done, for one thing, by preparing a tape in which all the instructions and numbers are represented in binary form, and reading in this tape by pressing down the "initial load switch". However, writing down the instructions and numbers in binary notation is by no means simple.
The basic input routine "R0" enables the instructions and numbers punched on tape in decimal, alphanumeric notation to be read and placed in the PC-1 memory. R0 decodes the teleprinter code of the PC-1 perforator, converts the decimal numbers into binary form, adds the operation codes and places the assembled words in specified locations in the memory. When all the instructions and numbers have been stored, R0 causes the machine to start the program by transferring control to a specified word in the memory. R0 provides further facilities of turning the relative addresses on tape into absolute addresses, and adding one or more parameters to the words before they are stored in the memory. Input routine R0 itself occupies the locations 0∼67 of the PC-1 memory and is stored there by placing the binary tape of R0 in the tape reader and pressing down the "initial load switch".

B.1 Basic Functions

Every instruction of the PC-1 is punched in exactly the same form as it is written in the text. That is, operation code consisting of letter, either followed or not followed by a letter "l", and followed by a decimal integer denoting the address, and terminated by a comma, such as `a40, x106, pl150,` ... Nonsignificant zeroes at the head of the address may be omitted, so that one may punch `a40` instead of `a0040`. Sequence of instructions punched one after another on tape is normally placed in consecutive locations in the memory. The location of the first instruction in a sequence must be specified by a "directive" in front of the sequence, which has the form `M:`, indicating that the sequence should occupy the storage locations M, M+1, M+2, ... in the memory.
Example: The tape `100:  pl150, vl152, sl154, tl156, jl130,` causes the memory locations 100 ∼ 104 to be loaded by the following words.

locatoin	instruction	contents of the memory
100	pl 150	001101100010010110
101	vl 152	001111100010011000
102	sl 154	010100100010011010
103	tl 156	000001100010011100
104	jl 130	011010100010000010

A program tape will consist of one or more sections of such sequences, or subprograms. Blank section of some ten centimeters should be left at the head of the program tape, and the program should begin with a "carriage return and line feed" (`CR LF`), which clears the working positions of R0 prior to reading essential information.
The complete program should end with a control code `jlM.` (terminated by a period "`.`"), which stops the operation of R0 and starts the program by causing the control to be transferred to the location M.
Notes: 1. When the address is 0, this may altogether be omitted, for example, "`jl0`" may be written simply as "`jl`".
2. It is preferable to leave a short blank section in front of each section of the program, and to punch a `CR LF` at the beginning of each. This serves the purpose of easy identification of the subprograms and also it enables the reloading of some part of the program by manually transferring control to the location 0.
3. R0 ignores "blank" symbols.
4. "Erase" symbol is not ignored by R0, so that mispunched characters overpunched by 6 holes should be removed in the tape-editing stage.

B.2 Input of Intergers

Input of numbers occurs just as the input of instructions, that is, the number is regarded as if it were an instruction without operation code, or with the operation code "0" (blank tape). Any positive integer N ($0 \le N < 2^{17}$) can be read in by punching N in decimal notation and terminating by a comma "`,`".

Ex. `150: 12345,` causes an integer 12345, or a fraction 12345×2^{-17} to be stored in memory location 150.

Negative integers $-$N can be read in by punching an integer $2^{18}-N$, where $0<N\leq2^{17}$.

Ex. -2 can be read in by punching `262142,`.

Long numbers are usually regarded as pairs of short positive integers (N_1, N_2) in the form $2^{18}N_1+N_2$. This is the standard way of reading in the numerical constants in library subroutines.

Note: For input of numerical data as well as constants in programs specially drawn up by the user, this would be quite inconvenient, and a universal input auxiliary routine R1 will be used for reading in signed long numbers, either integral or fractional. For input of signed, short numbers, R3 may be used instead of R1.

B.3 Relative Addresses

R0 offers an ample facility of using the preset parameters in the instructions and constants. Namely, any number of alphabets can be written as code letters in a word, except at the head, in which case it is interpreted as the operation code. Each code letter causes a number (a parameter), specified by a precursor tape, to be added to the word before it is stored in the memory. Actually, the specified parameter values for these code letters are stored in the storage locations 57∼87, but in general, the programmer is supposed to have no knowledge of exact storage locations for individual code letters.

There are several code letters that are reserved for specific purposes and cannot be assigned arbitrary values by the programmer. These are:

`l` =2048 (=2^{11}), used to indicate "long" instructions.

`t` =`tM` (M is the current address of the storage location where the word is to be stored).

`r` =$\mathtt{M_0}$: the storage location of the first wored of the current sequence of instructions. This is assigned by the directive $\mathtt{M_0}$`:`.

"`LF`" punched as a code letter is an erase symbol and serves the purpose of canceling the characters within the same word already read from tape. This function of `LF` is made use of by the `CR LF` at the head of the tape to insure uniform initial state in the operation of R0.

Setting the values to the code letters is done by a tape of the following form: e.g.

```
0h=131072, 0m=18, 0a=256, ...
```

using a new control symbol "`=`". Another way of setting the code letters is of course by storing these values by ordinary directives, e.g.

```
61:131072, 63:18, 80:256, ...
```

but this is inconvenient since it requires a precise knowledge of memory locations for individual code letters.

Notes:

1. Teleprinter control characters ("space"(SP), "upper case"(UC), "lower case"(LC), "carriage return"(CR) and "line feed"(LF) are treated by R0 as letters. Hence, these characters punched at the head of a word (that is, next to a terminating symbol) are regarded as (fictitious) operation codes. These symbols (except `LF`) punched in positions other than the head of a word is interpreted as code letters, and can in principle be used as variable code letters, except `CR`, the corresponding memory location for which is used by an instruction in R0. However, using teleprinter control characters as code letters is not recommended.

2. Code letters `o` and `g` are set to 28 and 68 respectively, usless otherwise specified (see next section). `SP`, `h`, `m` and `n` are set to zero by R0 tape.

3. Initial 0 in `0h=131072`, only serves the purpose of making the letter `h` to be interpreted as a code letter, and hence can be replaced by any other letter or teleprinter control symbol.

4. Setting of parameter values using "`=`" symbol destroys the current store address, in case it is done between program sections.

B.4 Use of Code Letters

The directive `M:` sets the code letter `r` automatically to M, the location of the first word in the current sequence of instructions, and hence any word within the same sequence can be referred to by relative address by use of this code letter, e.g.

```
jl1r,
```

means jumping to the second instruction in the same section of the program.

The code letter `t` is actually the instruction to store the assembled word to ultimate place in the memory, and hence its values is always equal to `tN`, where N is the current destination of the word. Hence this code letter can be used to refer to the word itself, e.g.

`it` becomes `pM` (i+t=p)

`gt` becomes `iM` (g+t=i)

tllt becomes oM (tllt+t=o)
where these words are supposed to be placed in location M. Use of code letter t for purposes other than the above is not recomended.
A standard use of variable code letters is to indicate the location of the first word in each subroutine, e. g. instead of

116: | program of print routine |,

we punch at the head of the complete program tape

0p=116: ...

and the subroutine tape begins with

0p: | program of print routine |,

so that this closed subroutine can be called in by an instruction pair

it, jlp, instead of it, jl116,.

This use of code letters greatly simplifies the use of library subroutines. The code letter o is used to refer to the storage location 28 which is used to store the independent variable in some closed subroutines. The code letter g(= 68) is used to refer to the working spaces of all kinds of program. However, the numerical value of either o or g can be modified for the programmer's convenience.
Code letters may appear in the directives as well as in instructions. It may also be used in setting of variable code letters, e.g.

0a=116, 0b=28a, 0c=36b,

is equivalent to 0a=116, 0b=144, 0c=180, and jla. has the same effect as jl116.

B.5 Special Directives

0t: This sets the r-parameter equal to the current store address.

0r: This causes the following instructions to be stored starting in the storage location specified by the previous directive.

Nr: This causes the following instructions to be stored starting in the N-th storage location of the preceding section of the program.

Nt: This causes the following instructions to be stored in a new location, skipping N storage location next to the preceding program.

0t:0h=0r, This sets the parameter h equal to the current store address. (0h=0t, is unacceptable since the symbol "=" destroys the store instruction.) This combination is placed before a subroutine having directive 0h:, to place it next to the end of preceding program. (h may be replaced by any other variable code letter.)

B.6 Control Code with "."

tN. (kN., zN.) Stores the following program starting in the location N, without resetting the r-parameter.

Nt. Stores the following program in a new location skipping N storage location next to the preceding program.

N. Stops tape. When the machine is restarted, the following program is stored starting in the location N.

tNt. Stops tape. When the machine is restarted, the following program is stored skipping N storage locations.

tt. Special case of tNt. Stops the tape without destroying the current store address.

8. Stops tape. When the machine is restarted by inital start key, the following program is stored starting in the current storage locations. (analogous to tt. in effect.)

t57.jl,zN. Replaces the store instruction by jlN, that is, transfers control to the word at N, leaving the word following in tape in the accumulator. Successive words will be read one by one and placed in the accumulator, each time R0 is called in, transferring control to the same word. (This is used when input of one word is called for during program.)

t57. Used to restore the operation part of the store instruction to the original one (t), in case it has been modified (to, say, jl).

t57.o56,X,t57. Print the letter X without disturbing the memory. (When more than one letter is to be printed, it is recommendd to use an interlude. See below.)

B.7 Input During Program

Although R0 is primarily designed for the input of the program, it is also suitable for such purposes as reading in numerical values called for by program a few at a time. A slight modification on R0 will make it still more adaptable for such uses.
If R0 is called for by jl40 in a program, the last current store address remains unaltered. If it is called for by jl55, the current contents of the accumulator replace the current store address, so that the word read can be stored in locations specified by the program.
Exit from R0 may be made in the usual way by punching jlN. at the end of the number sequence. However, it is more convenient to provide for the exit by means of some unused symbol. In case R1 is not used, either "+" or "-" symbol may be used to send control to a location N. This is done by replacing the word in 32 or 26 respectively by a pseudo-instruction (212980+N). CR and LF can also be very conveniently used for the exit from R0. This is effected by 23:jlN,.
Using the input routine thus modified, it is also possible to terminate a number by CR and LF, in which case, the number is left in 0oL (28L) as a long integer.
Note: The original program of R0 sets the machine to dynamic stop on reading + or -.

B.8 Interlude

It is sometimes useful to do some simple operations, such as calculating constants used in the main program, printing table heading, etc. by temporarily placing a short program in the memory and transferring control to this program, before the main program has been stored. This type of program is called the interlude. At the end of an interlude, control should be transferred to R0 by, say, jl40, and subsequent program is usually written over the interlude, so that no extra space need be reserved for it. Following are examples of short programs that are often used as interludes.

1. 0t:c49,jl4,jlr.:
   This has the effect of increasing the current store address by one, if it is odd, and hence may be placed before an even subroutine in order to make sure that the location of the first word be even.
2. 0t:s,o1r,l2,o3r,jl6r,jlr.:
   This causes carriage return and line feed to occur during input.
3. 0t:ir,zl40,o2r,jlr,jlr.:
   This causes the contents of the tape following it to be copied directly by the teleprinter, and may be used to print relevant information concerning the program, e.g. table heading. Blank tape at the end sends control back to R0.

C The Initial Input Routine R0

Location	Order	Notes
38→0	al 28	Decimal to binary conversion
1	jl 49	
2	0	Constant
3	(0)	Temporary storage for binary number
25→4	a 64	Add function
45→5	t 64	Store function and parameter
52,7→6	i 6	Read code, number or symbol
7	zl 6	Jump if blank
46→8	rl 12	Shift teletype code to address part
9	kl 20	Jump if code letter
10	a 33	Add address base
11	x 12	Assemble load order

```
        12   p ( 0)  Load number or base order
        13   kl 34   Jump if number
        14   a  47   Modify base order into switch order
    18→15   t  18   Set switch order
        16   p  29   Load address
        17   a  64   Add function and parameter
        18  (    0)  switch order(jl18,jl57,jl15,x66,p24)
        19   jl 55   Jump to "set transfer order"
     9→20   a  33   Add address base
        21   x  24   Assemble load order
        22   s   4   Examine whether code letter is LF
        23   z  40   Jump to clear order in case of LF
        24   p ( 0)  Load parameter
        25   jl  4   (4) Jump to "add function" order
        26  *s   6   (-) Base order for "jl 18" (* means 1 in sign bit)
        27   SP  8   (8) Constant, function part is SP
        28  (    0)       }Working space for
        29  (    0)       }decimal to binary conversion
        30  *s  45   (,) Base order for "jl 57"
        31  *s   3   (.)  Base order for "jl 15"
        32  *s   6   (+) Base order for "jl 18"
        33   CR 56   Address base, function part is CR
    13→34   x   3   (3) Store binary number
        35   pl 28        }
        36   rl  7   (7)  }Decimal to binary conversion
        37   ll  9   (9)  }
        38   jl  0
        39  *el 54   (:)  Base order for "x 66"
 56,23→40   p   2   (2) Load 0
        41   tl 28   Clear working space for conversion
        42   t  64
    44→43   i  43   Read function, number or symbol
        44   zl 43   Jump if blank
        45   kl  5   (5) Jump if function letter
        46   jl  8
        47  *nl 12   (=) Base order for "p 24"
        48   r   0   (0)
     1→49   l   1   (1)  }
        50   al  2        }Decimal to binary conversion
        51   tl 28        }
        52   jl  6   (6)
    58→53   p  57
        54   a  49   Increase "transfer order"
    19→55   x  57   Set "transfer order"
        56   jl 40   (Blank) Jump to clear order
    18→57   t (67)  (t) Transfer order
        58   jl 53   (CR)
        59      28   (o) o-parameter
        60       0   (SP)
        61       0   (h) h-parameter
        62   parity  (n) n-parameter
        63   digit   (m) m-parameter
        64  (    0)  (LF) Working space for function and parameters
        65    2048   (l) l-parameter for long word order
        66  (    0)  (r) r-parameter
        67      68   (g) g-parameter, End of tape
```

May 1, 1958
E. Wada

D Read Subroutine R1

R1 Input of Signed Long Integers and Decimal Fractions
Special; even; 28 storage locations (normally 88–115);
used with initial input routine R0.
R1 is used to read in long integers and decimal fractions of either sign into successive long storage locations in the memory. During the reading in of R1, the initial input routine is slightly modified so as to enable use of the control symbols "+" and "-" during its operation. With this routine, any number less than 2^{35} followed by a "+" will be read as a positive integer, and any number followed by a "-" as a negative integer. Numbers exceeding this limit will be interpreted modulo 2^{36}. If the number is preceded by a decimal point with or without a zero ("0." or ".") and followed by a sign it will be read in as a decimal fraction of that sign. Any number of digits up to 10 may follow the decimal point. For instance, 2^{-1} can be punched .5+ or 0.5+ instead of .5000000000+, but punching more than 10 digits will give incorrect result.
Long numbers and short words may appear mixed on the tape and they are stored successively in their natural order on the tape, but the location of long numbers must always be such that they are stored in even storage locations. Control symbols "+" and "-" cause storage address of the number to be increased by 2. Hence no extra "," should be placed after long numbers. No code letter nor function symbol can be used with long numbers.

```
           88:
    18→0   zl 2r     } decide whether the symbol just read is a decimal point
       1   jl 15     }or a control symbol.  If the latter, jump to 15
    0r→2   p 4r      place the "magic number" in 28 and 64
       3   jl 41     (by 41 and 42 of R0)
       4   ||156416  "magic number" 1.193359375
       5   ||nl 28
    18→6   p 12r     place "pl 28" in 23r on reading "+"
       7   jl 9r
    18→8   p 5r      place "nl 28" in 23r on reading "-"
    7r→9   t 23r     place " pl 28", "nl 28" in 23r.
      10   p 64      }test whether an integer or a fraction
      11   kl 19r    } by reading C(64)
      12   pl 28     } test whether multiplied
      13   kl 17r    } 10 times
  16r→14   v 48      }Multiply by 10 up to ten times
      15   ll 5
      16   k 14r
  13r→17   dl 26r    Divide by 10^10
      18   tl 28
  11r→19   p 57
      20   x 24r     Set store order for long number
      21   a 40      Increase address of store order
      22   x 57      by 2
      23   (    )    Becoms "pl 28" or "nl 28" depending on sign
      24   tl ( )    } Store a long number in ultimate
      25   jl 40     }storage location
      26   ||38146   }10^10
      27   ||254976  }
           t 26.
           212988r   (-)         jl 8r
           t 31.
           212980r   (.)         jl r
           212986r   (+)         jl 6r
           28r.
```

E Print Subroutine P2

P2 Print one signed decimal in 0oL (without layout or round off)
Closed; Self setting; 36 storage locations;
Time is determined by typing speed.
Prints the signed number in 0oL to H places of decimals, leaving $R \times 10^{H}$ in R register where R is the re-

mainder. H is determined by a preset parameter. Each number is preceded by a decimal point and followed by a sign.

Preset parameter:

`0h = H` Number of digits to be printed.

Notes:1. For example, 0 is printed, if H is 10, as
`.0000000000+`
and -0.5 as
`.5000000000-`
-1 will be printed as
`.0000000000-`

2. Before P2 is called in, digit position r_0 must be cleared.
3. The output tape perforated by P2 less than ten digits may be used as an input tape for use with R1. In this case, however, no space should be placed between numbers.

```
        0p:
     0  a 12r
     1  CR t      Print a decimal point
     2  x 35r     Plant link
     3  pl o      }Load the number in
     4  rl 35     }R register
     5  kl 18r    Test sign
     6  nl o      Change sign
     7  rl 35     Load the number in R register
     8  p 9r
     9  jl 19r
    10  ||139264  420000_8                 }
    11  ||143360  430000_8                 }
    12  ||j 2                              }
    13  ||s       10×2^-4                  }Table of
    14  ||g ( )   Counter                  }charcters
    15  ||LTRS h                           }
    16  ||155648  460000_8                 }
    17  ||f 10r   Base order               }
 5r→18  p 22r     Store address to select  }
 9r→19  x 33r     a code of sign           }
    20  p 15r
32r→21  x 14r     Store counter
    22  q 22r     Shift the number to acc.
    23  v 13r     }Multiply by 10×2^-17
    24  rl 13     }
    25  a 17r     Add base order
    26  x 27r     }Select the code
    27  a ( )     }
    28  CR t      Print a digit
    29  rl 18     Store fraction in R register
    30  a 14r     Subtract 1 from counter
    31  z 33r     Test count
    32  jl 21r
    33  a ( )     }Print sign
    34  CR t      }
    35  jl ( )    Link
```

References

[0] H. Takahasi ed: Parametron Computers, Iwanami Shoten, 1968. (in Japanese)

[1] M. V. Wilkes, D. J. Wheeler and S. Gill: The Preparation of Programs for an Electronic Digital Computer. Addison-Wesley Press, Inc. 1951.

[2] D. Knuth: The Art of Computer Programming, vol 2, Seminumerical Algorithms, 3rd ed. p. 291.

[3] H. Takahasi and E. Goto: Application of Error Correcting Codes to Multiway Switching, UNESCO International Conference on Information Processing (Paris 1959) G 2.9.