comunicaciones técnicas

THE PS-2000 SIMD COMPUTER: FUNCTIONAL DESCRIPTION AND INSTRUCTION SET*

Adolfo Guzmán
J. Miguel Gerzso
Kemer B. Norkin**
Boris Kuprianov**

serie naranja: investigaciones

INSTITUTO DE INVESTIGACIONES
EN MATEMATICAS APLICADAS
Y EN SISTEMAS
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Apdo. Postal 20-726 Admón. No. 20
Delegación de Alvaro Obregón
01000 México, D.F.
548-54-65

IIMAS
UNAM

Artículo 54

comunicaciones técnicas

1982 Serie Naranja: Investigaciones No. 323

THE PS-2000 SIMD COMPUTER: FUNCTIONAL DESCRIPTION AND INSTRUCTION SET*

Adolfo Guzmán
J. Miguel Gerzso
Kemer B. Norkin**
Boris Kuprianov**

* Technical Report AHR-82-23

** Institute for Control Sciences,
Academy of Sciences, USSR.

Recibida: 22 de noviembre de 1982.

INSTITUTO DE INVESTIGACIONES
EN MATEMATICAS APLICADAS
Y EN SISTEMAS
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

APDO. POSTAL 20-726 ADMON. No. 20
DELEGACION DE ALVARO OBREGON
01000 MEXICO, D.F.
550-52-15 ext. 4559

IIMAS
UNAM

ACKNOWLEDGEMENTS.

A special word of gratitude to the IIMAS administration, its past director, Dr. Tomás Garza, its present director, Dr. Alejandro Velasco, who made their best effort to help this project.

This project benefitted from the interchange of visiting scientists between the Institute for Applied Mathematics and Systems (Mexico), and the Institute for Control Sciences (USSR Academy of Science, Moscow), as a consequence of an Agreement for Scientific Collaboration between the two countries. We are grateful to the people of CONACYT (The National Council for Science and Technology, Mexico), and the USSR Academy of Sciences, who made this agreement possible.

The indispensible support of the Directors fo the Institute for Control Sciences is gratefully appreciated.

The group representing Mexico would like to thank Boris Kuprianov for describing the PS-2000 in such great detail, and Prof. Norkin for translating the description from Russian into English so meticulously and tirelessly. And to Prof. Vilenkin for proof reading this report.

And finally, and in a certain sense, most importantly, we gratefully acknowledge the members of the AHR project, IIMAS-UNAM, Mexico, who brought the first AHR machine into existance, for without them, the project reported herein would not have been possible.

INTRODUCTION:

The writing of this report came as a consequence of applying AHR concepts of parallel programming (processing) to the PS-2000, a single decoder multi-processor computer (SIMD). This machine resembles the ILLIAC-IV computer in its architecture.

The basic idea of AHR (Arquitectura Heterarquica Reconfigurable) it that it is possible to implement a system which carries out parallel computation in such a way that the programmer is freed from the necessity of making his programs "parallelizable" as he writes his code. Instead, this responsibility is shifted to the hardware. In IIMAS, Mexico City, Mexico, the AHR project built and tested a first version of this type of hardware.

However, in the Soviet Union, the design of the PS-2000 hardware was not done as consequence of taking into account AHR ideas, for its main goal was to create a machine for numerical calculation. Despite this fact, the present project of implementing a parallel AHR type LISP on the PS-2000 promises to demonstrate that AHR ideas can be applied to SIMD computers.

The immediate goal of applying LISP to the PS-2000 was to understand the functional characteristics of this machine. The intermediate goal was to use this information for designing the parallel AHR type LISP on the PS-2000. This was done. (see report: AHR-82-24; "Functional Design of a LISP Interpreter for the PS-2000".) The long term goals are to:

1-document and publish,in English, information on the PS-2000, such as in this report, for those people who may be interested in this type of computer.

2-study the behaviour of this LISP implementation as it is used for projects in the Institute of Constrol Sciences, and elsewhere.

THE GENERAL ORGANIZATION OF THE PS-2000.

The PS-2000 is organized according to the single decoder multi-processor architecture. (Figure 1)

CU= special purpose processor with memory for data.

G= memory for microinstructions.

M=memory of 4K bytes (to 16K bytes, depending upon the configuration), 24 bits per word. This PEi memory is for data only.

H=memory for programs. One instruction in H provokes a set of microinstructions in G to be broadcasted sequentially to all PE's.

T=activation triggers (mask bits) which permits or disables the activation of PEi. In α (see Figure 1), we write the contents of each of the Ti triggers.

PROCESS INTERCONNECTION.

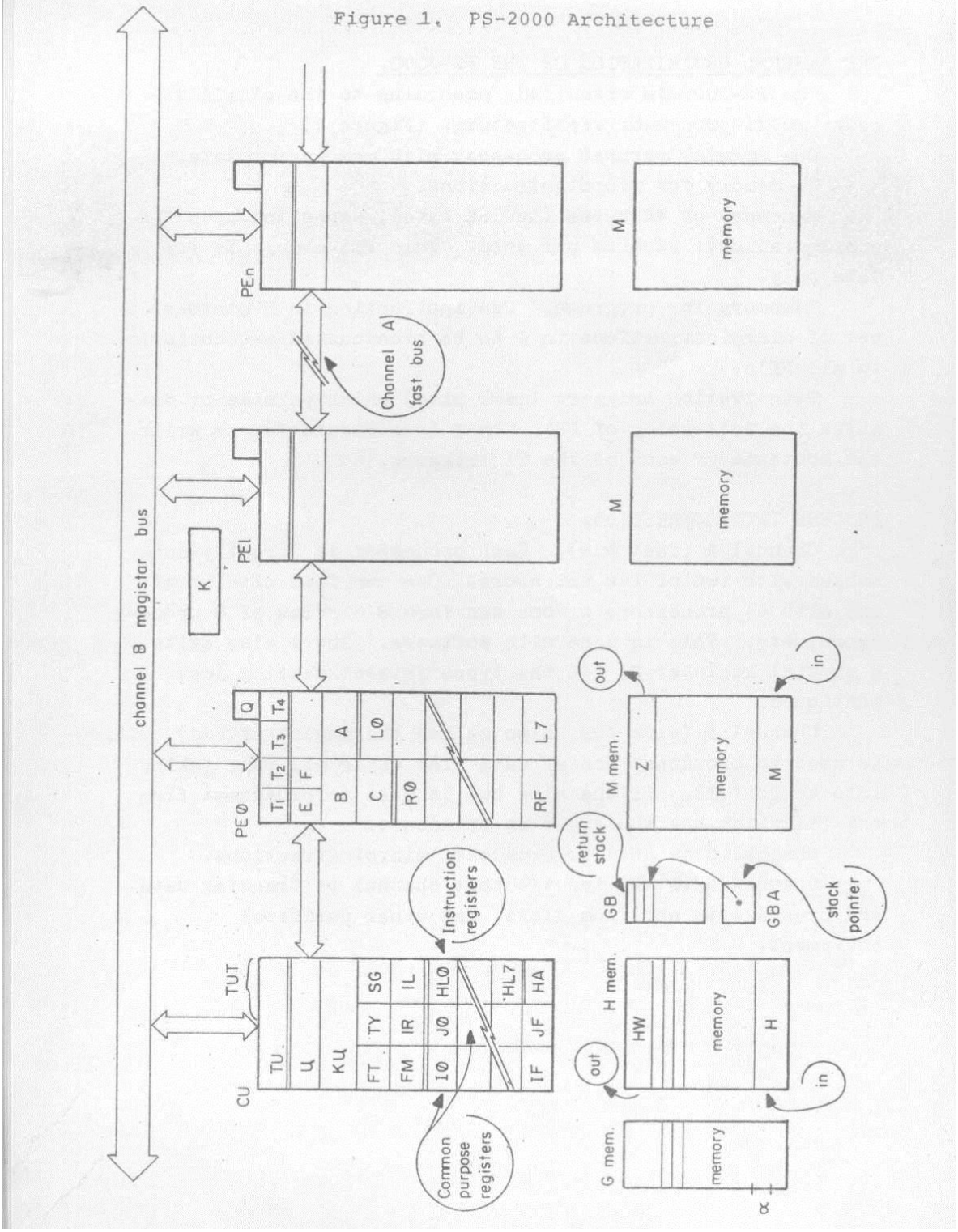
Channel A (fast bus). Each processor is directly connected with two of its neighbors. One can form circles of one with 64 processors or one can form 8 circles of 8 processors, etc. This is done with software. There also exits a special register SG for the types interconnection just mentioned.

Channel B (slow bus, also called the magistrar bus). This is used to broadcast scalar data from CU to all PEi, taking into account Ti. Or the slow bus is used to broadcast from any PEi which has the right to broadcast.

Channel C is used to broadcast microinstructions.

Channel D is the input/output channel to transfer data and programs to and from disks, and other periferal equipment.

Figure 1. PS-2000 Architecture



Other ways in which to carry out interprocessor connections:

TU is another activation trigger in CU (registers) which indicates (irrespective of Ti) which PEi are to be ignored.

When using Q, you can activate the following way:

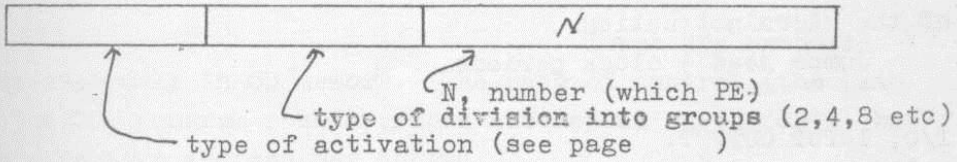
-Processor N
-Up to processor N.
-Processor after N without processor N.

There are additional ways to perform activation:

-Activate each j (for example, the third) PE in each group of 2,4,8,16, or 32 processors. (see page for details)
-Activate up to processor j in each group of 2,4, etc.
-Activate after each processor j in each group, etc.

Also, triggers Ti can be broadcasted from memory Mi.

DETAIL OF TU IN CU:



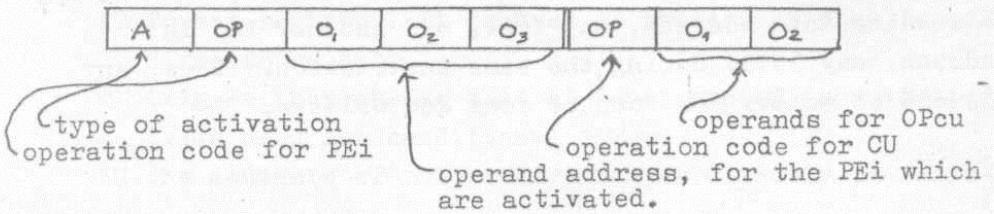
This is done dynamically in software.

Q, QUEUEING TRIGGER (1 bit in PE).

Activation holds only to the leftmost PEi which has the trigger on (in case there are several on). Then, perhaps, the instruction will change the status of the trigger and another PEj will be activated later.

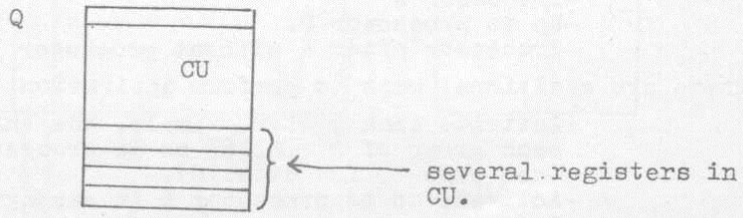
Some microinstructions will use the trigger Q, while other will ignore it.

MICROINSTRUCTION FORMAT, SIMPLIFIED DESCRIPTION, IN G.



Note: the exists microinstructions for going from one micro-instruction to another, that is, jump instructions.

DESCRIPTION OF SOME INSTRUCTIONS IN THE CONTROL UNIT (CU)



Each microinstruction demands 1 clock cycle. Each CU performs +,-,*, $\vee$, $\wedge$, with scalars which are stored in memory H. Scalars in G are not allowed. The operations involve the use of CU registers. They are executed in parallel with the micro-instructions affecting the PEi indicated in the left part of the microinstruction.

Jumps need 4 clock periods.

I/O, INPUT OUTPUT:

Two types of input/output instructions:

-those of CU.
-those of PEi.

In the firs design (initial), I/O of PEi was performed in parallel in all processors. However, in the present versions I/O in PEi may be ignored by using T or Q by some PEi.

There exists special additional I/O equipment which connects 1 (or 4) disks to the 64 PEi's, and thus, it appears that there are 64 disks.

The effective address in which I/O is done, is indicated by M field of OPpe, but it may be affected by the contents of one or some of the 8 registers of PEi. Then, one PEi may be reading into address, say 1024, and another PEj in address, say 3543, during the same instruction. Thus, any element of memory Mi can be read and written.

PROCESSING ELEMENT OPERATIONS

All of them are executed simultaneously in each PEi . The operations are:

+
-
+ addition complement 2's
+ * incremental step for addition and product.
*1 one step multiply
FMPY one step floating point multiply.

Other operations: -loading activity triggers
-writing to Ti, unconditionally, such as WRITE B into T
conditionally, such as if A 0, then WRITE B into T. This can be done in one instruction in parallel.
-broadcasting from PEi to its neighbor. Each PEi has two neighbors to which it can broadcast.
-accessing the slow channel.

There are symbolic addresses only in CU memory. The path of control (the path of the PC, program counter, does not depend on the data, but the effective address may depend.

REGISTERS IN THE CONTROL UNIT (CU)

The following list and description of registers should be done with reference to Figure 2.

U the universal register.

IO-IF 16 registers of 24 bits for common purpose.

JO-JF 16 registers of 12 bits for common purpose and for counters.

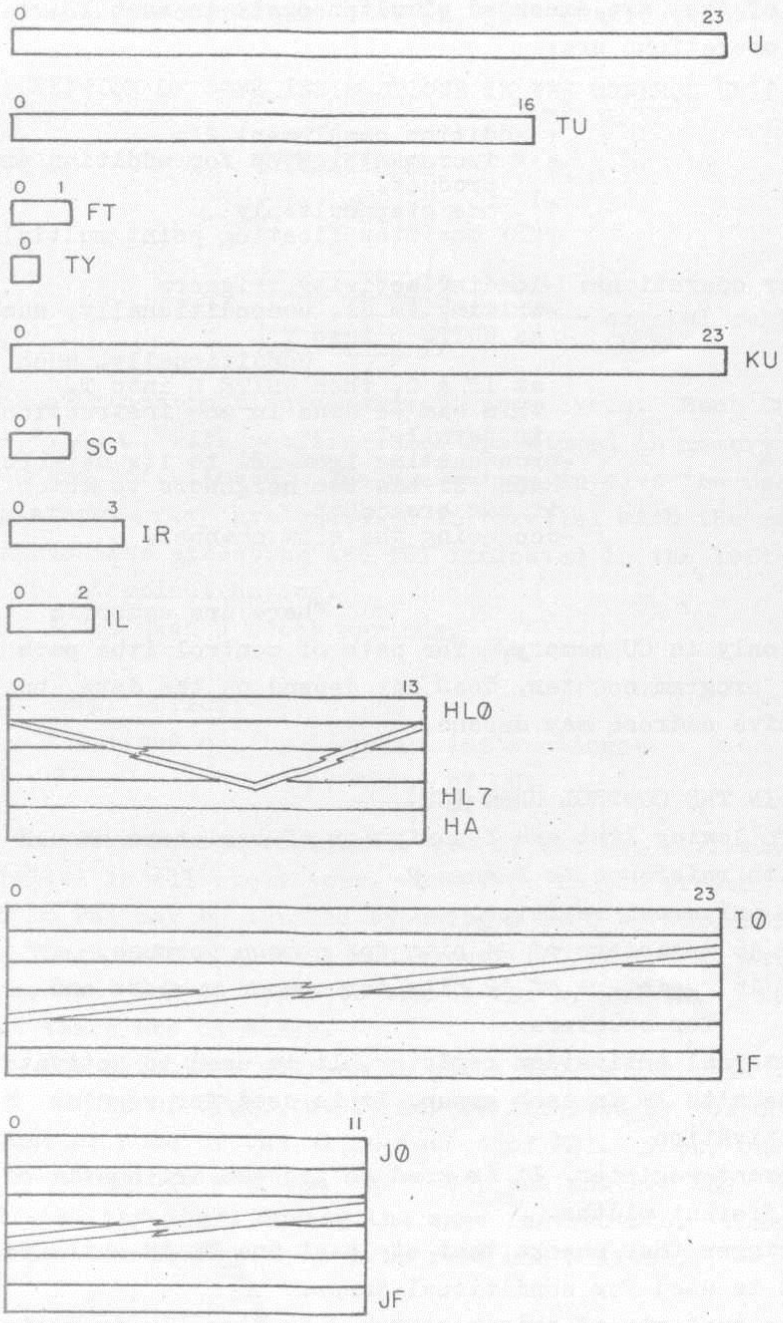
TU external activation register. It is used to activate the n'th PE in each group. It is used for regular activation.

FT format register. It is used to process arithmatic of different widths.

TY trigger that checks that at least one PE is activated. It is used for conditional jumps.

KU the contents of this register goes directly to regis-

Figure 2.


0
23
U
0
16
TU
0 1
FT
0
TY
0
23
KU
0 1
SG
0 3
IR
0 2
IL
0
13
HLØ
HL7
HA
0
23
IØ
IF
0
11
JØ
JF

ter K of PE's which are active.

SG segment register. It is used for dividing up the entire collection of processor into segments. Each segment must have a minimum of 8 members. Note: with TU, one can also divide the processors into groups of 2,4,8, etc. But there are groups that are not cyclic. The segmentation cuts the channels also, thus interconnecting the processors of each segment. This division only applies to the fast (channel A) bus, and the slow, magistral (channel B) bus.

FM masking for ignoring the overflow register of each PE.

H memory of 16 k bytes (now).

HW output buffer of the memory H. Memory contents upon extraction from H always goes to the HW register.

H register. It is the input buffer of the memory H and is used as HW, but for reading.

HA register for holding the address of a location in H memory. HA is based address.

HL0-HL7 registers for address arithmetic for control unit. It is used for different addressing modes.

GB register for the return address from the subroutines. This is the short stack.

GBA pointer to the top of the stack in GB. This is the stack pointer.

IR holds the number (address) of R0-RF registers in PEi. This is used in conjunction with indirect addressing registers of R0-RF. (4 bits).

IL same as IR, but for registers L0-L7 in PE. (3 bits).

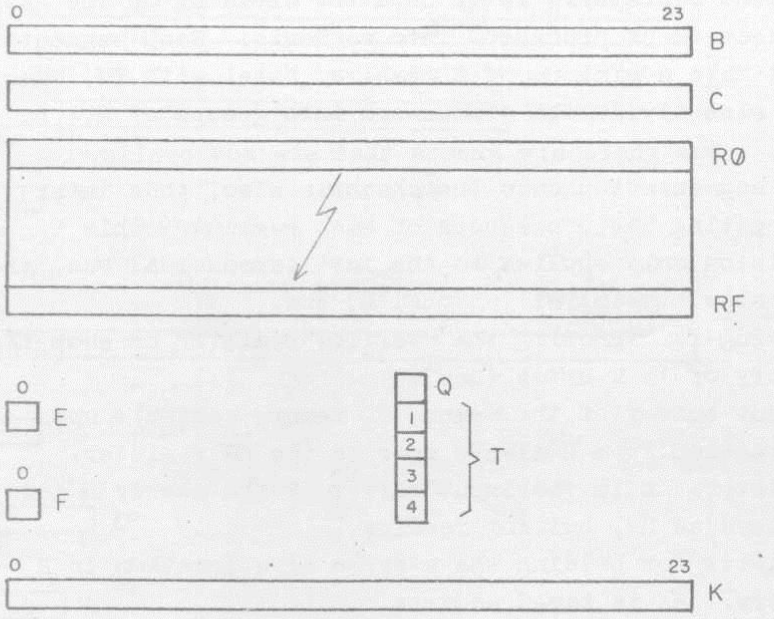
REGISTERS IN PROCESSING ELEMENTS (PE).

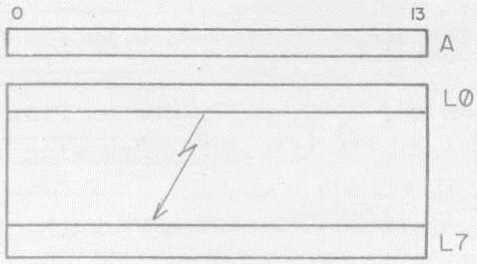
The following list and description of registers is understood easier with Figure 3.

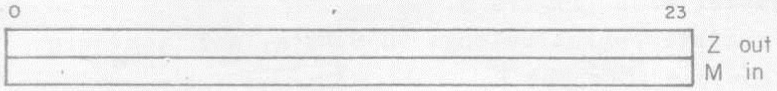
K is a register which belongs to 8 PE's, and controls the slow bus (Channel B).

B is a universal register. It allows the regular exchange of data between processors. It permits sending data to right and receive from the left processor, or opposite.

Figure 3.







C is also a universal register. One of the operands can be placed in this register, and the result will also be stored here immediately after the operation, like an accumulator.

R0-RF are 16 common registers for arithmetic and logic operations.

E is used for extended arithmetic operations, multiple precision.

F is an overflow register.

T is a register with four bits for activation. Each bit can be addressed seperately.

Q is the queueing register. The leftmost queue bit is the one that is 'obeyed' for triggering processors.

M is the memory of the processor (private memory). The maximum memory configuration will be 64 k words. The present configuration has only 4 or 16K words.

Z is a register associated with the memory M, and it is used to get data out of memory. (Output buffer)

M is the input buffer for memory.

L0-L7 are registers for address arithmetic, for different types of addressing modes.

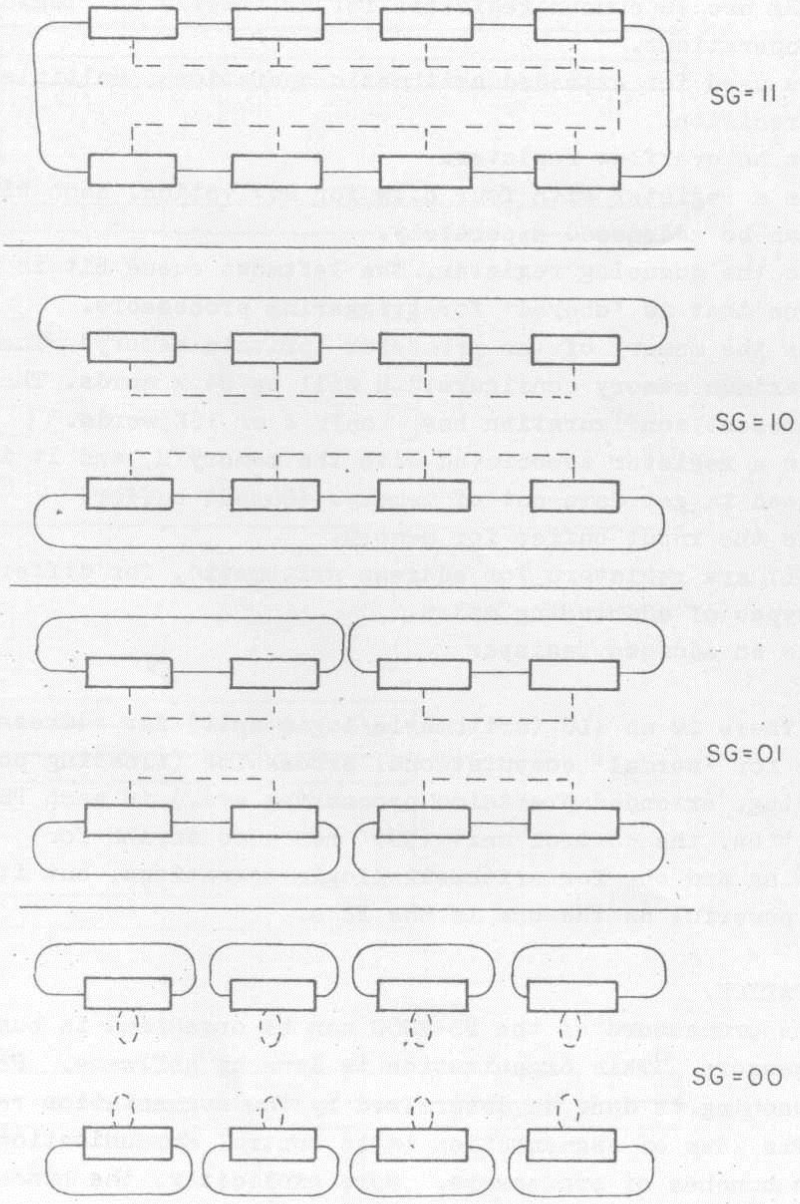
A is an address register

NOTE: There is an ALU (arithmetic/logic unit) for addressing and one for 'normal' computational processing (floating point processing, extended precision processing etc.) in each PE. In addition, the control unit (CU) has also an ALU for addressing and one for arithmetic/logic operations, but it is not as powerful as the one in the PE's.

SEGMENTATION.

The processors of the PS-2000 can be organized in bunches of processors. This organization is done by software. How this bunching is done is determined by the segmentation register. The idea of segmentation is to control communication between bunches of processors. More explicitly, the bunching

Figure 4.



is defined in terms of the following definitions:

segment- is $8x2^n$ processing elements (PE).

module - is 8 processors

group - are those processors which will be activated within a segment by using the TU register (bits 8:10) See page .

The maximum configuration is comprised of 8 modules, each of which have 8 processors. The configurations define "the neighbor to my left" and "my neighbor to my right".

NOTE: the fast channel permits bi-directional communication.

The meaning of the bit patterns in the SG register are seen in the following table.

SG bits	number of modules which are members of one segment	number of PE's which are members of one segment
00	1	8
01	2	16
10	4	32
11	8	64

The various configurations of the modules are described in Figure 4.

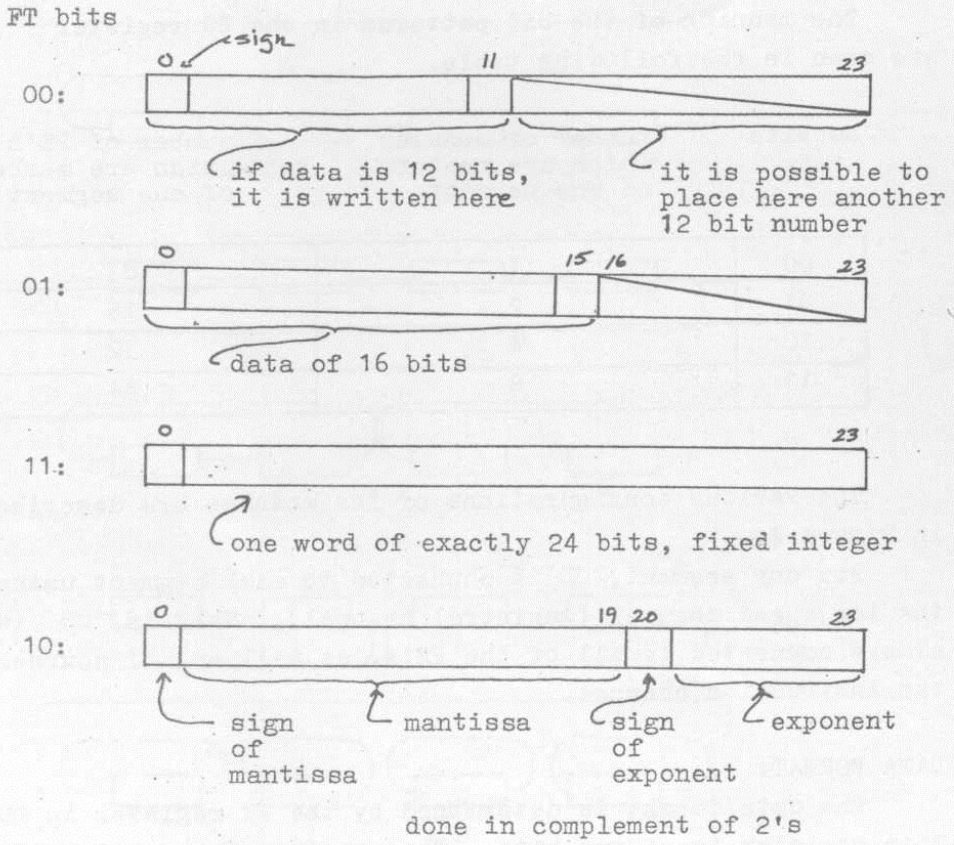
For any segment, CU is connected to each segment using the low speed channel (magistral channel). That is, CU is always connected to all of the PE's, as well as, of course, the instruction channel.

DATA FORMAT.

The data format is determined by the FT register in CU. This register is of two bits. The meaning of the bit configurations can be seen in the following table. Next page.

FT bits in CU	data format
00	12 bits, fixed point
01	16 bits, fixed point
10	24 bits, floating point
11	24 bits, fixed point

The above table has the following corresponding diagrams for making explicit the use of each bit.



done in complement of 2's

INSTRUCTIONS.

Notational conventions:

The alphabet is latin, plus the cyrilic letters for labels and comments.

The digits are 0,1,2....9

Special characters include +,-,/,(,)

The format of microcode instructions is:

label: field of PE $\S$ field of CU; comment

Notes on the format and microinstructions: 1- the field of PE is the computing field which may contain one or more microinstructions of the PE, which are performed in parrallel and which are joined by ",". 2-there is a special table which indicates which microinstructions can be performed in parallel. (see page 72). 3-the instructions in the field of CU can be one or more and are joined by ",". They are also performed in parallel to the ones being exxecuted by the PE's. 4-each instruction is member of either field, and one must be sure to make this clear in writing code, or that, given an instruction, the assembler can deduce the correct membership. This will be clarified on page . 5- any field can be absent. 6-the $\S$ marker can be absent if the PE field disappears. 7-if the CU field disappears, then the $\S$ marker can also disappear. But if the programmer puts it in, it is redundant because the assembler knows it is there or should be there. 8- the semicolon must always be present.

In the following description of the assembler of the PS-2000, the notational conventions are:

Underlining a string, such as, MUMBLE , means that it can be omitted, that is, it is optional.

If we write T, for register T, then it is understood to stand for T1, T2, T3, and T4.

Curly brackets indicate that several alternatives are allowable. In general, one should select exactly one.

{ }

To make the subsequent descriptions of instructions more compact, there are several subset of registers denoted by A1, A2,....A12. Inside the curly brackets are the names of the registers or memory locations.

A1= { C, Z, R∝I, A }

A2= { B, C, K, R∝I }

A3= { B C M } you can choose one or more of these registers

A4= { B C }

A5= { C Z T A }

A6= { C, K, R∝I }

A7= { I, U, HW, TU }

A8= { J, U, IL, IR }

A9= { K, FM, TU, TU1, SG, H, FT }

A11= { I, J, TU, SG, HA, FT, HW, FM, KU, IL, IR }

A12= { I, J }

∝ is any 1 digit base 16, that is, 0 to F.

Note: in microcode, there are three constants: 0, 1, -1.

0 means that 0 will be placed in each bit.

1 means that 1 is placed in the least significant bit, also refered to as the youngest bit.

-1 means that 1 is placed in each bit, that is, 2's complement.

All digits are represented in 2's complement.

$$A10=\begin{cases} \beta\beta\beta\beta\beta B & \text{direct address} \\ U & \text{jump to contents of U} \\ *\pm \delta & \text{current addr. } \pm \text{ decimal} \\ \text{label } \pm \underline{\delta} & \text{label } \pm \text{ some offset, dec.} \\ *\pm U & \text{current addr. } \pm \text{ contents of U register.} \end{cases}$$

β=0,1,...7 octal, ie., one digit base 8.

IMPLICIT INSTRUCTIONS.

Implicit notation in this assembler is a compact way of expressing operations. However, one must remember what the instruction means, since it is not obvious from the notation of implicit instructions themselves.

An instruction in implicit form for arithmetic and logical operations is:

$$SF\begin{Bmatrix} A \\ L \end{Bmatrix}\varphi(A3=\underline{A1},\underline{A2},\underline{\varepsilon})$$

φ=0...F is the hexadecimal number of necessary micro operand.

A is for arithmetic
L is for logical
S means that this instruction is performed by PE.
F permits the influ-influence of overflow control in each PE. That is, the PE must take into account the F bit

A1, and A2 are classes or sets of registers. The first and the second are optional (underlined).

$\varepsilon=\begin{Bmatrix} 1 \\ E \end{Bmatrix}$ means modification of operation code.

absent, no modification
1=modifies with value 1
E=modifies with reg. E.

Examples:

. SA2(A3=A1,A2,ε) means A3$\leftarrow$(A1 $\vee$!A2)+ε

! means not
✓ means "or"
+ means binary unsigned add with possible overflow

SA9(C=A,B) means C←A+B

+ in this case, it is 2's complement arithmetic

All implicit arithmetic and logical operations are performed under the control of the format register FT. If the FT register indicates floating point arithmetic or operands, then the operations are done on 20 digits of the mantissa. In addition, one must make sure by software that the exponents are equal in performing the operation. However, the 4 bits which represent the exponent will change their contents during the operation in an unpredictable way, thus forcing the programmer to store the values of these exponents someplace else.

All of these operations take 1 clock cycle.

There also exists macro instructions which expand into several microinstructions.

EXPLICIT INSTRUCTIONS, ARITHMETIC.

In contrast to the implicit instructions, explicit instructions indicate with greater clarity the meaning of the instruction. As in the last section, the SFA instruction is presented here. The general format is:

SFA (A3=A1 ω A2 ± $\underline{\varepsilon}$)

ω=means or refers to general operations, arithmetic

List of operations:

1	SFA(A3=A1)	means	A3←A1 — contents of A1 is placed in A3
2	SFA(A3=A1+E)		A3←A1+E — contents of A1 is placed in A3
3	SFA(A3=A1+1)		A3←A1+1
4	SFA(A3=2A1)		A3←2*A1 contents A3 receives 2 times contents of A1

5	SFA(A3=2A1+1)	means A3← 2*A1+1	
6	SFA(A3=2A1+E)	A3← 2*A1+E	
7	SFA(A3=A1-1)	A3← A1-1	
8	SFA(A3=A1-1+E)	A3← A1-1+E	
9	SFA(A3=-1)	A3← -1	Each digit is 1
10	SFA(A3=-1+E)	A3← -1+E	
11	SFA(A3=0)	A3← 0	
12	SFA(A3=A1∨A2)	A3← A1∨A2	
13	SFA(A3=A1∨A2+E)	A3← A1∨A2+E	
14	SFA(A3=A1∨A2+1)	A3← A1∨A2+1	
15	SFA(A3=A1∨!A2)	A3← A1∨¬A2	
16	SFA(A3=A1∨!A2+E)	A3← A1∨¬A2+E	
17	SFA(A3=A1∨!A2+1)	A3← A1∨¬A2+1	
18	SFA(A3=A1-A2-1)	A3← A1-A2-1	
19	SFA(A3=A1-A2-1+E)	A3← A1-A2-1+E	
20	SFA(A3=A1-A2)	A3← A1-A2	
21	SFA(A3=A1&!A2-1)	A3← A1&¬A2-1	
22	SFA(A3=A1&!A2-1+E)	A3← A1&¬A2-1+E	
23	SFA(A3=A1&!A2)	A3← A1&¬A2	
24	SFA(A3=A1+A2)	A3← A1+A2	
25	SFA(A3=A1+A2+E)	A3← A1+A2+E	
26	SFA(A3=A1+A2+1)	A3← A1+A2+1	
27	SFA(A3=A1&A2-1)	A3← A1&A2-1)	
28	SFA(A3=A1&A2-1+E)	A3← A1&A2-1+E	
29	SFA(A3=A1&A2)	A3← A1&A2	

Note: in the above instructions, the E register is set in the following situations:

1-in overflow (plus or minus)
2-in A2E and EA1, shift operations.

The contents of E is determined by the operation.
For the F register, the same happens as in the case of the E register, except that if 1 is sent to F, then F assumes the value 1 (its contents is set to 1), but if 0 is sent, then the contents of F is left unchanged.

For further information (and if you can read Russian) see page 53 of the manual, PS-2000, for the complete table.

EXPLICIT INSTRUCTIONS, LOGICAL.

There are the corresponding implicit logical operations, but they are not listed here.

The general form of the explicit operation is:

SL (A3=A1 ω A2)

Notice that the F register is always absent.

The list of operations are:

1	SL(A3=!A1)	means	A3←¬A1
2	SL(A3=A1)		A3←A1
3	SL(A3=A2)		A3←A2
4	SL(A3=!A2)		A3←¬A2
5	SL(A3=0)		A3←0
6	SL(A3=-1)		A3←-1
7	SL(A3=!!A1∨A2)		A3←¬[A1∨A2]
8	SL(A3=!A1&A2)		A3←¬A1&A2
9	SL(A3=!!A1&A2)		A3←¬[A1&A2]
10	SL(A3=A1#A2)		A3←A1#A2
11	SL(A3=A1&!A2)		A3←A1&¬A2
12	SL(A3=!A1∨A2)		A3←¬A1∨A2
13	SL(A3=!!A1#A2)		A3←¬[A1#A2]
14	SL(A3=A1&A2)		A3←A1&A2
15	SL(A3=A1∨!A2)		A3←A1∨¬A2
16	SL(A3=A1∨A2)		A3←A1∨A2

"not" the contents of A1 is sent or loaded into A3

$\omega = \{\vee, \&, \#\}$

\# is "or" addition, mod. 2, exclusive "or
!! is "not" but applied to entire result
! is "not" to immediate operand.

SHIFT OPERATIONS.

There are three types of shift operations: arithmetic, logical, and cyclic. They can be used with or without the E register. Note, the difference between the E and F register is that F (of the PE) can be used by the control unit and the E (of the PE) can only be used by the PE. The operation of shifting is influenced by the FT register: 12 bit shift, 16 bit shift, etc. The bits that are not included in the format are altered in an unpredictable manner during the shift operation. In the case that the FT register holds code for floating point number (10), only the mantissa is shifted. The exponent bits are altered unpredictably.

The general format of the shift operation is:

$$C \begin{Bmatrix} A \\ L \\ C \end{Bmatrix} \begin{Bmatrix} L \\ R \end{Bmatrix} (A3=A2)$$

C stands for shift
A for arithmetic
L for logical
C (second one) stands for circular.
L is left
R is right

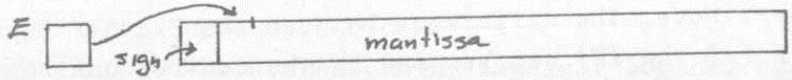
Notes: the left arithmetic shift does not change the contents of the sign bit, lsb (least significant bit) becomes 0. The right arithmetic shift propagates the sign bit. If the bit that disappears is not equal to sign bit, then the contents of the F register is set to 1. Otherwise, F register is not changed. The operation of shifting takes one microinstruction cycle, and only one bit is shifted at a time. In general, register E's contents are not altered and do not intervene.

In the case of the logical shifts, the sign bit is treated as any other bit. New bits are filled with zeroes. The shift operation is done one bit at a time, so if one wants to shift several bits at a time, one must write the shift operation codes the required times.

In the circular shift case, the sign goes to the lsb bit.

The three shift operations can use the E register to store an additional bit. In general, one uses the mnemonics

EA2, EA3, A2E, and A3E. In the arithmetic shift, the bit of the E register is moved as seen in the following diagram.



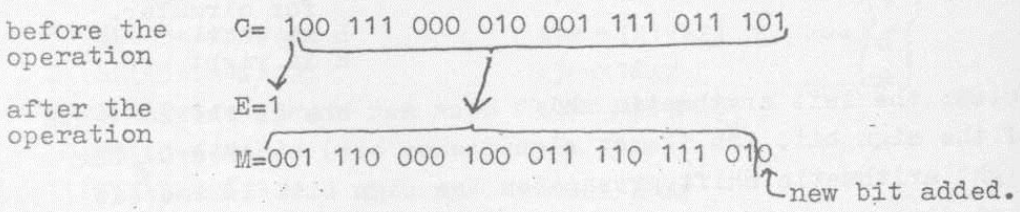
In the CLL(A3,EA2) case, the shift is done by using register EA2 (25 bits), but the result is as if one copied the contents of A2 (24 bits) to A3 (24 bits). One bit is "left" in E.

In all of the cases and examples in which A3, A2 and A1 are mentioned, one should always remember that they are sets of registers.

There are more examples involving shift operations in the manual on page 59.

Example,

Suppose that FT is equal to "11" (24 bit fixed point). We then perform CLL(EM=C) with the contents of the following registers thus:

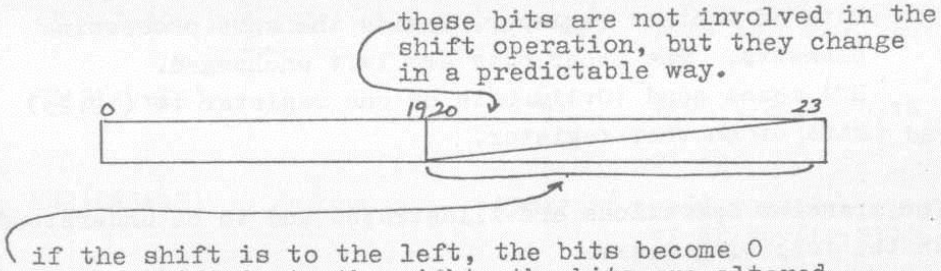


MULTIPLE (STEPS) SHIFT OPERATION.

This type of operation permits that several bits may be shifted in only one clock period. The general format of the instruction is:

$$CA \begin{Bmatrix} 4 \\ 8 \\ 12 \\ 16 \end{Bmatrix} \begin{Bmatrix} L \\ R \end{Bmatrix}$$

This is only arithmetic shift; L means "left" and R means "right". The numbers refer to the number of bits to be shifted. This operation only shifts the C register, and only the first 20 bits are involved in the shift. See figure on the next page.



OPERATIONS INSIDE PROCESSING ELEMENTS (PE's): TRANSFERENCE.

The instructions to perform transference all start with "P", which stands for the word, "perisilka" which means transference in Russian.

The FT register is ignored.

The "P" instructions are defined as:

$$P \begin{Bmatrix} 1 \\ M \\ C \\ H1 \\ H2 \\ H12 \\ H21 \end{Bmatrix} (\ \langle \text{a transfer operation} \rangle \ )$$

The meaning of the modification number or mnemonics are:

1 means send the 24 bits (0:23) bits

M means send the mantissa (0:19) bits

C means send the exponent (20:23) bits

H1 means send the lowest half (12:23) the youngest bits.

H2 means send the upper half (0:11) the oldest bits.

H12 means send (12:23) bits of one register to (0:11) bits of another register, inside the same processing elements. The other bits are left unchanged.
H21 means send (0:11) bits of one register to (12:23) bits of another register.

The transfer operations are illustrated and to be understood in the following cases.

	P1(A3=A2)	means	A3←A2	send all 24 bits from A2 to A3
2	P(A3=A1)	means	A3←A1	same as above, it does does not have a "1"
3	P(A3 R∝I=A5)		A3 ←┐A5 R∝I↵	send 24 bits to both A3 and R∝I . If "I" is included, then add the number in I to the number of the register R. For example, if we say R2I, and IR has 3, then we send to R5.
4	P(A3 H1R∝I=A5)		A3 ←┐A5 R∝I↵	send 24 bits of A5 to A3, and then the youngest half of A5 to R∝I. The oldest bits do not change.
5	P(A3 H2R∝I=A5)		A3 ←┐A5 R∝I↵	same as above, but the oldest bits are sent.
6	PM(A3=A2)		A3←A2	send the mantissa of A2 to the mantissa of A3. But if A3 is M (memory) then all bits are sent, Once you write into M, then data becomes inaccessible. M can only be written into. If one wants to read, then store into memory, then read from Z; and Z can only be read from.
7	PM(A3=A2,R∝I=A5)		A3←A2 R∝I←A5	same as above, but does two sets of registers. All operations done in one clock period. Remember, A3 is B C M .

8	PC(A3=A2, R∝I=A5)	means	A3← A2 R∝I←A5	same as above, but for exponent.
9	PH12(A3=A2)		A3←A2	send (12:23) bits of A2 to (0:11) bits of A3.
10	PH21(A3=A2)		A3←A2	send (0:11) bits of of A2 to (12:23) bits of A3.
11	P(Z=!M)		Z←¬M	send to Z the complement of M. It is the same as sending into M and then read this into Z.

ACTIVATION INSTRUCTIONS.

These instructions are in G memory, and use register TU in CU, and registers T in PEi. Some of these instructions are executed in PE, and some in CU. They are executed for one cycle. And, if one wants to have five instructions that satisfy the activation condition, one must write five activation condition, one for each instruction. That is, one can only write one activation instruction per microinstruction.

The general form of activation instruction is:

/ω

The several cases for ω are:

1 /ω T1234 where

ω = {
- ! inversion (0→1, 1→0)
- V disjunction
- & conjunction (also written as ·)
- N addition modulo 2, exclusive "or"
- !N not(exclusive "or")
- !& not"and"
- !V not "or" (first "or" each Ti, then not

example: /&T24 means that process elements will be activated when processing element T2 and T4 are both set to 1.

2 /FCΘO where

Θ = { < <= = ≠ >= > }

The purpose of this activation instruction is to match the contents of the C register with 0 in PE. The length of the matching is determined by FT. If FT is "10" (floating point), then only the mantissa is matched. If in the activation instruction, one writes F, then the PEi becomes activated when the contents of the register F is 1, or if C⊛0 is satisfied. If one does not write F, then only the condition C⊛0 is tested.

3 /F means to select and activate those PEi which have overflow, F set to 1.

4 /Q In this mode, one activates only one PEi per segment which a Q register is set to 1. But only that Q which is the left most register. For example, if the Q registers are set:

```
0  0  0  1  0  1  1  1  0  1
Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10  etc.
```

The processing element with Q4 will be triggered first, then Q6, etc.

5 /QF This is the same as above, but the PE will be activated if the left most Q register is set, but after the PE is triggered, the Q register is turned off, that is set to 0. For example,

```
before   0  0  0  1  0  1  1  1  0  1
         Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 etc.
                  ↓
after    0  0  0  0  0  1  1  1  0  1
         Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10
```

6 /TUF means to activate PEi according to the contents of TU, which is in CU. The register TU is used as follows:

Bits

0:1 activation mode
2:7 not used
8:10 number of PE in group (ie. how many PE's in a group)
11:16 the number of the PE in the group, (ie. which PE in the group)

These last bits correspond to the portion in TU refered to as TU1. In transfer intructions, one can send bits to TU1, but one cannot write an activation instruction such as /TU1. TU1 appears as part of the set of registers of A9.

Note about groups. A group, as mentioned before in a different way, is not the same as a segment. One divides the computer into segments, and the segments into groups. The number of elements PE in a group is a power of 2. The number of elements in a group must be less than or equal to the number of elements in a segment. If one makes the mistake of including more elements in a group, the system will ignore it, and it will make or include the maximum number of elements in a group allowable.

The following table is for bits 0:1 of TU. It should not be confused with the table of bits for segmentation.

bits 0:1 (activation mode in a group)

00	in each group, activate PEi
01	all processor except PEi
10	all processor up to and including PEi
11	all processor after PEi

bits 8:10 (how many PE's in a group)

000	64 PE's in a group
001	32 PE's in a group
010	16
011	8
100	4
101	2

Again, it is impossible to have groups which have more elements than in a segment. If one designates more elements in a group than in a segment, then the system will make the size of the group equal to the size of the segment.

If F is written in the TU activation instruction (last page), then the number of the PEi in bits 11:16 will be increased by 1. (IMPORTANT: this F has nothing to do with the register F.) It is the programmers responsibility to make sure that new PEi number after the increment operation is less than or equal to the size of the segment. Otherwise, the system will report an error.

If no instruction of the form /Z exists, then all of the PE's will execute the microinstruction.

SENDING DATA TO TRIGGER REGISTERS.

All of these instructions do not depend of the format register FT.

1 E= {1 0} means E←0, E←1 — place 0 into E. The previous value is lost. E is also is set after addition, previous value is lost

2 F= {1 0} means F←0, F←1 — F is also set after overflow in addition. F holds previous value if no overflow. Thus, if one set F=1, and there no overflow, then it remains 1.

3 T1234={TU, Q, CC, F, FC⊕0} — TU has 17 bits in CU. This instruction is executed by all PE's, and 0 or 1 is stored in the registers to the left of the equal sign, depending upon whether the rules of interpretation of TU render this processor inactive (0) or active (1).

Cases (or example):

T13=TU means that (according to the rules of interpretation of TU) 0 or 1 will be set in T1 and T3 of each PE.

T24=Q means send the contents of the Q bit to registers T2 and T4.

T12=C means send the (20:21) to T1 and T2 respectively of register C. If one writes T1234=C, then bits (20:23) are sent from C to registers T1...T4.

T34=F means send the contents of F bit to T3 and T4.

T1234=FC⊕0 means one is comparing (matching) the contents

of register C. Θ can be $<, \leq, =, \neq, \geq, >$. For example, T12=C>0/&T12. This means that we write 1's in T1 ant T2 in the case that C>0, and that T1 and T2 are active (that they already have 1) . However, 0's will be written in T1 and T2, which might have 1's, if C$\leq$ 0. In addition, it will leave T1 and T2 unchanged for those PE's which have their number (11:16) of TU are not simultaneously active.

SENDING DATA TO REGISTER Q TRIGGER.

The general form of the instruction is:

Q/τ

It places a 1 into the Q's in each PE where is satisfied. If one writes only Q with no conditional, then 1's will be written in all PE's.

Example: Q/TU means write 1's in those PE's which would be active according to the rules of interpretation of TU, and in other Q's, write 0's.

SENDING DATA TO REGISTER TY.

The general form of this instruction is:

TY/τ

Note: the use of TY is that it is set to 1 if there is at least one active processing unit; otherwise it is set to 0. In the case of the above mentioned instruction, TY is set to 1 depending upon -at this moment- there is or there is not some PE obeying τ .

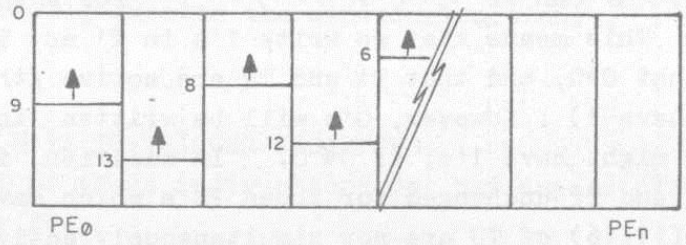
PROGRAMMING EXAMPLE.

The objective of this example is to show how to fill a matrix with 0 up to a given index level. The levels are stored in a vector V and it contains:

$V=[9 \quad 13 \quad 8 \quad 12 \quad 6]$

This vector is prestored in the machine with 9 in the first PE, 13 in the second PE, 8 in the third, etc. The filling of 0's is done from location of the index down toward location 0.

To illustrate what we want to do, see the following diagram.



The program is as follows: (code is in syntactic "free form")

```
        C=1          C is the current index, set to 1
        C=V-C        in all PE's. Begin with V-1 and
                     ending with 0. In reality, V is
                     stored inside a register.
L:      T1=C>=0      Set T1=1 in those PE's where C
                     is not yet negative.
        M(C)=0/T1    Store 0 in those PE's which are
                     active. This instruction is not
                     strictly correct syntactically.
        C=C-1        Decrement the index.
        TY/C >=0     Execute this instruction to store
                     1 in TY if there is at least one
                     PE still active; and C>=0.
        GOTO L/TY    Go to label L if there is at least
                     one active PE, else continue.
```

SENDING DATA OVER THE FAST CHANNEL (CHANNEL A).

This instruction, (described below) does not use or does not pay any attention to register FT.

The general format of the instruction is:

$B(\pm\delta)$ where + means send to the right
- means send to the left.
δ is any decimal number from 1 to 32.

If one writes B(+8), the translator (assembler) will produce eight B(+1), each one will take 1 clock period. The actual effect of this instruction is to shift the contenst of register B in one PE to another register B in the neighboring PE. It is important to note that to which PE (that is which PE is considered the neighbor to the left and neighbor to the right) is determined by the SG register. But grouping only influences activation of PE's, and in this case, it is not taken into account.

It is also possible to use:

$$B(\pm\delta)/\tau$$

In this case, data is sent out of all PE's, whether they are active or not, but the writing into the register B is done only in active processes. The inactive PE's have registers B whose contents remain unchanged.

Example, given the instruction B(+1)/T1, the following will happen. The contents of the B register is shifted to the right if the T1 bit of the neighboring processor is set to 1.

number of PE		1	2	3	4	5	6	7	8	etc.
contents	T1	0	1	0	1	1	1	0	0	
contents	B	11	47	567	8	35	666	88	69	BEFORE
contents	B	11	11	567	567	8	35	88	69	AFTER

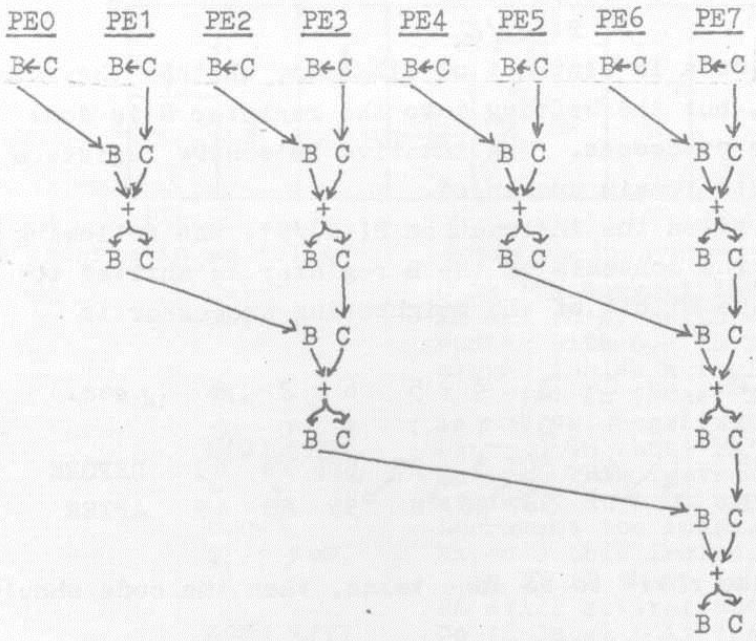
If one wants the shift to be done twice, then the code should be

```
B(+1)/T1
B(+1)/T1
```

Example: performing an add using eight PE's and shift to the neighbor operation or instruction. The goal is to have the sum of Ci, the contents of eight numbers of C which have be loaded in each PE. This sequence of operations for adding requires seven clock periods. The code is as follows:

```
P(B=C);          Copy C into B, in parallel
B(+1);           Send B if PEi to right
                 neighbor
SA(B,C=C+B);     Add C to B and store in
                 both C and B
B(+2);           Send to the right neighbor
                 two times.
SA(B,C=C+B);     Do the addition again.
B(+4);           Send to the right neighbor
                 four times.
SA(C=C+B);       Do the addition for the
                 last time. Finished.
```

The result of the program is in the register C of the seventh processor; and by symetry, the result will appear in all of the C registers of all PE's. The diagram of the operation appears on the next page.



THE USE OF LOW SPEED CHANNEL (MAGISTRAR OR CHANNEL B).

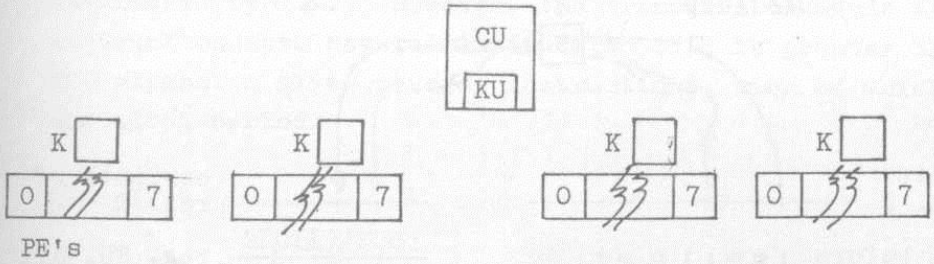
The instructions for accessing this channel uses KU in CU, and K associated with the PE's. Note: there are eight PE's for one register K.

The general syntax of the instruction is:

$$KU=B/\left\{\begin{matrix}QF\\TUF\end{matrix}\right\}$$

This instruction sends to the register KU the contents of EXACTLY one register B in one PEi. The way in which the PEi is selected is done using the Q register, OR by using the register TU. In this last case, (because of the rules of interpretation of TU) more than one PE is selected, then the system indicates an error.

Diagram of the relationship between the KU register and the K registers for each 8 PE's is seen on the next page.



In executing the instruction, the contents of register B is sent to the corresponding register K, then after that, the contents of register K is sent to register KU in CU. So, again, it is important to take into consideration that only one PE must be selected.

The register FT, for format, is ignore.

If F is present in the instruction, while using the Q register, then the Q bit (left most processor) is set to 0, making the corresponding PE passive.

If F is present in the instruction which uses TU, then the number of the PE (which one) in (11:16) bits is increased by 1, to permit the access of the next PE. Note again, if the programmer does not initialize the number correctly, then upon increment of the number, the system will indicate an error.

If one wants to send an array, it must be done word by word. Parallel copying of data is good for replication, and the appropriate instructions for this task will be seen later.

ANOTHER INSTRUCTION FOR CHANNEL B, MAGISTRAL CHANNEL.

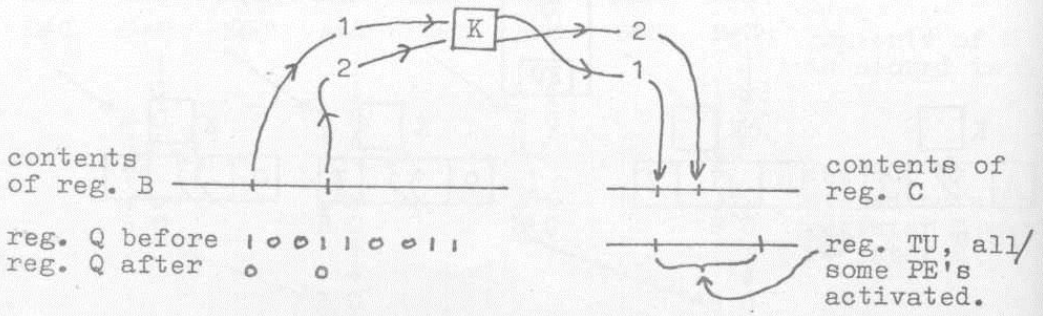
The general format of this instruction is:

$$BKC(\delta)/ \begin{Bmatrix} QTU \\ TUQ \end{Bmatrix}$$

The register FT is not taken into account, all of the bits are sent.

Example: (cases)

BKC(δ)/QTU

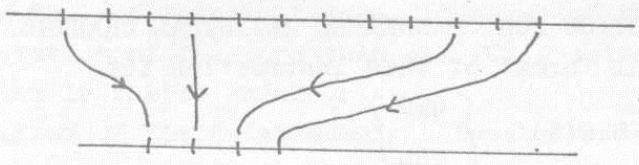


The processes for transfering data is done in the following way:

1st step- data is sent from the register B of the PE which has the first left most Q bit set, to register K. Then, the transfered data into K is sent to register C. The PE which receives the data into C is selected according to the rules of interpretation of TU. TU should be in the mode which only one PE is activated. This is done by setting (0:1) bits to "00".

2nd step- after the first step, Q register is set to 0, and the number of the PE in TU (which one) (11:16) bits are incremented by 1. Now, the instruction is executed again, but the transfer is done from the next PE which has the left most Q bit set to 1, and the next PE number is stored in TU.

The net effect of executing this instruction several times is to concentrate data that is widely disperesed in various PE's.



The number of times that the instruction is executed is determined by δ. However, the translator expands this instruction into several instruction if δ is greater than 1. The expansion gives several instructions, each of which take one clock period.

Example:

BKC(δ)/TUQ

This instruction does the opposite as the previous instruction in that in is useful for dispersing concentrating data.

However, in contrast to the previous instruction, B is controlled or the B of PEi is activated by the rules of interpretation of TU. And C is selected by register Q.

The effect of executing this instruction is to first move data from B register to register K, and then from K register to C register.

NOTE:

If, for some reason, a segment is made up of 2 modules of eight processors, only one of the two K registers is selected and used in the transfer process. The other K register is ignored.

ACCESS TO PE's MEMORY.

The memory of each PE is private memory.

The syntax of this instruction is:

$$M\phi\{\mathcal{X}\};$$

$$\phi = \begin{cases} R & \text{read} \\ W & \text{write} \end{cases}$$

$$\mathcal{X} = \begin{cases} U & \text{in CU processor} \\ A & \text{in PE} \\ L\beta\underline{I} & \text{in PE} \\ A+\overline{1} & \text{in PE} \\ L\beta\underline{I}+1 & \text{in PE} \end{cases}$$

β is an octal number.

Note: one can index $L\beta$ by using IL. One can also send data to the address stored in A, and after the operation, increment A. For example: MR(A+1) means read from memory at the location stored in register A. After this reading, then increment

the contents (the address) by 1.

The meaning of L∝I is: use address that is stored in the register L5 , which register number is 5. But, in the code of the program, we write L3I, but we have stored in register IL (the "I" in the instruction is short for "IL") the value 2. Upon the excution of the instruction, 2 in IL is added to 3 in the instruction L3I.

Notes: 1-it is not allowed to write two M instructions, one right after the other. The reason being that memory requires 2 machine cycles, each one being 300 nanoseconds.

2- It is possible to write:

P(B=Z) § MR(U) for example

and the net effect is that the two instructions will be executed in parallel. That is, old data from register Z to register B, and at the same time, the contents of memory location indicated by the address in register U is transfered into Z. As a consequence, Z contains a new value.

3-It is not possible to invert the order of the instruction above and write:

MR(U) § P(B=Z)

The common rule for connecting instructions is that the instruction of the PE should be written first, and then the instruction for the control unit last.

4-The is a way to write "M" instructions using the U register that is different than described above. The following is also permitted:

MUR is the same as MR(U)
MUW is the same as MW(U)

5-If one wants to connect (join) two instructions for PE's, or two for CU, they have to be separated by a comma, ",". However, if one wants to connect one PE instruction and one CU instruction, they have to be separated by "§" . But the instruction for the PE must go first. The symbol "§" also indicates that the instruction on either side of it will be executed in parallel.

6-Restriction: If one wants to write into memory, one must load register M at least one instruction before performing

the operation of writing into the memory. That is, the writing into M register and writing into M memory can not be performed in parallel. The explanation for this restriction is that each instruction is decoded in CU. The control unit is the only place in which instructions are decoded. For example, if one decodes:

$$P(A=Z)\,\S\,MW(U)$$

then the first instruction will be performed by the PE's and the MW instruction will be performed by memory. This part does work in parallel. However, if one writes:

$$P(M=Z)\,\S\,MW(U)$$

then this violates the restriction because one is copying into register M the contents of register Z, and at the same time (parallel), one is writing into memory the contents of register M at the address stored in register U.
7-Another restriction: On cannot write into register M immediately after performing an MR or MW instruction. It is necessary to execute some other instruction in between.
8-In using the MR and MW instructions, the FT register is not used.

INSTRUCTIONS TO FILL REGISTERS A AND Lβ. ADDRESS REGISTERS

The FT register for these instructions is not used. The instructions have the following general formats:

$$A=\begin{Bmatrix} U \\ O \\ C \\ L\beta\underline{I} \end{Bmatrix} \qquad A+\begin{Bmatrix} 1 \\ U \\ L\beta\underline{I} \end{Bmatrix}$$

$$L\beta\underline{I}+1 \qquad L\beta\underline{I}=\begin{Bmatrix} A \\ O \end{Bmatrix}$$

β is an octal number

In the case that one writes "A+ MUMBLE", means that "A=A+ MUMBLE". The "A=.." is omitted in the spirit of optimization. But in the case that we are adding A register and LβI register, such as "A+LβI", means that both "LβI=A+LβI" and "A=A+L I". The specific cases and their meaning are seen in the following list: (next page)

1	A=U	means	A←U	contents of reg. U is sent to reg. A.
2	A=0		A←0	
3	A=C		A←C	
4	A=L/ØI		A←L/ØI	
5	A+1		A←A+1	
6	A+U		A←A+U	
7	A+L/ØI		A←A+L/ØI L/ØI←	
8	L/ØI+1		L/ØI←L/ØI+1	
9	L/ØI=A		L/ØI←A	
10	L/ØI=0		L/ØI←0	

These instructions are found on page 169 of the PS-2000 manual.

All of these instructions can have the /6 feature, that is, they can be conditionally activated or executed.

INSTRUCTIONS OF ACCESSING H MEMORY OF THE CU.

The H memory is the one that contains the so-called macro instructions, that is, the pointers to bunches of microinstructions in G memory.

The FT register is ignored for these instructions.

The general format of these instructions is:

$$H\phi\ (\mathcal{X}1);$$

$$\phi = \left\{ \begin{matrix} R \\ W \end{matrix} \right\} \qquad \mathcal{X}1 = \left\{ \begin{matrix} U \\ HA \\ HL \\ HA+1 \\ HL+1 \end{matrix} \right\} \text{ in CU processor}$$

Note: indexed addressing is not possible.

These instructions are performed in three clock periods. This means that two or more H memory instructions cannot be written or coded one after the other. Therefore, one must write two other instructions in between two H memory instructions. This is true if the memory is of 16 k of size. However, if it is 4 k, then one must write only one other instruction between two H memory instructions.

The same restrictions to H memory with respect to the

input register H, and the output register HW hold as those that apply the input/output buffers of M memory, of the PE's.

As in the case of M memory instructions, one can write the following H memory instructions thus:

HUR is the same as HR(U)
HUW is the same as HW(U)

INSTRUCTIONS FOR ADDRESS ARITHMETIC IN CU. (ADDRESS REGISTERS)

These instructions are related to those instructions of address registers for M memory, in that they perform similar functions.

The general formats are as follows:

$$HA = \begin{Bmatrix} U \\ 0 \\ HL\beta \end{Bmatrix} \qquad HA + \begin{Bmatrix} 1 \\ U \\ HL\beta \end{Bmatrix} \qquad HL\beta = 0$$

β is an octal number.

As in the case of the instructions of M memory, the following notational shortcuts hold: in the case of "HA+ MUMBLE", except for $HL\beta$, is meant to be understood as "HA=HA+ MUMBLE", If "HA+$HL\beta$" is written, then it is the same as "HA=HA+$HL\beta$", and "HL =HA+$HL\beta$". Specific examples of instructions are:

1	HA=U	means	HA←U	the contents of register U is loaded into reg. HA
2	HA=0		HA←0	
3	HA=$HL\beta$		HA←$HL\beta$	
4	HA+1		HA←HA+1	
5	HA+U		HA←HA+U	
6	HA+$HL\beta$		HA←HA+$HL\beta$ $HL\alpha$	
7	$HL\beta$=0			

INSTRUCTIONS FOR ARITHMETIC/LOGIC UNIT OF CU.

These are the same as the operations for the ALU of the PE's.

The FT register is ignored. The operations are performed on all 24 bits, without taking into account the overflow.

Negative numbers are represented as 2's complement. The arithmetic and logic operations take one clock cycle.

The format is: (implicit type instruction).

$$W \begin{Bmatrix} A \\ L \end{Bmatrix} \varphi(U=\underline{A7},\underline{A8},\underline{1});$$

W refers to ALU in CU.
A arithmetic
L logic
φ is a hexadecimal digit.

This implicit representation is probably useful for program- who have a great deal of experience in using the PS-2000, because they can remember what the instruction stands for. However, for readability and maintaince of code, it may be counter productive.

For implicit intructions, there is a set of explicit instructions, which correspond to the implicit ones exactly. The format is: (explicit instructions)

$$W \begin{Bmatrix} A \\ L \end{Bmatrix} (U=\underline{A7}\ \omega\ \underline{A8}\ \underline{+1})$$

Of the sets of registers and number that are underlined (optional), at least one must be present. The list of cases (examples) is:

1	WA(U=A7)	means	U←A7	load the contents of A7 register into U
2	WA(U=A7+1)		U←A7+1	
3	WA(U=2A7)		U←2*A7	load two times the value in register type A7 into U.
4	WA(U=2A7+1)		U←2*A7+1	
5	WA(U=-1)		U←-1	
6	WA(U=0)		U←0	load immediate constant zero into register U.
7	WA(U=A7-1)		U←A7-1	
8	WA(U=A7)		U←A7	
9	WA(U=A7∨A8)		U←A7∨A8	
10	WA(U=A7∨A8+1)		U←A7∨A8+1	
11	WA(U=A7∨¬A8)		U←A7∨¬A8	load into reg. U the result of reg. type A7 "ored" with not A8 type.
12	WA(U=A7∨¬A8+1)		U←A7∨¬A8+1	

13	WA(U=A7-A8-1)	means	U← A7-A8-1
14	WA(U=A7-A8)		U← A7-A8
15	WA(U=A7&!A8-1)		U← A7&¬A8-1
16	WA(U=A7&!A8)		U← A7&¬A8
17	WA(U=A7+A8)		U← A7+A8
18	WA(U=A7+A8+1)		U← A7+A8+1
19	WA(U=A7&A8-1)		U← A7&A8-1
20	WA(U=A7&A8)		U← A7&A8

LOGICAL OPERATIONS FOR PROCESSOR CU.

The list of logical operations is:

1	WL(U=!A7)	means	U← ¬A7
2	WL(U=A7)		U← A7
3	WL(U=A8)		U← A8
4	WL(U=!A8)		U← ¬A8
5	WL(U=0)		U← 0
6	WL(U=1)		U← 1
7	WL(U=!!A7∨A8)		U← ¬[A7 ∨ A8]
8	WL(U=!A7&A8)		U← ¬A7&A8

SHIFT INSTRUCTIONS IN THE ALU OF CU.

The form of the instruction is:

$$U \begin{Bmatrix} A \\ L \\ C \end{Bmatrix} \begin{Bmatrix} L \\ R \end{Bmatrix}$$

A arithmetic
L logical
C circular
L (second one) left
R right

In using the circular option of this shift operation, the E register is not used since it does not exist in the CU processor.

SENDING DATA TO THE IR AND IL REGISTER.

Remember that the IR register is of 4 bits and the IL register is 3 bits. The several instances of these instructions are:

1	IR=U	means	IR← U(20:23)	the contents of bits 20:23 of register U are sent to reg. IR

2	IR+1	means	IR←IR+1	
3	IR-1		IR←IR-1	
4	IL=U		IL←U(21:23)	
5	IL+1		IL←IL+1	done module 8
6	IL-1		IL←IL-1	

INSTRUCTIONS TO SEND DATA TO I, J REGISTER GROUPS IN CU.

The register group I has 24 bits, and register group has 12 bits. The general form is:

$$I\alpha, J\alpha = UC$$

Cases:

1	Iα=U	means	Iα←U	the contents of reg. U is sent to reg. I .
2	Iα=UC		Iα←U	the contents of reg. U is shifted to the right 12 bits (circular option) U reg. remains unchanged. Useful for storing two 12 bit numbers.
3	Jα=U		Jα ←U(12:23)	bits (12:23) of reg. U is sent to reg. J .
4	Jα=UC		Jα ←U(0:11)	sends all 24 bits.
5	Iα, Jα=U		Iα ← U(0:23) Jα ←U(12:23)	sends all 24 bits. sends the youngest bits. the least significant bits.

INSTRUCTIONS FOR DATA EXCHANGE BETWEEN ADDRESS REGISTERS OF CU.

The general format of this instruction is:

U=A11

where A11= {I J TU SG HA FT HW FM KU IL IR}

Note: if the source of the bit pattern has less than 24 bits, the bits which are not sent from the source register are set to 0 in register U.

Cases:

1	U=Iα	means	U←Iα	all 24 bits are received.
2	U=Jα		U(12:23)← Jα U(0:11) ← 0	

3	U=TU	means	U(7:23) TU U(0:6) 0	TU register has 17 bits
4	U=IL		U(21:23) IL U(0:20) 0	IL has 3 bits.
5	U=IR		U(20:23) IR U(0:19) 0	IR has 4 bits.
6	U=KU		U KU	KU has 24 bits.
7	U=SG		U(14:15) SG U(0:13) 0 U(16:23) 0	SG has 2 bits.
8	U=FM		U(10) FM U(0:9) 0 U(11:23) 0	FM has 1 bit.
9	U=HW		U HW	HW has 24 bits, it is output of memory.
10	U=FT		U(22:23) FT U(0:21) 0	FT has 2 bits.
11	U=HA		U(10:23) HA U(0:9) 0	HA has 14 bits

INSTRUCTIONS TO LOAD CONSTANTS INTO REGISTER U.

Note: constants can be written in the microinstructions themselves, instead of loading them into areas of memory which are seperate from the code.

The format is:

Y(U=η)

where η = { βB ; *± βB ; *± δ ; <label> ± { βB ; δ } }

β octal digits

δ decimal digits

* means "where the program counter (PC) is".

Decimal numbers range from 0 to $2^{23}-1$. Only positive decimal numbers are allowed.

INSTRUCTIONS FOR SENDING FROM U TO OTHER REGISTERS.

There are two equivalent forms:

Y(A9=U) or A9=U

Each form is used in particular cases. The table which

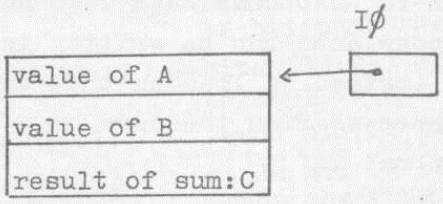
indicates how to join instructions for execution in parallel also indicates which of the above mentioned forms are applied.*

Programming example: how to write 5 in H memory location 100 octal.

```
Y(U=5);          load 5 into reg. U
Y(H=U);          load contents reg. U into
                 memory buffer H, input
Y(U=100B);       load memory address into U.
HUW;             perform write operation
                 into memory using address
                 in reg. U
```

On cannot write a constant directly into H memory due to the fact that it is difficult to relocate.

Another programming example: how to add the contents of one location in memory to the contents of another location in memory. The operating system O.S. reports -for example- that the first free word in H memory assigned to our program is stored in register IO. The situation is seen in the following diagram:



What needs to be done is to add the contents of location A to the contents of location B, and leave the result in location C.

```
U=IO;              get the address of A
HUR;               read memory H at location in U
NOP;               need to wait two cycles for
NOP;               memory to do its thing
U=HW;              load contents of A read from
                   memory via output register and
                   load into U.
J5=U,WA(U=IO+1);   store A in reg. J5, increment the
                   addr. in IO to location of B
                   note: original value in IO is
                   left unchanged. The result of
                   increment is stored in U.
HR(U),U=J5;        read H memory using addr. in
                   reg. U, and load A into U.
```

*see Table A. (appendix I).

```
NOP;              we let time pass for memory to
NOP;              perform its task.
WA(U=U+HW);       add A -in U- to B -in ouput reg.
                  HW- and leave the result in U.
H=U;              store result in input reg. H of
                  H memory
Y(U=2);           load immediate value 2 into U.
WA(U=U+IO);       add 2 to memory location of A to
                  get memory location of C in H
                  memory.
HW(U);            write the sum of A and B stored
                  in H register into H memory
                  at location pointed to by U.
```

ADDITIONAL INSTRUCTIONS FOR SENDING CONTENTS OF U TO OTHER REGISTERS.

There are two formats for the following instructions. For example, in the instruction TU1=U, one can also write Y(TU1=U). But if one writes the instruction using this last format, then one has to consider the rules of joining instructions, so as to assure parallel processing. (See Table A). If one does not pay attention to the format of these instructions, then one does not have the assurance that the instructions one wants to have performed in parallel will actually be performed in parallel.

The list of instructions is:

1	TU1=U	means	TU1←U(18:23)	load bits (18:23) of U into reg. TU1
2	TU=U		TU←U(7:23)	
3	H=U		H ←U	all 24 bits sent
4	SG=U		SG←U(14:15)	
5	K=U		K ←U	
6	FT=U		FT←U(22:23)	
7	FM=U		FM←U(10)	

INSTRUCTION FOR SENDING DATA FROM REGISTER HW TO REGISTER K.

This instruction sends the contents of HW to K simultaneously. This facilitates sending data from H memory to PE's. The instructions are:

1	K=HW	means	K←HW
2	Y(HW=:H)		HW←¬H

For this last instruction, only this instruction format exists.

SENDING DATA INTO FT REGISTER.

In performing this operation, one has to write two instructions. The pairs of instructions depends on the format of the data.

```
1    Y(U=0B);                    for 12 bit arithmetic
     Y(FT=U);
2    Y(U=1B);                    for 16 bit arithmetic
     Y(FT=U);
3    Y(U=3B);                    for 24 bit arithmetic
     Y(FT=U);
4    Y(U=2B);                    for floating point arithmetic
     Y(FT=U);
```

SENDING DATA INTO SEGMENT REGISTER.

```
1    Y(U=0B);                    for 8 PE's
     Y(SG=U);
```

Note: an instruction being executed in parallel to the last instruction of the above pair will be doing it in accordance to a previously defined segmentation. Only after this last instruction of the pair is executed will the new segmentation be used.

The above pair is used for defining segmentation; only the octal value of the first instruction is different. THe rest of the values are:

```
a    400B          for 16 PE's
b    1000B         for 32 PE's
c    1400B         for 64 PE's
```

When there are several users running, they time-share; they do not compute in parallel, because there is only one CU. The O.S. saves and restores all of the registers in the case that the context changes from one user to another.

SENDING DATA TO REGISTER TU OF CU.

The general form of the instruction is:

**** next page ****

$$TU\eta = \left\{ \begin{array}{l} P \\ !P \\ P1>\aleph \\ P>\eta \end{array} \right\} \qquad \eta = \left\{ \begin{array}{l} 2 \\ 4 \\ 8 \\ 16 \\ 32 \\ 64 \end{array} \right\}$$

possible number of PE's in each group

$\aleph$ is the processing element number within a group. It values range $\phi \leq \aleph < \eta$.

1 TUηP activates in each group, processing element P$\aleph$

2 TUη!P activates in each group, all processing elements except P$\aleph$

3 TUηP1>$\aleph$ activates in each group, processing elements 0:$\aleph$ inclusive.

4 TUηP$\aleph$>η activates in each group, processing elements $\aleph$+1:η

In the table on page 146-149 of the manual, one can see that there are two instructions which are equivalent to the mnemonics above. For example:

TU2P1 means Y(U=500B)
Y(TU=U)

This table is used to see which pairs of "Y" instructions are equivalent to the mnemonics above in case that such mnemonics are not known to the assembler/translator. However, there is another approach which permits avoiding the use of the table. One can use the information stored in TU to accomplish the same purpose. In particular, one can remember that TU has the following bits for the following purposes:

register TU

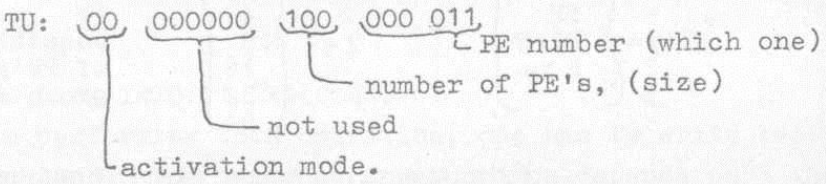
bits	0:1	activation mode
	2:7	not used
	8:10	size of group: how many PE's
	11:16	PE number: which PE in group

Now, what needs to be done -as an illustration of the use of TU- is to divide the computer into groups of 4 PE's, and activate the 3rd PE in each group. The mnemonic for this would be:

TU4P3

but one can also set the TU register with the following bit pattern to do the same thing:

next page



The above bit pattern is 000403B in octal. Thus, the simple way to do the same as the mnemonic on the previous page is:

```
Y(U=403B);          load in U the bit pattern
Y(TU=U);            stash it in register TU
```

INSTRUCTIONS FOR JUMPING.

The general form of the jump instruction is:

G(A10)/τ2 where G stands for G memory and the "/" is for conditional jumping.

and where

$\tau 2 = \begin{cases} U \ominus A12 \\ TY=1 \\ Q=0 \end{cases}$ match contents of U to A12; check against at least 1 active PE; check if all Q regs. are 0

$A12 = \begin{cases} I\alpha \\ J\alpha \end{cases}$ these registers are in CU

If the condition 2 is not satisfied, then the next instruction is executed. We can say that the "G" instruction is a CU instruction. "G" instruction takes 1 clock period if $\tau 2$ is false, and 4 clock periods if $\tau 2$ is true.

If /$\tau 2$ is not written, then the instruction becomes an unconditional jump to a location specified by A10 (defined previously, at the beginning of this report).

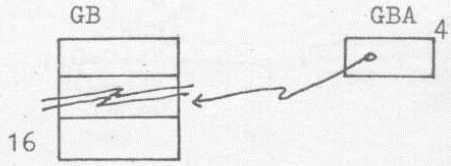
Note: it is possible to overlap a "G" instruction with another 1 clock period instruction for PE's.

Restriction: if one checks the Q registers (triggers), one cannot change their contents at the same time one checks them. The reason is that U is in CU, and "G" is a CU instruction.

SECOND TYPE OF JUMP INSTRUCTION. (JUMPING TO SUBROUTINE).

This instruction permits the jumping to a subroutine, and the return is done with the jump instruction of the last section, but without 2, that is, condition.

Associated with jumping to a subroutine is the set of "registers" called GB, and a register which points to the last return address at the top of the stack in GB. See diagram:



The process for going to a subroutine is: first add 1 to register GBA, then load into GB pointed to by GBA, the address of the next microinstruction to be executed after returning from the subroutine. And finally, jump to subroutine indicated by A10. Upon returning from the subroutine, jump to the location stored in GB pointed to by GBA. Subtract 1 from GBA.(The subtraction and addition to GBA is done modulo 16,hexadecimal).

THE NOP INSTRUCTION.

The NOP instruction in the PS-2000 is an important/useful instruction in that it "wastes time" when processors have to wait for memory instructions (or other instructions) to finish their operation. In general, this is not important, but in parallel computer systems, it is very important.

The NOP instruction can appear on either side of the sign. In each case, it will be executed by either the CU or the PE's.

NOP § a CU instr. ; CU instr. executed

a PE instr. § NOP ; PE instr. executed.

examples: in the case that the machine has 16 k memory, it takes 2 cycles to execute; in the case of 4 k memory, it takes only 1 cycle.

16 k memory	4 k memory
HUR; NOP; NOP; U=HW;	HUR; NOP; U=HW;

There are some situations in which the NOP instruction can be dispensed with. For example, the code on the right side is equivalent, and less redundant than the code on the left side.

HUR; SA(C=0) NOP; P(B=C) NOP; U=HW;	HUR; SA(C=0); P(B=C); U=HW;

INSTRUCTIONS FOR FLOATING POINT MANIPULATION.

The FT must be used for these instructions.

The formats are: (execution instructions)

$Z\Omega_1(Ai\nu_1Aj)$ where Ai, Aj are any registers.

$\nu_1 = \{ *, :, +, - \}$ $\Omega_1 = \{ 1, 2, C, M, MC \}$

1	preperation
2	perform
C	exponent
M	mantissa
MC	joint exponent and mantissa

this refers to the fragment of operation designative by ν_1.

(normalization instructions)

$ZAi\Omega_2Aj\nu_2$

$\nu_2 = \{ *, : \}$

$\Omega_2 = \{ LL, L, 4R, AP, DN, NL \}$

LL	logical shift left by 1 (see note LL)
L	logical shift left by 1
4R	arithmetic shift 4 bits to the right to either A_i, A_j
AP	rounding
DN	denormalize
NL	

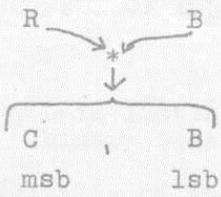
the two words A_i and A_j are processed, separately.

INSTRUCTIONS FOR FIXED POINT MULTIPLICATION:

Example: a program for fixed point multiplication, using the one step instructions.

FT can be either 24, 16, or 12 bit mode; fixed point arithmetic.

Registers R∝ and C have the operands, and the result is stored in C and B. Thusly:



The code:

```
Z1(R∝I * B);        preliminary/execution
Z2(R∝I * B);        performing/normalization
      .
      .
      .
```

The Z2 instruction is repeated depending upon the size of the operands, that is, depeding upon the contents of FT.

if FT is set for 24 bits, then repeat Z2 6 times
" 16 " 4
" 12 " 3

Thus, or as a consequence of this phenomena, it is evident that the machine needs 4 clock periods for every 12 bits.

There is an assembler construct that permits easier coding for repetition of Z2 instructions. For example:

```
REPEAT=12          for 12 bit manipulation
Z1(R∝I * B);
Z2(R∝I * B);
```

If one needs to process 16 or 24 bit numbers, then replace the 12 with the appropriate number. THe assembler will produce the appropriate number of Z2 instructions.

Note: it is important to consider the rules for coupling instructions of CU and instructions of PE. There are two ways to do it. The first way is to write the code with coupled

instructions, and then let the translator/assembler determine if it is a correct coupling. But one must beware. It is also the case that the translator does not detect all of the incorrect couplings, which brings us to the second method. It simply involves consulting Table A, which indicates all of the legal couplings of instructions.

********** P A R T T W O **********

In this part, more complete programming examples are given, as well as rules of syntax of the organization of entire programs. In addition, input/ouput of the system is dealt with along with the "software tools" (editors, debuggers etc) which are needed to make a "useful" program.

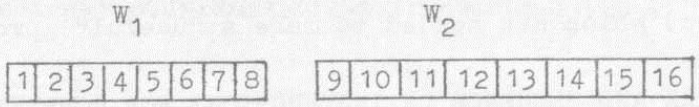
GENERAL SYNTAX AND SEQUENCE OF INSTRUCTIONS FOR COMPLETE PROGRAMS.

The sequence is:

```
PS=L,B,T;        L is listing, B is symbol table, T is
                 binary code output. These are options.

TITLE=<text>;    title of the program to be placed at
                 top of each page, optional

PUNCH=<text>;    text to appear at the beginnig of each
                 paper tape. It is optional.

START=<num> ;    starting load address of the program
                 in memory in decimal or octal. ( nnnnB)
                 If one says START=;, it means load at
                 0. One can have several "starts" in
                 several parts of the program to have
                 a disjoint or noncontigous program.
                 The assembler is either absolute or
                 relocatable. In addition to START,
                 there is a command to start the execu-
                 tion of the program at an address
                 other than the load address.

NAM=llll;        program name, no more than four letters.
    .
    .
microcode
    .
    .

END;             the end.
```

In between NAM=llll; ... END; , one can have other instructions for title and start (load address). The title command makes the printer begin with a new page, and the start command begins loading the program at a new location.

Programming example: assume that there are W processing elements. The program is to calculate the natural numbers. After each step of the program (or iteration), there will appear in each PE one of the natural numbers. See the diagram below:



The first group of numbers are under W_1, and the second group are under W_2, etc.

Note: this program is the same as iota in APL.

The following example is for 16 PE's.

```
      PS=L,T,B;
      START=100B;          load beginning at 100B, absolute.
      NAM=INDX;            name of program is INDX
      Y(U=3);              load code for format 24 bits
      FT=U;                stash it in the format register
      Y( U=400B);          load code for segmentation, 16 PE's
      Y(SG=U);             store it in segmentation reg. SG
*               line that starts with star is comment
*
*               for this program to run, the following data
*               must be loaded in the following way:
*
*            1- JD=16
*            2- I4 contains reg. number of R.  Registers R are
*               from R0 to RF.
*            3- M(I3)= 1,2,3,4,5....,16   The number corres-
*               to the number of PE's.  This data is loaded
*               in such a way so that 1 is in I3 in PE1, 2 is
*               loaded in I3 in PE2, etc.
*            4- JE is the complement to tail length. In some
*               cases one wants to produce a set which has the
*               length, say 67 for 8 elements. The natural
*               divider is 64; the tail is 3.  Complement to
*               tail is 64. THUS JE CONTAINS 64.
*            5- IB is the integer (N/n)+1, where N is the last
*               natural number, n is the number of PE's, in
*               case, 16.
*            6- I0 contains the number of superwords that one
*               needs.  The first superword is 1,2,....16 ;
*               the second superword is 17,18,...32;  and the
*               third superword is 33,34....48;
*               I0 has the values 1  I0  IB.
*               I0 indicates which superword is needed.
*
*
```

```
*    U=I3;                          load U with I3, address of loc. in
*                                   memory M.
*    MUR, WA(U=IO-1);               read from M  mem., at addre. point-
*                                   ed to by U, load new  superword
*                                   number.
*    K=U,WL(U=JD);                  load K with U, and load U with
*                                   number of PE's, ie 16
*    P1(C=K),K=U,WA(U=I4);          load C in PE from K, load K from U
*                                   and U from I4, the number of the
*                                   register in which there is an addr.
*                                   which -in each PE- the final
*                                   result will be placed
*    IR=U;                          now we copy this addr. into U
*                                   AT THIS POINT, MUR HAS  FINISHED
*                                   ITS ACCESS TO M MEMORY.
*    P1(B=K),ROI=C;                 contents of JD loaded via U and K
*                                   into B via slow channel to all PE's
*                                   in parallel. in each PE, load Rn
*                                   indexed by IR. value comes from
*                                   operation IO-1, above.
*                                   NOW THE MULTIPLICATION IS PERFORM-
*                                   ED IN FIXED POINT MODE.
*    Z1(ROI*B)$ U=JE;               do the execution of the multiplica-
*                                   tion, and in parallel, load the
*                                   natural divider, ie 64
*    Z2(ROI*B)$ Y(TU=U);            do first normalization, load
*                                   activation register TU with 64
*    Z2(ROI*B)$ Y(U=300000);        load U with bit pattern which is
*                                   code to activate after given PE
*                                   number, inclusive.
*    Z2(ROI*B)$ WA(U=U+JE);         "or" the bit pattern with the
*                                   PE number, ie 64
*    Z2(ROI*B)$ I1=U;               temporarily stashes the bit pattern
*                                   in I1
*    Z2(ROI*B)$ WA(U=IB);           load U with value (N/n)+1 prestor-
*                                   ed in IB register.
*    Z2(ROI*B)$ NOP;                this NOP is not necessary but we
*                                   put it in for fun.
*
*    SA(C=Z+B);                     by MUR, number in Z ouput register
*                                   is stored, and it is "ored" (exclu-
*                                   sive) to B and stored in C
*
*    P1(B=C,ROI=C)$ G(TAIL)/U=IO;
*                                   save in B the result in C for
*                                   future use, and in parallel save
*                                   in Rn indexed by IR.  If U is
*                                   equal to IO, then go to TAIL. IO
*                                   indicates the superword needed.
*                                   If it is not equal (U to ROI),
*                                   then execute the next statement.
*    G(GB);                         return from this routine via
*                                   return address pointed to by top
*                                   of stack in GB.
```

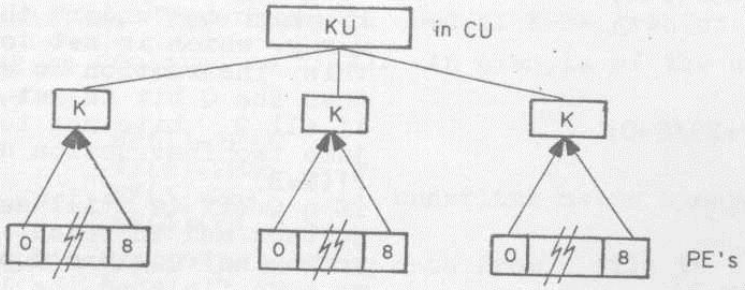
```
*
*
TAIL:KU=B/TU$ U=I1;          send contents of reg. B from
*                            active PE. There is only one PE
*                            activated by what is stored in JE.
*                            The contents that are sent are
*                            placed in KU. Note, when data
*                            is placed in KU, it will also be
*                            in K. The "ored" value placed in
*                            I1 is taken out to be used.
     P1(C=K)$ Y(TU=U);       contents of K loaded into C, and
*                            at the same time load into TU
*                            the activation bit pattern
     ROI=C/TU$ G(GB);        load ROI with C for active PE's
*                            determined by TU, and in parallel
*                            return from this routine.
     END;
```

COMMENTS TO THE PROGRAM.

1-in the code, the sign "$" is to mean § .

COMMENTS OF THE RELATIONSHIP OF K REGISTER AND KU REGISTERS.

The relationship of the KU register and the K registers and the modules of 8 PE's is diagramed thus:



If the KU register is loaded, then automatically the contents of KU are written into the K underneath.

If all of the K's registers are loaded, then the KU register is also automatically loaded. However, if the contents of each K register is different, then what will be loaded into KU will be unpredictable.

If a segment is defined, using instructions related to the SG register, then the K registers are "physically" connected to the 8 PE's. This is done with an electronic switch.

The transfer to and from the KU register and the K register is done in parallel. Consequently, it is evident that if data is to be sent from several K's to KU, then the data must be the same.

A second example in programming the PS-2000: the goal is to write a 1 in PE0, a 2 in PE1, and ... 16 in PE15. The code is:

```
     Q;                          set all Q's equal to 1
     SL(C=0);                    initialize all C regs. to 0
     SA(C=C+1);                  now set C to 1, all C's in the
*                                all PE's. They will be incre-
*                                mented later.
     P1(B=C)/QF;                 load reg. B with value in C in the
*                                PE which is "under" the left most
*                                Q reg. which is set to 1. After
*                                this  instruction is executed,
*                                then the Q bit is set to 0.
     G(*+2)/Q=0;                 if all Q  bits set to 0, then
*                                jump two instruction down, to
*                                P1(M=B)
     G(*-3);                     if a Q bit is still set to 1,
*                                go back and increment C by 1
*                                three instructions "up"
     P1(M=B);                    we have finished the loop, store
*                                the value in input reg. of M.
          .
          .
          .
      do other things
```

INPUT/OUTPUT CONTROL INFORMATION.

The following programs, which can be called from FORTRAN, are only stored into memory, but are not read from memory.
1- PUTG is a program which sends a program from the memory of CM2 (the host of PS-2000), or from the disks into G memory.
2- PUTH does the same as above, but sends the program into H memory.

The calling convention of these programs from FORTRAN is:
1- CALL PUTG(K,M1,IB,NF,M,PB) where:

K= number of logical unit from which one receives the information; the values of which mean:

>0 Disk Number (number on drive)
=0 Random Access Memory (RAM)
<0 Device which is not disk, eg. Paper tape, or whatever.

M1=name of array in SM2 in which the data exists. This is for K=0 (RAM). The array is of integer numbers. The maxi-

mum size is 10,000. If wants to load an array which is large, then one calls PUTG again. If K is not equal to 0, then M1 is ignored.

IB=this is an output variable of this program, and it tells how the call did. If IB(1)=0, then the call did alright. If IB(1) is not equal to 0, then IB(2) will contain more information about the error. If this happens, then consult the staff about the meaning of the error.

NF=is the name of the disk file. There are two characters per array position. This array is used if K is greater than 0. The array is declared as NF(4). An example of the use of this array is:

```
NF(1)=2HMA
NF(2)=2HSH
NF(3)=2HA_        underline means blank space.
NF(4)=2H__
```

M=is a buffer used for sending data from a file to G memory. It is declared as M(144).

PB=an integer, which is used to tell the loader where (beginning a which address) the program is to be loaded. When PUTG finishes, then PB points to the first free location not used in G after the load operation. This last value is used by the next load operation upon calling PUTG. In addition, in saying, PB=PB+M, one leaves a space of M words.

One has to define PB is a double precision integer, if one is loading in a memory which is greater than 32 k. The following trick is used:

```
DP (double precision) PB      DIM I(2)
INT                   PB EQUIV (DP,I(1))
```

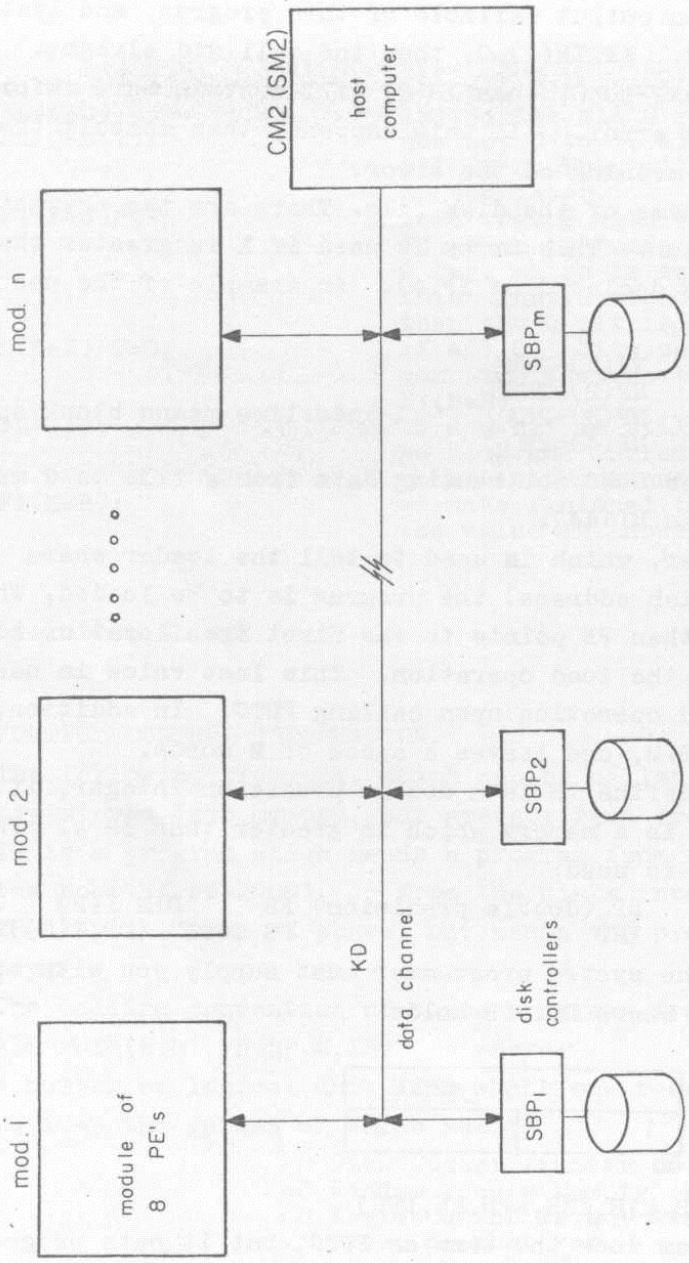
In this case, the system programmer must supply you with a a modified PUTG where DP, PB holds:

DP	
1	2

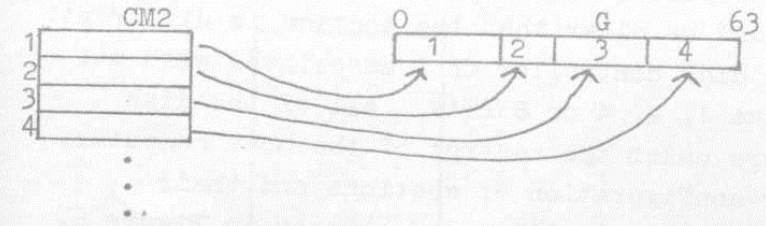
2- CALL PUTH(KH,M1H,IBH,NFH,MH,PBH)

This program does the same as PUTG, but it puts programs in H memory instead of G memory. It is useful to note that the way in which the program transfers information from the

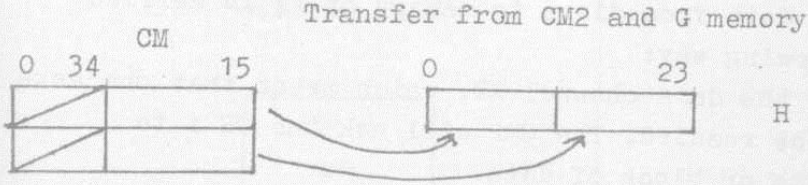
Figure 5.


CM2 (SM2)
host computer
mod. n
mod. 2
mod. 1
module of 8 PE's
KD
data channel
disk controllers
SBP1
SBP2
SBPm

CM2 computer and the PS-2000 machine.



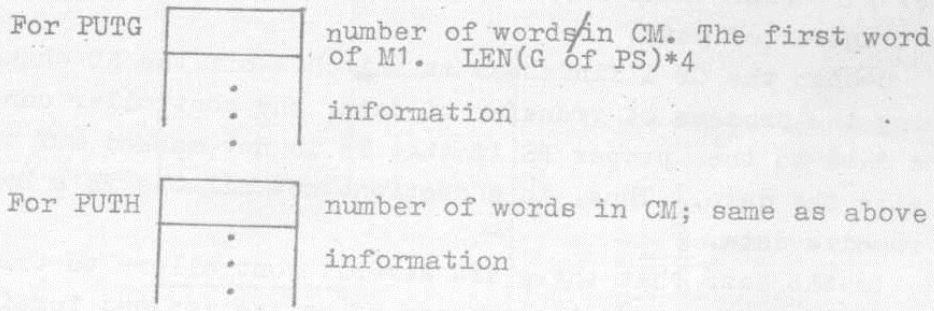
Transfer from CM2 and G memory.



Transfer from CM2 and H memory.

Copy data from CM to PS(G) and PS(H) memories leaves unchanged the contents of the memory of the CM2 machine.

Note on the format of M1.



HOW TO LOAD DATA FROM RAM OF CM2 OR FROM DISK.

The relationship of the disks and the memory of the PE's can be seen in Figure 5. The data is transfered via an I/O channel. This channel is connected to the host computer CM2 as well as the PS-2000. In between the I/O channel and the disks themselves are disk controllers, or so-called special purpose microprocessors which interface the disks to the system. The I/O channel is also called the KD channel.

The process of transfering data, the programmer has to

subdivide the PE's (and corresponding memory) into so-called sections. These sections can coincide with groups and segments, but it must be clear that the section is different. Each section has a disk controller CB i associated with it. The section may have 1, 2, 4 or 8 PE's. All of the disk controllers CB i are under the control of the host computer, CM2. A particular configuration of sections and their corresponding PE's, and controllers.can be seen in Figure 6.

In sending data from disks to memory of PE is carried out in the following way:

a-activate the data channel KD, which means that one asks the channel to be readied. The CM2 will ask the CB i to transfer "x" size or block of data.

b-ask the CB i to send the data.

c-the data will be transfered directly into memory, from the point of view of the programmer. However, internally, the transfer is not direct because it involves several intermediate steps. This internal system employs buffers, etc. When the data starts to flow, the CM2 becomes free, and thus available for other purposes. This happens because the CB i takes over in the transfer of data.

d-when the CB i finishes, it switches off the KD channel. During the process of transfer of data, the controller can send data to the proper PE if that PE is not masked NOT to receive the data. Thus, in a section, not all the PE's have to receive data.

In the case that there are several controllers to transfer data to the PE's, then the process of activation and termination is:

a-activate each controller sequentially. There can be several controllers working at the same time.

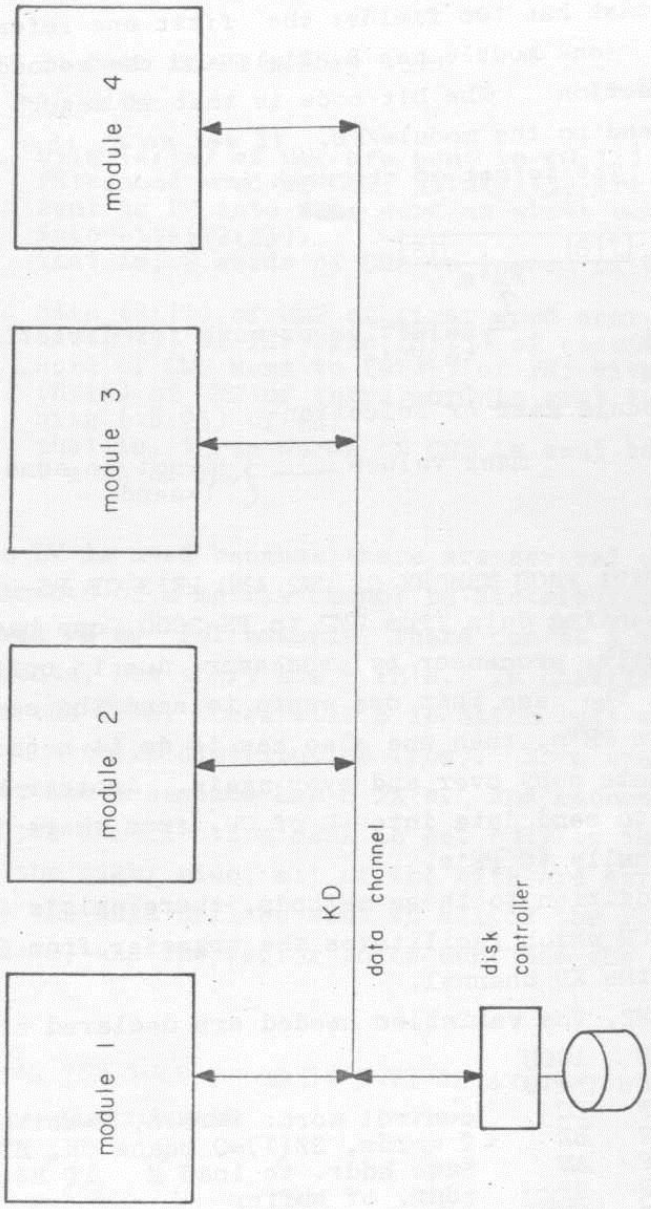
b-each controller will stop when it finishes its task.

c-the KD channel remains "on" while there is at least one controller active.

d-the last controller finishes it task, and it will turn off the KD channel.

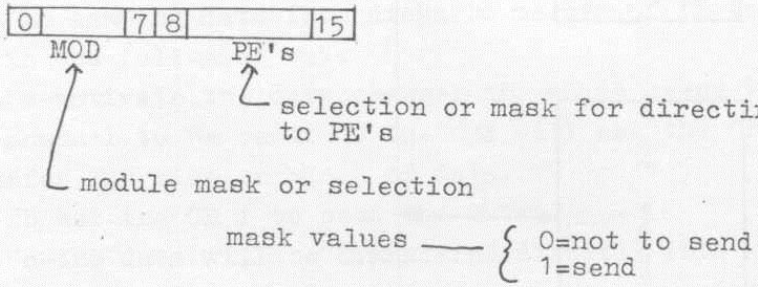
Figure 6.

One Section.



Given a section, the way to control the flow of data from the disk controller and the PE's is by means of a mask.

The mask has two fields; the first one refers to module selection (one module has 8 PE's), and the second one refers to PE selection. The bit code is that if set to 1, then the data is send to the module/PE, if set to 0, then the data is not sent. The format od the mask is:



SENDING DATA FROM MEMORY OF CM2 AND PE'S OF PS-2000.

In sending data from CM2 to PS-2000, one has to do it sequentially, processor by processor, due to software restriction. In the case that one wants to send the same data to all of the PE's, then one also has to do it sequentially, sending the same copy over and over again. Another better way to do it is to send data into U of CU, from there to register K, and finally to PE's.

In addition to these methods, there exists a program called IOCM which facilitates the transfer from CM2 to the PE's via the KD channel.

In CM2, the variables needed are declared as follows:

```
JSB   IOCM
DEF   *+10
DEF   YC      control word: 1=read, 2=write
DEF   BZ      2 words, BZ(1)=0 means OK, BZ(2) error code
DEF   AM      base addr. to load M   if BZ(1)=0
DEF   BF      addr. of buffer
DEF   LBF     length of buffer
DEF   FT      type of data transformation
DEF   MA      module/PE mask to send data
DEF   LV      length of vector
DEF   CB      address in memory of CM2 of location of
              array, up to 36 words.
```

Comments on paramters: in FT, there are several codes for requesting data transformation: 161, 212, and 38. The explanation:

	24 bits in PE memory word
161	16 bits
212	bits (4:15) of CM2 are sent to (0:11) of PE;second word of CM2, bits(4:15) are sent to PE into same word as above but into bits(12:23). that is, 2 words of CM2 to 1 word of PE
38	bits (8:15) of CM2 of first word sent to bits(0:7) of PE; bits (8:15) of second word of CM2 sent to (8:15) of PE; bits (8:15) of CM2 of third word is sent to bits (16:23) of PE. that is, three words of CM2 is sent to 1 word of PE.

The variable LV is used because there are several cases in which the elements of a matrix cannot be distributed evenly over several PE's. For example, there can be a vector which has 18 elements, but there are 8 PE's. In distributing the elements over the PE's, there will 6 location left over (with zeroes) in the third distribution (row). More specifically, assume that the a section has 8 PE's. And assume that the left side of the distribution mask is set only to that section. But on the right side, all of the bits are set to 1 indicating that the data will be sent to all of the PE's. Now, LV is set to 18, and the vector to be sent has the folwing contents:

11 23 444 6 45 789 1000 7 98 34 5 77 12 432 5 7 8 100

The vector will be distributed:

11	23	444	6	45	789	1000	7
98	34	5	77	12	432	5	7
8	100	0	0	0	0	0	0

The last vector is padded with zeroes.

CB is a variable the way to tell the OS where to put things. The organization of CB is:

cell	contents	meaning
1	-1	magic number
2	addr.	first free address of M memory before loading
3	aaddr.	last free address before loading
4 to 7		NOT USED
8	mod. num.	number of modules in physical configuration. OS writes this information in this location.

The way to call the routine from FORTRAN using these parameters is by calling IOCM. The format is:

```
CALL IOCM(US,BZ,AM,BF,LBF,FT,MA,LV,CB)
```

Thus, if one wants to send data from the disk to PE, you must first send the data to the memory of CM2 and then use IOCM to send the data to the PE's sequentially. The routine for transfering data directly from the disks to the PE's directly, will be done soon (through the disk controller, and not via CM2).

GETTING A PROGRAM CODED, LOADED, AND DEBUGGED.

The general process to get a program running on the PS-2000 is to code a program, have in punched on paper tape, read the program from the paper tape into the host machine CM2, edit the program, and debug it. During the intermediate stages of the debugging process, a user can have his program punched on paper tape as backup. The program usually resides on disk. The final version is also stored on paper tape.

During the process of editing, it is very useful to always have a listing of the program, since there is no screen editor. Not only that, the program must be assembled at least once, because the assembler is the program that generates line numbers on the listing for entering line numbers in the

editor.

For any program to run, logical numbers must be assigned. However, if two programs use the same logical unit number for the same purposes, then the logical unit number does not have to be reassigned between running the first and second program run. But in most situations, the logical unit numbers must be assigned before running anything; the operating system does not have the feature of assigning logical unit numbers by default. It is assumed that the programmer/operator is responsible for all assignments.

One final comment: the operating system does not control the access to the PS-2000. It is the responsibility of the programmer/operator to make sure that the PS-2000 is free to run a program. Otherwise, he/she must wait.

COMMANDS FOR THE FILE CONTROL PROGRAM, AND UTILITIES.

1-Initialization of the user console.

:OP
the machine responds to *.

2-Assignment of logical unit numbers to files/physical devices.

:ΛH,lun,{<file name> ::nd / nu}

This associates an lun to a file or physical device. nd= disk (physical) number, lun=logical unit number, nu=unit (physical) number. Machine responds with / if assignment successful.

3-Test (and display) logical unit numbers.

:TΛ
the machine responds with a list of lun's and the corresponding assignments.

4-Start (or run system routines) program.

:CT, < name of system routine >

5-File management program called POF (ΠΟΦ).

:CT,ΠΟΦ starts POF
the machine responds with / if POF starts OK.
POF is now expecting commands for file operations.

5-a Copying files.

Кф,{<source file name>::nd / nu},{<destination file name>},{С / А}

Note that if the destination file already exits, then it must be erased before performing the operation of copying. The machine will respond with / if operation OK.

5-b Printing files.

Пф, <file name> ::nd [,{С / А}]

[ ] square brackets means "optional".

In requesting to print, alphanumeric file is taken as default. The contents of the file will appear on the screen of the display.

5-c Print file attributes.

ПМ, <file name >::nd

5-d List files on disk.

ПС,nd

5-e Delete file.

Уф, <file name> ::nd

5-f Rename file name.

Нф, <old file name> ::nd, <new file name>

5-g Exiting POF.

КР

6-The Editor.

Before editing a file, one must assign the logical unit numbers in the following way:

: Н,5, <editor input file name> ::nd
: Н,4, <new file name, from editing> ::nd

To start the editor:

:СТ,РСИА,,1

6-a Insert text.

I,n

Insert command inserts text after line "n". One can insert several lines if necessary. However, if one wants to terminate the input sequence, one must type "/" IN THE FIRST COLUMN OR POSITION IN THE LINE.

6-b Delete lines (inclusive).

D,n1[,n2]

Deletes lines from n1 to n2. If one wants to delete one line, then it is not necessary to include "n2".

6-c Replace text.

R,n1 [,n2]

This command deletes lines n1 to n2, and then replaces them with newly inserted lines. One does not have to insert the same number of lines that are deleted.

6-d Delete previous line.

/↑

One must press both keys. The effect is to delete the previous line. It can be done as many times as necessary.

6-e End of editing session (part 1).

E

6-f End of editing session (part 2).

:ПЧ,РСИД

Upon exiting the editor, one must request the creation of the new version of the file that was edited. The new file has possibly a new file name. The old file version is NOT automatically erased by the editor. Such erasure has to be done by hand.

NOTES OR COMMENTS IN RELATION TO THE EDITOR: 1- there are no change instructions or commands in the editor. That is, one cannot have the editor search for a pattern of

characters, and then replace it with another pattern.
2-the line numbers serve to reference parts of the file by line ONLY. There is no way to reference parts of the file any other way. 3-the file names are a simple single name. The system does not support the feature of names with the form < mumble >. < extension > , where extension refers to the file type, such as text, source code, binary, or assembler.

7-Invoking Language Translators.

To invoke the translator programs, first one has to enter the assignment statements:

```
:ΛH,5, < input file name > ::nd
:ΛH,4, < ouput file name> ::nd
:ΛH,6,nu
```

The last instruction is to assign logical unit numbers for printing a listing. This listing includes a symbol table, and the line numbers necessary for editing. To invoke the assembler for the PS-2000, one types:

```
:CT,TPΠC
```

This means: "CTart TPanslator of Παrallel Ctructure (micro-code)".

To invoke the FORTRAN compiler, then do:

```
:CT,FTN
```

8-DEBUGGER (AND LOADER)

There is only one assignment statement for the debugger:

```
:ΛH,6,nu
```

To start:

```
:CT,DBPSH [:: < sys disk number which is usually 33>]
```

After starting, the debugger will start but with no prompt. Then one types:

```
O
```

to gain absolute control over the PS-2000.

The next set of commands is for loading one or more modules into the PS-2000, and to exit the loader part of debugger.

```
B, <load address>          start loading at this address
D, <file name> ::nd        now perform the load of program

LDG                        if another module is to be
                           loaded or some previous load
                           did not work.

B, <load address>
D, <file name> ::nd
        .
        .
LDG
B, <etc.>
D, <etc.>
        .
        .

/E                         finish the load session.
```

When one invokes the debugger with the above start command, one is really invoking the loader. When one exits the debugger, one is really exits the loader and AUTOMATICALLY enter debugger portion for setting breakpoints. This is done by typing:

```
P,a1,a2,.....an
          .
          .                "ai" is an octal address.
          .
P,a1,a2,.....an
          .
        etc.
```

One can type one or more than one line. However, the TOTAL number of breakpoints cannot be greater than 132. Now, one can run the program:

```
RUN[,a]                    "a" is an octal address.
```

The address after the run command is used the first time, but optionally afterward. If the program runs in such a way that the breakpoints are no longer necessary, then one types:

```
D,a1,a2,....an             "ai" is an octal address.
```

to delete breakpoints. To display (print or list) breakpoints, it is necessary to type:

next page

TCP display breakpoints

There will be situations in which the program must be interrupted. For this purpose, one need only hit any key in the keyboard, and the program will stop.

In case it is necessary to alter a value in a particular location in memory of one or more PE's, then use this command:

WM {B / D / H},n,p,a

Where: B means binary
D decimal
H hexidecimal
n is module num. 0 to 7, if n=8, then all module modules.
p is the mask for selection the processors within the module.
a is the address which to be modified.

\# is the response of the machine if it accepts the command.

For example, one may want to alter location 563 octal, with the data 5490 decimal, in module 5, with processor mask 01257. The mask means "do alteration in processors 0,1,2,5,7". The command is:

```
WMD,5,01257,563
#       positive confirmation of machine.
```

Now, it is necessary to input the data. In general, the format is:

<1st num.>, <2nd num.>, <3rd num.>, <n number>

The above sequence of number coincides (in number) with the number of processors which are masked to accept data in the "WM" command.

On finishing input data values into the processors, one types:

/E exit input of data in debugger.

Notes on the debugger: 1-as the machine stops and reports that it has arrived at a breakpoint, it prints the contents of registers of the control unit (CU) and the processors (PE's). 2-one uses the listing of the translator for determining the location of an instruction. The number of the location is the one labeled "LOC". 3-WARNING: it is not allowed to select a location which has in its location an instruction of the type:

$$G(*\pm U)/2$$

The problem is that the instruction is a conditional instruction tion, or more specifically, it is a conditional branch instruction. However, if one has an unconditional branch instruction to an absolute address, it MIGHT work. The limitation is that the jump is not done with the use of the U register. 4-REMINDER: in programming a module, one should make the load address (that is, the address from which the loader starts loading the module) coincide with the start address of execution of the program. 5-ANOTHER WARNING: it is not possible to modify the contents of the registers of neither the CU or the PE's with the use of the debugger. 6-WARNING: register IF is reserved for use of the debugger.

APPENDIX I. Table A.

Joining Rules of Instructions
in the PS-2000.

MNEMONICS OF THE PS-2000

INSTITUTE OF CONTROL SCIENCES,
ACADEMY OF SCIENCES, MOSCOW, SOVIET UNION

Table Translated: October, 1982. IIMAS, UNAM, MEXICO

THE LEFT PART OF THE INSTRUCTION TO BE JOINED:
--> PART 1 <--
INSTRUCTIONS OF PROCESSING ELEMENTS (PE's)

arithmetic, losical instructions denoted by S
loadins functions is done with resister R.

S,R@I=A5/T A5=A1 FT 1 CYCLE

S-arithmetical operators

exlicit: implicit:

exlicit:	implicit:
A1+A2	1
A1+A2+1	2
A1+A2+E	4
A1-A2	5
A1-A2-1	6 A2
A1-A2-1+E	SFA7(A3=A1,A2,1)
A1+1	8 A2,E
A1+E	9
2A1	A
2A1+E	B
SFA(A3= A1-1)	D
-A1-1+E	E
A1VA2+1	
A1VA2+E	0 A1
A1V!A2+1	SFAC(A3=A1,1)
A1V!A2+E	F A1,E
A1&A2-1	
A1&A2-1+E	
A1&A2-1	A3
A1&!A2-1+E	SFA3(A3,1)
2A1+1	A3,E
0	
SFA(A3=-1)	
-1+E	

codes (0-F) arithmetic operations

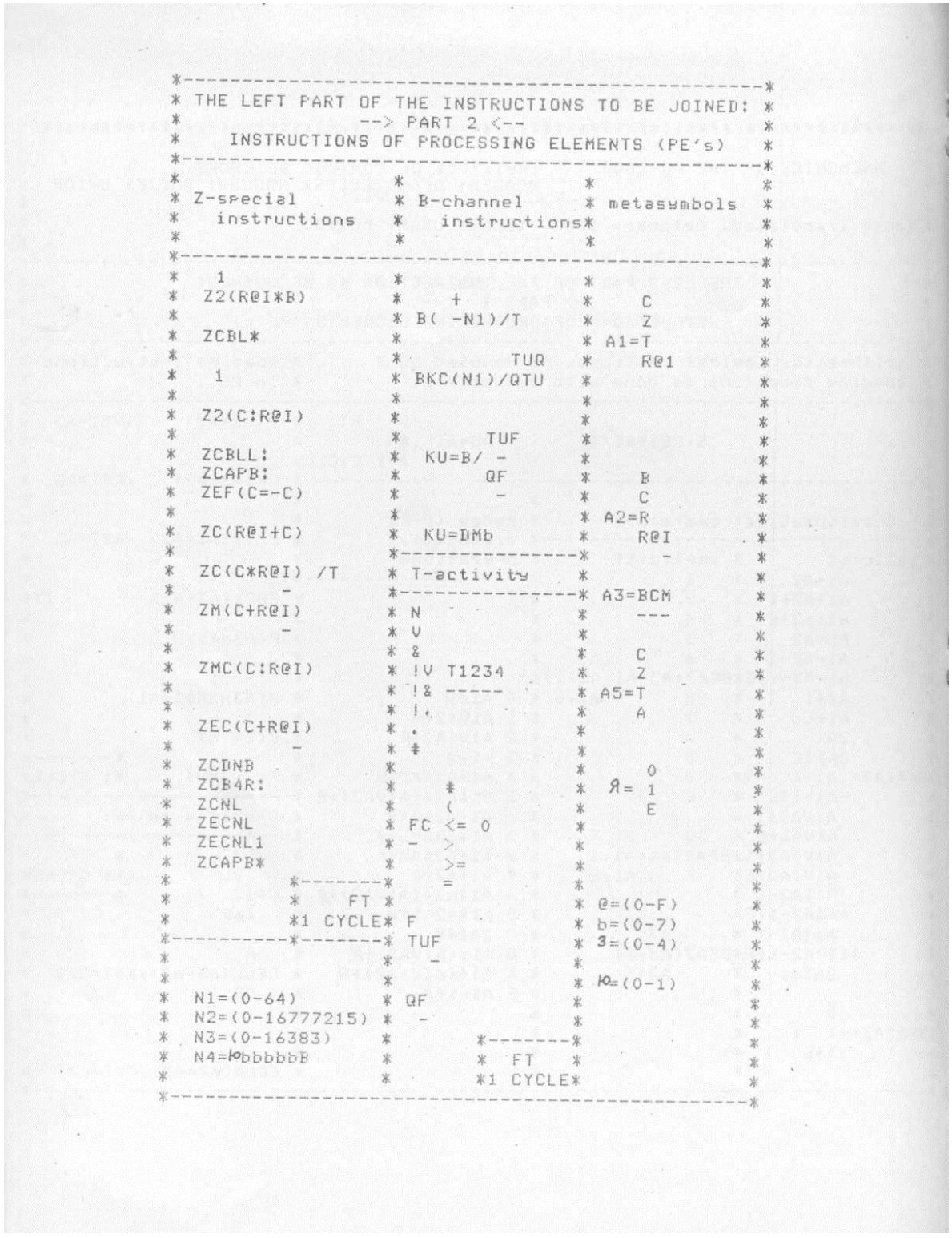
0	A1+Я
1	A1VA2+Я
2	A1V!A2+Я
3	-1+Я
4	A1+A1&A2+Я
5	A1&A2+(A1VA2)+Я
6	A1-A2-1+Я
7	A1&!A2-1+Я
8	A1+A2&A2+Я
9	A1+A2+Я
A	A1&A2+(A1VA2)+Я
B	A1&A2-1+Я
C	2A1+Я
D	A1+(A1VA2)+Я
E	A1+(A1V!A2)+Я
F	A1-1+Я

loading instructions in PE i

P(A3=A2) ,R@I=A5
C
PM(A3=A2) ,R@I=A5
PH12(A3=A2) ,R@I=A5
PH21(A3=A2) /T
P(A3=A1)
1
P(A3H2R@I=A1)
P(Z=!M)
P(Z=!KD) *1 CYCLE

C-shifts in PEi

4	FT
8L	*1 CYCLE
CA12 /T	
16R	
A	
CELL(A3=A2),R@I=C/T	
CR	
L	
ECLR(A3=A2),R@I=C/T	


THE LEFT PART OF THE INSTRUCTIONS TO BE JOINED:
--> PART 2 <--
INSTRUCTIONS OF PROCESSING ELEMENTS (PE's)
Z-special instructions
B-channel instructions
metasymbols
1
Z2(R@I*B)
ZCBL*
1
Z2(C:R@I)
ZCBLL:
ZCAPB:
ZEF(C=-C)
ZC(R@I+C)
ZC(C*R@I) /T
ZM(C+R@I)
ZMC(C:R@I)
ZEC(C+R@I)
ZCDNB
ZCB4R:
ZCNL
ZECNL
ZECNL1
ZCAPB*
FT
1 CYCLE
N1=(0-64)
N2=(0-16777215)
N3=(0-16383)
N4=#bbbbbB
+
B(-N1)/T
TUQ
BKC(N1)/QTU
TUF
KU=B/ -
QF
-
KU=DMb
T-activity
N
V
&
!V T1234
!& -----
!.
.
#
(
FC <= 0
-
>
>=
=
F
TUF
-
QF
-
FT
1 CYCLE
C
Z
A1=T
R@1
B
C
A2=R
R@I
-
A3=BCM

C
Z
A5=T
A
0
1
E
@=(0-F)
b=(0-7)
3=(0-4)
#=(0-1)

```
*-------------------------------------------------------*
*   THE LEFT PART OF THE INSTRUCTION TO BE JOINED:      *
*                --->  PART 3  <--                      *
*     INSTRUCTIONS OF PROCESSING ELEMENTS (PE's)        *
*-----------------------------------*-------------------*
* S-losical operators               * codes of losical  *
*-----------------------------------* operators         *
* explicit:       * implicit:       *                   *
*       0         *  1              *-------------------*
*       1         *  2              *                   *
*       !A1       *  4              *                   *
*       A1        *  6              * 0 !A1             *
*       !!A1VA2   *SL7(A3=A1,A2)    * 1 !!A1VA2         *
*       !A1&A2    *  8              * 2 !A1&A2          *
*       !!A1&A2   *  9              * 3 0               *
*SL(A3=A1#A2    ) *  B              * 4 !!A1&A2         *
*       A1&A2     *  D              * 5 !A2             *
*      !A1VA2     *  F              * 6 A1#A2           *
*      !!A1#A2    *                 * 7 A1&!A2          *
*      A1&A2      *  0              * 8 !A1VA2          *
*      A1V!A2     *SLF(A3=A1)       * 9 !!A1#A2         *
*      A1V!A2     *                 * A A2              *
*      A1VA2      *  A              * B A1&A2           *
*                 *SL5(A3=A2)       * C -1              *
*      !A2        *                 * D A1V!A2          *
*SL(A3=A2 )       *  3              * E A1VA2           *
*       0         *SLC(A3)          * F A1              *
*       1         *                 *                   *
*-----------------*-------------------------------------*
```

```
*-----------------------------------------*
*  THE LEFT PART OF THE INSTRUCTION TO    *
*              BE JOINED:                 *
*           --> PART 4  <--               *
*     INSTRUCTIONS OF THE PROCESSING      *
*           ELMENTS (PE's)                *
*-----------------------------------------*
*      loadins trissers  E,F,T,Q,TY       *
*-----------------------------------------*
*                      0                  *
*                      1                  *
*                      TU                 *
*   0     0            Q                  *
* E=1 , F=1 ,   T1234=CC   ,Q,TY/T        *
*                ---- F                   *
*                                         *
*                                         *
*                          <              *
*   0     0              <=               *
* E=1 , F=1 ,   T1234=FC >   0 ,Q,TY/T    *
*                ---- -- >=               *
*                        =                *
*                        #                *
*                                         *
*                             *---------* *
*                             *    FT   * *
*                             * 1 CYCLE * *
*-----------------------------------------*
```


THE RIGHT PART OF THE INSTRUCTION TO BE JOINED:
--> PART 1 <--
INSTRUCTIONS FOR MEMORIES AND CONTROL UNIT (CU)
control of M and H memory addressing arithmetic and registers: A,HA,IL and IR
pseudocommands
metasymbols
A,L,HA,HL,
IL,and IR
1 CYCLE
M,H
*2(3) CYCLES
PS=L,B,T,16
END
TITLE=<string>
EJECT
PUNCH=<string>
START=<string>
CONST=N4
N4 , N4 ,N4
REPEAT=FT24
ENT=<list>
EXT=<list>
NAM=<name>
binary codes in B FT,SG,TU
jumps and subroutines
shifts in CU.
FT12 Y(U=0)
FT16 Y(U=1)
FT24 Y(U=3)
FTP Y(U=2)
SG8 Y(U=0)
SG16 Y(U=400B)
SG32 Y(U=1000B)
SG64 Y(U=1400B)
A
L
U L
R
C
/TY=1 GT0
/Q=0 GT1
/GT=1 KP1=U
/IQ
/DO=1
G-1,5(6)
CYCLES
GT,KP1
1 CYCLE
CYCLES
I@
U
A/=HW
TU
J@
A8=IL
IR
FM
TU1
TU
SG
A9=H
K
FT
KP1
bbbb
A10=*-U
<label>
A11=KP
HW
bbbbbbbbB
N2
A12=<label>+N3
<label>-N3

```
*-------------------------------------------------------------*
*      THE RIGHT PART OF THE INSTRUCTION TO BE JOINED         *
*                  --> PART 2 <--                             *
*      INSTRUCTIONS FOR MEMORIES AND CONTROL UNIT (CU)        *
*-------------------------------------------------------------*
*                control of loading M,H                       *
*-------------------------------------------------------------*
*   R                                                         *
* M (U), A9=U,I@J@=UC,W                                       *
* H                                                           *
*  W                                    *------------------*  *
*  R                                    * Y,T,J,W -1 CYCLE *  *
* M (U),Y(HW=!H),I@J@=UC,W               *                  *  *
*  W                                    * M,H-2(3) CYCLE   *  *
*--------------------------------------*------------------*  *
*   W-arithmetic operators     *    W-logical operators     *  *
*------------------------------*----------------------------*  *
* implicit:     *              * implicit:   *               *  *
*  1            * Я:  A7       *  1          *  Я:  !A7     *  *
*  2            * B:  A7+1     *  2          *  B:  A7      *  *
*  4            * H:  2A7      *  4          *  H:  !A8     *  *
*  5            * Ы:  2A7+1    *  6          *  Ы:  A8      *  *
*  6            * E:  A7-1     *  7          *  E:  0       *  *
*  7            *---  0        *WL8(U=A7,A8) *---    -1     *  *
*WA8(U=A7,A8,1)*      -1       *  9          *       !!A7VA8*  *
*  9            *      A7VA8    *  B          *       !A7&A8 *  *
*  A            *      A7VA8+1  *  D          *WL(U=!!A7&A8)*  *
*  B            *WA(U=A7V!A8  )*  E          *      A7#A8   *  *
*  D            *      A7V!A8+1 *             *      A7#!A8  *  *
*  E            *      A7-A8    *  0          *      !A7VA8  *  *
*               *      A7-A8-1  *WLF(U=A7)     *      !!A7#A8 *  *
*  0            *      A7&!A8-1 *             *      A7&A8   *  *
*WAC(U=A7,1)    *      A7&!A8   *  A          *      A7V!A8  *  *
*  F            *      A7+A8    *WL5(U=A8)     *      A7VA8   *  *
*               *      A7+A8+1  *             *              *  *
*WA3(U,1)       *      A7&A8    *  3          *              *  *
*               *      A7&A8-1  *WLC(U)        *              *  *
*-------------------------------------------------------------*  *
*              load instruction in CU          * no           *  *
*--------------------------------------------* operation    *  *
*          IR=U                               * operator     *  *
*          IR+1                               *--------------*  *
* Y(A9=U)  IR-1                               *              *  *
*        ,     ,I@J@=UC,U=A11,K=HW,QY=1       *    NOP       *  *
*          IL=U                               *              *  *
* Y(HW=!H) IL+1                               *              *  *
*          IL-1                    *---------*--------------*  *
* Y(U=A12),K=HW,QY=1               * 1 CYCLE *  1 CYCLE     *  *
*----------------------------------*---------*--------------*  *
*    format of TU; bits of TU      *                           *
*----------------------------------*                           *
*    * 01* 234567*   * 89A* BCDEFG*                           *
*----*---*-------*---*----*-------*                           *
* P  * 00* 000000* 64* 000* number*                           *
* !P * 01*       * 32* 001*       *                           *
*<P-1* 10*       * 16* 010* P OT  *                           *
* >P * 11*       *  8* 011* 000000*                           *
*    *   *       *  4* 100*   DO  *                           *
*    *   *       *  2* 101* 111111*                           *
*-------------------------------------------------------------*
```