

The routines to communicate with R2 are written, needing only a few additions (I hope). Tape 8 needs to be thrown out and the subroutine DOR2 put in as the final tape (with .END on it). DOR2 needs, on the top of page 2, the code to tell the R2 to det up the instruction and do it. The addresses in the definig stuff in the beginning of the program need to be put in. The subrt SEND should go on the same tape as DOR2, at the end.



INFLEC = ?  
INSTR = ?

COMMUNICATING WITH R2

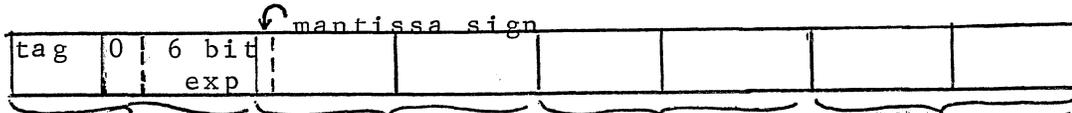
Sending data to R2:

- Send U and S for all instructions.
- Send R if U,R/S or XTRMCP (option C is 4, 6, or 10).

Receiving data from R2:

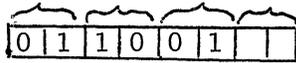
- Receive U for all instructions.
- Receive R for everything except:
  - a) integer and fixed point add and subtract
  - b) logic instructions

R2 Word as represented in the PDP: 4 words of core

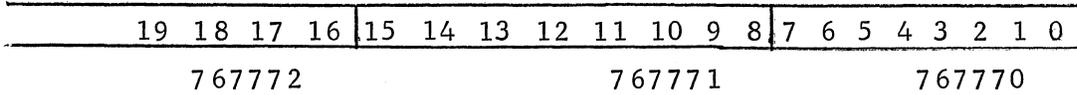


The Tag Byte:

This byte is kept as a 4 or 5 until it is sent to the R2. Then 2 bits are used to represent each bit, resulting in an 8bit representation for the 4 bit R2 field. In each pair, the left bit is 1 if the bit is 1, the right bit is 1 if the bit is 0. Thus, the tag byte as transmitted to the R2 is either 145 (tag 4) or 146 (tag 5).



Memory Address Register:



bit	octal	meaning	bit	octal	meaning
0	1	LOADU	11	4000	R→AD
1	2	U→U	12	10000	S→AD
2	4	U!→U	13	20000	AD→CD
3	10	LOADR	14	40000	
4	20	R→R	15	100000	
5	40	R!→R	16	1	
6	100	LOADS	17	2	
7	200	S→S	18	4	
8	400	S!→S	19	10	
9	1000	LOAD OPCODE			
10	2000	U→AD			

## R2 LOGIC INSTRUCTIONS

The condition code is set for all logic instructions by the routine SETCC.

XTRMCP: This routine does the following for each bit position:

X field( the X option)  
0 S to U if R is 1  
1 S to U if R is 0  
2 1 to U if S and U are different and R is 0  
3 1 to U if S and U are different and R is 1

For inflections 0 and 1, it copies U to PDPU and puts the address of the extract routine (XTRCT) on the stack. For 2 and 3, the address of the masked comparison routine (MCPRT) is stacked. Then a routine called LOGIC is called: it manipulates a 1 bit mask to cover the 54 bits of an R2 word, using the routine whose address is on the stack to do the appropriate action based on U, R, and S, and the bit position indicated by the mask.

AND: The X field interpretation:

0 U AND S to PDPU  
1 NOT (U AND S) to PDPU  
2 U AND S to PDPU ( in the R2, only the condition code is set with inflection 2)  
3 NOT U to PDPU

For inflections 0 thru 2, the address of the routine ANDRT is stacked. Then LOGIC is called to manipulate the 1 bit mask for each bit position, and use ANDRT to do the appropriate action for that bit. Inflection 3 copies U to PDPU and complements it.

ORU: The ORU routine copies S to PDPU and then does the following; as indicated by the X field:

0 S OR U to PDPU  
1 NOT (S OR U) to PDPU  
2 S OR U to PDPU (in the R2, inflection 2 just sets CC)  
3 NOT S to PDPU

SYD: Exclusive OR of U and S. This routine copies S to workspace and U to PDPU. It then clears all the bits of S from PDPU and all the bits of U from the workspace. Then the routine ORU is called to do the OR of PDPU and the workspace. ORU also takes care of the inflections 1-2 (no option 3).

IMPRT: The implication routine does NOT U OR S to PDPU. This routine moves the operand addressed by R0 to PDPU and complements it. Then ORU does the OR and options 0-2 (no option 3 with IMP).

RIMPRT: Reverse implication does NOT S OR U to PDPU. This sets R0 and goes to IMPRT, which takes care of everything.

## ARITHMETIC R2 INSTRUCTIONS

**ADDI:** Integer add. This routine is called with R0, R1, and R2 addressing the low order byte of the operands and the storage location for the result. The bytes addressed by R0 and R1 are moved to work space, added together with the carry from the last addition, and stored in the result location. End-around carry from the high order byte is added back in and propagated. If there is overflow, an indicator (OFLOW) is set, and the sign of the result is corrected.

**SUBI:** Integer subtract. This is called with the same parameters as ADDI. SUBI complements the mantissa of the operand addressed by R1 and calls ADDI. On return from ADDI, it restores the operand for use in R2 arithmetic.

**FADD:** Floating add. The parameters are the same. FADD moves the operand with the smaller exponent to work space and shifts it to compensate for the exponent difference, after propagating its sign in PDPR (it is shifted into PDPR). ADDI is then called to add the mantissas. Overflow is checked; if none, the sign of the result is compared with the sign of PDPR. If they do not match, and the result is 0, the result is set to match the sign of PDPR. If the result is not zero, 1 is added to or subtracted from PDP, as the sign of PDPR is minus or plus. If the signs matched, this correction is skipped. The result is then normalized.

**MPY2:** This routine multiplies S X U. It sets the parameters (operand addresses) and goes to MPYI.

**MPYI:** If called directly, this does U X S. The parameters are the same as for ADDI. The result goes to PDP and PDPR. First, SIGN is set for sign control (0 if the result will be positive). Then the operands are copied to the stack, in order to leave the original untouched. The 6 PDP words of the result are shifted right and the final carry tested. If the carry is set, ADDI is called to add the multiplicand into the partial product. The counter is decremented and the shift repeated as necessary. Then the type is tested: type 0 is an integer operation, in which case PDP and PDPR are swapped and the routine exited. If it is a floating MPY, then the result is normalized, the exponents added, and OFLOW set if necessary.

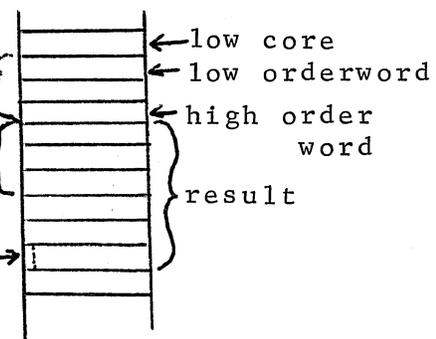
Multiplier addressed by R1.  
Multiplicand addressed by R0.    {    As called.  
Destination addressed by R2.

The stack:

multiplicand

multiplier

sign bit →



## UTILITY ROUTINES

COMPAR: This subroutine does the comparison of results after the PDP and R2 are finished. If the operation was a logic instruction with inflection = 2, only the condition codes are compared. Otherwise, PDPU and R2U are compared. If R1 was not 0 on entry, PDPR and R2R are also compared. The value returned in R3 indicates the result: R3=0 if no comparison failed.

NORMZ: Normalizes the words addressed by R1 and R2. R2 is set to 0 for a single length operand.

ITOFI: Integer to floating point conversion, of the word addressed by R1. The data is first copied to storage (called COPY) so that it can be restored before being used by the R2.

RANDM: This routine generates random numbers which are stored at the location given by R1. The tag byte is not disturbed and the exponent byte is cleared to 6 bits.

SETCC: Sets the condition code (CCODE) for logic instructions. The code is set on the basis of PDPU: 0 if all zeros, 1 if all ones, 2 for mixed even, 3 for mixed odd.

SETACC: Sets the condition code for arithmetic instructions. The code is set on the basis of PDPU: 0 for result of 0, 1 if negative, 2 if greater than 0, 3 if overflow.

## INPUT-OUTPUT ROUTINES

TALK: Parameters are: R1 is a character to be output on the TTY. R2 is the address at which to store the input.

TALK is used in setting the various options. It accepts only octal digits; a carriage return causes exit from the subroutine; anything else causes a repeat of the option request.

OUTSP: Outputs one space on the TTY.

OUTC: Outputs one character on the TTY (char in R1).

INCH: (INput CHAracter) Receives one character from the TTY. Octal digits, spaces, and carriage returns are echoed. There are 3 exits from INCH: the normal return plus 2 instructions if an octal digit is received, the normal return plus 1 if a CR is input, and the normal return for anything else.

DATAIN: Parameters are: R1 is a character to be output. R2 addresses the exponent of the storage area for the data to be accepted. DATAIN handles the input of the operands U, R, and S. It looks for the tag first (a 4 or a 5), then a separator (anything except a CR or an octal digit), then the exponent (1 or 2 octal digits), a separator, up to 16 octal digits for the mantissa, and finally a CR terminates the input. If the first character received is not an octal digit, the subroutine does not return to the calling point: it restarts the request for all options. If more than 2 digits are input for the exponent, or more than 16 for the mantissa, the last digits received will be used.

OUTWRD: Parameters are: R1 is a character to print. R2 addresses the word (4 PDP words) to be output. This subroutine is used to print the results of PDP and R2 arithmetic. The tag is output in octal; the exponent and mantissa in binary.

OUTPR1: Spaces the printer.

OUTPR: Outputs the character in R1 on the printer.

Printer output is as follows:

Whenever the instruction code (C) or inflection (X) is changed, the first line is:

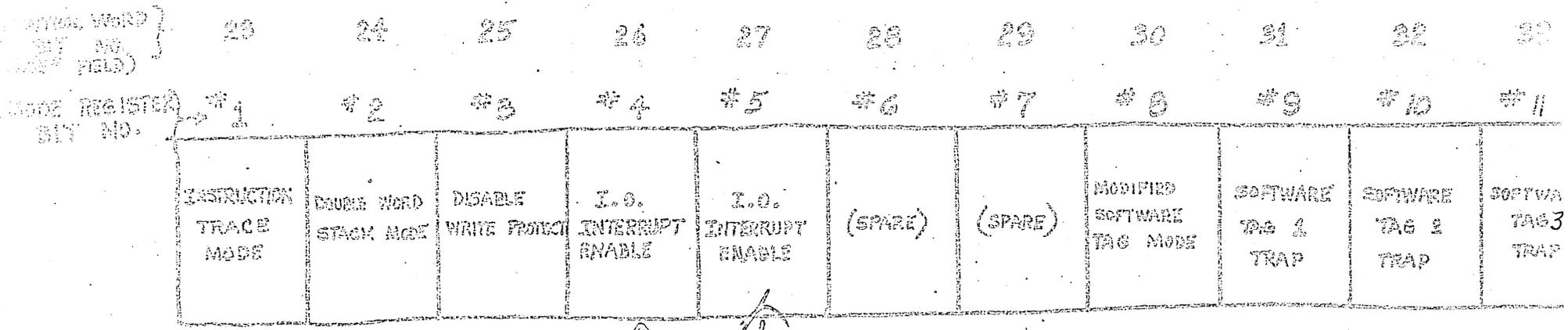
C nn            X n

This is followed by the data, where t is the tag (octal), e the exponent (binary), and n the mantissa (binary, spaced into bytes):

S t eeeee nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn nnnnnnnn  
U            (same format as S)  
R            (same, but printed only if required by the instruction)  
X1           (same; this is the R2 U register)  
PU           (same; this is the PDP equivalent of R2 U)  
XR           (same; this is the R2 R register)  
PR           (same; this is the PDP equivalent of R2 R)

This is followed by the condition codes:

PDPCC=n       R2CC=n



MODE BITS #4 AND #5 CONTROL EXTERNAL INTERRUPTS AS FOLLOWS:

BIT #4	BIT #5	
0	0	NO EXTERNAL INTERRUPTS
0	1	ALLOW LEVEL 1 INTERRUPTS ONLY
1	0	" " 1 AND 2 " "
1	1	" " 1, 2, AND 3

ASSIGNMENT OF DEVICES TO EACH PRIORITY LEVEL IS DONE BY SOFTWARE IN THE PDP/11

MODE BIT #9 CAUSES THE SOFTWARE TAG REGISTER (ST) TO HOLD THE TAG OF OPERANDS LOADED INTO X1 ONLY. WHEN MODE BIT #8 IS OFF, ST IS SET BY THE SECOND OPERAND OF ALL INSTRUCTIONS.

# ASCII/FLEX

	0	1	2	3	4	5	6	7								
00					.											
01	bs	bs	tab	tab	nl	cr	ff	27								
02																
03																
04	sp	sp	!	σ	"	π	#	#	\$	Σ	%	β	&	lcΣ	'	π
05	(	(	)	)	*	*	+	+	,	,	-	-	.	.	/	/
06	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
07	8	8	9	9	:	→	;	Δ	<	<	=	=	>	⚡	?	
10	@	α	A	A	B	B	C	C	D	D	E	E	F	F	G	G
11	H	H	I	I	J	J	K	K	L	L	M	M	N	N	O	O
12	P	P	Q	Q	R	R	S	S	T	T	U	U	V	V	W	W
13	X	X	Y	Y	Z	Z	[	↓	\		]	↑	^		-	
14	`		a	a	b	b	c	c	d	d	e	e	f	f	g	g
15	h	h	i	i	j	j	k	k	l	l	m	m	n	n	o	o
16	p	p	q	q	r	r	s	s	t	t	u	u	v	v	w	w
17	x	x	y	y	z	z	{	Σ(			}	Σ)	~			

27 (stop code) must be punched as overpunch +7

⚡ is punched as uc ≤ bs |

the symbol ≤ is not permitted - use <=