Instructions

for the

EXTENDED BENTON HARBOR BASIC (Version 10.02.00)

Model HC8-13

Table of Contents

INTRODUCTION

This Manual describes the new and different functions of the Extended Benton Harbor BASIC Program (version 10.02.00). It is not meant to replace the H8 Software Reference Manual, only to supplement it. Read through this Manual to become familiar with the new commands.

It is a good idea to make notes or references to this Manual in your Software Manual to remind you of the new and different commands contained in version 10.02.00 of BASIC. Keep this Manual with the H8 Software Reference Manual.

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

595-2164

SYSTEM DATA FORMAT

Extended Benton Harbor BASIC (version 10.02.00) contains the three original file types (001, 002, and 003) plus three new file types (004, 005, and 006).

001 Memory Image.

002 BASIC Programs using Extended Benton Harbor BASIC version 10.01.00.

003 Compressed Text.

004 BASIC Programs using Extended Benton Harbor BASIC version 10.02.00.

005 BASIC Data using Extended Benton Harbor BASIC version 10.02.00.

006 BASIC Programs and Data using Extended Benton Harbor BASIC version 10.02.00.

The three original file types (001, 002, and 003), which are explained in the "Tape Files" section of the H8 Software Reference Manual, do not change. The three new file types are explained in the following paragraphs.

BASIC PROGRAMS (Type = 004)

This file type is used by BASIC (version 10.02.00) when you load and dump **programs.** It is the same as file type 002 except that it uses a different format to dump and load the program. This file always has a label record (#0), a table record (#1) that contains a table describing what data has been stored, and a data record (#2) containing the actual data (program text).

BASIC DATA (Type = 005)

This file type is the same as file type 004 except that it is used to load and dump **data** (program variables).

BASIC PROGRAM AND DATA (Type = 006)

This file type is the same as file type 004 except that it is used to load and dump both the **program text** and **program variables** together.

READING THE DISPLAYS

When the H8 computer is reading or writing data on a tape transport, the front panel displays are continually displaying data about the tape operation. The information contained in the "Reading The Displays" section of the H8 Software Reference Manual explains how to read the displays. However, the data-type LED (the one on the right) displays the type of data being read or written. This information is displayed as:

DISPLAY	DATA/TYPE
1	Memory Image
2	BASIC Program Text (version 10.01.00)
3	Compressed Text
4	BASIC Program Text (version 10.02.00)
5	BASIC Program Variables (version 10.02.00)
6	BASIC Program Text and Program Variables (version 10.02.00)

NEW COMMAND MODE STATEMENTS

The command mode statements in this version (10.02.00) of BASIC either replace or supplement those listed in the H8 Software Reference Manual. Refer to the statements in this Manual when you run version 10.02.00 of Extended Benton Harbor BASIC.

DUMP

The DUMP statement is the same as the dump statement for Extended Benton Harbor BASIC version 10.01.00 except that it produces a file type 4. A program dumped using the DUMP statement can only be loaded using the LOAD statement.

LOAD

The LOAD statement is the same as the load statement for Extended Benton Harbor BASIC version 10.01.00 except that the variables in memory are not destroyed when the program (file type 4) is loaded. However, the variables will be cleared to zero if you use the RUN command to execute the program after it is loaded. Therefore, use the CONTINUE statement when you want to run the program without destroying the variables. The LOAD command enters the LOCK mode after it has loaded the program text.

OLDLOAD

The OLDLOAD statement lets you load programs (file type 2) that were dumped using the old version (10.01.00) of Extended Benton Harbor BASIC. The OLDLOAD statement, unlike the new LOAD statement, destroys the program variables currently in memory. The form of the OLDLOAD statement is:

*OLDLOAD "name" ㊞

PUT

The PUT statement is a form of dump statement, in that it saves the program variables on tape (file type 5). It does not save the program, only the variables. The "name" that you give to the variables is written on the tape so you can reload the variables in the future using the specified name. The form of the PUT statement is:

*PUT "name" ㊞

The string "name" may consist of up to 80 ASCII characters. Any normal ASCII character string is permitted. Make sure the tape drive is ready before you enter the PUT statement. BASIC starts the drive, writes the data, and stops the drive. You can use CONTROL-C to abort the PUT routine; however, the file will not be complete. A program dumped using the PUT statement can only be loaded using the GET statement.

GET

The GET statement loads variables (file type 5), previously stored on tape, into memory. The current variables in memory are destroyed, but the program text is not affected. Since the variables are stored in mass storage under a specified name, you can load them from storage using their specified name. The form of the GET statement is:

*GET "name" ㊞

SURE?

A Y reply to the question "SURE?" causes BASIC to scan the mass storage device until it finds a variable (file type 5) whose name matches the specified string "name." It then destroys current variables in memory and loads the new variables. Any other response cancels the GET routine. If the name in mass storage device is longer than the specified name you enter, a match on the supplied characters in the string "name" is valid. Thus, a program may be dumped (PUT) with extra information entered in the name (such as program version number). This lets you load a program without entering the extra information. The GET command enters the LOCK mode after it has loaded the program variables.

FDUMP

The FDUMP statement is a combination of the DUMP and PUT statements in that it dumps both the program text and the variables (file type 6). The "name" that you give to the program is written on the tape so that you can reload the program and variables in the future using the specified name. The form of the FDUMP statement is:

*FDUMP "name" ㊞

The string "name" may consist of up to 80 ASCII characters. Any normal ASCII character string is permitted. Make sure the tape drive is ready before you enter the FDUMP statement. BASIC starts the drive, writes the data, and stops the drive. You can use the CONTROL-C to abort the FDUMP routine; however, the file will be incomplete. A program dumped using the FDUMP statement can only be loaded using the FLOAD statement.

FLOAD

The FLOAD statement is a combination of the LOAD and GET statements in that it loads both the program and its variables (file type 6) at the same time. Since the program and its variables are stored in mass storage under a specified name, you can load them from storage using the specified name. The form of the FLOAD statement is:

*FLOAD "name" ⏎

SURE?

A Y reply to the question "SURE?" causes BASIC to scan the mass storage device until it finds a program (file type 6) whose name matches the specified string "name." It then destroys the current program in memory and loads the new program text and program variables. Any other response cancels the FLOAD routine. If the name in the mass storage device is longer than the specified name you enter, a match on the supplied characters in the string "name" is valid. Thus, a program may be dumped (FDUMPed) with extra information entered in the name (such as program version). This lets you load a program without entering the extra information. The FLOAD command enters the LOCK mode after it has loaded the program text and the program variables.

LOCK

The LOCK statement protects your program by preventing the execution of the following command mode statements:

BUILD	SCRATCH
DELETE	CLEAR
LOAD	GET
RUN	FLOAD

It also prevents the entry or deletion of program text. Variables can be changed, but not deleted. The form of the LOCK statement is:

*LOCK ㊞

A lock error (LOCK) is generated if you attempt to enter a "locked out" command mode statement, such as RUN. Use the UNLOCK statement to abort the LOCK mode.

UNLOCK

The UNLOCK statement aborts the LOCK mode and restores the use of all command mode statements. The form of the UNLOCK statement is:

*UNLOCK ㊞

ERROR TABLE

The following new error listing is in addition to those errors given in the "BASIC Error Table" in the H8 Software Reference Manual.

BASIC	EXTENDED BASIC	COMMENTS (Cause of Error)
—	LOCK	Data LOCK engaged. Attempting to change data using the RUN, CLEAR, SCRATCH, BUILD, LOAD, DELETE, GET, or FLOAD commands. Or, attempting to add or delete lines of BASIC text.

OPTIONAL PATCHES

An optional patch is supplied with Extended Benton Harbor BASIC version 10.02.00. This option is for systems that use a terminal device that requires two stop bits (such as the Teletype Model ASR 33). Refer to the "Installing A Patch" section of the H8 Software Reference Manual if you need to use either of these optional patches.

EXTENDED BENTON HARBOR BASIC
10.02.XX.

OPTION PATCH #1

2 STOP BITS

USE: THIS PATCH IS INSERTED FOR SYSTEMS WHICH USE A TERMINAL DEVICE REQUIRING 2 STOP BITS. THIS SHOULD NOT BE USED FOR DEVICES WHICH CAN RUN WITH ONLY ONE STOP BIT.

NOTES: NONE.

041010 316
111261 001

PATCH

BASIC UTILITY ROUTINES

The following pages contain a description of several utility routines included in Extended Benton Harbor BASIC version 10.02.00. You can use them with user-written machine language routines called by the USR function. Refer to the section "Entry Points to BASIC Utility Routines" in this Manual.

```
000.000                  3          XTEXT   MTR


                        81  ***     BASIC - *WINTEK* BASIC INTERPRETER.
                        82  *
                        83  *       J. G. L., 09/76, FOR *WINTEK* CORPORATION,
                        84  *                              LAFAYETTE, IN.
                        85  *
                        86  *       H. W. S., 01/78, FOR HEATH COMPANY.
                        87  *                              BENTON HARBOR, MI.


                        89  ***     COPYRIGHT 09/1976, *WINTEK* CORPORATION,
                        90  *                          902 N. 9TH ST.
                        91  *                          LAFAYETTE, IN. 47901
                        92  *
                        93  *       COPYRIGHT 01/1978, HEATH COMPANY,
                        94  *                          BENTON HARBOR, MICH.
                        95  *                                         49022


                        97  **      LOW-MEMORY CELLS USED BY BASIC
                        98
040.064                 99          ORG     7*3+.UIVEC
                       100
040.064                101          DS      2               ACCX TYPE
040.066                102  ACCX    DS      4
040.072                103          DS      2               ACCY TYPE
040.074                104  ACCY    DS      4
```

```
                        107  **      USR - CALL USER ASSEMBLY LANGUAGE FUNCTION.
                        108  *
                        109  *       THE *USR* FUCTION IS ACTUALLY A CALL TO A USER-WRITTEN ROUTINE
                        110  *       WHICH MUST HAVE BEEN PREVIOUSLY LOADED INTO MEMORY BY THE USER.
                        111  *
                        112  *       THE ADDRESS OF THE FUNCTION'S ENTRY POINT MUST BE IN *USRFCN*.
                        113  *
                        114  *       BASIC MUST HAVE BEEN PREVIOUSLY CONFIGURED SO THAT THE USER
                        115  *       FUNCTION RESIDES IN MEMORY ABOVE THE STACK POINTER (HIGH MEMORY)
                        116  *       OR ELSE BASIC WILL OVERLAY THE FUNCTION WITH DATA.
                        117  *
                        118  *       THE FUNCTION IS ENTERED WITH A POINTER TO THE SINGLE ARGUMENT (IN
                        119  *       FLOATING POINT), AND MAY RETURN A FLOATING POINT OR STRING ARGUMENT.
                        120  *       IF NO RETURN VALUE IS PLACED IN *ACCX*, THEN THE ORIGINAL ARGUMENT
                        121  *       REMAINS THERE, AND USR( RETURNS ITS ARGUMENT AS ITS VALUE.
                        122  *
                        123  *       ENTRY   (BC) = #ACCX
                        124  *       EXIT    (ACCX) CONTAINS VALUE
                        125  *       USES    ALL
                        126
                        127
                        128
---.---  000 000        129  USRFCN  DW      0               ADDRESS FO USER FUNCTION ENTRY POINT
                        130  *                               IF = 0, USR( IS NOT LEGAL
```

```
                    133  **      ERROR PROCESSING.
                    134  *
                    135  *       THESE ERROR PROCESSORS ARE ENTERED WHEN AN ERROR IS DETECTED.
                    136  *
                    137  *       CONTROL PASSES DIRECTLY BACK TO COMMAND MODE.
                    138
                    139
---.---             140  ERR.CC  EQU     *               CONTROL-C
                    141
---.---             142  ERR.CB  EQU     *               CONTROL-B
                    143
---.---             144  ERR.DE  EQU     *               DATA EXHAUSTED
                    145
---.---             146  ERR.DO  EQU     *               /0
                    147
---.---             148  ERR.IN  EQU     *               ILLEGAL NUMBER
                    149
---.---             150  ERR.IU  EQU     *               ILLEGAL USAGE
                    151
---.---             152  ERR.LK  EQU     *               DATA LOCK ENGAGED
                    153
---.---             154  ERR.NV  EQU     *               NEXT VARIABLE MISSING
                    155
---.---             156  ERR.OV  EQU     *               OVERFLOW
                    157
---.---             158  ERR.RE  EQU     *               RETURN ERROR
                    159
---.---             160  ERR.SL  EQU     *               STRING LENGTH
                    161
---.---             162  ERR.SN  EQU     *               STATEMENT NUMBER
                    163
---.---             164  ERR.SY  EQU     *               SYNTAX ERROR
                    165
---.---             166  ERR.TC  EQU     *               TYPE CONFLICT
                    167
---.---             168  ERR.TO  EQU     *               TABLE OVERFLOW
                    169
---.---             170  ERR.SR  EQU     *               SUBSCRIPT RANGE
                    171
---.---             172  ERR.SC  EQU     *               SUBSCRIPT COUNT
                    173
---.---             174  ERR.ND  EQU     *               NOT DIMENSIONED
                    175
---.---             176  ERR.IC  EQU     *               ILLEGAL CHARACTER
                    177
---.---             178  ERR.UD  EQU     *               UNDEFINED FUNCTION
                    179
---.---             180  ERR.TP  EQU     *               TAPE ERROR
```

```
                    183  **      CVX - COPY VALUE INTO 'X' ACCUMULATOR.
                    184  *
                    185  *       CVX COPIES A 4 BYTE VALUE INTO THE X ACCUMULATOR.
                    186  *
                    187  *       ENTRY   (DE) = ADDRESS OF VALUE
                    188  *       EXIT    COPIED
                    189  *       USES    A,F
                    190
                    191
---.---             192  CVX     EQU     *


                    194  **      CXY - COPY (ACCX) TO (ACCY)
                    195  *
                    196  *       ENTRY   NONE
                    197  *       EXIT    NONE
                    198  *       USES    A,F,D,E
                    199
                    200
---.---             201  CXY     EQU     *


                    203  **      CXV - COPY X TO VALUE.
                    204  *
                    205  *       CXV COPIES THE CONTENTS OF THE 'X' ACCUMUALTOR INTO A MEMORY
                    206  *       LOCATION.
                    207  *
                    208  *       ENTRY   (DE) = TARGET ADDRESS
                    209  *       EXIT    COPIED
                    210  *       USES    A,F
                    211
                    212
---.---             213  CXV     EQU     *


                    215  **      IFIX - SPLIT NUMBER INTO INTEGER AND FRACTION.
                    216  *
                    217  *       IFIX FIXES ((DE)) INTO IN INTEGER.
                    218  *
                    219  *       ENTRY   (DE) = ADDRESS OF NUMBER
                    220  *       EXIT    (DE) = INTEGRAL PART OF 0<=N<=65535
                    221  *               TO ERR,IN OTHERWISE
                    222
                    223
---.---             224  IFIX    EQU     *
```

```
                    226  **      IFLT - FLOAT NUMBER.
                    227  *
                    228  *       ENTRY   (DE) = VALUE
                    229  *       EXIT    (ACCX) = NUMBER VALUE
                    230  *               (DE) = #ACCX-1
                    231
                    232
---.---             233  IFLT    EQU     *



                    235  **      TDI - TYPE DECIMAL INTEGER.
                    236  *
                    237  *       TDI TYPES AN INTEGER AS A 5 PLACE NUMBER. LEADING ZEROS ARE
                    238  *       SUPPRESSED.
                    239  *
                    240  *       ENTRY   (DE) = NUMBER
                    241  *       EXIT    TYPED
                    242  *       USES    A,F,D,E
                    243
                    244
---.---             245  TDI     EQU     *



                    247  **      XCY - EXCHANGE (ACCX) WITH (ACCY)
                    248  *
                    249  *       ENTRY   NONE
                    250  *       EXIT    NONE
                    251  *       USES    A,F
                    252
                    253
---.---             254  XCY     EQU     *



                    256  **      ZRO - ZERO MEMORY.
                    257  *
                    258  *       ZRO ZEROS A FIELD OF MEMORY.
                    259  *
                    260  *       ENTRY   (HL) = ADDRESS
                    261  *               (DE) = COUNT
                    262  *       EXIT    NONE
                    263  *       USES    A,F,D,E,H,L
                    264
                    265
---.---             266  ZRO     EQU     *
```

```
                ;**      H8 FLOATING POINT FORMAT:
                ;*
                ;*      SINGLE-PRECISION FLOATING POINT NUMBERS ARE REPRESENTED
                ;*      BY A 4-BYTE VALUE.  THE NUMBER CONSISTS OF A 3 BYTE
                ;*      TWO'S COMPLEMENT MANTISSA, AND A ONE-BYTE BIASED BINARY
                ;*      EXPONENT, THE NUMBER FORMAT IS:
                ;*
                ;*
                ;*      N+0     LEAST SIGNIFICANT MANTISSA BYTE
                ;*      N+1     MID   SIGNIFICANT MANTISSA BYTE
                ;*      N+2     MOST  SIGNIFICANT MANTISSA BYTE
                ;*      N+3     BIASED EXPONENT
                ;*
                ;*
                ;*      EXPONENT:
                ;*      ---------
                ;*
                ;*      EACH FLOATING POINT NUMBER CONTAINS A BINARY EXPONENT.
                ;*      THE EXPONENT CONTAINS A BIAS OF 128 (200 OCTAL).
                ;*
                ;*      THUS AN EXPONENT OF 0 IS ENTERED AS 200Q, AN EXPONENT OF -10
                ;*      CODED AS 166Q, ETC. THE NUMBER ZERO IS TREATED AS A
                ;*      SPECIAL CASE: ITS EXPONENT (AND MANTISSA) IS ALWAYS 0.
                ;*
                ;*
                ;*      MANTISSA:
                ;*      ---------
                ;*
                ;*      THE MANTISSA OCCUPYS 3 BYTES, FOR A TOTAL OF 24 BITS. THE NUMBERS
                ;*      ARE STORED IN TWO'S COMPLEMENT NOTATION.  THE HIGH ORDER
                ;*      BIT IS THE SIGN BIT, THE NEXT BIT (100Q) IS THE MOST SIGNIFICANT
                ;*      DATA BIT.  NOTE THAT FLOATING POINT NUMBERS SHOULD ALWAYS
                ;*      BE NORMALIZED, SO THAT THE MOST SIGNIFICANT DATA BIT IS THE
                ;*      OPPOSITE OF THE SIGN BIT'S VALUE.
                ;*
                ;*      THE FLOATING POINT ROUTINES SUPPLIED WILL NOT OPERATE ON NON-
                ;*      NORMALIZED DATA.
                ;*
                ;*      EXAMPLES:
                ;*      ---------
                ;*
                ;*      DECIMAL NUMBER          M3  M2  M1  EX
                ;*
                ;*       1.0                    000 000 100 201
                ;*       0.5                    000 000 100 200
                ;*       0.25                   000 000 100 177
                ;*      10.0                    000 000 120 204
                ;*       0                      000 000 000 000
                ;*       0.1                    146 146 146 175
                ;*      -1.0                    000 000 200 200
                ;*      -0.5                    000 000 200 177
                ;*     -10.0                    000 000 260 204
```

```
                        323  *       FPADD - FLOATING POINT ADD.
                        324  *
                        325  *       ACCX = ACCX + (DE)
                        326  *
                        327  *       ENTRY   (DE) = POINTER TO 4 BYTE FP VALUE
                        328  *       EXIT    ACCX = RESULT
                        329  *               SUPPLIED VALUE UNCHANGED
                        330  *       USES    A,F
                        331
                        332
    ---.---             333  FPADD   EQU     *


                        335  **      FPSUB - FLOATING POINT SUBTRACT.
                        336  *
                        337  *       FPSUB COMPUTES (DE) - ACCX
                        338  *
                        339  *       ENTRY   (DE) = POINTER TO 4 BYTE FP VALUE
                        340  *       EXIT    ACCX = RESULT
                        341  *               SUPPLIED VALUE UNCHANGED
                        342  *       USES    A,F
                        343
                        344
    ---.---             345  FPSUB   EQU     *


                        347  **      FPNRM - FLOATING POINT NORMALIZE.
                        348  *
                        349  *       FPNRM NORMALIZES THE CONTENTS OF (ACCX).
                        350  *
                        351  *       ENTRY   NONE
                        352  *       EXIT    (ACCX) NORMALIZED
                        353  *       USES    A,F
                        354
                        355
    ---.---             356  FPNRM   EQU     *


                        358  **      FPNEG - FLOATING POINT NEGATE.
                        359  *
                        360  *       FPNEG NEGATES THE CONTENTS OF ACCX.
                        361  *
                        362  *       ENTRY   NONE
                        363  *       EXIT    (ACCX) = -(ACCX)
                        364  *       USES    A,F
                        365
                        366
    ---.---             367  FPNEG   EQU     *
```

```
                        369  **      FPMUL - FLOATING POINT MULTIPLY.
                        370  *
                        371  *       ENTRY   (DE) = ADDRESS OF Y
                        372  *       EXIT    ACCX = ACCX * Y
                        373  *       USES    A,F
                        374
                        375
    ---.---             376  FPMUL   EQU     *
```

```
                        378  **      FPDIV - FLOATING POINT DIVIDE.
                        379  *
                        380  *       ACCX = ACCX/Y
                        381  *
                        382  *       ENTRY   (DE) = POINTER TO Y
                        383  *       EXIT    (ACCX) = RESULT
                        384  *       USES    A,F
                        385
                        386
    ---.---             387  FPDIV   EQU     *
```



```
                        390  **      ATF - ASCII TO FLOATING.
                        391  *
                        392  *       ATF CONVERTS AN ASCII STRING INTO A FLOATING POINT VALUE
                        393  *       IN ACCX.
                        394  *
                        395  *       SYNTAX
                        396  *
                        397  *        NNNN [.NNN] [E [+-] NN]
                        398  *
                        399  *       ENTRY   (HL) = ADDRESS OF TEXT
                        400  *       EXIT    (HL) UPDATED
                        401  *               (ACCX) = VALUE
                        402  *       USES    A,F,H,L
                        403
                        404
    ---.---             405  ATF     EQU     *
```

```
                                407  **      FTA - FLOATING TO ASCII.
                                408  *
                                409  *       FIA CONVERTS A FLOATING POINT NUMBER INTO AN ASCII
                                410  *       REPRESENTATION.
                                411  *
                                412  *       ENTRY   (ACCX) = VALUE
                                413  *               (HL) = ADDRESS TO STORE TEXT
                                414  *       EXIT    (A) = LENGTH OF STRING DECODED
                                415  *               (DE) = ADDRESS OF LAST BYTE
                                416  *       USES    A,F,D,E
                                417
                                418
   ---.---                      419  FTA     EQU     *
```



```
                                422  **      FLOATING POINT VALUES.
                                423  *
                                424
                                425          LON     G               LIST GENERATED BYTES
                                426
   ---.---  000 000 100         427  FP1.0   DB      0,0,100Q,201Q
            201
                                428
   ---.---  000 000 120         429  FP10.   DB      0,0,120Q,204Q
            204
                                430
   ---.---  000 000 000         431  FP0.0   DB      0,0,0,0
            000
                                432
   ---.---  022 170 233         433  NPI.2   DB      022Q,170Q,233Q,201Q     -PI/2
            201
                                434
   ---.---  022 170 233         435  NPI2    DB      022Q,170Q,233Q,203Q     -PI*2
            203
                                436
   ---.---  022 170 233         437  NPI     DB      022Q,170Q,233Q,202Q     -PI
            202
                                438
   ---.---  022 170 233         439  NPI.4   DB      022Q,170Q,233Q,200Q     -PI/4
            200
                                440
   ---.---  356 207 144         441  PI.4    DB      356Q,207Q,144Q,200Q     PI/4
            200
                                442
                                443
   ---.---                      444          END
ASSEMBLY COMPLETE
   444 STATEMENTS
     0 ERRORS DETECTED
 41178 BYTES FREE
```

ENTRY POINTS TO BASIC UTILITY ROUTINES

ADDRESSES FOR

EXTENDED BENTON HARBOR BASIC
ISSUE # 10.02.

ACCX	040066	ACCY	040074	ATF	101070	CVX	067374
CXV	070023	CXY	070007	ERR.CH	065115	ERR.CC	065105
ERR.DO	065136	ERR.DE	065125	ERR.IC	065333	ERR.IN	065143
ERR.IU	065151	ERR.LK	065157	ERR.ND	065324	ERR.NV	065166
ERR.OV	065175	ERR.RE	065206	ERR.SC	065311	ERR.SL	065214
ERR.SN	065226	ERR.SR	065274	ERR.SY	065237	ERR.TC	065245
ERR.TO	065262	ERR.TP	065355	ERR.UD	065346	FP0.0	106135
FP0.1	106131	FP1.0	106121	FP10.	106125	FPADD	076125
FPDIV	100025	FPMUL	077070	FPNEG	077047	FPNRM	076347
FPSUB	076333	FTA	101355	IFIX	070340	IFLT	070376
NPI	106151	NPI.2	106141	NPI.4	106155	NPI2	106145
PI.4	106161	TDI	074046	USRFCN	106106	XCY	074341
ZRO	074373						