SETUP AND DECODE IOT'S

TABLE OF CONTENTS

13 October 1966

SETUP AND DECODE IOT'S

GENERAL FORM OF SETUP IOT'S

All the setup IOT's work in the following way unless described differently in the individual specifications. A setup IOT accepts the number to be set up in the AC. It converts it to a text string and stores it according to the character pointer found in STS. The setup IOT adds an EOM and leaves STS pointing to the EOM. (Note: In order to retain EOM's between successive setups, execute a LCH I STS between them).

Setup IOT's have only 1 return. The general form of a setup IOT is this:

```
LAC  (NUMBER                    /character ptr to buffer is STS
SETUP
R1
```

The setup IOT's are transparent to the AC and IO.

Any setup IOT may type out instead of setting up the output string in core. It is subject to the Job Hunter IO suppression flag JMODE -- unless JMODE is positive, nothing will be typed out. If it receives a Teletype error while outputting, it traps to TTTSU. The address of the IOT will be found in TRAPPC.

SETUP NUMBER

```
LAC  (NUMBER
SNM                             /set up number
R1
```

SNM converts a number to a decimal text string.

SNM+1 /type

SNM+1 types out the number.

SNM+2 /octal

SNM+2 sets up the number as an unsigned octal number.

SNM+4 /set up a + sign

SNM+4 sets up a + sign instead of a space if the number is positive.

SNM+1Ø /set up no sign for positive

SNM+1Ø omits the sign for a positive number.
(SNM+1Ø overrides SNM+4)

SNM+2Ø /zeroes instead of spaces

SNM+2Ø supplies zeroes instead of spaces when used with the following option:

SNM+4Ø /right-adjust

SNM+4Ø will columnate numbers. The IO contains the length of the string to be set up. If the number of characters specified is more than the number of characters available, SNM+4Ø fills out with spaces between the sign and the number. If the number of characters specified is smaller than the actual number of characters, the whole number is set up anyway.

SETUP UNIT NUMBER

```
LAC  (XXX          /xxx (character pointer in STS)
LIO  (XXXX         /-xx-xx
SUN                /set up unit number
R1                 /AC + IO unchanged, STS updated
```

SUN accepts a unit number in standard internal format, first three digits in the AC, last four in the IO, and sets it up in standard external format (xxx-xx-xx).

```
SUN+1
```

SUN+1 types out.

SETUP TIME AND DATE

```
LAC  (DATE         /set up time and date (char. ptr. in STS)
LIO  (TIME
STD                /AC, IO unchanged
```

STD accepts the date in the AC and time in the IO and sets up a text string of the following form:

3:25 PM 3/6/1864

```
STD+1              /type
```

STD+1 types.

```
STD+2              /compress date
```

STD+2 sets up a string of the form:

3:25 PM 3/6

STD+2 ignores year completely.

```
STD+4              /compress time
```

STD+4 sets up a string of the form:

12A␣3/6/1864

STD+4 throws away minutes.

STD+1Ø /suppress date

STD+1Ø sets up only the time
(STD+1Ø overrides STD+2)

STD+2Ø /suppress time

STD+2Ø sets up only the date
(STD+2Ø overrides STD+4 and STD+1Ø)

STD+4Ø columnates

STD+4Ø sets up the time and date like this:

␣5:49␣AM␣␣4/13/1966
12:49␣AM␣12/06/1966

STD+42 ␣1:23␣PM␣␣1/23

STD+44 does not columnate.

GENERAL FORM OF DECODE IOT'S

All the decode IOT's work in the following way unless described differently in the individual specifications. A decode IOT converts the character string to an octal number, returning the result in the AC. Decode IOT's are called with a character pointer to the input string in FSA.

The general form of the IOT is:

```
DECODE          /(FSA)
RETURN 1        /illegal format
RETURN 2        /result in AC
```

DNM expects the character string pointed at by FSA to contain numbers, i.e., Ø-9. It converts this text, character by character, until it reaches a non-numeric character (not Ø-9). It leaves FSA pointing to the first non-numeric character. If the input string is not the format expected by the IOT, it gives return 1. The AC, IO, and FSA are unchanged. Otherwise, return 2 is given and the result is returned in the AC. The IO is transparent and FSA is updated so that a LCH I FSA will give the first uninterpreted character.

DECODE NUMBER

```
DNM             /decode number (FSA)
R1              /AC = Ø illegal format, FSA unchanged
                /AC = -Ø overflow, FSA points to the
                      first non-numeric character
R2              /number in AC, FSA updated
```

DNM decodes a decimal number. There are two error conditions that cause return 1. They are differentiated by the state of the AC.

1) AC = Ø illegal format
2) AC = -Ø overflow

The maximum allowable absolute value for decimal numbers is

131071, and the maximum allowable value for octal numbers is 777777. Any larger number causes an overflow condition.

DNM terminates on any character other than a digit although it may accept a sign followed by spaces before the number. An initial space is not accepted.

DNM+1

The contents of the AC, which must be positive, specify the inclusive maximum magnitude to be used in decoding this number.

DNM+2

DNM+2 decodes an octal number. The overflow test for an octal number treats the word as an unsigned 18 bit quantity. (If you plan to use this variation with the previous one, note that there are no negative octal numbers -- only very large positive ones).

DNM+4

DNM+4 considers a minus sign illegal before a number.

DNM+10

DNM+10 considers a plus sign illegal before a number.

DECODE UNIT NUMBER

```
DUN                    /decode unit number (FSA)
R1                     /illegal format, FSA, IO, AC
                        unchanged
R2                     /unit number in AC and IO, FSA
                        updated
```

DUN accepts a number only in the form

XXX-XX-XX

On a good return the AC holds the first 3 digits as an ordinary binary number; the IO the last 4 digits.

DECODE TIME

```
DTM     /decode time (FSA)
R1      /illegal format; AC, IO, FSA unchanged
R2      /time in AC; IO transparent; FSA updated
```

DTM decodes a time string.

DTM returns the time (minutes since midnight) in the AC.

The allowable input formats for time are, in example form:

3:25P
3:25 P
3:25PM
3:25 PM
} meaning 3:25P

3P, 3PM or 3 PM meaning 3:00P

12A, 12 A, 12 AM meaning Midnight

12P, etc. meaning Noon

T (time taken from lower core)

DECODE DATE

```
DDT     /decode date (FSA)
R1      /illegal format; everything unchanged
R2      /date in AC; IO transparent; FSA updated
```

DDT decodes a date string

DDT returns the date (days since 1 January 1849) in the AC.

The allowable input formats for date are the following:

3/6	meaning 3/6 current year
3/6/64	meaning 3/6/1964
3/6/1864	meaning 3/6/1864
3/6/ØØ	meaning 3/6/19ØØ
T	meaning today (taken from lower core)

```
T1          meaning tomorrow
T2          meaning day after tomorrow
Tn          meaning n days hence
Y1          meaning yesterday
Y2          meaning day before yesterday
Y4          meaning n days ago
```

BREAKDOWN TIME AND DATE

```
LAC  (DATE      /days since 1 January 1949
LIO  (TIME      /minutes since midnight
TDNUM
TBUF            /ptr to 5-word buffer
R1
```

TDNUM accepts the date and time in the AC and IO in internal format and breaks it down into hour, minute, month, day, and year. The location following the IOT contains a traceable (indirect bit is meaningful) word pointer to a five register buffer area.

TDNUM sets up the buffer in this way:

1) hour
2) minute
3) month
4) day of month
5) year

EDIT

```
LAW BUF         /character pointer in the AC
EDIT
R1              /RUBOUT terminated string
R2              /EOM terminated string
```

EDIT scans the designated input string removing characters deleted by \. If a rubout terminates the string, EDIT replaces it with an EOM and gives return 1. In either case the AC contains an updated character pointer to the character past the EOM. The IO is unchanged.

```
EDIT+4Ø         /leaves the AC unchanged.
```

EDIT scans the designated input string removing characters deleted by\. If a rubout terminates the string, EDIT replaces it with an EOM and gives return 1. In either case the AC contains an updated character pointer to the character past the EOM. The IO is unchanged.