8. IBM MACROS

8.1 TIME Macro

The `TIME` macro can be used to obtain the current date and time from the operating system. The macro is located in the `SYS1.MACLIB` macro library and may be invoked in an Assembler program if the library is made available on the `SYSLIB DD` statement in the Assembler step.

Use the following format:

```
label   TIME   DEC,stor-location,LINKAGE=SYSTEM,DATETYPE=MMDDYYYY
```

`DEC`

The keyword parameter `DEC` requests that the time and date be returned in packed decimal format but it is WITHOUT a sign digit.

`stor-location`

The name of a 4-fullword (16 byte) storage area where the time and date will be placed by the macro. It must be defined in the program's storage area.

`LINKAGE=SYSTEM`

A keyword parameter specifying the type of linkage to use. Use `SYSTEM` instead of `SVC` as recommended by IBM for application programs.

`DATETYPE`

The keyword parameter `DATETYPE` specifies the format in which the date portion of the time and date is returned. Among others, the choices are `YYYYDDD` (Julian date) or `MMDDYYYY` (standard format with four-digit year).

Example of the Returned Time and Date

The following is a dump of the four fullwords of stor-location after calling `TIME` at `13:00:17:75` on 03/31/2014:

```
13001775 33790000 03312014 00000000
```

Using the Time Returned:

The first fullword holds the time of day in military time in the form of `HHMMSShh` with two digits for the hour with range 00 to 23, two digits for minutes with range 00 to 59, two digits for seconds with range 00 to 59, and two digits for 100ths of seconds with range 00 to 99.

For example, if the time is 5:05:23:33 PM, the first eight digits of the time will be returned in four bytes as:

```
17052333
```

The `ED` instruction can be used to "edit" the time into an output field. It CANNOT be used in a packed decimal instruction, though, as it has no sign digit and would therefore cause a `S0C7` – Data Exception.

Using the Date Returned:

If the date is December 15, 2013, the third fullword of the 16-byte `stor-location` will be `12152013` and, if the date is June 2, 2013, the third fullword of the 16-byte `stor-location` will be `06022013`.

In most cases, you want the leading 0 to print regardless. To do this, declare the receiving field 13 bytes long instead of 10. Set significance ON in the third byte of the 13 and use the `ED` instruction.

For example:

```
         MVC   SETDATE(13),=X’40202120206120206120202020’  X’61’ = ‘/’
*
         ED    SETDATE(13),DATETIME+7
         MVC   OUTDATE(10),SETDATE+3
*
DATETIME DS    4F             STORAGE FOR THE TIME MACRO
*
SETDATE  DS    CL13           TEMPORARY STORAGE FOR FORMATTING DATE
*
HEADER1  DC    C’1’           TOP OF PAGE HEADER
OUTDATE  DS    CL10           WHERE THE DATE GOES
         ...etc.
```

8.2 SNAP Macro

The IBM macro named SNAP is useful for debugging purposes. It can be used to print out the contents of registers, the PSW, specified regions in storage, or any of many control blocks.

JCL Requirement:

Because the SNAP writes output, it requires a DD statement. This is often declared as:

```
//ddname   DD SYSOUT=*
```

where ddname may be any name except SYSABEND, SYSMDUMP, or SYSUDUMP.

DCB Requirement:

So that it can produce the output requested, SNAP also requires its own DCB declared in the following format:

```
dcbname  DCB   DDNAME=ddname,                                          X
               DSORG=PS,                                               X
               RECFM=VBA,                                              X
               LRECL=nnn,                                              X
               BLKSIZE=mmm,                                            X
               MACRF=(W)
```

where we have the usual X in column 72 to indicate continuation.

Notice that we have variable-length blocked records with ASA Carriage control.

Simply declare LRECL=125 and either BLKSIZE=882 or 1632. SNAP will normally print 120 characters per line in a standard dump.

Before issuing the SNAP, the DCB should be opened for output, and it should be closed at some point afterward.

Register Usage:

SNAP will normally change the values of registers 0, 1, 14 and 15.

Return Code Values:

SNAP provides a return code in register 15. The values are 00 if all went well, 04 if the DCB was not opened before the SNAP was issued, or larger values for more serious errors.

Format:

SNAP has many options. The following covers only a few of them. This discussion assumes the program is running in 24-bit mode.

```
         SNAP  DCB=dcbaddress,options
```

`dcbaddress` represents either a label or a `D(X,B)` address.

`PDATA` Option

```
PDATA=choice
```

or

```
PDATA=(choice1,choice2,...)
```

Examples of choice are:

`PDATA=PSW`	Dump the PSW at the time the `SNAP` was issued.
`PDATA=REGS`	Dump the contents of all registers at the time the `SNAP` was issued.
`PDATA=SAH`	Print save area linkage information and program call linkage information.
`PDATA=SA`	Same as `SAH`, plus a back trace through save areas.

`STORAGE` Option

```
STORAGE=(start address,end address)
```

or

```
STORAGE=(start address1,end address1,start address2,end address2,...)
```

The starting and ending addresses are rounded down and up, respectively to the nearest fullword boundaries.

`STRHDR Option`

```
STRHDR=hdr address
```

or

```
STRHDR=(hdr address1,hdr address2,...)
```

This enables printing headers for the storage that is to be dumped. Each header address is the address of a one-bye header length field, which is immediately followed by the test of the header (up to 100 bytes long). The first header address is matched with the first region of storage, the second with the second, etc.

If there are several fields listed in `STORAGE`, and the printing of a header for each is not desired, a comma can be coded as a placeholder as in the following example:

```
         SNAP  DCB=(8),                                           X
               STORAGE=(ITEM1,ITEM1END,ITEM2,ITEM2END),           X
               STRHDR=(,LABEL1)
```

where in storage there is

```
LABEL2   DC    X'5'
         DC    C'ITEM2'
```

Here there is no label for `ITEM1`, and but there are labels `'ITEM2'` for `ITEM2`.

8.3 ABEND Macro

The `ABEND` macro is an IBM macro used to terminate a running program abnormally.

The format is as follows:

```
[label]  ABEND ccode[,REASON=nnn][,options]
```

where

`ccode` = completion code = a number 0 to 4095, decimal or hexadecimal, or a register value such as (R1) through (R12).

The options include:

REASON=reason code This is a numerical value (31-bit decimal or 32-bit hexadecimal or register value (R2) through R12)).

(The following three parameters are positional, in this order.)

DUMP This requests a dump of the storage areas and control blocks belonging to the program being terminated.

STEP This indicates that the entire job step is to be terminated; this might be needed if, for instance, the ABENDing program is a child process of another program. Without STEP, only the child process is terminated.

code type This indicates whether the completion code should be treated as a USER or SYSTEM code. Valid values are USER and SYSTEM; the default is USER.

Examples:
```
ABEND 253

ABEND 317,DUMP

ABEND X'0C7',,,SYSTEM

ABEND (R5),DUMP,,USER

ABEND 101,REASON=X'C'
```

Note: the ABEND macro also has other options involving, for instance, user-written error recovery routines.

If we have a SYSABEND, SYSMDUMP, or SYSUDUMP DD statement, the DUMP parameter will be ignored.

Normally the completion code and reason code (if any) are printed in the JES job log. If the ABENDing program is a child process of another program (created, for instance, using the ATTACH macro), then the completion code and reason code are otherwise handled.

See *Z/OS Programming: Assembler Services Reference* for more details and related macros.