XDUMP/XSNAP
360 ASSEMBLER LANGUAGE          writeup revised January  1973 (v.5.0)

EXTENDED  SNAP  AND  DUMP  MACROS

INTRODUCTION

       XDUMP provides a very simple command for obtaining either a dump of
the general purpose registers, or of a single area of main storage.  The
output is identified by printing the location at  which  the  XDUMP  was
invoked.  XDUMP is a special case of XSNAP, and calls it to create code.

       XSNAP is a much more flexible macro.  General purpose registers may
be stored for later inspection if it is not desired to have them printed
immediately.  In addition to the  printing  of  the  GP  registers,  the
floating point registers and any number of storage areas can be  printed
by one XSNAP, in any desired combination.  Any  storage  areas  may  be
displayed, and may be specified  by  labels,  base-displacements,  dummy
section symbols.  An optional label may be provided to further  identify
the output.  XSNAPs may be cancelled at assembly  time  using  the  XSET
macro, and may be made conditional at execute time by  making  tests  on
values in registers or storage.  XSNAP modifies  neither  the  condition
code nor the registers, and can thus be used at any point in  a  program
at which addressibility exists.

TYPICAL USAGE

a)   Print GP registers (XDUMP with no operands).

          XDUMP

b)   Print a single block of storage (up to 4095 bytes).

          XDUMP AREA,LENGTH    LENGTH bytes starting at AREA

c)   Print  GP  registers,  floating  registers,  and  a  storage  area.
     (labeled as shown, with 100 bytes starting at X).

          XSNAP T=FL,LABEL='REGS,AREA X',STORAGE=(X,X+100)

d)   Print 2 storage areas, no registers.  Areas are  specified  by  RX-
     type addresses in this case, including DSECT  symbols  X1  and  X2.

          XSNAP T=NO,STORAGE=(*0(R1,R2),*40(R1,R2),*X1,*X2)

e)   Print (GP registers in this case) only when specified test is  met,
     in this case, only if value in register 2 is  greater  than  100  .

          XSNAP LABEL='AFTER 100TH TIME IN LOOP',IF=((2),H,=F'100')

XDUMP

USE

XDUMP can be used in one of two basically different ways:

a)   GENERAL PURPOSE REGISTER DUMP

        XDUMP

Coding XDUMP with no operands prints the contents of the user's general purpose registers, in hexadecimal notation. The registers are preceded by a header line like the following:

BEGIN XSNAP - CALL    # AT CCAAAAAA USER REGISTERS

#    is the number of calls made to XDUMP so far, for identification.

CCAAAAAA shows the last 32 bits of the user's PSW, in hexadecimal.

CC   gives the ILC, CC, and Program Mask at the time of the XDUMP.

AAAAAA gives the approximate address of the XDUMP macro expansion, and thus can be used to distinguish between the output of different XDUMP statements. *NOTE* XDUMP , is the same as XDUMP with no operand.

b)   STORAGE DUMP

        XDUMP area,length

area      is any RX-type address (anything allowed in a LA instruction)

length   is an absolute expression having value from 1 to 4095.

Coding XDUMP with an address and length produces a dump of a user storage area, beginning at the address given by area, and ending at address area+length.

The resulting output includes a header line like the above, followed by a hexadecimal and alphanumeric dump of the selected storage area. The storage is printed in lines showing two groups of four fullwords, preceded by the memory address of the first word in each line, and followed by the alphanumeric representation of the 32 bytes on the line, with letters, numbers, and blanks printed directly, and all other characters translated to periods. The storage printed is also preceded by a line giving the address limits specified in the XDUMP.

If the length is omitted, the value 4 is used as a default.

EXAMPLES OF XDUMP USAGE

        XDUMP AREA+10,80
        XDUMP 8(1,4),100
        XDUMP FULLWORD            use default value of 4
        XDUMP TABL(3),12

                         XSNAP

USE

     XSNAP may be coded with any of the operands shown,  in  any  order,
since all are keyword operands.  It is called as follows:

label    XSNAP T=,LABEL=,STORAGE=,IF=

label     is an optional statement label.


T=          (Type of action to be performed for registers)

T=PRINT   requests that the GP registers be printed, as given by XDUMP..
     Storage areas may of course  be  printed  in  addition  (STORAGE=).

T=NOREGS  requests that GP registers NOT be  printed.   If  no  STORAGE=
     is specified, this produces exactly 1 line of output, which can  be
used for program tracing.

T=FLOAT   requests both GP and floating point registers to  be  printed.

T=(FLOAT,NOREGS)   requests floating point registers, but NOT GP  regs.

T=STORE   causes the GP registers to be saved into a well-labeled  area,
     but does not create any  printed  output.   The  register  area  is
originally  generated  filled  with  -1's  (hexadecimal 'FF's), and  is
immediately preceded by the LABEL= string if it exists, or by the  label
generated on the register area (of form XX####B), if no LABEL= was used.
The area is followed by 'XXXX' to make it easy to  find.   By  examining
such an area in a dump, the user can immediately determine:

a) whether control ever passed through a given point (register area  has
values other than X'FF').

b) the contents of the registers the LAST time  control  passed  through
the given XSNAP.

     Note that the LABEL= option helps identify each XSNAP.   If  it  is
not used, the XX####B label placed before the area also appears  in  the
assembly Cross-Reference listing, so it is still easy  to  locate. Note
that this option is especially useful for  placing  inside  heavily-used
loops in a program in  which  intermittent/unpredictable  errors  occur.

     Any of the types above may be abbreviated by the first two letters:
PRINT = PR, NOREGS = NO, FLOAT = FL, STORE = ST .

**DEFAULT**   T=PRINT  is  the  default  value  used  if  not  supplied.


LABEL='string'      (identification  LABEL  printed  as  XSNAP  heading)

     The 'string' is any character string usable as a C-type constant in
a DC statement, and is used as the XSNAP heading if it  prints  anything
at all.  The register area label (XX####B) is used if LABEL= is omitted.

STORAGE=(list)        (areas of main STORAGE to be dumped)

This option accepts a list of ADDRESS PAIRS, each  pair  specifying
the limits of an area of memory to be printed, from  the  address  given
by the first operand to that given by the second, from the third to  the
fourth, etc.  There may be any even number of  addresses,  separated  by
commas.  Each of the address specifications in each pair may be  written
in either of the following forms (in any combination):

a)   any label or expression  usable  in  an  A-type  address  constant.

b)   *expression , where expression is 40 or less characters  long,  and
is acceptable as the second operand of a Load Address instruction. Thus,
base-displacement forms, doubly-indexed expressions, and  dummy  section
symbols can all be used if preceded by an * .  The following  is a  legal
example of a STORAGE= operand:

     STORAGE=(A,B+7000,*4(2,R3),*AREA+3(4),*DSECT1,*DSECT1A,A,*A)


IF=(opa,relation,opb)     (perform  XSNAP  only  IF  condition   exists)

Operands opa and opb are compared in some way.   The  XSNAP   prints
output only whenever the relation between  the  two  operands  is  true.
The relation is specified using the same codes as the Extended Mnemonics
of the Branch on Condition:  H,L,E,O,P,M,Z,NH,NL,NE,NO,NP,NM,NZ.   Thus,
an XSNAP WITH  IF=(FLAG,NE,1) dumps only when value of FLAG is Not Equal
to 1.  The operands are compared in different ways depending on  whether
they are registers, storage, or immediate operands.  The  3  cases  are:

| OPERAND FORMS | COMPARISON GENERATED | MEANING |
|---|---|---|
| (opa)  (opb) | CR   (opa),(opb) | opa, opb both designate registers |
| (opa)  opb | C   (opa),opb | opa: register, opb: storage  fullword |
| opa    opb | CLI opa,opb | opa: address, opb: Immediate  operand |

For flexibility, adding a fourth operand to the sublist  allows  it
to be used as the opcode in the comparison:   IF=(TAG,O,X'F0',TM)  would
generate  TM TAG,X'F0'  and print only if  result  is  Ones  condition.

The IF option can be a powerful debugging  tool.   Dumping  can  be
allowed only when a specific condition exists.  For example, if an error
occurs only after a loop is executed several thousand times, IF can make
an XSNAP print nothing the first (2000, for example) times, then  print:
IF=((register with loop  counter),H,=F'2000')   .  For  programs  which
process many input cards, it becomes possible to turn  debug  output  on
just before an input card known to be causing trouble, without  printing
output for the preceding cards.  This is done by checking the input  for
a special debug flag value, then setting a flag byte  in  memory,  which
can be tested by many XSNAPs to determine whether printing  is  desired.
In fact, if an 8-bit value  can  be  obtained  from  an  input  card, 8
separate groups of XSNAPs can be individually controlled.

SUPPRESSION OF XDUMP AND XSNAP CODE GENERATION

   After a program is debugged, it is  often  useful  to  be  able  to
suppress all debug code, without actually removing the  cards  from  the
deck.  Coding   XSET XSNAP=OFF  suppresses all XDUMP  and  XSNAP  code
following the  XSET.  They can be restored by  coding  XSET XSNAP=ON  ,
and this process repeated as desired.  Also note that XSET can  be  used
similiarly for  other  X-MACROS, with multiple operands  a  desired:
XSET XSNAP=OFF,XSAVE=OFF,XRETURN=OFF  is quite common.

RESTRICTIONS AND NOTES

   XDUMP and XSNAP are usable ONLY  where  addressibility  exists.   A
common error is to write the following code at an exit point of a CSECT

```
      L    13,4(13)              RETORE SAVE AREA POINTER
      LM   14,12,12(13)          RESTORE REGISTERS
     XSNAP . . .
      BR   14                    RETURN
```

   The above code sequence typically results in  a  branch  SOMEWHERE,
since the program has destroyed whatever base  register  the  XSNAP  was
assembled  under.   Such  errors  are  extremely  difficult  to  locate.


   In order to receive dumping output from XSNAP or  XDUMP,  the  user
should supply a DD card for the execution step of his program,
using the DDNAME XSNAPOUT. Typical such cards are:

```
 //DATA.XSNAPOUT DD SYSOUT=A        (at Penn State)
 //GO.XSNAPOUT DD SYSOUT=A          (standard IBM  cataloged  procedures)
```

  If XSNAPOUT is omitted, XSNAP will use one of the following DD names:
        XPRNT, SYSPRINT, FT06F001 (in that order)

   XDUMP and XSNAP operate by calling a CSECT  named  XXXXSNAP,  which
performs the actual dumping.  If it does not find  the  DDNAME  XSNAPOUT
it will issue a message to the system log and ABEND  U0300.   It  should
be noted that it is impossible  for  XXXXSNAP  to  cause  an  interrrupt
unless the user program has destroyed  some  of  its  code.   Thus,  any
abnormal end with PSW inside XXXXSNAP is probably caused by user  error.

   If XSNAP and XPRNT are used in the same program, the output of each
is usually separate.  The following XSNAPOUT card may be used instead to
obtain XPRNT and XSNAP output merged together:

```
 //DATA.XSNAPOUT DD UNIT=AFF=FT06F001   (at PSU, may not work elsewhere)
```


REFERENCES

   The  following  documents  of  other  X-macros  may  be  of  use :
XMSYSGEN, XREAD/XPRNT/XPNCH .

XGET/XPUT
360 ASSEMBLER                                    V.5.0. Jan 1973

EXTENDED ASSEMBLER I/O COMMANDS

PURPOSE
     Two macros XGET (eXtended GET) and XPUT (eXtended PUT), provide
easy assembler I/O facilities for up to 20 files each simultaneously.
     a) Only 2 statements are needed for each operation.
          LA      R1,=CL8'ddname'
          XGET or XPUT  area address,length
Example:
          LA    1,=CL8'CARDS'
          XGET AREA,80
 //DATA.CARDS DD *


 The above example will read in a card file.

     b) Only register 1 is destroyed, but the user destroys it with the
      LA instruction.
     c) Condition code is set to indicate end-of-file, error conditions,
      and normal execution.


     THESE MACROS ARE EXACTLY LIKE THE XREAD/XPRNT/XPNCH MACROS IN
their form of calling sequence, except that the maximum length is 32767.
I.e., the area address may be any RX-type address, or (0) or (R0), the
latter two indicating that the address is already present in register 0.
Likewise, the length is an absolute expression giving the length, a
register enclosed in parentheses (such as (1) or (R2)), or a default
value  (80 for XGET, 133 for XPUT) if the operand is omitted.


CONDITION CODES
     XGET and XPUT set the CC as follows:
          CC=0 --- I/O occurred;
          CC=1 --- XGET only --- end-of-file encountered;
          CC=2 or 3 --- ERROR occurred.  Either file not opened or error
               occurred while processing, file closed if possible or
               necessary.

CARRIAGE CONTROLS
     If XPUTting to a printer file, the first character of the area must
a valid carriage control character.

CLOSING OF FILE
     Performing an XGET or XPUT with a length of zero causes the
designated file to be closed, so that it may be reread, for example.

JOB CONTROL LANGUAGE
     There are no extra DD cards in the catalogued procedures,
therefore, unless using a dd that is in the procedure, the user must
specify everything.

```
     Card Reader        //DATA.XXXXX DD *            any DD name allowed
     Printer         //DATA.XXXXX DD SYSOUT=A,DCB=(RECFM=FA,BLKSIZE=133)
     Card Punch        //DATA.XXXXX DD SYSOUT=B,DCB=(RECFM=F,BLKSIZE=80)
     Tape              //DATA.XXXXX DD VOL=SER=YYYYY,DSN=ZZZ,UNIT=2400,
                       //         DCB=(RECFM=FB,LRECL=dd,BLKSIZE=ddd)
     Disk              //DATA.XXXXX DD DSN=&&TEMP,UNIT=SYSDA,
                       //  SPACE=(CYL,(5,1)),DCB=(RECFM=FB,BLKSIZE=ddd)
```
Note that these macros need the dcb to be specified, (or on the file),
Thus, many DD cards in the procedure may require the DCB to be added.

GENERAL INFORMATION
     Each macro has an associated CSECT which it calls to perform the
actual I/O operations.  The characteristics are as follows:

| MACRO | MAX LENGTH | CSECT NAME | CSECT LENGTH |
|-------|-----------|------------|--------------|
| XGET  | 32767     | XXXXGET    | 964          |
| XPUT  | 32767     | XXXXPUT    | 996          |

STORAGE REQUIREMENTS
     Each call on any of the macros generates from 32-42 bytes of code,
depending on initial alignment and options use.  This is in addition to
the storage required by any of the CSECTS used.
     Each macro calls the inner macro XIONR, which is the same macro
called by the XREAD, etc, macros.

***NOTE***  User must load into register 1 the address of an eight byte
character string, left justified and padded with blanks if necessary
This string represents the XXXXX of the //DATA.XXXXX DD cards.

REFERENCES:   see the XREAD/XPRNT/XPNCH writeup for the simpler and
more specific I/O commands provided.  Also see the XMSYSGEN writeup
which describes the various properties and generation of the entire
X-MACRO system.

EXAMPLE OF XGET AND XPUT USAGE

        The following program reads/writes several files in parallel.

```
TEST1     CSECT
          BALR  12,0
          USING  *,12
          SR    0,0
*
*     THIS PROGRAM WILL PROCESS A FEW FILES IN PARRALLEL,
*
LOOP      LA    1,=CL8'CARD'       point to an input file
          XGET  AREA,80            do the input
          BNE   DONE               branch on endfile,
*                                  file automatically closed
          XREAD AREA2,80           do normal input
          LA    1,=CL8'PAPER'      point to a printer file
          XPUT AREA-1,81           do output, note carriage control
          LA    1,=CL8'PAPER2'     point to other printer file
          XPUT  AREA2-1,81         do output on other file
          B     LOOP                try again
DONE      BR    14   RETURN, IMPLICITLY CLOSE OTHER FILES
           DC    CL1' '
AREA      DS    CL80
          DC    CL1' '
AREA2     DS    CL80
          END
```

The extra JCL for the above is as follows:
```
//DATA.PAPER DD SYSOUT=A,DCB=(RECFM=FA,LRECL=133,BLKSIZE=133)
//DATA.PAPER2 DD SYSOUT=A,DCB=(RECFM=FA,LRECL=133,BLKSIZE=133)
//DATA.CARD DD *
THIS STUFF IS READ
   AT THE SAME TIME AS ANOTHER
  FILE IS READ
 ******  THE LAST CARD *******
//DATA.INPUT DD *
THIS IS THE NORMAL INPUT FILE
 AND IS READ AT THE SAME TIME AS ANOTHER FILE
  IS READ
    ********* THE LAST CARD *********
```

THE PENNSYLVANIA STATE UNIVERSITY
Computation Center

CONTRIBUTED PROGRAM
Contributor: Alan Artz

XHEXI/XHEXO
360 Assembler language                    April,1972

Extended Hexadecimal Conversion Macros

INTRODUCTION

XHEXI and XHEXO provide easy conversion  of  hexadecimal  numbers
for  input  and  output.  The value  of  a hexadecimal  number  can  be
read  from  a  card  using  XREAD, converted  from  character  mode  to  a
hexadecimal  number, and the converted number is placed in the specified
general  purpose  register  with  XHEXI.  XHEXO  provides  an  easy  way
to  convert  internal  hexadecimal  to  an  output  form  that  can   be
printed using XPRNT.
XHEXI  also  places  the  address  of  the  first  non-hexadecimal
number in register one, but if  more  than  eight  digits  are  scanned,
the address of the ninth is placed in register 1.

TYPICAL USAGE

The following shows the most  common usage of the  macros  XHEXI
and XHEXO:

A)      Take a character number from an area called  CARD  and  convert
   it to hex in Register 2.

                XHEXI    2,CARD    convert number

B)      Take a series of numbers from a card and store them on  a   full-
   word-boundary.

                LA    3,AREA      address of storage
                LA    1,CARD      address of numbers
        TOP     XHEXI 2,0(1)      convert number
                ST    2,0(3)      store it
                LA    3,4(3)      advance storage pointer
                BNO   TOP         keep on going

   This  bit of code  takes  character numbers from an area called CARD,
converts them to internal hex, and then stores them in a location called
AREA.  The card was probably read in with the XREAD instruction.

C)      Take a number in a register  and  convert  it  and  place  it  in
   an output area to be printed.

                XHEXO  2,AREA+1        convert number
                XPRNT  AREA,8          print number
        AREA    DC     CL9' '         storage for number

XHEXI


USE:


XHEXI    REGISTER,ADDRESS


XHEXI, in the general form shown above where REGISTER is any general purpose register  and ADDRESS is anything legal in an RX instruction, is used to do the following:

1.  Beginning at the location ADDRESS, memory is  scanned  until  the first non-blank character is found.

2.  If the first character  found  is  anything  but  a  legal  hexa- decimal character(0-9,A-F),  the  condition  code  is  set  to  overflow and  this  address  is  placed  in  register  1.   If  the  REGISTER  is anything but register 1, its contents remain unchanged.

3.  One to eight  hexadecimal  characters  are  scanned,  the  number converted  to  hexadecimal,  and  the  result  is  placed  in  REGISTER. The  value  placed  in  the  register  is  internal  hexadecimal   with leading zeros included and the number is right justified.

4.  Register one is set to  the address of the first  non-hexadecimal character.  With this in mind, the user  should not code register one as REGISTER.  This  allows  you  to scan  across the card for any number of character strings.  The  strings should be separated by blanks.  The end of the string could be flagged with any non-hexadecimal  character and a test could be made after a Branch Overflow (see sample program).


5.  If more than eight hex digits are found, register one  is  set to the  address  of  the ninth.  This  allows the user to scan  across long strings of numbers.


XHEXO


USE:


XHEXO    REGISTER,ADDRESS


XHEXO  in  the  general  form  shown  above  converts  the  value in REGISTER and places it in a right-justified 8-byte field beginning at ADDRESS.  It can be easily printed using an XPRNT instruction.

SAMPLE PROGRAM USING XHEXI AND XHEXO

      This program reads a data card with an unknown number of hexadecimal numbers on it. The end of the data string is denoted by a '%' punched after the last number. The numbers are stored after being converted using XHEXI, and then converted for output using XHEXO.

```
            LA     3,STORAGE          WHERE NUMBERS STORED
            XREAD  CARD,80            READ IN CARD
            XPRNT  CARD,80            ECHO PRINT
            LA     1,CARD             ADDRESS OF CARD FOR SCANNING
LOOP        XHEXI  2,0(1)             CONVERT NUMBER PUT IN 2
            BO     ILLEGAL            CHECK FOR END
            XHEXO  2,AREA             PUT NUMBER IN OUTPUT AREA
            XPRNT  REP,28             PRINT CARD AND MESSAGE
            ST     2,0(3)             STORE NUMBER
            LA     3,4(3)             INCREASE INDEX
            B      LOOP               GET NEXT NUMBER
ILLEGAL     CLI    0(1),C'%'          SEE IF END OF STRING
            BE     DONE               YES DONE
            XPRNT  =CL50' ILLEGAL CHARACTER STOP',50
DONE        ....MORE INSTRUCTIONS.....
CARD        DC     81C' '             STORAGE FOR CARD
STORAGE     DS     20F                STORAGE FOR NUMBERS
REP         DC     C' THE NUMBER IN R2 IS'
AREA        DC     CL8' '             STORAGE FOR OUTPUT NUMBER
```

REFERENCES

      The following documents of the other X-macros may be of use: XREAD/XPRNT/XPNCH, XMSYSGEN.

    360 ASSEMBLER                       XREAD/XPRNT/XPNCH
                                        V.5.0. Jan  1973

                ASSEMBLER UNIT-RECORD INPUT-OUTPUT MACROS

PURPOSE
      Three macros XREAD (card READer), XPRNT (line PRINTer),
and XPNCH (card PuNCH) provide easy assembler 1/0 facilities
with the following features:
      a)  Only 1 statement is needed for each operation.  No
          OPEN's, CLOSE's, DCB's, etc are required.
      b)  REGISTERS - no registers are destroyed by these macros.
      c)  CONDITION CODE - XPRNT and XPNCH preserve the con-
          dition code.  XREAD sets it to indicate end-of-file
          occurrence.
      d)  JOB CONTROL LANGUAGE - under most circumstances,
          little or no extra JCL will be needed.
      e)  number of characters - may be specified in several
          different ways, may be any length up to maximum
          for device, and can be varied at execution time for
          maximum flexibility.
      f)  XSET macro may be used to temporarily cancel output
          messages, input statements to be omitted, etc, during
          assembly time.


USE
      All three macros use the same form of calling sequence:


      label     is an optional statement label

      Xmacro    is one of XREAD, XPRNT, XPNCH

      area      the address of the area to be read or written.
                This may be specified in two different ways:
      a)  Anything valid in a LA instruction, for example HERE,
          SYM+8(5), 2(4,6), =CL10'OMESSAGE' are all acceptable.
      b)  Either  (O) or (RO), indicating that register O con-
          tains the area address.

      length    specifies the number of bytes to be transferred.
                This length should be in the range 1 to maximum
length for the device (80 for XREAD, XPNCH, l33 for XPRNT).

If the length is incorrect, it will be ignored, and the maximum length used.  The length may be specified in three different ways:

    a)  Directly, by providing an expression in the macro
        call.
    b)  By default, by providing only an area address, which
        will cause the appropriate maximum length to be used.
    c)  Indirectly, at execution time, by coding an expression enclosed in parentheses, indicating a general
        purpose register which contains the number of bytes
        to be read or written.


CONDITION CODE
    XREAD sets the CC as follows:
    CC = 0 - a card was read, and length characters transferred
    CC = 1 - end-of-file was encountered-no more cards


CARRIAGE CONTROLS
    XPRNT requires the first character of the area to be
a valid carriage control character.

JOB CONTROL LANGUAGE
    The macros will accept DD cards with various DDNAMES,
but have been designed to use JCL already present in certain cataloged procedures, in order to minimize the
amount of extra JCL.  The DD cards required for each macro
are listed below.  Note that the PSU catalogued procedure
ASGCLG already contains a FT06F001 DD card, so this need
not usually be supplied by the user.

```
MACRO    DD CARD FORM                 DDNAMES ALLOWED(for xxxxx)
XREAD    //DATA.xxxxx DD *            XREAD, INPUT, FT05F001
XPRNT    //DATA.xxxxx DD SYSOUT=A     XPRNT, FT06F001
XPNCH    //DATA.xxxxx DD SYSOUT=B     XPNCH, FT07F001
```

***NOTE*** if you want to use the  macros in combined FORTRAN-
Assembler programs, you must supply the X-ddnames, rather
than using the FORTRAN ones.

USAGE OF XSET
    The macro XSET (described in writeup on XSNAP/XSTOP/XSET),
may be used to selectively cancel and restore generation of
any of these macros.  It is used as follows:
          XSET name=onoff
where name is XREAD, XPRINT, or XPNCH, and onoff is either
ON or OFF, which either permits or deletes generation of
the specified macro until it is changed again by another XSET.

GENERAL INFORMATION
     Each macro has an associated CSECT which it calls to
perform the actual 1/0 operations.  The characteristics of
these CSECTS are listed below, together with their DCB's.
If the DCB's are unsatisfactory, they may be altered prior
to the first macro call, since the DCB names are all entry
point names also.

```
MACRO   MAX LENGTH   CSECT NAME   CSECT LENGTH   DCB NAME
XREAD      80          XXXXREAD        576          XXREDCB
XPRNT      133         XXXXPRNT        700          XXPRDCB
XPNCH      80          XXXXPNCH        596          XXPNDCB
                       XXXXOPEN        532
```

     XXXXOPEN is called by all the csects to do the openning
of their dcb's for  them.

     The relevant paramaters of the current DCB's used are
as follows:

XXREDCB DCB DSORG=PS,MACRF=GM,RECFM=F,BUFNO=1,LRECL=80,
BLKSIZE=80

XXPRDCB DCB DSORG=PS,MACRF=PM,RECFM=FA,BUFNO=1,LRECL=133,
BLKSIZE=133

XXPNDCB DCB DSORG=PS,MACRF=PM,RECFM=F.BIFMP=1,LRECL=80,
BLKSIZE=80

STORAGE REQUIREMENTS
     Each call on any of the macros generates from 32-42 bytes
of code depending on initial alignment and options used.  This
is in addition to the storage required by any of the three
CSECT's used.

GLOBAL SET VARIABLES
     Each of the macros used a GBLB variable of the form
&nameST, where name is the name of the given macro.  These
are used in conjunction with XSET to control macro generation.

INNER MACROS CALLED
     Each of the macros uses the inner macro XIONR.

SAMPLE PROGRAM
     The following program reads, prints, and punches cards,
and shows some the various ways of specifying the macro
operands.

```
// EXEC ASGCLG
//SOURCE.INPUT DD *
         TITLE  ' TEST PROGRAM FOR 1/0 MACROS'
TESTX10  CSECT
         XSAVE   TR=NO          set up linkage, base register
         XSET    XPRNT=OFF      cancel XPRNT for a while
         XPRNT  =CL50' THIS MESSAGE IS NOT GENERATED', 50
         XSET    XPRNT=ON       allow XPRNT's again
         LA      0,CARD         save address of card
         XPRNT  =CL40'1 TEST EXAMPLES', 40 skip to new page
         LA      2,50           set up length in register
         XPRNT  =CL50'0 WILL READ,PRINT,PUNCH',(2)
TLOOP    XREAD   CARD           use default length=80
         BNZ     TDONE          branch on end-of-file
         XPNCH  (0)             use previously loaded address
         XPRNT   CARD-1,81      print card, with carriage cont
         B       TLOOP          go back for next card
TDONE    XPRNT MESSAGE,L'MESSAGE print ending message
         XRETURN SA=*,TR=NO
MESSAGE  DC      CL100'OSUCCESSFULL COMPLETION OF TEST'
CARD     DS      CL80
         end
//DATA.XPNCH DD SYSOUT=B        required for card punch
//DATA.INPUT DD *
     THIS IS THE FIRST TEST CARD
     THIS IS THE SECOND TEST CARD
     THIS IS LAST TEST CARD-END-OF-FILE FOLLOWS
```

REFERENCES:  see the XGET/XPUT writeup for more generalized macros
allowing access to any sequential files.  Also see XMSYSGEN for an
overview of the X-MACRO system.

                    THE PENNSYLVANIA STATE UNIVERSITY
                         Computation Center

                          CONTRIBUTED PROGRAM
               Contributor: John R. Mashey, Computer Science Department

                                                    XSAVE/XRETURN
          360 ASSEMBLER LANGUAGE           writeup revised April 1972 (v.4.0)

                      EXTENDED SAVE AND RETURN MACROS


INTRODUCTION


        XSAVE and XRETURN provide various services for  assembler  language
program linkage using OS/360  conventions.   While  offering  many  more
options than SAVE and RETURN, they  use  defaults,  thus  allowing  that
few operands need  be  coded.    Some  of  the  services  provided  are:

a) Saving or restoring any range or ranges of registers on program entry
or exit, in a more general way than SAVE or RETURN.
b) Automatic initialization of 1 or more base registers for  a  program.
c) Creating standard entrypoint identifier coding at  a  program  entry.
d)  Generating  reentrant  code,  using  GETMAIN  and  FREEMAIN   calls.
e) Creating an AUTOMATIC PROGRAM  TRACE  or  REGISTER  DUMP  whenever  a
module is entered or exited.
f)  Permitting  easy  usage  in  CSECT's  with  multiple  entries/exits.


TYPICAL USAGE


        The following shows the most common ways to use XSAVE and  XRETURN:

a)    CSECT with a single entry point, at the CSECT name:

      ACSECT    CSECT
                XSAVE
      .....  the code for the csect.
                XRETURN SA=*    EXIT POINT: ALSO CREATE SAVE AREA


b)    CSECT with single entry and exit, using register 13 as both a  base
register and save area  pointer,  generating  save  area  inside  XSAVE:

      BCESCT    CSECT
                XSAVE BR=13    CAUSE SAVE AREA TO BE HERE
      ..... the code for the csect.
                XRETURN


c)    CSECT with multiple entry points:

      CCSECT    CSECT
                ENTRY C1,C2     SEVERAL ENTRIES
      C1        XSAVE
      ..... the code for section C1 of CCSECT.
      C1EXIT    XRETURN
      C2        XSAVE
      ..... the code for C2.
      C2EXIT    XRETURN SA=*    SAVE AREA AFTER LAST XRETURN ONLY

     The macros are described below, with arguments listed basically  in
order of importance.  A programmer not yet familiar with  S/360  linkage
should concentrate on the options:  RGS=,SA=, and BR= where they  exist.

                                   XSAVE
PURPOSE

     XSAVE is used at any entry point  (CSECT, START, or ENTRY label) in
an assembler program to  save  registers  acccording  to  the  standard
conventions, set up new base register(s), and possibly  perform  program
tracing and various other optional services.

USE

     The macro allows 8 keyword operands, which can be  used  in  almost
any combination or omitted as desired.  Reasonable defaults  values  are
supplied so that few operands need normally be  coded.   It  is  called:

label    XSAVE RGS=,BR=,SA=,ID=,TR=,REEN=,OPT=,AD=

label    is an optional statement label.


     RGS=        (ReGisterS)

     This operand specifies the register(s) to be saved into the calling
program's save area in the standard locations.  It can specify  that  NO
registers, a single range of registers, or multiple ranges of  registers
be saved, with the following operand types:

RGS=NO          no registers are saved.

RGS=(list)      list specifies the the registers to be saved, where  each
     item in the list is either:
a) a  single  register  designation (number  or  equate  symbol)     OR
b) a pair of register designations, separated by  a  dash,  showing  the
(inclusive) range of registers that should be stored.
     For example, RGS=(14-15,2-12) causes registers 14,15, and all of  2
through 12 to be saved appropriately.

**DEFAULT VALUE** If omitted, RGS=(14-12) is assumed    (normal linkage).

     BR=        (Base Register(s))

     BR lists the base register(s) to be set up  for  the  program  just
entered.  It is either a single register, or a  list  of  2-4  of  them,
enclosed in parentheses, and separated by commas.  If the latter  format
is used, a multiple USING  statement  and  appropriate  register-loading
statements are generated.  If the first (or only) register is 13, a save
area is generated in the middle of the XSAVE, so that register 13 can be
used both as a base register and as a save area  pointer.   WARNING:  if
register 13 is specified, do not code XRETURN SA=* later in  the  csect.

**DEFAULT** BR=12 is assumed if this option is omitted.

     SA=        (Save Area)


     This option specifies the name of a save area to be used, and  also
controls whether the program actually has a save area or not (if it is a
lowest-level  routine  calling  no  others,  it  might  not  have  one).

     SA=NO    no code is generated for linking save areas, and no save  area
     is referenced by XSAVE code.  Register 13  is  unchanged,  so  this
option should  only  be  used  by  a  routine  which  calls  no  others.

     SA=*      this is normal linkage, with following actions:

a) register 13(calling program's save area) is loaded into another  reg.
b) the address of the called program's save area  is  placed  into  R13.
c) the two registers are saved correctly to fill in the standard forward
and backward linkage between calling and called  programs'  save  areas.

     Part b) creates the statement  LA  13,saveareaname  ,  as  follows:
a) If this is the first XSAVE in a CSECT, a 'standard name' (unique  to
that CSECT) is created by the macro and referenced.
b) If this is not the first XSAVE, the current 'standard name'  is  used
regardless of how it was originally created.

     SA=name   this specifies the same kind of linkage as SA=*,  except  that
     the name becomes the 'standard name' .

     Briefly, the 'standard name' allows XSAVE and XRETURN macros to use
and generate appropriately-named save areas,  allowing  either  complete
control over naming, or the convenience of giving no names.

     **DEFAULT** If omitted,  SA=* is assumed, giving standard linkage,  with
all  XSAVE(s)  in  a  CSECT  referring  to  the  SAME  save  area  name.


     ID=        (IDentifier)


     ID specifies an identifying character string  to  be  created  just
after the entry point.  It aids debugging by making the entry point easy
to locate in the dump, and may be printed  in  dump  save  area  traces.

     ID=NO     no  identifier  is  created.   This  saves  space  and  time.

     ID=*      the identifier created is the first of  the  following  found:
     the statement label, if one is coded on the XSAVE; the name of  the
CSECT in which the XSAVE is used, or $PRIVATE if the CSECT  is  unnamed.

     ID=string the string (1-255 characters) is the identifier.

     **DEFAULT**  If omitted, ID=* is assumed, thus creating some kind of ID.

TR=        (TRace)

     TR provides for  automatic  program  tracing  or  register  dumping
whenever the entry point is  given  control,  a  useful  debug  feature.

TR=NO      no trace code is provided .

TR=*       the message '*** name ENTERED ***' is printed on entry,  where
     name is either the label on the XSAVE, or the name  of  its  CSECT.

TR='message'    'message' is printed on entry to the program.

TR=SNAP    the message (as in TR=*) is printed, followed by a dump of the
     GP  registers,  before  they  get  changed   by   called   program.

TR=('message',SNAP)     'message' and registers are printed.

**DEFAULT**  TR=* is assumed, thus normally providing a  dump  (but  see
section at end called SPECIAL CONSIDERATIONS FOR TRACE CODE).


REEN=exp  (REENtrant code)

     REEN requires that reentrant code be generated, and give the number
of bytes to be acquired via GETMAIN, IN ADDITION TO the 72 bytes  needed
for the standard savearea.  The address of the acquired area  is  placed
in register 13, so that first usable working storage begins  at  72(13).
If the called program uses the original values of register 15,0,1,  they
must be included in the registers saved by the RGS= option.   The  value
exp may be any absolute expression.  WARNING: this method is  slow  for
heavily-used routines.


OPT=        (OPTional declarations)

     OPT provides for  declaration  of  CSECT  or  ENTRY  and/or  TITLE.

OPT=CSECT the label  (required)  on  the  XSAVE  is  declared  a  CSECT.

OPT=ENTRY the label(required) on the XSAVE is declared to be  an  ENTRY.

OPT=TITLE a TITLE is generated before any other code, and  displays  the
     label on the XSAVE statement.

OPT=(TITLE,xxxxx)    xxxxx is either CSECT or ENTRY, combining the above.


AD=        (ADdress constant addressibility)

     In some cases it may be necessary to have several entry  points  in
a CSECT which share common USING conditions;  AD can be used to do this.
An address constant of the name specified is loaded into the  first  (or
only) base register (BR=), and addressiblity  declared.   Typically  the
address constant refers to the name of the CSECT.

PURPOSE

     XRETURN is used to return from a module entered at an XSAVE, and is
used to restore registers, set return code in R15, perform program trace
and possibly generate a 72-byte save  area  referenced  by  XSAVE  code.

USE
     The macro has 7 keyword operands,  coded  or  omitted  as  desired:

label     XRETURN  RGS=,SA=,RC=,RP=,T=,TR=,REEN=

label     is an optional statement label

RGS=      (ReGisterS)

     This specifies the registers  to  be  restored  from  the  original
calling program's save area, and is coded just like  XSAVE  RGS= option.

**DEFAULT**     RGS=(14-12)  is  assumed,  restoring  all  registers.

SA=       (Save Area)

     This option can be used to control the return linkage code, and  if
desired, creates an 18-fullword save area (filled with zeroes for  debug
purposes), immediately following XRETURN's executable code.

SA=NO     the macro assumes that the routine is a  lowest-level  routine
     which calls no others.  It generates neither the save area  itself,
nor code to reload register 13 with the  address  of  th  caller's  save
area. R13 is  assumed  to  still  point  at  the  caller's  save  area.
WARNING:  use only paired with XSAVE  SA=NO,  else  errors  will  occur.

SA=*      an 18-fullword  save  area  is  generated,  labeled  with  the
     current  'standard  name'  described  under  XSAVE  SA=*  above.

SA=name   the save area is generated, and labeled with the  name  given.

**DEFAULT**     If SA is completely omitted, no save area  is  generated,
     but a save area is assumed to exist somewhere, and  code  generated
to restore the previous save area pointer.  The reason for this  default
is that any CSECT having several entry points still needs  only  1  save
area, which is typically generated only by the LAST XRETURN, so that  it
is addressible by ALL of the XSAVEs.  Typically, a  maximum  of  1  SA=*
should appear in the XSAVEs and XRETURNs in one CSECT.

RC=       (Return Code)

     Specifies a return code value to exist in register 15 on return  to
the calling program.  The user need not worry about  possible  conflicts
with RGS=, since RC overrides any inclusion of R15 in the  RGS=  option.

RC=exp    expression less  than  4096  to  be  placed  in  register  15.

RC=(r)    specifies  register  currently  containing  the  return  code.

RP=exp    (Return Past register 14)

     RP provides a form of multiple (nonstandard) return to the  calling
program.  The last instruction generated is not the usual  BR   14,  but
B  exp(14)   instead.  The value of  exp  is usually a multiple of 4, so
that the calling program has 1 or more  Branch statements following  the
call to the routine.


T=*       (Tag save area)

     This requests that byte 12 of the calling program's  save  area  be
set to  X'FF' just before control  is  returned.   This  is  useful  for
debugging, and is sometimes used  by  FORTRAN  error-handling  routines.


TR=       (TRace code)

     This option is coded exactly as is XSAVE TR=,  and  does  the  same
thing, except that the message printed is  '*** name EXITED ***',  where
name is either the label on the XRETURN or the CSECT name.


REEN=exp  (REENTRANT code)

     REEN specifies reentrant code generation,  and  should  be  matched
only with an XSAVE REEN=exp, where the two  expressions  are  the  same.
The area addressed by register 13, of length  exp+72, is freed using the
FREEMAIN macro.   WARNING:  if this option is used, it is impossible for
the XRETURN code to ever restore the values of registers 15, 0, 1  which
existed in the calling program prior to entering the called module. Note
that normal conventions require only that registers  2-12  be  restored.

SPECIAL CONSIDERATIONS FOR TRACE CODE

     Note that the TR= options call the macro XSNAP (for SNAP  options),
or XPRNT for the others, so that the relevant writeups should be studied
if any problems arise.  In particular, the SNAP options  may  require  a
card like the following to be inserted for the  execution  step  of  the
program (before or after a SYSUDUMP card):

 //DATA.XSNAPOUT DD SYSOUT=A  (non-PSU users may replace DATA with  GO).

     The trace code (and ONLY the  trace  code)  created  by  XSAVE  and
XRETURN can be eliminated by coding  XSET  XSAVE=OFF,XRETURN=OFF  ,  and
can be allowed again by replacing OFF by ON, as many times  as  desired.
This allows leaving trace code in a program until it is  debugged,  then
eliminating  it by adding 1 XSET card at  the  beginning of the program.

SYMBOLIC REGISTER EQUATES

     Symbolic register equates may be used  anywhere,  and  can  be  any
symbols, except that any used for registers 13,14, or  15  must  end  in
those characters, i.e.,  WORK13, R14, REG15 are acceptable.

SAMPLE USAGE
     The following sample program assumes that EQU's have  been  set  up
for  R0  EQU 0, ...  R15 EQU 15 .

```
*          TYPICAL USAGE FOLLOWS.
           XSAVE
           CALL PROG2
           XRETURN SA=*,RC=4    RETURN TO OS WITH RETURN CODE OF 4
*          FOLLOWING CALLS EXTREME AND UNTYPICAL.
PROG2      XSAVE RGS=(R14,R0-R1,5-12),BR=(11,10),OPT=(TITLE,CSECT)
           CALL  LOWEST           CALL LOWEST LEVEL ROUTINE
           LA    R3,5             SET UP LIMIT ON RECURSION DEPTH
           CALL  RECURSE          CALL RECURSIVE ROUTINE
           CALL  PROG3
           CALL  PROG4
           B     BAD              RETURN HERE× OR SKIP BRANCH VIA RP=
           SR    15,15            CLEAR FOR RETURN CODE = 0
BAD        XRETURN  RGS=(14,0-1,5-12),SA=*,T=*,RC=(R15)
*           LOWEST LEVEL ROUTINE FOLLOWS: NOTE CODE GENERATED.
LOWEST     CSECT
           XSAVE SA=NO,BR=15,TR=NO,ID=NO  ZAP OPTIONS
           XRETURN SA=NO,TR=NO
*           FOLLOWING CSECT USES REEN FOR RECURSIVE CALLS.
PROGRAM    CSECT
RECURSE    XSAVE TR=('RECURSION',SNAP),REEN=0,OPT=ENTRY
           BCT   R3,CONTINU            LOOP, RECURSING
           B     RETB                 GIVE UP NOW
CONTINU    CALL  RECURSE              CALL MYSELF AGAIN
RETB       XRETURN REEN=0             RETURN, FREE STORAGE
*           ENTIRES PROG3 AND PROG4 HAVE SAME ADDRESSIBILITY, WITH AD.
PROG3      XSAVE OPT=ENTRY,SA=P3SAVE,BR=(12,11,10),AD=PROG3
PROG3A     XRETURN RC=(R3),T=*
PROG4      XSAVE OPT=ENTRY,AD=PROG3,BR=(12,11,10),ID=PROG3ENTRYOFPROGRAM
           LTR   R0,R0                TEST TO DECIDE
           BZ    ZERO                 SKIP TO RP=0
           XRETURN RP=4               BRANCH OVER BAD ABOVE
ZERO       XRETURN RP=0,T=*,SA=*     SAVE AREA, RP
           END
```

REFERENCES

     The  following  documents  of  other  X-macros  may  be   of   use:
XMSYSGEN, EQUREGS, XDUMP/XSNAP, XREAD/XPRNT/XPNCH.

     The reader is also referred to the following manuals:

C28-6827  IBM S/360 OS FORTRAN IV (G&H) Programmer's Guide – Appendix C.
     (a concise explanation on save areas and linkage).

C28-6646   IBM  S/360  OS  Supervisor  and  DAta  Management   Services
     (pp. 10-18 approx: linkage, reentrancy).

C28-6647  IBM S/360 OS Supervisor and Data Management Macro Instructions
     (macros CALL, SAVE, RETURN)