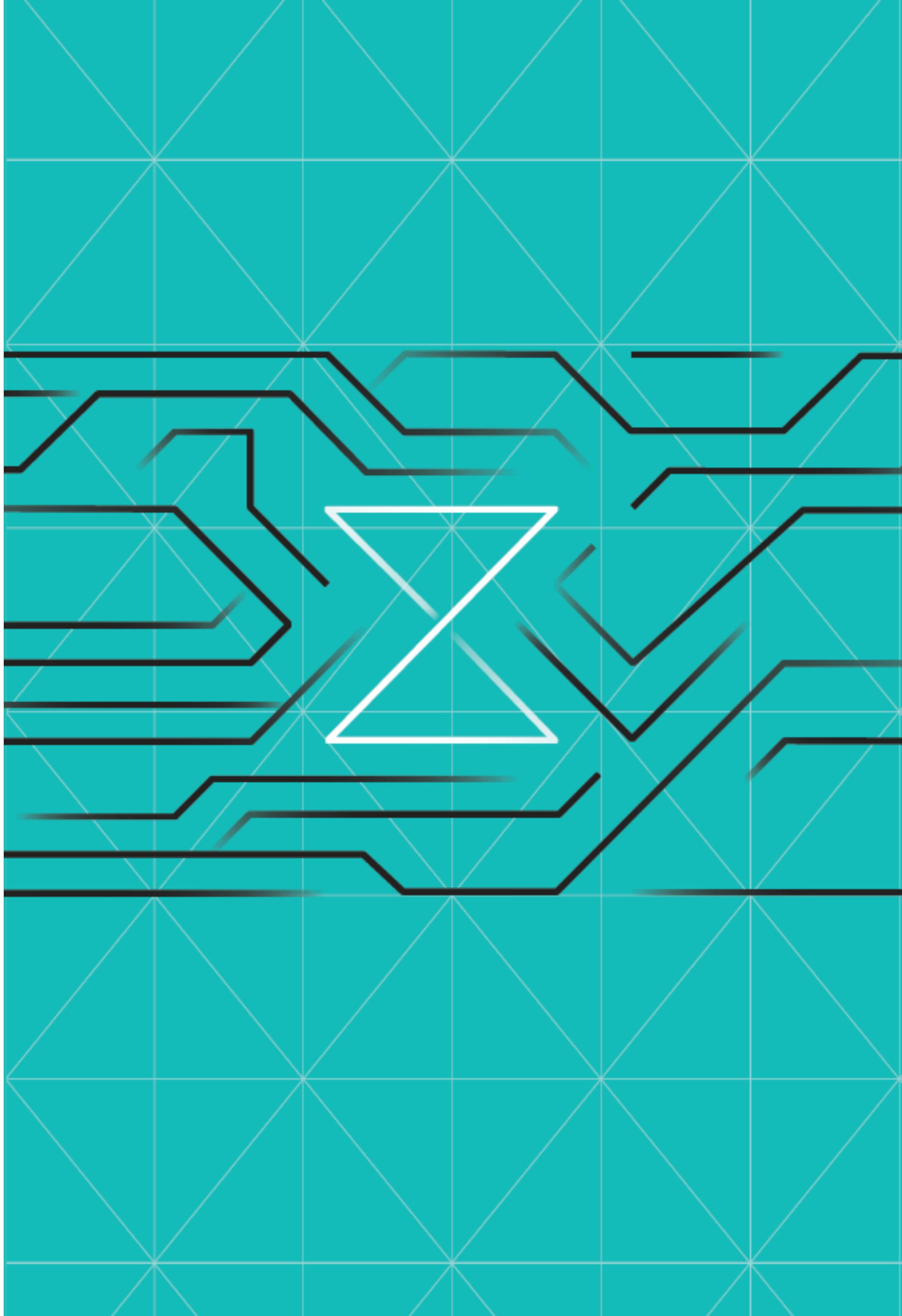
Advanced *230808-2334*

ASM2

Developers Assemble!



Z Xplore

1 BACKGROUND

This challenge takes you a bit deeper into IBM Z Assembler Language.

Some large companies need programmers, developers and architects that understand IBM Z Assembler language.

These technical positions pay well and lead to promotions in the technical ranks. Familiarity with IBM Z Assembler on a resume is a major “attention-getter” for securing an interview. You can say you have familiarity with IBM Z Assembler as a result of completing these challenges.

Do not let the technical details in the challenge scare you away. By the end of this challenge you will realize you are capable of beginning to understand the technical details of how IBM Z computer software works.

ASM2|230808-2334

2 PREPARATION

A relatively simple assembler program will be used and explained.

Each computer architecture has machine instructions unique to the architecture. All computer languages supported by the architecture must be translated into the unique machine instructions of the underlying computer architecture. Each computer architecture has an assembly language which includes mnemonics that are assembled into machine instructions understood by the computer. Compilers and interpreters translate supported computer languages into the unique machine instructions understood by the hosting computer.

Computer processing memory is used to load and store the machine instructions and data for processing. The operating system keeps track of processing memory locations using addresses where some of the memory is free, some of the memory has machine instructions, and some of the memory has data.

Higher level languages such as C/C++, Java, COBOL, etc. were created to make programming the computer easier by hiding the complexity of the underlying machine instructions, addressable memory, and registers. Registers are at the top of the memory hierarchy, and provide the fastest way to access data.

IBM Z mainframe computer architecture, z/Architecture, is what is commonly known as Complex Instruction Set Computing (CISC); intel x86 and compatible processors are also CISC architectures.

The assembler code that follows is explained in greater detail using the assembler compiled output.

```
*---------------------------------------------------------------------* 00060000
*        register equates                                             * 00070000
*---------------------------------------------------------------------* 00080000
R0       EQU   0                        register 0                      00090000
BASEREG  EQU   12                       base register                   00100000
SAVEREG  EQU   13                       save area register              00110000
RETREG   EQU   14                       caller's return address         00120000
ENTRYREG EQU   15                       entry address                   00130000
RETCODE  EQU   15                       return code                     00140000
         EJECT                                                          00150000
*---------------------------------------------------------------------* 00160000
*        standard entry setup, save area chaining, establish          * 00170000
*        base register and addressibility                             * 00180000
*---------------------------------------------------------------------* 00190000
         USING ASMPGM,ENTRYREG          establish addressibility        00200000
         B     SETUP                    branch around eyecatcher        00210000
         DC    CL8'ASMPGM'              program name                    00220000
         DC    CL8'&SYSDATE'            program assembled date          00230000
SETUP    STM   RETREG,BASEREG,12(SAVEREG)  save caller's registers      00240000
         BALR  BASEREG,R0               establish base register         00250000
         DROP  ENTRYREG                 drop initial base register      00260000
         USING *,BASEREG                establish addressibilty         00270000
         LA    ENTRYREG,SAVEAREA        point to this program save area 00280000
         ST    SAVEREG,4(,ENTRYREG)     save address of caller          00290000
         ST    ENTRYREG,8(,SAVEREG)     save address of this program    00300000
         LR    SAVEREG,ENTRYREG         point to this program savearea  00310000
         EJECT                                                          00320000
```

The section shown above is responsible for “standard linkage”, a technique for passing data and processing control from a caller program to a called program.

Read the comments to the right of the assembler mnemonics.

```
*---------------------------------------------------------------------* 00330000
*        program body                                                 * 00340000
*---------------------------------------------------------------------* 00350000
         L     2,=C'Begin'                                              00360001
         LA    2,=C'Begin'                                              00370001
LOOPINIT DS    0H                                                       00380000
         SR    2,2                                                      00390000
         L     2,=F'4'                                                  00400001
         L     3,=F'1'                                                  00410000
LOOP     DS    0H                                                       00420000
         A     3,=F'1'                                                  00430001
         BCT   2,LOOP                                                   00440001
STOP1    LH    3,HALFCON                                                00450001
STOP2    A     3,FULLCON                                                00460001
STOP3    ST    3,HEXCON                                                 00470001
         EJECT                                                          00471001
```

Next comes the logic to be executed, the program body.

```
*         storage and constant definitions.                      *  00590000
*         print output definition.                               *  00600000
*----------------------------------------------------------------*  00610000
SAVEAREA DC    18F'-1'                 register save area           00710000
FULLCON  DC    F'-1'                                                00711001
HEXCON   DC    XL4'9ABC'                                            00712001
HALFCON  DC    H'32'                                                00713001
         END   ASMPGM                                               00720000
```

Generally, the bottom of an Assembler source code contains the definition of program constants, and other structures for handling data in processing memory storage.

```
//GO      EXEC PGM=ASMPGM                                           00730000
//STEPLIB  DD DSN=&SYSUID..LOAD,DISP=SHR                            00740000
//PRINT    DD SYSOUT=*                                              00750002
```


And finally, the job-step that runs the program created by the Assemble and Link steps

3 THE CHALLENGE

Compile assembler program

- copy ‘ZXP.PUBLIC.SOURCE(ASMPGM)’ to your SOURCE dataset.
- submit the program with right-click from the DATA SETS view in VSCode, and select Submit Job.
- Review the assembler compilation details by checking the job output called ASMPGM:SYSPRINT

```
    Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                      HLASM R6.0  2023/02/14 06.17
  000000                00000 000ED     2 ASMPGM   CSECT                                              00050000
                                        3 *-----------------------------------------------------------* 00060000
                                        4 *        register equates                                  * 00070000
                                        5 *-----------------------------------------------------------* 00080000
                        00000           6 R0       EQU   0                 register 0                 00090000
                        0000C           7 BASEREG  EQU   12                base register              00100000
                        0000D           8 SAVEREG  EQU   13                save area register         00110000
                        0000E           9 RETREG   EQU   14                caller's return address    00120000
                        0000F          10 ENTRYREG EQU   15                entry address              00130000
 i                      0000F          11 RETCODE  EQU   15                return code                00140000
```

ASM2|230808-2334

Things to look for in the compiler listings:

The column headings in the program listing:

- Loc - abbreviation for location, program location column
- Object Code - machine instructions column. commonly called op codes
- Addr1 - memory address location column
- Addr2 - memory address location column
- Stmt - line numbers associated with each line of code

- Source Statement - assembler code labels, mnemonics, and mnemonic operands

The program code itself:

- Stmt 2 is the first assembler code line where:
  Loc 00000 represents the beginning location
  Addr1 00000 Addr2 000FD represents the starting and ending location of the program in hexadecimal – effectively the program length in bytes - x’FD’ or 253 decimal
  ASMPGM is a user chosen label
  CSECT is an assembler directive declaring a Control Section
- Stmt 3, 4, 5 are comments - identified by the character in column 1 being and asterisk “*”
- Stmt 6 to 11 show user selected labels equated (EQU) to computer register numbers - names to use in the program instead of just the register numbers.

Following Stmt 11 is assembler code frequently reused for [“standard linkage”]() - storing the location and register contents of the calling program to return control to when this program is finished.

```
                                  13 *-------------------------------------------------------------------* 00160000
                                  14 *        standard entry setup, save area chaining, establish        * 00170000
                                  15 *        base register and addressibility                           * 00180000
                                  16 *-------------------------------------------------------------------* 00190000
                  R:F  00000      17          USING ASMPGM,ENTRYREG        establish addressibility        00200000
 000000 47F0 F014          00014  18          B     SETUP                  branch around eyecatcher        00210000
 000004 C1E2D4D7C7D44040          19          DC    CL8'ASMPGM'            program name                    00220000
                                  20          DC    CL8'&SYSDATE'          program assembled date          00230000
 00000C F0F261F1F461F2F3           +          DC    CL8'02/14/23'          program assembled date          00230000
 000014 90EC D00C          0000C  21 SETUP    STM   RETREG,BASEREG,12(SAVEREG)  save caller's registers    00240000
 000018 05C0                      22          BALR  BASEREG,R0             establish base register         00250000
                                  23          DROP  ENTRYREG               drop initial base register      00260000
                  R:C  0001A      24          USING *,BASEREG              establish addressibilty         00270000
 00001A 41F0 C06E          00088  25          LA    ENTRYREG,SAVEAREA      point to this program save area 00280000
 00001E 50D0 F004          00004  26          ST    SAVEREG,4(,ENTRYREG)   save address of caller          00290000
 000022 50F0 D008          00008  27          ST    ENTRYREG,8(,SAVEREG)   save address of this program    00300000
i000026 18DF                      28          LR    SAVEREG,ENTRYREG       point to this program savearea  00310000
```

- Stmt 18 includes assembler mnemonic **B**, a branch instruction.

  The operand is SETUP, a program label that is a quick way to resolve addressable location in memory.

- To the left of Stmt 18 you can see an operation code ("OPCODE") that begins with 47, which is the opcode for the mnemonic **B**.
- Stmt 18 Addr2 is 00014. Loc 00014, Stmt 22, is label SETUP with more assembler code to execute.

While the explanation below is elaborate, it is just preparation for you to see these changes happen at breakpoints during program execution later in the challenge.

The program body includes -

Statement	OPCODE	Mnemonic	Description
34	x'58'	**L**	Load register 2 with characters 'Begin'
35	x'41'	**LA**	Load Address register 2 with address of 'Begin' address location. You will see the register content difference between the 2 instructions later. You will might notice the Addr2 is 000F8 memory address location for 'Begin', =C'Begin' resulted in assembler assigning a memory address location for the characters "Begin"
37	x'1B'	**SR**	subtract register 2 from itself
38	x'58'	**L**	load 4 into register 2

Statement	OPCODE	Mnemonic	Description
39	x'58'	**L**	load 1 into register 3
42	x'5A'	**A**	add 1 to register 3
43	x'46'	**BCT**	decrement register 2 by 1 and branch to the label loop until register 2 is zero
45	x'48'	**LH**	load halfword into register with content of memory location assigned to HALFCON label
46	x'5A'	**A**	add memory location assigned to FULLCON label into register 3
47	x'50'	**ST**	store register 3 in memory location of HEXCON

ASM2|230808-2334

```
                        30 *------------------------------------------------------------------* 00330000
                        31 *       program body                                               * 00340000
                        32 *------------------------------------------------------------------* 00350000
 000028 5820 C0CE  000E8  33          L     2,=C'Begin'                                         00360001
 00002C 4120 C0CE  000E8  34          LA    2,=C'Begin'                                         00370001
 000030                   35 LOOPINIT DS    0H                                                  00380000
 000030 1B22              36          SR    2,2                                                 00390000
 000032 5820 C0C6  000E0  37          L     2,=F'4'                                             00400001
 000036 5830 C0CA  000E4  38          L     3,=F'1'                                             00410000
 00003A                   39 LOOP     DS    0H                                                  00420000
 00003A 5A30 C0CA  000E4  40          A     3,=F'1'                                             00430001
 00003E 4620 C020  0003A  41          BCT   2,LOOP                                              00440001
 000042 4830 C0BE  000D8  42 STOP1    LH    3,HALFCON                                           00450001
 000046 5A30 C0B6  000D0  43 STOP2    A     3,FULLCON                                           00460001
i00004A 5030 C0BA  000D4  44 STOP3    ST    3,HEXCON                                            00470001
```

At Stmt 56 you will see ‘WTO’:

- Left is blank - no op code, no operands, and no addresses
- WTO, abbreviation for Write To Operator, is an assembler macro
- The text following WTO is written to the system log

Below, you can see commonly reused code for ‘standard linkage’, restoring the caller program registers. This is how this program returns control to the caller.

```
                        46 *------------------------------------------------------------------* 00480000
                        47 *       standard exit -  restore caller's registers and            * 00490000
                        48 *       return to caller                                           * 00500000
                        49 *------------------------------------------------------------------* 00510000
 00004E                   50 EXIT     DS    0H                    halfword boundary alignment     00520000
 00004E 58D0 D004  00004  51          L     SAVEREG,4(,SAVEREG)   restore caller's save area addr 00530000
 000052 58E0 D00C  0000C  52          L     RETREG,12(,SAVEREG)   restore return address register 00540000
 000056 980C D014  00014  53          LM    R0,BASEREG,20(SAVEREG) restore all regs. except reg15 00550000
                          54          WTO   'Giving control back to system'                        00551001
```

ASM2|230808-2334

Next up: Stmt 58 to 64 with a '+' following the statement numbers is the WTO macro expanded code

Stmt 65 with op code beginning with x'07' associated with mnemonic BR, branch

> The address location represented by label RETREG is register 14 which contains the address location of the caller program.

So, the execution result is a branch to an address location of the caller program.

```
   00005A 0700                     56+        CNOP  0,4                                    01-WTO
   00005C A715 0013         00082  57+        BRAS  1,IHB0001A          BRANCH AROUND MESSAGE    @LCC 01-WTO
   000060 0021                     58+        DC    AL2(33)             TEXT LENGTH          @YA17152 01-WTO
   000062 0000                     59+        DC    B'0000000000000000'  MCSFLAGS                    01-WTO
   000064 C789A58995874083         60+        DC    C'Giving control back to system'               X01-WTO
   00006C 9695A39996934082           +                                  MESSAGE TEXT          @L6C
   000082                          61+IHB0001A DS   0H                                         01-WTO
   000082 0A23                     62+        SVC   35                  ISSUE SVC 35         @L6A 01-WTO
  i000084 07FE                    63         BR    RETREG              return to caller            00560000
```

And finally there are some defined constants with assembly-assigned addressable memory locations

The opcode area is used as content at the addressable memory location when the program starts execution.

A few observations -

- Label SAVEAREA at location 98 into the program is 18 fullwords in length
- Label FULLCON at location E0 into the program is 1 fullword in length
- Label HEXCON at location E4 into the program is a hexadecimal length of 4.
- Label HALFCON at location E8 into the program is a halfword where of decimal value H'32'   but inspection of the stored value in memory is x'20'

- Program body included =F'4', =F'1', and =C'Begin', where the = resulted in the assembly to machine code assigning addressable locations for the values at the end of the executable program, F0, F4, and F8, respectively.

```
138                                        65 *----------------------------------------------------------------------* 00580000
139                                        66 *       storage and constant definitions.                              * 00590000
140                                        67 *       print output definition.                                       * 00600000
141                                        68 *----------------------------------------------------------------------* 00610000
142     000086 0000
143     000088 FFFFFFFFFFFFFFFF            69 SAVEAREA DC    18F'-1'             register save area          00710000
144     0000D0 FFFFFFFF                    70 FULLCON  DC    F'-1'                                            00711001
145     0000D4 00009ABC                    71 HEXCON   DC    XL4'9ABC'                                        00712001
146     0000D8 0020                        72 HALFCON  DC    H'32'                                            00713001
147     000000                             73          END   ASMPGM                                           00720000
148     0000E0 00000004                    74                =F'4'
149     0000E4 00000001                    75                =F'1'
150     0000E8 C285878995                  76                =C'Begin'
```

Fullword	32 bits - 4 bytes - of storage
Halfword	16 bits - 2 bytes - of storage

Other terms that might be useful: [Glossary of z/OS Terms and Abbreviations]

Use a calculator to convert between hexadecimal and decimal (if you need it)

4 USE TSO TEST FACILITY

The TSO TEST facility is used to execute a program with the added capability to stop the program execution at chosen address locations, to inspect changes in registers, and make changes in other address locations in the program.

The TSO Ready prompt is required to use the TSO TEST facility - if you are using ISPF, you will need to exit that environment, and get to TSO READY.

In this challenge, you can execute all the commands through the Zowe CLI in the same way as the REXX1 challenge.

To start things off, create an active TSO connection with Zowe CLI and capture the Address-Space Key (ASKEY) -

```
export ASKEY=`zowe tso start address-space --sko`
echo $ASKEY
```

If you are using a Windows Command shell, then the following will work better:

```
for /f "delims=" %i in ('zowe tso start address-space --sko -a FB3') do set ASKEY=%i
echo %ASKEY%
```

You should see something similar to

```
Z#####-153-aacwaaat
```

where Z##### is your Z userid.

Once you have your TSO session key, you need to prepare TSO to recognise “short-hand” dataset names - for this challenge, it means that the TEST program, which will try and use a **LOAD** dataset, will use *your* LOAD dataset.

`zowe tso send address-space $ASKEY --data "PROFILE PREFIX(Z#####)"` (make sure to replace Z##### with your Z-userid!)

Send `zowe tso send address-space $ASKEY --data "PROFILE"` to see the resulting profile setting.

Now tell TSO to start the TEST environment with your assembled program

`zowe tso send address-space $ASKEY --data "test (asmpgm)"` - this will look in the LOAD dataset

For Windows Command Shell - `zowe tso send address-space %ASKEY% --data "test (asmpgm)"` - this will look in the LOAD dataset

Run through the following sets of commands by sending them as data to the TSO session - for example, for a command **LISTPSW**, use:

`zowe tso send address-space $ASKEY --data "LISTPSW"`

For Windows Command Shell - `zowe tso send address-space %ASKEY% --data "LISTPSW"`

(You should find it easy by pressing the cursor-up key and overtyping the last command)

NOTE: *if your zowe address-space times out between commands, you will see a message like :*

> IZUG1126E: z/OSMF cannot correlate the request for key "Z#####-153-aacwaaat" with an active z/OS application session.

if that happens, you just need redo the command to start a new address-space, redo the command to start the TEST of asmpgm, then send the last `at` *command that you got to, followed by* `go`*; that should take you back to where you were …*

- `LISTPSW`

  Enter the above command to list the current address location of your ASMPGM under the **INSTR ADDR** column

- `l 0r:15r`

ASM2|230808-2334

List the 'current' content of the 16 registers, 0-15.Observe register 15 (15r) is the address of ASMPGM which will be executed shortly. Register 14 is the address of TSO Ready environment which called TEST facility. When TEST facility is ended, control is returned to the address in register 14

- `l 15r% length(240)`

  List the content starting at address location based on register 15. Observe the content is machine instructions and data from the compile output.

  For example, the first byte, x'47' machine instruction (OPCODE) was created by the branch, B mnemonic at Stmt 18

- `l 15r% length(240) c`

  List the content starting at address location based on register 15 in character format. Observe the literal text embedded in the assembler program. The first instruction in the program is to branch around the literal "eye catchers" embedded in the executable program

- `l +0 length(240)`

  List the content starting 'relative' address location 0. Observe content is the same as listing 'absolute' address location starting at 11F00.

  The program executable area can be referenced by either 'relative' or 'absolute' addressing

- `at +14 (l 0r:15r)`

  at +14 is Stmt 22 – SETUP label and STM assembler mnemonic Stopping at +14 means the instruction at +14 is NOT executed until proceeding past +14 location

  Enter the above command to stop program execution at 'relative' address location 14, then list the contents of all the registers (using `l 0r:15r`)

- `at +28 (l 0r:15r)` at +28 is Stmt 34 – L 2,=C'Begin' to load register 2 with content of character string 'Begin'.

Enter the above command to stop program execution at ‘relative’ address location 28, then list the contents of all the registers

- `at +2C (l 2r)`

  at +2C is Stmt 35 – LA 2,=C’Begin’ to load register 2 with address location of character string ‘Begin’. Enter the above command to stop program execution at ‘relative’ address location 2C

- list content of register 2 which now contains the result of L 2,=C’Begin’
- `at +30 (l 2r;l +E8 length(5) c)`

  at +30 is Stmt 37 – subtract register 2 from itself, SR 2,2, filling register 2 with zeroes Enter the above command to stop program execution at ‘relative’ address location 30, then

- list content of register 2 containing result of LA 2,=C’Begin’

  Observe register 2 contains an ‘absolute’ address location, then the ‘absolute’ address location is listed.The content of the ‘absolute’ address location is displayed

- `at +32 (l 2r)`

  at +32 is Stmt 38 – load register 2 with the number 4, L 2,=F’4’

  Enter the above command to stop program execution at ‘relative’ address location 32, then - register 2 should now contain zeros as a result of the previously executed SR 2,2

- `at +36 (l 2r)`

  at +36 is Stmt 39 – load register 3 with the number 1, L 3,=F’1’

  Enter the above command to stop program execution at ‘relative’ address location 36, then check register 2. Observe register 2 changed from 0000000 to 0000004 as the result of L 2,=F’4’

- `at +3A (l 2r:3r)`

  at +3A is both Stmt 41 and Stmt 42 where:

Stmt 41 has label LOOP at the +3A Defined Storage location, LOOP DS 0H

Stmt 42 results in add 1 to register 3, A 3,=F'1'

Enter the above command to stop program execution at 'relative' address location 3A

```
l 2r:3r
```

list content of registers 2 and 3 to that registers 2 equals 00000004 and register 3 equals 00000001 for the first time through the loop.

The program execution is designed to branch back to the +3A location, therefore, setting a stopping point at +3A will be executed again

- at +3E (l 2r:3r)

  at +3E is Stmt 43 - branch on count, BCT 2,LOOP

  Enter the above command to stop program execution at 'relative' address location 3E.

  Each BCT execution results in decrement of register 2 by 1

  The result of BCT 2,LOOP is to branch to label LOOP at +3A until register 2 is zero

  The next stop is at +3A, storage address location of label LOOP adding 1 to register 3 until register 2 is zero

- Enter the following commands to be executed when register 3 is zero, terminating the loop

```
at +42
at +46
at +4A
at +84
```

- list content of registers 2 and 3, then branch to LOOP

  Once register 2 is zero, then execution will stop at +42

You will need to enter go numerous times to proceed with program execution

When execution stops at +42, then

- l 2r:3r Register 2 and 3 show the final results of the loop Register 2 contains 00000000 as a result of BCT count decrement during each loop execution Register 3 contains 00000005 as a result of adding 1 during each loop execution
- l +d8

  Location +D8 has the content of HALFCON HALFCON is defined as a halfword, ‘H’, therefore the first 4 bytes, 0020 represents the halfword

- go to continue execution

  When execution stops at +46

  l 3r Stmt 45 - STOP1 LH 3,HALFCON was just executed

  Register 3 is a full word, therefore load halfword, ‘LH’, resulted in register 3 containing 00000020

  l +D0

  Content of value FULLCON is displayed, ‘FFFFFFFF’

  However, the assembler Define Constant statement is DC F’-1’

  Why? FULLCON is an example of Two’s Complement

  Two’s Complement Explained

  For the purpose of computer arithmetic operation, FULLCON is -1

- go
- When execution stops at +4A

Stmt 46 - STOP2 A 3,FULLCON was just executed

    l 3r

Register 3, 0000001F, is the result of adding x'20' to x'-1'

Before continuing with program execution, list value of address location for HEXCON (+D4)

    l +D4

Observe address location currently shows 00009ABC

- go
- When execution stops at +84,

Stmt 47 - STOP3 ST 3,HEXCON was just executed ST 3,HEXCON stores the content of register 3 into address location HEXCON

    l 3r

Register 3 still contains the result of the previous add operation unaltered by ST op code

    l +D4

HEXCON address location was written over using the ST, Store, op code with content of Register 3

- go

The last go resulted in ASMPGM program termination returning control to caller TSO TEST facility

Enter end to terminate TSO TEST facility, returning control to TSO

ASM2|230808-2334

5 MODIFY AND RE-COMPILE

Copy your SOURCE member ASMPGM to ASM2PGM:

1. Update the JCL to ensure output produces a load module called **ASM2PGM**
2. Use register 6 for all operations where register 2 was previously used
3. Use register 7 for all operations where register 3 was previously used
4. Initialize the register used to sum each add operation in the loop with a zero value
5. Execute the loop 10 times adding 5 to the register being used to sum each add operation
6. Submit and verify results via the Job output.

Remember that the register used to store the sum of the arithmetic add operation is a hexadecimal value.

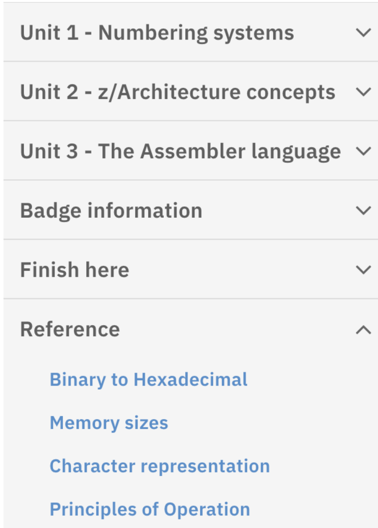
ASM2|230808-2334

6 VALIDATION AND NEXT STEPS

Submit the **CHKASM2** jcl located in **ZXP.PUBLIC.JCL** and review completion code for **0000**

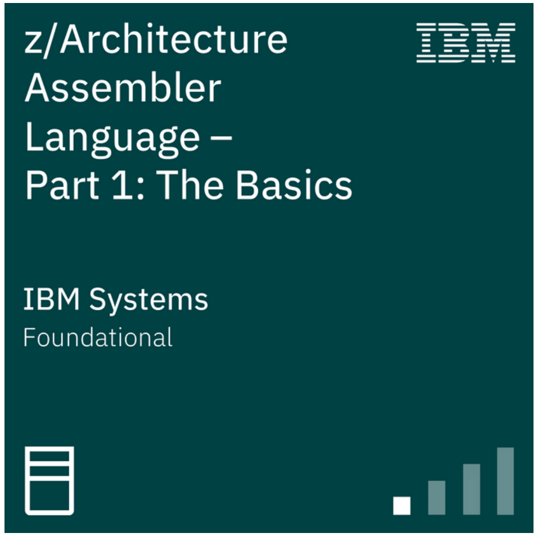
Once you have completed the updates correctly and received credit, maybe you would like to get a bit more detail about how the IBM System 390 intruction set works, how data is represented for computation, and larger-scale applications are buit.

Check out the The z/Architecture Assembler language course series which has just launched on the IBM Training platform.



ASM2|230808-2334

You will be able to earn the z/Architecture Assembler Language - Part 1: The Basics Credly badge.



ASM2|230808-2334