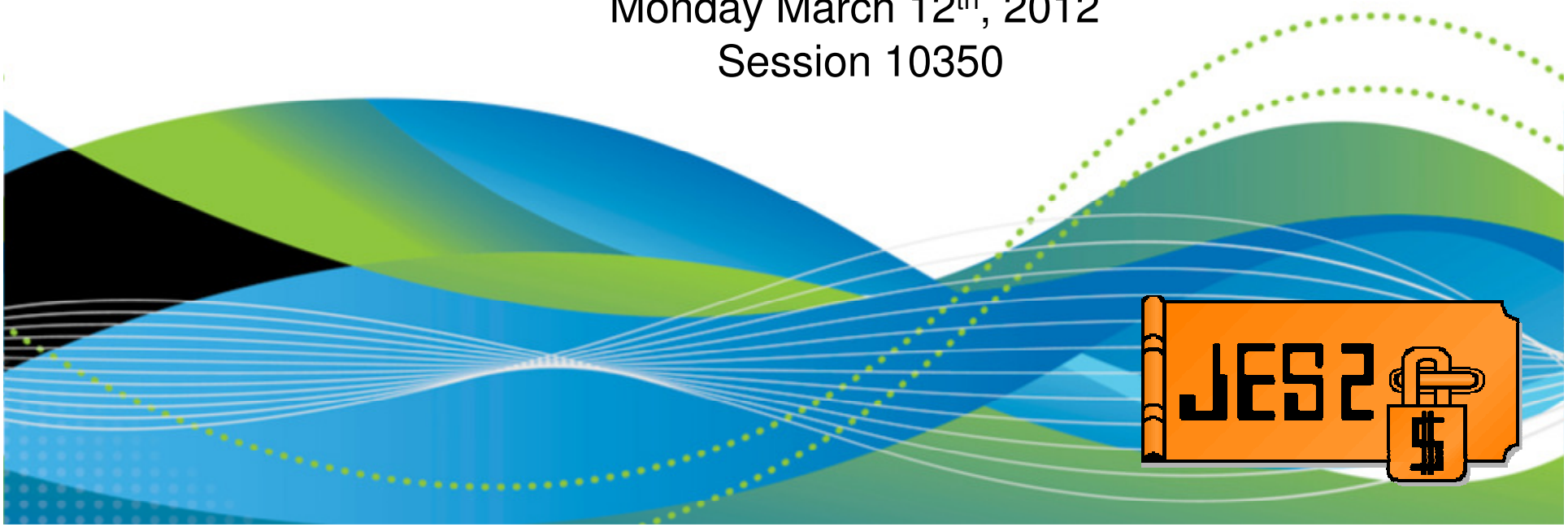
#SHAREorg

SHARE
Technology · Connections · Results

My Favorite HLASM Features
A JES2 Perspective
(These are a few of my favorite things)

Tom Wasik
IBM

Monday March 12th, 2012
Session 10350



A bit of history

- Older IBM assemblers (F, XF, H)
  - Did the job, not “feature rich”
  - Early feature was Extended Mnemonics
    - B, BE etc instead of BCR x
- SLAC assembler mods
  - Enriched the “language” of assembler
    - Dependent and Named usings
  - Improved readability
    - Using at the top of the page
  - SLAC - Stanford Linear Accelerator Center
- HLASM – The high level assembler
  - SLAC plus so much more
  - Assembler as a language and not just machine code

Assembler Programmer Stereotypes

- Assembler is THE programming language
  - You control what the machine does
  - Operating system services are only available in assembler
  - High level languages are too slow
- Real programmers don't need listings
  - Review using compare listing
  - Assembler listing are hard to read on a 24x80 green screen
  - What can the listing tell me that I cannot see in the source

Reality Check

- Assembler code is hard
  - Must keep track of many details
    - Oh what the heck is in that register
    - Which register is available
  - Low information density
    - Lots of code to do a simple function
    - Cannot always get the big picture
- Must leverage EVERY feature to make your life easier
  - Comments are a great help but require human to update
  - Assembler features can help document the code
- Listings can help understand the code
  - 24x80 screen? Get a bigger screen! Try 62x160.

Named USINGs

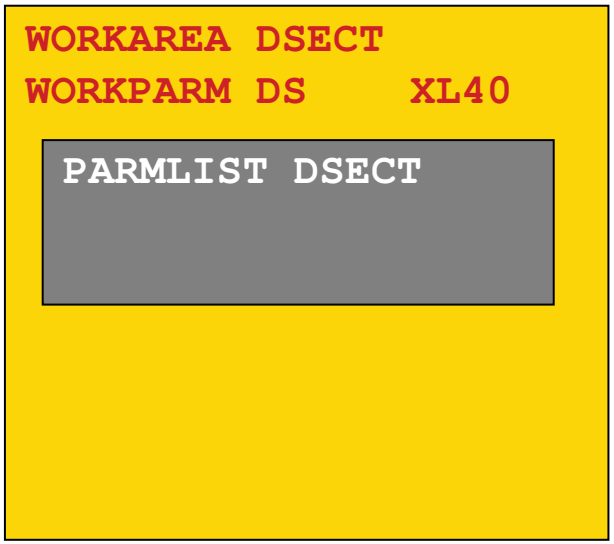
- Same DSECT pointed to by 2 or more registers
  - Adding to a linked list
    - Previous and current list element pointer
  - Copy data from one instance to another
    - Old list element copied to new list element
- Multiple USINGs on same DSECT with unique labels

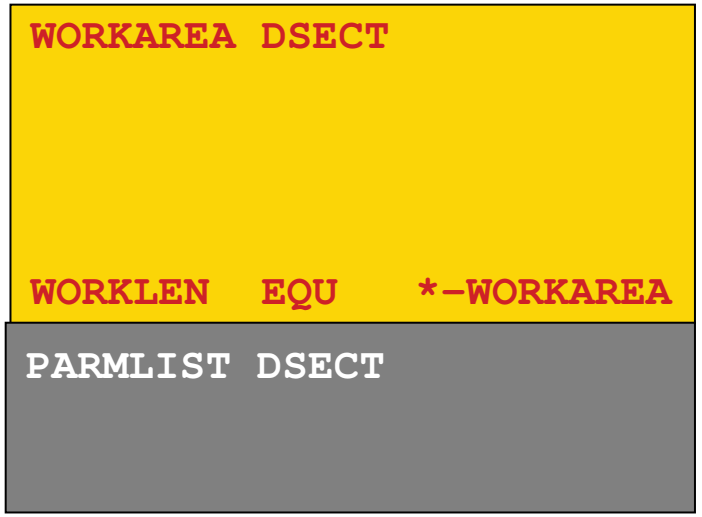
```
NEW      USING ELEMENT,R2       New list element
OLD      USING ELEMENT,R3       Original element
         MVC   NEW.FIELD,OLD.FIELD
```

- Did not need to name both but it adds to understanding
- Short USING names aids readability
  - In example use N instead of NEW and O instead of OLD

Dependent USINGs

- Multiple DSECTs adjacent to or imbedded in one another






```
USING WORKAREA,R8
USING PARMLIST,WORKPARM
```

```
USING WORKAREA,R8
USING PARMLIST,WORKPARM+WORKLEN
```

- Reduces number of base registers needed

Dependent USINGs Example

- Examine the case of a request header and multiple request mappings

- Traditional mapping uses ORG statements
- Needs only one base register to address structure
- But what happens if code to process request 1 type references REQ3COD?
- Wrong data will be accessed
- Imagine if I did not use numbers in label names
  - How would you spot error?

```
REQUEST DSECT ,
REQID   DS    CL4
REQTYPE DS    X
   :
REQORG  DS    0F
REQ1DAT DS    A
REQ1PRM DS    A
        ORG   REQORG
REQ2CLR DS    CL8
REQ2TIM DS    D
        ORG   REQORG
REQ3INF DS    CL32
REQ3COD DS    F
```

Dependent USINGs Example

- Recode example using multiple DSECTs

- Code to process header
  ```
  USING REQUEST,R4
  ```
- Code to process request 1
  ```
  USING REQ1,REQORG
  ```
- At end of request 1 code
  ```
  DROP  R4
  USING REQUEST,R4
  ```
  Clears REQ1 using
- Code to process request 2
  ```
  USING REQ2,REQORG
  ```
- Code in request 1 referencing REQ3COD gets error!

```
REQUEST DSECT ,
REQID   DS    CL4
REQTYPE DS    X
   :
REQORG  DS    0F
REQ1    DSECT ,
REQ1DAT DS    A
REQ1PRM DS    A
REQ1    DSECT ,
REQ2CLR DS    CL8
REQ2TIM DS    D
REQ3    DSECT ,
REQ3INF DS    CL32
REQ3COD DS    F
```

Dependent USINGs Example

- What if you cannot recode the DSECT (IBM DSECT)

- Code that processes header can be prevented from accessing request data
  `USING (REQUEST,REQORG),R4`
- Limits what fields can be addressed
  - Based on OFFSET not location in DSECT
- No help with code that processes requests
- That is why Label naming conventions are important

```
REQUEST DSECT ,
REQID   DS    CL4
REQTYPE DS    X
   :
REQORG  DS    0F
REQ1DAT DS    A
REQ1PRM DS    A
        ORG   REQORG
REQ2CLR DS    CL8
REQ2TIM DS    D
        ORG   REQORG
REQ3INF DS    CL32
REQ3COD DS    F
```

USING Warning Level

- What happens when (unnamed) USINGs overlap?
  - 2 Registers can address the same data
  - USINGs tell the assembler register to use to access data
  - One of the using is ignored, but which one?
    - The one with the lower register is ignored
- Avoid all confusion by not having overlapping USINGs
- Tell assembler to flag all ambiguities as errors
  - USING(WARN(15)) flags potential problems as errors
  - Specify on *PROCESS statement, first line in assembly
- First time may flag things that work as errors
  - Fix the errors, it is worth it

USING Warning Level

- Here is a common case WARN(15) flags

```
            USING CODE,R8
CODE        ……
            LA    R10,WORKAREA
            MVC   0(L'TEMPLATE,R10),TEMPLATE
            USING TEMPLATE,R10
            MVC   MGSJOB,JOBNAME
            :
TEMPLATE    DC    C'THE JOB NAME IS xxxxxxxx'
MSGJOB      EQU   *-8,8
```

- The code “works” but both R8 and R10 can address TEMPLATE

USING Warning Level

- Here is an easy fix

```
            USING CODE,R8
CODE        ……
            LA    R10,WORKAREA
            MVC   0(L'TEMPLATE,R10),TEMPLATE
W           USING TEMPLATE,R10
            MVC   W.MGSJOB,JOBNAME
            :
TEMPLATE    DC    C'THE JOB NAME IS xxxxxxxx'
MSGJOB      EQU   *-8,8
```

- There are other ways around this

*PROCESS and SUPWARN

- *PROCESS conflicts with PARM can cause warnings
  - Results in non-zero assembler return codes (RC=2)
  - Can mess up JCL and other process that expect RC=0
- Suppress unneeded warning messages with SUPRWARN
  - Code SUPRWARN(436,437) on *PROCESS

- In general, specify parameters on *PROCESS instead of PARM
  - As code writer you control the assembly and do not depend on JCL or other techniques

FLAG(PAGE0)

- Spot the error in this code

```
CLI    COMEWORK,C'R'
JE     COF3098A
MVC    TKNWRK-TKN(R10),COMTKN
CLI    COMEWORK,C'L'
JNE    COF3098U
CLC    R1,=A($MAXLNES)
JH     COF3098U
LLGH   R0,$NUMLNES
LTR    R0,R0
JZ     COF3098U
```

FLAG(PAGE0)

- Did you see BOTH errors?

```
CLI    COMEWORK,C'R'
JE     COF3098A
MVC    TKNWRK-TKN(R10),COMTKN
CLI    COMEWORK,C'L'
JNE    COF3098U
CLC    R1,=A($MAXLNES)
JH     COF3098U
LLGH   R0,$NUMLNES
LTR    R0,R0
JZ     COF3098U
```

FLAG(PAGE0)

- Both are (were) common things I coded wrong
- FLAG(PAGE0) flags implicit references to location 0 as an error
  - Specify on second line (or anywhere else) as
    - ACONTROL FLAG(PAGE0)
  - This could flag legit references as errors
  - In this case code the 0 reference explicitly

```
MVC   PSATOLD-PSA(,0),TCBADDR
```

- What about pesky system macros?

```
PUSH  ACONTROL,NOPRINT
ACONTROL FLAG(NOPAGE0)
…Pesky macro call…
POP   ACONTROL,NOPRINT
```

Listing Features
Using at the Top of the Page

- Helps answer the question what's in a register
- Keep USINGs accurate to minimize errors
  - Don't set using for entire routine if base not immediately set

```
COMM      HASP COMMAND SERVICES  -- ACE SERVICE ROUTINES
  Active  Usings: ACT,R2  ACE,R3  MONCB,R4  ASEO(X'FE0'),R4+X'20'
  Loc     Object Code       Addr1     Addr2     Stmt  Source Stateme
00000466 A774 00B9                   000005D8  56001+          JNZ
                     R:5 00000000                56003          USING
0000046A 1851                                    56005          LR
0000046C BF68 C11F                   0000EC1B    56006          ICM
00000470 5060 5004                   00000004    56007          ST
00000474 D203 5000 C0C4 00000000     0000EBC0    56008          MVC
```

Listing Features
Using at the Top of the Page

- Consider dummy USING for non-DSECT register usage
  - Counts, RCs Etc
- Define a DSECT with nothing in it

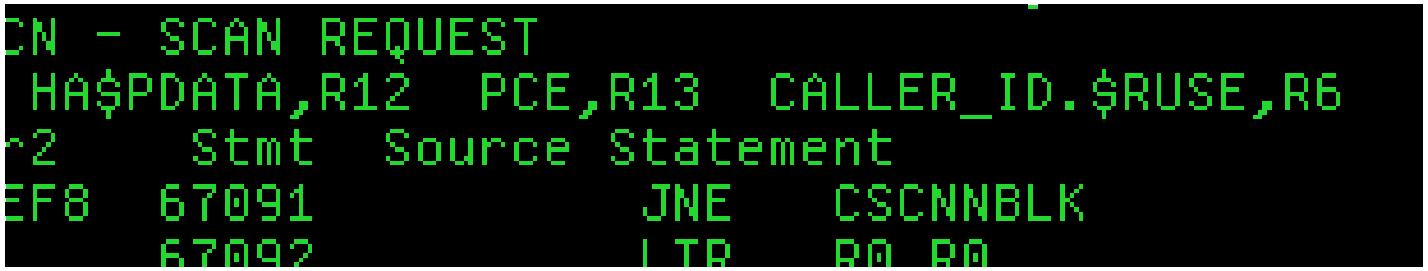
```
$RUSE   DSECT ,                  Dummy DSECT
```

- Use with a named using to document register usage

```
CALLER_ID USING $RUSE,R6 Establish USING
```

- When done just drop the named using

```
DROP  CALLER_ID
```



Macro and Copy Source in Listing

- Tells you where all Macros came from

```
L2 MZ13PTF  MACLIB   J1                    J2-112    $BAT      $BUFFER   $CADDR    $DAS      $HASPEQU  $HASPGBL
                                                     $HCCT     $HCT      $JQE      $MODULE   $PARMLST  $QSE
                                                     $SBWA     $SDB      $SJB      $SJXB
L3 MZ13PID  MACLIB   J1                    J2-112    $AMODE    $CALL     $CB       $CCW      $CHEK     $CMB
                                                     $CRTSYSN  $CSPLOPN  $DCT      $DECODE   $DORET    $DSCA
                                                     $DSIX     $ENTRY    $ENVIRON  $ERPL     $ERROR    $FREEBUF
                                                     $FREG     $FREMAIN  $GETADDR  $GETFLD   $GETMAIN  $GETRTN
                                                     $HASB     $HASXB    $HFAME    $IOT      $IRE      $IRIS
                                                     $IRWD     $JCT      $JRW      $LRC      $LRCGET   $MIT
                                                     $MITETBL  $MODEND   $MPRINT   $NJEWORK  $PADDR    $PAIR
                                                     $PATCHSP  $PBLK     $PCE      $PDDB     $POST     $PSV
                                                     $REGTEST  $RUSE     $SAFINFO  $SCAT     $SCR      $SIG
                                                     $SJIOB    $SPID     $SYMCB    $S35D     $TAB      $TQE      $TRE
                                                     $TRKCNV   $TRX      $VERIFY   $WSP      $XECB     $XIT      $XPL
                                                     IAZDSINF  IAZSPLIO
L5 MVSZ13A  MACLIB   T1                    J2-7D1    ABEND     ACB       ACBVS     ADSR      ALESERV   ATTACH    CVT
                                                     DCBD      DEQ       ENQ       ESTAEX    IDARMRCD  IDAVSACB
                                                     IDAVSCB3  IDAVSOPT  IDAVSRPL  IEAMSCHD  IECDIOSB  IECDRQE
                                                     IEESMCA   IEFTCT    IEZBITS   IEZDEB    IEZWPL    IFGACB
                                                     IFGACBVS  IFGEXLST  IFGEXLVS  IFGRPL    IFGRPLVS
L6 MVSZ13B  MACLIB   T1                    J2-7D1    IHAASCB   IHAASSB   IHAASXB   IHAECB    IHAPSA    IHAPSAE
                                                     IHAPSW    IHAPVT    IHARB     IHASCB    IHASDWA   IHASPP
                                                     IHASRB    IHASSL    IHASTCB   IHAVRA    IHBINNRR  IHBRDWRS
                                                     IHBSETR   IKJRB     IKJTCB    IOSDIEDB  IOSDIOBE  ISGYCON
                                                     ISTACBEX  ISTEXLEX  ISTGAPPC  ISTGLBAL  ISTRPLEX  ISTRPLFB
                                                     ISTRPL6X  ISTUSFBC
L7 MVSZ13C  MACLIB   T1                    J2-7D1    LOAD      NIL       OIL       PGSER     POST      READ      RPL
                                                     RPLYS     SDUMPX    SETRP     SPLEVEL   STIMERM   STORAGE
                                                     SYMREC    SYSSTATE  TCBTOKEN  VRADATA   VSMLOC
```

- Helps debug assembler setup problems

Unreferenced Labels

- Unreferenced labels slow understanding of code
  - Who branches into the middle of this logic?
- I generally try to delete them in code when possible
- Table at bottom of listing
- Specify on PARM using XREF(SHORT,UNREFS)

```
COMM                                Unreferenced Symbols Defined in CSECTs
  Defn  Symbol
 70627  CJS7954B
 70732  CJS8012B
 91454  CMDEND
 61834  CMVCENT
 56979  COFCLRTY
 56980  COFJOBE
 56976  COFJOBS
 60855  COFJSQUE
 56798  COFLIM
 59571  COFRTCNB
 59576  COFRTCSR
 58702  COFRTRSC
 56981  COFSTCE
```

Unreferenced Labels

- Table often has fluff in it (labels in MACROs, equates)
- Copy table to a file and sort by first column
  - Statement where label was defined
- Then focus in on statements where your code is located

LOCTR

- Not many programmers know what LOCTR is
  - Stands for location counter
  - Lets you group code next to each other in the source but far apart in final object deck
- Great when trying to get rid of base register for a routine
  - When routines get too large for one base register
    - Or however many base registers there are
- Instead of loading R12 with code base, make it a data base
- Use LOCTR to put data elsewhere

LOCTR Example

```
MAIN     CSECT
         USING SORT,R12
SORT     SAVE ,
         LR   R12,R15
         :
         MVC  MSGWRK,ERR1
         :
         EX   R4,SORTCLC
         :
SORTCLC  CLC  0(*-*,R3),0(R4)
         :
         RETURN ,
         :
ERR1     DC   C'BOOM!'
         LTORG ,
```

```
MAIN     CSECT
         USING SORTDATA,R12
SORT     SAVE ,
         LARL R12,SORTDATA
         :
         MVC  MSGWRK,ERR1
         :
         EX   R4,SORTCLC
         :
SORTDATA LOCTR
SORTCLC  CLC  0(*-*,R3),0(R4)
MAIN     LOCTR
         :
         RETURN ,
         :
SORTDATA LOCTR
ERR1     DC   C'BOOM!'
         LTORG ,
MAIN     LOCTR ,
         :
```

LOCTR

- This assumes you have moved to JUMP
- Branch tables are also an issue with this
  - Do a LARL to set a base

```
          LARL   R14,TABLE
          B      0(R15,R14)
TABLE     B      XXXXXXXX    +0
```

- What about those pesky system macros
  - Set up temp (limited) base register for macro

```
          BASR   R9,0                    Est temp local
          USING  (*,AFTERLAB),R9           addressability
          Pesky macro call
AFTERLAB  DS     0H                      Limit of addr'blty
```

Miscellaneous Stuff

- CEJECT allows for prettier listing
  - CEJECT 10 does an EJECT if less than 10 line left on page
  - I use when writing repeated segments of code to get page breaks where I needed them
  - MVS IPCS example:

```
BLSQMFLD NAME=PCEID,VIEW=X'0200'
SPACE 1
CEJECT 10
BLSQMFLD NAME=PCEUSER0,VIEW=X'0200'
SPACE 1
CEJECT 10
BLSQMFLD NAME=PCEUSER1,VIEW=X'0200'
SPACE 1
CEJECT 10
BLSQMFLD NAME=PCEPOSTD,VIEW=X'0200',NEWLINE
```

Miscellaneous Stuff

- Amaze your fellow programmers with &SYSSEQF
  - Use in a MACRO to get the callers sequence number
    - Columns 73 to 80
  - Pass this and &SYSECT (section name) to your macro
  - If you have an error, put both in the error message

    Error in XYZ service called from MAIN at sequence 12340000
  - If they wonder how you did it, tell then you read the listing in your service and watch their puzzled look.

Miscellaneous Stuff

- You can control what OPCODEs HLASM allows
  - PARM OP(xxx) specified the opcode table
  - Can be updated by ACONTROL
  - Allow only what your hardware supports
    - Current z/OS support use ZOP
  - Complete list
    http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/asmr1020/5.3
  - Newer values
    - ZS3 - z9-109 instructions
    - ZS4 - z10 instructions
    - ZS5 - z196 instructions (with PK97799)
- Want an OPCODE list? PARM(OP(xxx,LIST))

Jump to Wrong Code

- Ever copy some code and forget to change something?
  - Say the target of a jump instruction?
  - And copied it from the same module?
- Wild jumps are easier to create than wild branches
- Results can be very hard to detect
  - Copied code that jump to a return macro
    - Original code just returned
    - New code was supposed to set a return code

Jump to Wrong Code

- Try added the following to your SAVE (or routine entry) macro

```
        DS    XL65535
```

- Add before the label that gets generated
- Could get lots of strange errors but look for

```
ASMA320W Immediate field operand may have
  incorrect sign or magnitude
```

- Potential jumps to other routines will show up
- DO NOT FORGET TO DELETE THE CHANGE AND RE-COMPILE
- When co-workers ask how you found the bad jump just say you couldn't sleep and were reading the code.

Questions?

Session 10350