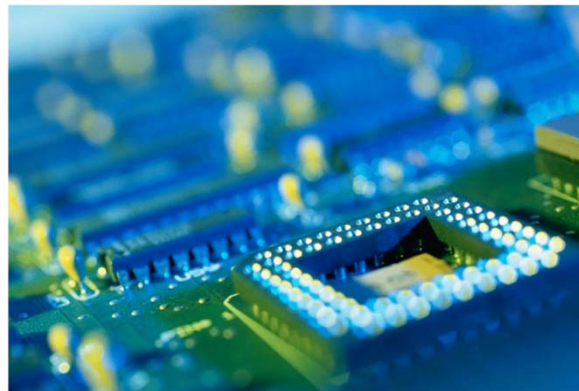




Why Assembler is a 21st Century Language





Authors

This presentation was prepared by:

Kristine Harper
Associate Developer

NEON Enterprise Software, Inc.
14100 Southwest Freeway
Sugar Land, TX 77479
Tel: 281.207.4978
Fax: 281.207.4973
E-mail: kristine.harper@neonesoft.com

This document is protected under the copyright laws of the United States and other countries as an unpublished work. This document contains information that is proprietary and confidential to NEON Enterprise Software, which shall not be disclosed outside or duplicated, used, or disclosed in whole or in part for any purpose other than to evaluate NEON Enterprise Software products. Any use or disclosure in whole or in part of this information without the express written permission of NEON Enterprise Software is prohibited.

© 2006 NEON Enterprise Software (Unpublished). All rights reserved.



Presentation Outline

- Little about myself
- Why use Assembler?
- My Father's Assembler...
- Today's Assembler – new and cool features!
- The future of Assembler is bright



A Little About Me





About Me

- Graduated from the University of Arizona 2005 with a degree in Computer Science
- Was on the gymnastics team there all 4 years
- Moved back to Houston from Tucson to work for NESI
- For 5 summers, I worked for NESI as a software developer, coding in HLASM and enhancing mainframe software
- zNextGen Project Manager for SHARE





About Me

- How did I even get into this business?
- It all began with “Take Your Daughter To Work Day” when I was 12...





Why Use Assembler?





Why Use Assembler?

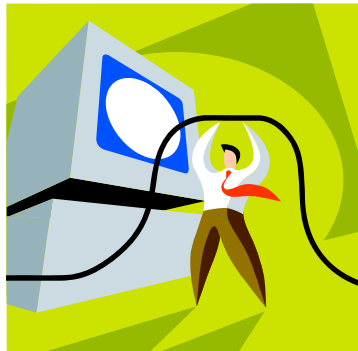
One Cool Fact...

- 98% of all instructions *executed* on the mainframe are written in Assembler!
- Consider DFSORT, IEBGENER, ISPF, TSO and IMS itself
 - Anything that is high-volume execution is written in Assembler
- Some shops may have a lot of COBOL programs, but these source code lines are instructions written, not instructions executed
- SMF records can prove this
 - Run an SMF record filter and see what is consuming CPU time!



Why Use Assembler?

- Most of you are probably familiar with the benefits of this language
- The benefits that existed when Assembler was born still exist today
- But there are some new advantages that could be used to convince even my generation 😊





Why Use Assembler?

Classic arguments to use Assembler

- Optimization
 - Coding in Assembler forces a programmer to focus more on the inner-workings of the machine
 - Knowing the hardware and which instructions are best can help your programs run faster and more efficiently
 - “Assembly language continues to hold a core position in the programming world because of its similar structure to machine language and its close links to underlying computer-processor architecture and design. These features allow for high processing speed, low memory demands and the capacity to act directly on the system’s hardware.” –Sivarama Dandamudi



Why Use Assembler?

Classic arguments to use Assembler



- Memory
 - Application programs can run above the 16MB line
 - Dynamic storage allocation
 - In-storage objects are only as big as they need to be
- Error deflection
 - Simple Assembler routines can help avoid simple problems
 - Errors can be “deflected” and solved



Why Use Assembler?

Classic arguments to use Assembler

- Operating System services
 - These facilities offer performance benefits and can be used via Assembler with little overhead
 - Data spaces, subtasks, VLF (Virtual Look-aside Facility), reenterability, and concurrent access to multiple datasets
- Code tends to be more condensed and can execute faster
- It's a rich and stable host language



Why Use Assembler?

New arguments to use Assembler

- **Structured programming is possible!**
 - No more “spaghetti” code filled with GOTO’s
 - Can write clean, readable and understandable code
- HLASM has a powerful macro facility
 - SPMs: Structured Programming Macros
- New tools and techniques make programs are easier to debug



Confidential Material of NEON Enterprise Software, Inc.



Why Use Assembler?

New arguments to use Assembler

- Using ISPF to edit and create Assembler programs is easier than ever
- Supports interfacing with other languages
- New instructions are popping up all the time
 - Including new 64-bit instructions
- Storage use above the 2GB bar



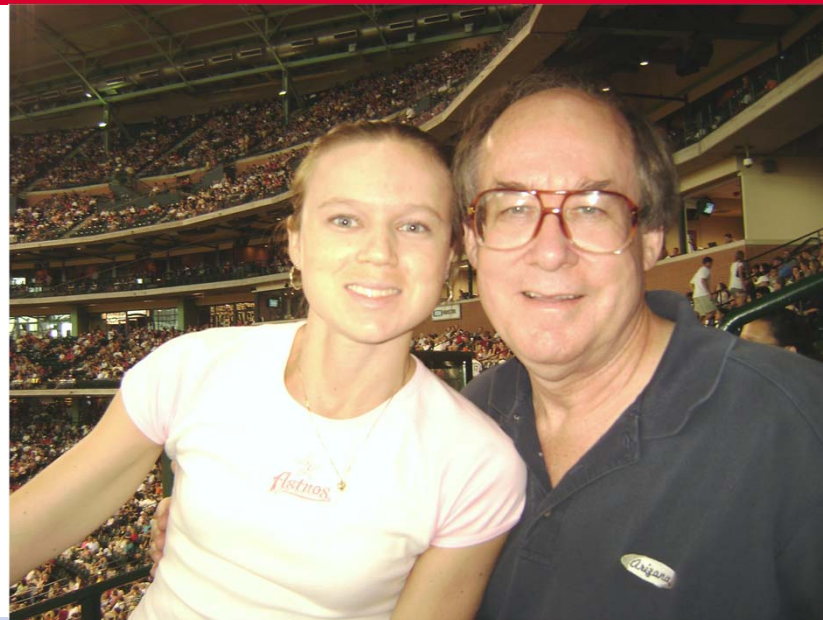
Why Use Assembler?

New arguments to use Assembler

- HLL concepts such as inheritance, abstraction, polymorphism and reuse
- When programmers take advantage of the new features of Assembler (and HLASM), code is easier to analyze and maintain...and pass on to the next generation!



My Father's Assembler





My Father's Assembler

Does
this
look
familiar?

```
154270      CLI      0(R1),VALUE_A
154280      JNE      LABEL_1
154290      .
154300      .          (VALUE_A code)
154301      .
154302      J          LABEL_4
154303 LABEL_1      DC      0H
154304      CLI      0(R1),VALUE_B
154305      JNE      LABEL_2
154306      .
154308      .          (VALUE_B code)
154309      .
154310      J          LABEL_4
154311 LABEL_2      DC      0H
154312      CLI      0(R1),VALUE_C
154313      JNE      LABEL_3
154314      .
154315      .          (VALUE_C code)
154316      .
154317      J          LABEL_4
154318 LABEL_3      DC      0H
154321      .
154322      .          (all other code)
154323      .
154324 LABEL_4      DC      0H
```



My Father's Assembler

Or this?

```

000100 RTN1      DC      0H
000200
000300 * Find the entry
000400 FIND      DC      0H
000500          L        x,y
000600          L        a,b
000700 FIND_1    DC      0H
000800          LTR      x,a
000900          JNE      FIND_2
001000          LHI     ...
001100          GOTO     SETMSG2
001110 FIND_2    DC      0H
001200          C        ...
001300          JE        PROC
001310          L        a,b
001320          GOTO     FIND_1
001330
001340 * Process the entry
001400 PROC      DC      0H
001500          MVC      ...
001600          L        ...
001700          ST       ...
001800          TM       ...
001900          JZ        PROC_1
001910          MVI     ...
002000          GOTO     ACCOUNT
002010 PROC_1    DC      0H
002020          MVC      ...
002021          MVC      ...
002022          CLC     ...
002080          JE        ACCOUNT
002090          ST       ...
002100

```

```

002200 * Perform Accounting
002300 ACCOUNT    DC      0H
002500          L        ...
002510          L        ...
002520          LHI     ...
002530 ACCT_1    DC      0H
002600          CLI     ...
002800          JE        SETMSG1
002810          L        ...
002820          A        ...
002830          ST       ...
002900          TM       ...
003000          JZ        ACCT_2
003010          L        ...
003020          A        ...
003030          ST       ...
003100 ACCT_2    DC      0H
003200          LTR      ...
003500          JH        SETMSG1
003501          LA     ...
003510          JCT      x,ACCT_1
003520          GOTO     RTN2
003521
003530 * Set Messages
003540 SETMSG1    DC      0H
003550          .
003560          .
003570 SETMSG2    DC      0H
003580          .
003590          .
003591          .
003600 RTN2      DC      0H
003800          .
003900          .

```

Enterprise Software, Inc.

(Courtesy of
Ed Jaffe)



My Father's Assembler

- GOTOs and Jumps were as popular as reality TV today!
- Labels, labels, labels everywhere!
 - Branch targets...tempting!
 - I want to change this label name...
- Ever try to cut/paste a code fragment filled with labels and GOTOs?





My Father's Assembler

- “Top Down” design was rare – most Assembler programs stuck to “flat” design techniques
- Difficult to create a mainline of control flow
 - Do you really know where that Branch is going to?
- No nesting
 - Code duplication was common
 - Size of routines was out of control!



My Father's Assembler

- Code indentation was rare
 - Do you like to read books with no paragraphs or chapters?
- Order of complexity of code was larger than necessary
 - From program to program, there wasn't much standardization or uniformity
- Do you really want to read someone else's code over and over just so you can understand what's going on?

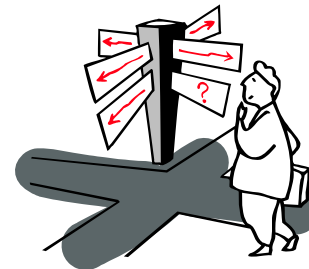


e Software, Inc.



My Father's Assembler

- No more line ID requirement
 - IBM used to require every line be marked with an ID
- Lots of switches were used to control looping
- Passing on assignments was difficult
 - New programmers could not easily understand author's intent
 - Maintaining and updating someone else's code took more time than it should





My Father's Assembler

- Programmers didn't code that way "just because"
- Machines were `s l o w e r` and programs needed to be coded as efficiently as possible
 - Every instruction counted
 - Care was taken to create least amount of overhead as possible
- Memory was precious
 - 256K didn't get you very far!



Today's Assembler: New and Cool Features





Today's Assembler

- Many new features and tools have been added over the years
- Make this language competitive with other HLLs...Assembler is not dead!
- Still efficient, just in a more contemporary way
- HLASM is enriched every year and its SPMs make coding easier without sacrificing performance



Today's Assembler

New Enhancements

- New USING statements
 - Can make simultaneous references to multiple instances of a control section
 - Can map > 1 DSECT per register
 - Can specify fixed relationships among DSECTs
 - Labeled, dependent and labeled dependent USINGs
 - Makes coding simpler – clean and readable



Today's Assembler

New Enhancements

- Many updates/extensions to the macros
- Diagnostics have been improved
- ADATA file captures all assembly information
- Interfaces with I/O exits





Today's Assembler

New Enhancements

- AMODE and RMODE operand extensions for 64-bit addressing
- New instructions for the z9 machine
- HLASM for Linux on zSeries is completely compatible with HLASM on MVS, VM and VSE



Today's Assembler

New Instructions

- BAS/BASR: Branch and Save (Register)
 - Replaces BAL/BALR
 - Added to support 31-bit return addresses
- AHI, CHI, LHI, MHI: Halfword Immediate's
 - Work the same as AH, CH, LH and MH, but a 16-bit binary value is used instead
 - Ex: CH R1,=H'3' → CHI R1,3
- MSR: Multiply Single
 - Works like most Multiply's, but uses only 2 registers





Today's Assembler

New Instructions

- CUSE: Compare Until String Equal
 - Compares 2 strings, looking for an identical substring that exists in both
 - Uses 6 registers
- CLST: Compare Logical String
 - Compares 2 strings whose addresses are in the 2 registers
- MVST: Move Logical String
 - Moves the string at the location in the 2nd register to address specified in the 1st



Today's Assembler

New Instructions

- SRST: Search String
 - Searches a string for a specific character
 - Simpler and faster than TRT or CLI loop
- CVBY, CVBG, CVDY, CVDG: Convert Binary and Decimal Extended
 - Allows for extended displacement
 - CVBG and CVDG use 64-bit register
- CSP/CSPG: Compare and Swap Purge (Grande)
 - Same as CS, but a purging operation is performed if equal



Today's Assembler

New Instructions

- IIHH, IIHL, IILH, IILL: Insert Immediate
 - Places halfword into specific register location
- NIxx, OIxx, XIxx: Boolean Immediate
 - xx: HH, HL, LH or LL
 - Halfword boolean operations
- LPQ/STPQ: Load/Store Register Pair
 - Works with pair of 64-bit registers and 16 bytes of storage
 - Can be used in place of two LGs/STGs



Today's Assembler

New Instructions

- TMH/TML: Test Under Mask High/Low
 - Test the high/low order bytes of the register with a 16-bit operand
- TRE: Translate Extended
 - Super translate instruction – no 256 byte limit!
- TRTR: Translate and Test in Reverse
 - Similar to TRT, but proceeds right to left



Today's Assembler

New Instructions

- FLOGR: Find Leftmost One Bit Grande Register
 - Scans 64-bit register left to right to find 1st one bit
- SLLG, SRLG, SLAG, SRAG: Shift Grande
 - 64-bit register shifting
 - Right, left, logical, arithmetic
- MANY OTHERS!



Today's Assembler

Recent Tools That Boost Productivity Have Been Upgraded (HLASM)

- Disassembler: ASMDASM
 - Creates symbolic Assembler language source from object code
- Cross-Reference Facility: ASMXREF
 - Analyzes code, summarizes symbol and macro use, and locates specific tokens
- Program Understanding Tool: ASMPUT
 - Provides graphic displays of control flow within and among programs



Today's Assembler

Recent Tools That Boost Productivity Have Been Upgraded (HLASM)

- Structured Programming Macros: SPMs
 - Complete set includes if-then-else, do loops, case, select, etc.
- File Comparison Utility: SuperC
 - Enhanced SuperC includes remarkable file-search and date-handling capabilities



Today's Assembler

Debugging is a lot easier with z/XDC!

- z/XDC helps programmers to easily find & fix bugs – systems-level debugging tool
- In object code within any CSECT within any module running under any RB under any TCB within any address space on the system...including much of the operating systems' internals!
- A ***really neat tool*** not only for debugging, but for seeing what your module is really doing, learning how instructions work & how registers change
- Very useful – NEON Enterprise Software, Inc. loves it!
- www.colesoft.com



Today's Assembler

Structured Programming!

- Program flow is easily understood just by looking at the code
- Hierarchical call/return paths
- HLL-like code but without the overhead
- HLASM SPMs keep getting better
- “Hide” branching
- SPMs are an additional cost item – HLASM Toolkit (PN5696234)...but it's worth it!

**Developers Against
Spaghetti Code!**





Today's Assembler

Structured Programming!

- IF (condition)
 code...
 ELSEIF (condition) ←Optional
 code...
 ELSE ←Optional
 code...
ENDIF



Today's Assembler

Structured Programming!

- IFs

IF (LTR,R15,R15,Z)

If clean result

IF (ICM,R14,15,FIELD1,NZ)

If FIELD1 has value

IF (CLI,0(R15),EQ,0),AND, If byte is available

(TM,FLAG1,FLAG1_ON,O)

And our flag is on



Today's Assembler

Structured Programming!

- DO Loops
 - DO, DO FROM/TO/BY, DO INF, DO WHILE/UNTIL
 - Can loop a certain number of times, while/until a condition is met, or infinitely
 - Can use DOEXIT to opt out of a loop
 - Can nest DO loops and IFs



Today's Assembler

Structured Programming!

- DO Loops

- DO WHILE=(LTR,R1,R1,NZ) While R1 has a value
- DO FROM=(R1) From value in R1
 - Loops n times as specified in R1, decrementing R1 as it goes
- DO UNTIL=(CLI,0(R1),EQ,C' ') Until space is found



Today's Assembler

Structured Programming!

- SELECT
 - WHEN (condition)
 - WHEN (condition)
 - OTHRWISE
 - ENDSEL
- Condition is an “IF-like” expression



Today's Assembler

Structured Programming!

- SELECT

SELECT TM,FLAG1,O

WHEN HEX

BAS R14,RTNHEX

WHEN CHAR

BAS R14,RTNCHAR

WHEN NUM

BAS R14,RTNNUM

OTHRWISE

BAS R14,RTNOTHER

ENDSEL

Select based on data type

Hex data

Character data

Numeric data

All other data



Today's Assembler

Structured Programming!

- CASEENTRY Rx, VECTOR=listtype, POWER=n
CASE a,c,d
CASE b,c
CASE f
...
ENDCASE



Today's Assembler

Structured Programming!

- Not used often, but works well as a branch table after returning from a lower-level module
- POWER=n (optional)
 - Case numbers are multiples of that power of 2
 - POWER=3 means case numbers are multiples of 8
- VECTOR=B/BR (optional)
 - Specifies that a branch vector is to be generated instead of an address vector (fewer instructions)



Today's Assembler

Structured Programming!

- CASEENTRY R14,VECTOR=B,POWER=2 Act on Ret. Code

CASE 4 Macro error

BAS R14,MACROERR

CASE 8 Invalid value error

BAS R14,INVALERR

CASE 12 Severe error

BAS R14,ABORTERR

ENDCASE



Today's Assembler

Structured Programming!

- STRTSRCH (any DO operands)
EXITIF (condition)
ORELSE
ENDLOOP
ENDSRCH
- Also not used too often, but is self documenting
- Can be used to code more complex loops



Today's Assembler

Structured Programming!

- STRTSRCH UNTIL=(LTR,R2,R2,Z)

LR R3,R6
offset

LR R6,R2
offset

LH R15,BLK_LEN(R2)

EXITIF (C,R15,GE,MAXLEN)

ST R3,PREV_VAL
we're done

ORELSE

LH R2,BLK_OFFSET

ENDLOOP

LA R15,8

ENDSRCH

Search for space

Save previous

Save current

Get length

Length ok?

Save value -

Length too short

Try next value

R2 is zero

Set return code



Today's Assembler - Examples

DO loop: Convert reason code

```
149800      ICM    R4,15,DCFDL_RSN      Get the reason code
149900      LA     R1,L'DCFDL_RSN      Set length
150000
150100      DO     FROM=(R1)
150200          SRDL  R4,8                Move first character R5
150300          SRL   R5,24               Put to low order
150400          IF    (C,R5,GE,=A(X'000000F0')),AND,
150500          (C,R5,LE,=A(X'000000F9'))
150600              S    R5,=A(X'000000F0')
150700          ELSE
150800              S    R5,=A(X'000000B7')
150900          ENDIF
151000          LR     R14,R5              Save nibble
151100          SRDL  R14,4                Save in R15
151200      ENDDO
151300
```



Today's Assembler - Examples

IF/DO: Message checking

```

112310 ***
112320 *** Check for successful message
112330 ***
112340 L      R14,IMSCMTST_LENSAVE      Get string length
112350
112360 IF      (CLC,=CL8'DFS0488I',EQ,IMSCMTST_AREA) Command OK msg?
112370     SL      R14,=F'13'          Adjust length for checking
112380     LA      R4,IMSCMTST_AREA+9  Point past message ID
112390     DO      FROM=(R14)          Loop thru to look for 'RC= 0'
112400         DOEXIT (CLC,=CL5'RC= 0',EQ,0(R4)) Cmd successful
112410         AHI      R4,1          Keep checking
112420     ENDDO
112430
112440 IF      (LTR,R14,R14,NZ)          RC= 0 found
112450     IF      (CLC,DBBKKEYE,EQ,=A(DBBK EQU)) Increment correct count
112460         L      R1,DBBK_CMDECB+12 Current successful DBD cmd count
112470         LA      R1,1(,R1)        Increment
112480         ST      R1,DBBK_CMDECB+12 Save
112490 *
112500 *      Set DBBLK flag bit
112510 *
112520     L      R1,DBBK_CMDECB        Get flag value
112530     OI      DBBK_FLG4,X'00'     *** Executed Instruction ***
112540     EX      R1,*-4              Set appropriate success flag
112550 ELSE
112560     L      R1,DSBK_CMDECB+12     Current successful DBR cmd count
112570     LA      R1,1(,R1)            Increment
112580     ST      R1,DSBK_CMDECB+12    Save
112590 *
112600 *      Set DSBLK flag bit
112610 *
112620     L      R1,DSBK_CMDECB        Get flag value
112630     OI      DSBK_FLAG2,X'00'     *** Executed Instruction ***
112640     EX      R1,*-4              Set appropriate success flag
112650     ENDIF
112660     ENDIF
112670     ENDIF

```



Today's Assembler - Examples

SELECT/IF/DO: Update based on function

```

067100      SELECT CLI,DCFEIB_FUNC,EQ          Select depending on function
067200      WHEN GS#TERM                        TERM
067300      WHEN GS#ENQUEUE                     ENQUEUE
067700      OTHERWISE                           All other functions
067800      STCK DCFEXIB_TIMESTAMP
067900 P      USING DCFEIB,R8                    Establish addressability
068000 N      USING DCFEIB_IMSTBL_AREA,R2
068100 O      USING DCFEIB_IMSTBL_AREA,R1
068200      IF (ICM,R8,15,NT_TOKNAREA,NZ),AND, If we have a token and +
068300      (CLC,P.DCFEIB_TMSTMP,GT,DCFEXIB_TIMESTAMP) Timestamp OK
068400      LA R1,P.DCFEIB_IMSTBL_AREA
068500      LA R2,DCFEB_IMSTBL_AREA
068600      L R3,P.DCFEIB_IMSCNT
068700      ST R3,DCFEB_IMSCNT
068800      MVC DCFEIB_TMSTMP,P.DCFEIB_TMSTMP
068900      IF (LTR,R3,R3,NZ)                    If count is not zero
069000      DO FROM=(R3)                          Loop for each one
069100      MVC N.DCFEIB_IMSID,O.DCFEIB_IMSID
069200      MVC N.DCFEIB_MVSID,O.DCFEIB_MVSID
069300      MVC N.DCFEIB_CTLASID,O.DCFEIB_CTLASID
069400      LA R1,DCFEB_IMSTBL_ENTRY_LEN(,R1)
069500      LA R2,DCFEB_IMSTBL_ENTRY_LEN(,R2)
069600      ENDDO
069700      ENDIF
069800      ELSE                                  Else if we don't have token
070000      IF (C,R15,GT,=A(RC_WARN))           More than a warning?
070100      J DCFEXIB_EXIT
070200      ENDIF
070300      L R14,DCFEXIB_TIMESTAMP
070400      AL R14,=A(60)                         DBRC info good for 60 sec.
070500      ST R14,DCFEB_TMSTMP                   Save timestamp
070600      ENDIF
070700      DROP P
070800      ENDSEL

```



Today's Assembler - Examples

DO/IF/CASEENTRY: Examine statement

```
079300      ST      R14,BLDCMA_CPA5000      A(where to go back to)
079400
079500      DO      UNTIL=(CR,R1,GT,R9)      Examine statement
079600          LR      R14,R9                A(end)
079700          SR      R14,R1                L(to scan)
079800          LA      R2,0                  Initialize
079900          LR      R15,R1                A(Start of text scan)
080000          EX      R14,CPATRT1          Scan statement
080100          IF      (LTR,R2,R2,NZ)        If something found
080200              CASEENTRY R2,VECTOR=B,POWER=2 Act on result of TRT
080300
080400              CASE 4                    A quote was found
080500                  BRAS R14,BLDCMA_6000 Analyze & copy data
080600
080700              CASE 8                    A slash was found
080800                  BRAS R14,BLDCMA_6300 Analyze & copy data
080900
081000              CASE 12                  A plus or minus was found
081100                  BRAS R14,BLDCMA_6600 Analyze & copy data
081200
081300              ENDCASE
081400
081500          ELSE
081600              LR      R1,R9                A(last byte of data)
081700              BRAS R14,BLDCMA_6900        Copy remaining data
081800          ENDIF
081900          LA      R1,1(,R1)              Point past byte examined
082000      ENDDO
082100
082200      L      R14,BLDCMA_CPA5000          A(where to go back to)
082300      BR      R14                        Return
```



Today's Assembler - Examples

SELECT: Convert values correctly

```
052600      SELECT CLC,0(12,R2),EQ      Select on space allocation type
052700          WHEN =CL12'CYLINDERS'
052800              M      R0,DSVA_TRKS_CYL      Convert Cylinders to Tracks
052900          WHEN =CL12'TRACKS'
053000      OTHRWISE ,
053100          LA      R2,0      Convert Blocks to Tracks
053200          ICM      R2,3,DSVA_BLOCKS_TRK
053300          DR      R0,R2      Divide
053400          IF (LTR,R15,R0,NZ)      Remainder?
053500              LA      R1,1(,R1)
053600          ENDIF ,
053700      ENDSEL ,
```



Today's Assembler - Examples

STRTSRCH/DO: Search for segment

```
045600      L      R6,DBOPSEGA          Load A(SGMT)
045700      USING SGMT,R6
045800
045900      L      R5,DBOPNOSEGF        R5 = # of segments
046000      STRTSRCH FROM=(R5)         Search for current segment.
046100
046200      EXITIF (CLC,SGMTSEGNM,EQ,XPABCURSEG) Exit if found.
046300
046400      DO UNTIL=(ICM,R6,B'1111',SGMTPARADDR,Z) Now backup looking
046500      DOEXIT (CLC,SGMTSEGNM,EQ,XPSYMSEGNAME) for referenced seg.
046600      ENDDO
046700
046800      ORELSE
046900      AHI     R6,SGMTELEN          R6 = A(Next segment)
047000      ENDLOOP
047100
047200      ENDSRCH ,                   Continue looking for current seg
047300      DROP   R6
```



The Future of Assembler is Bright





The Future of Assembler is Bright

- These are just some of the aspects of Assembler that make it a language of the 21st century
- There are many other new instructions and enhancements to come
- HLASM is ever improving – 5 releases since 1992



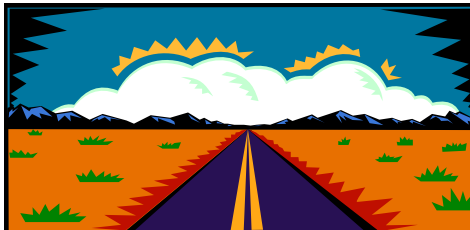
The Future of Assembler is Bright

- Ok, so the language has moved into the 21st century...
- But what about the programmers??
- The language isn't dying, but the programmers...are retiring
- Kids aren't just coming out of college knowing the capabilities of Assembler
 - It's Java, java, java!



The Future of Assembler is Bright

- Who will learn this language and take advantage of its 21st century powers?
- It is a problem, but things are being done
 - Note the IBM Academic Initiative
- Assembler programmers will arise!
- I am sure our future is bright!





Recap

- Many reasons to use Assembler – classic and new arguments
- My Father's Assembler got the job done, but many of the techniques and thought-processes were outdated
- Today's Assembler has evolved into a 21st century language, with updated functionality, instructions and control flow
- Assembler isn't going anywhere – we just need to make sure its programmers aren't



Resources

- A.F. Kornelis – High Level Assembler © 2003, <http://www.hlasn.com/english/hlasn.htm>
- Sivarama Dandamudi – *Introduction to Assembly Language Programming*, 1998
- Edward Jaffe – “Structured Assembler Language Programming Using HLASM”, SHARE 106 Session 8175, March 2006
- John Ehrman – “High Level Assembler: New and Useful Features in Release 5”, SHARE 106 Session 8164, March 2006
- John Ehrman – “Extending the Life Cycle of Legacy Applications”, SHARE 103 Session 8132, August 2004
- John Ehrman and John Dravnieks – “HLASM: Stalking the New Opcodes”, SHARE 106 Session 8162, March 2006



Resources

- Avri Adleman - “More zArchitecture Goodies”, SHARE 106 Session 8173, March 2006
- Avri Adleman - “You Don’t Have to Run AMODE 64 to Exploit the Joys of z/Architecture”, SHARE 105 Session 8172, August 2005
- Vlad Pirogov – *The Assembly Programming Master Book*, 2004
- IBM Press Release - *High Level Assembler for z/OS & z/VM & z/VSE V1.5 Offers Support for New Capabilities and Increases Ease of Use*, June 8, 2004
- Randall Hyde – “Why Learning Assembly Language is a Good Idea”, May 2005,
<http://www.onlamp.com/pub/a/onlamp/2004/05/06/writegreatcode.html>



Thank you!



Intelligent Solutions for Enterprise Data. Depend On It.