Assembler BootCamp Plus: Instructions Everyone Can Use

(Created by)
John Dravnieks, IBM Australia
(dravo@au1.ibm.com)

(Presented by Dan Greiner and John Ehrman)

SHARE 117, Orlando, FL
Thursday, August 11, 2011
Session 9286

Agenda

- Bit shifting
- Single byte operands
- Halfword operands
- Multiple byte operands
- Variable length operands
- Character translation
Definitions

- Characters used in instruction mnemonics
  - \textbf{G} - \textit{Grande} - 64-bit operand
  - \textbf{F} - \textit{Fullword} - 32-bit operand
  - \textbf{H} - \textit{Halfword} - 16-bit operand
  - Single byte operands
    - \textbf{B} - \textit{Byte} (signed 8 bit value)
    - \textbf{C} - \textit{Character}
  - \textbf{L} - \textit{Logical} - unsigned, or Load and clear
  - \textbf{Y} - 20-bit displacement

Definitions

- Parts of a 64-bit register

<table>
<thead>
<tr>
<th>HH</th>
<th>HL</th>
<th>LH</th>
<th>LL</th>
</tr>
</thead>
<tbody>
<tr>
<td>HF</td>
<td>LF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \textbf{H}=\textit{High}, \textbf{L}= \textit{Low}, \textbf{F}=\textit{Fullword}
Bit shifting

- RS-type format instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>R₁</th>
<th>B₂</th>
<th>D₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>

- R₁ - Source and target

- 2nd operand address
  - NO storage reference
  - Last 6 bits used as shift amount

Bit shifting (continued)

- Two directions, two types, and two sizes
  - Left or Right
  - Logical or Arithmetic
  - Single or Double register

- 8 mnemonics - Shift ...
  - SLA  SLDA
  - SLL  SLDL
  - SRA  SRDA
  - SRL  SRDL
Bit shifting \textit{(continued)}

- 64-bit register instructions
  - \textbf{NO} 64-bit-register-pair (128-bit) shifts
  - Single-length: \texttt{SLAG}, \texttt{SRAG}, \texttt{SLLG}, \texttt{SRLG}

- Separate source (R\textsubscript{3}) and target (R\textsubscript{1}) registers

- Example:
  \texttt{SLAG R\textsubscript{1}, R\textsubscript{3}, D\textsubscript{2} (B\textsubscript{2})}
  - Shifted contents of R\textsubscript{3} goes into R\textsubscript{1}

---

Bit shifting \textit{(continued)}

- Arithmetic shifts:
  - Sign bit not modified
  - Right shifts copy sign bit
  - Left shifts may overflow
  - Condition code set

- Logical shifts:
  - No sign bit
  - Always inserts 0's
  - Condition code not changed
### Bit shifting: example 1

- **SRA 5,16**
  - Object code: X'8A50 0010'
  - c(r5) before: X'8001 0000'  (sign is propagated)
  - c(r5) after: X'FFFF 8001'
  - Condition code 1 set (result < 0)
- **SRA 5,7**
  - Object code: X'8A50 0007'
  - c(r5) after: X'FF00 0200'
- **SRA 5,20**
  - c(r5) after: X'FFFF F800'

### Bit shifting: example 2

- **SRL 5,16**
  - Object code: X'8850 0010'
  - c(r5) before: X'8001 FFFF'
  - c(r5) after: X'0000 8001'  (sign not propagated)
- **SRL 5,7**
  - Object code: X'8850 0007'
  - c(r5) after: X'0100 03FF'  (3 = 0011)
- **SRL 5,20**
  - c(r5) after: X'0000 0800'
Bit shifting: example 3

- **SLA 5,16**
  - Object code: X'8B50 0010'
  - c(r5) before: X'0000 8001'
  - c(r5) after: X'0001 0000'
  - Condition code 3 set (Overflow)

- **SLA 5,7**
  - Object code: X'8B50 0007'
  - c(r5) after: X'0040 0080' (CC2, no overflow)

- **SLA 5,30**
  - c(r5) after: X'4000 0000' (overflow)

---

Bit shifting: example 4

- **SLL 5,16**
  - Object code: X'8950 0010'
  - c(r5) before: X'0000 8001'
  - c(r5) after: X'8001 0000'

- **SLL 5,7**
  - Object code: X'8950 0007'
  - c(r5) after: X'0040 0080'

- **SLL 5,30**
  - c(r5) after: X'4000 0000'
Bit shifting (continued)

- **Rotate Left Single Logical**
  
  - \( \text{RLL} \ (G) \ R_1, R_3, D_2 \ (B_2) \)
  - Separate target \((R_1)\) and source \((R_3)\) registers

  - **Example:** \( \text{RLL} \ 7, 8, 12 \ (0) \)

  - Before: \( c(R7) = \text{'????????'} \), \( c(R8) = \text{'FEDC0000'} \)
  - After: \( c(R7) = \text{'C0000FED'} \), \( c(R8) = \text{'FEDC0000'} \)

---

Bit shifting: uses

- **Arithmetic Operations**
  - Fast multiplication or division by a power of 2
  - Hashing algorithms

- **Masking**
  - In conjunction with Boolean operations
    - Exclusive OR (XOR), OR, AND
  - Extracting data
    - Merged or compressed data

- **Encryption**
Single byte operands

- **Insert Character**
  - IC \( R_1, D_2 (X_2, B_2) \)
  - Copies a single byte from storage into low order byte of \( R_1 \)
  - **Note:** rest of \( R_1 \) register unchanged

- **STore Character**
  - STC \( R_1, D_2 (X_2, B_2) \)
  - Copies the low order byte of \( R_1 \) into storage

---

Single byte operands: example 1

- IC \( 7, 0 (0, 11) \)
  - Object text \( \text{x'4370 B000'} \)
  - \( R11 \) points to storage byte containing \( \text{x'A5'} \)

- \( c(R7) \) before \( \text{x'1234 5678'} \)
- \( c(R7) \) after \( \text{x'1234 56A5'} \)
  - Remainder of register \( R7 \) is unchanged

- Condition code is unchanged
Single byte operands

- **Load Logical Character**
  - \( \text{LL(G) CR} \ R_1, R_2 \)
  - \( \text{LL(G) C} \ R_1, D_2 (X_2, B_2) \)
  - Clears the register and copies a byte from register or storage into low order byte of \( R_1 \)

- **Load Byte**
  - \( \text{L(G) BR} \ R_1, R_2 \)
  - \( \text{L(G) B} \ R_1, D_2 (X_2, B_2) \)
  - Single byte from register or storage is sign extended and updates the *entire* register

Single byte operands: example 2

- **LLC 7, 0 (0, 11)** Load Logical Character
  - Object text \( 'E370 B000 0094' \)
  - \( R_{11} \) points to storage byte containing \( 'A5' \)
  - \( c(R_7) \) before \( '1234 5678' \)
  - \( c(R_7) \) after \( '0000 00A5' \)
    - Remainder of register \( R_7 \) is zeroed
  - Condition code is unchanged
Single byte operands: example 3

- **LB 7,0 (0,11)**  
  ```
  Load Byte
  ```
  - Object text X'E370 B000 0076'
- **R11 points to storage byte containing X'A5'**
- **c(R7) before X'1234 5678'**
- **c(R7) after X'FFFF FFA5'**
  - Leftmost bit of X'A5' extended to left
- **Condition code is unchanged**

Single byte operands: uses

- **Translation example (we'll use it again):**
  ```
  ... UNPK STRING(L'STRING),HEXDATA(L'HEXDATA+1)  
  * Get data into zoned format.  
  LA 3,STRING Point to STRING.  
  LHI 4,1 Load JXLE increment.  
  LA 5,L'STRING-1(,3) Point at last byte.  
  LOOP IC 2,0(,3) Get next character.  
  NILL 2,X'000F' Remove zone.  
  IC 2,TABLE(2) Use c(R2) as index.  
  STC 2,0(,3) Store "translated" digit.  
  JXLE 3,4,LOOP Loop until finished.  
  ...  
  TABLE DC C'0123456789ABCDEF'
  ```
- **The low-order hex digit of each byte referenced by R3 is replaced by its character representation**
Halfword (two byte) operands

- RX instructions
  - Mnemonic $R_1, D_2 (X_2, B_2)$

- Operand 1 is entire $R_1$ register
  - $\text{STH}$ ignores high order 16 bits of $R_1$, stores only rightmost 16 bits

- Operand 2
  - Halfword in storage
  - Signed value - $\text{LH}$ expands to fullword with sign extension

---

Halfword (two byte) operands (continued)

- Add Halfword $\text{AH}$
- Compare Halfword $\text{CH}$
- Load Halfword $\text{LH}$
- Multiply Halfword $\text{MH}$
- Store Halfword $\text{STH}$
- Subtract Halfword $\text{SH}$
Halfword (two byte) operands

(continued)

- Halfword immediate format
  - Mnemonic $R_1, I_2$
    where $I_2$ is a signed 16-bit field in the instruction

- Add Halfword Immediate $AHI$
- Compare Halfword Immediate $CHI$
- Load Halfword Immediate $LHI$
- Multiply Halfword Immediate $MHI$

Halfword (two byte) operands

(continued)

- Halfword-immediate operands for 64-bit registers:
  ▶ $AGHI, CGHI, LGHI, MGHI$
  ▶ $LGH(R)$

- Long displacement facility (instructions with signed 20-bit displacement)
  ▶ $AHY, CHY, LHY, STHY, SHY$
Halfword (two byte) operands

(continued)

- Register-to-register form: \textbf{L(G)HR}
  - Source is in bits 48-63 of 2nd-operand register
- Load Logical form: \textbf{LL(G)HR, LL(G)H}
  - Remainder of 1st-operand register zeroed
- Load Logical Immediate form: \textbf{LLIxx}
  - Source is in bits 16-31 or 16-47 of the instruction
- Insert Immediate form: \textbf{IIxx}
  - Remainder of 1st-operand register unchanged
- Where \textbf{xx} - HH, LH, HL, LL (See slide 4)

Halfword operands: example 1

- \textbf{LH 0,0 (0,12)}
  - Object text \texttt{X'4800 C000'}
- R12 points to storage containing \texttt{X'B1A4'}
- c(R0) before \texttt{X'FEDC BA98'}
- c(R0) after \texttt{X'FFFF B1A4'}
  - High-order bit of \texttt{X'B1A4'} extended to left
- Condition code is unchanged
Halfword operands: example 2

- CH 10,0(0,11)
  - Object text X'49A0 B000'
  - R11 points to storage containing X'B1A4'
    - Expanded internally to X'FFFF B1A4'
    - If c(R10) = X'FFFF B1A4'
      - Condition code set to 0 (equal)
      - R10 unchanged
    - If c(R10) = X'0000 B1A4'
      - Condition code set to 2 (greater)

Halfword operands: example 3

- CH 10,0(0,11)
  - Object text X'49A0 B000'
  - R11 points to storage containing X'B1A4'
  - If c(R10) = X'FFFF A5A5'
  - Resulting Condition Code?
  - Is R10 unchanged?
**Halfword operands: example 4**

- **LLILH 0,X'\text{A5A5}'**
  - Load Logical Immediate Low High
  - Object text X'A50E A5A5'

- c(R0) before X'FEDC BA98'
- c(R0) after X'A5A5 0000'
  - Remainder of target register is zeroed

- Condition code is unchanged

**Halfword operands: example 5**

- **IIILH 0,X'\text{A5D6}'**
  - Insert Immediate Low High
  - Object text X'A502 A5D6'

- c(R0) before X'FEDC BA98'
- c(R0) after X'A5D6 BA98'
  - Remainder of target register is unchanged

- Condition code is unchanged
Halfword operands: uses

- Record lengths (DCBLRECL)
  - V format records: RDWs, BDWs
- Database records
- Small integers

Multiple byte operands

- Insert Characters under Mask
  - ICM R₁,Mask,D₂(B₂)
  - Copies 0 to 4 bytes from storage into mask-selected bytes of R₁
  - Condition code set
  - Note: Unselected bytes unchanged
Multiple byte operands \textit{(continued)}

- Mask operand is a 4 bit field
  - Bits correspond one to one with bytes of register
    - \texttt{B'1001'} refers to the first and last byte

- Storage bytes are contiguous
  - \texttt{ICM 2,B'1010'},{\texttt{=X'12345678'}}
  - \texttt{c(R2) = X'12??34??'}

\begin{itemize}
  \item \textbf{Compare Logical Characters under Mask}
    \begin{itemize}
      \item \texttt{CLM \(R\_1,\text{Mask},D\_2(B\_2)\)}
      \item Compares 0 to 4 contiguous bytes from storage with mask-selected bytes of \(R\_1\)
      \item Condition code is set
    \end{itemize}
  \item \textbf{STore Characters under Mask}
    \begin{itemize}
      \item \texttt{STCM \(R\_1,\text{Mask},D\_2(B\_2)\)}
      \item Stores 0 to 4 bytes from selected bytes of \(R\_1\) register into contiguous storage bytes
    \end{itemize}
\end{itemize}
Multiple byte operands (continued)

- z/Architecture instructions:
  - CLMY, CLMH
  - ICMY, ICMH
  - STCMY, STCMH

- $H = \text{High-order 32 bits of 64-bit register}$

- Long-displacement format (RSY)

---

Multiple byte operands: uses

- STCM $R_1, B'0111', D_2(R_2)$

  - Stores low-order 24 bits of $R_1$ into contiguous storage bytes

  - Historically important use:
    - STCM $R5, B'0111', \text{Label+1}$
    - Label DC X'bits', AL3(address)
      - DCB address fields
      - CCW address field
Multiple byte operands: uses (continued)

- ICM with mask B'0001'
  ▶ Same as IC, but condition code is set

- ICM with mask B'1111'
  ▶ Same as Load, but condition code is set

- ICM 5,B'1111',24(8) is equivalent to:
  ▶ L 5,24(,8) this
     LTR 5,5 plus this
  ▶ NO index register with ICM

Fullword operands

- z/Architecture with extended immediate facility

  ▶ Load and Test - LT (like L + LTR)
  ▶ 32-bit Fullword Immediate operands:
    - Arithmetic: AFI, ALFI, SLFI
    - Logical AND, XOR, OR: NIHF, NILF, XIHF, XILF, OIHF, OILF
    - Compare: CFI, CLFI
    - Load immediate: LGFI, LLIHF, LLILF
    - Insert immediate: IHIHF, IILF
Variable number of operand bytes

- Q: How would we store HLASM symbols, from 1 to 63 bytes long?
- A1: Update MVC instruction in storage?
  - Reentrancy violation
  - Difficult to debug
  - Data / Instruction cache conflicts?
- A2: Use IC and STC in a loop?
  - Slow
- A3: Use EXecute instruction!

EXecute instruction

- \( \text{EX } R_1, D_2 (X_2, B_2) \)

- Operand 2 - Address of target instruction

- If \( R_1 \) is not general register 0, then low order byte is ORed internally with the second byte of the target instruction

- The target instruction is then performed
  - The target instruction in memory is unchanged!
EXecute instruction (continued)

- Three important points
  - Operands 1 and 2 are not modified
  - The operation is a logical OR
  - When EXecuting variable-length instructions, lengths in object text are one less than actual length

- An example follows

EXecute instruction example

- **EX R4,MOVEIT**

- **MOVEIT MVC TARGET(0),SOURCE**
  - Object text X'D200 bddd bddd'

- c(R4) = X'1234 5602'

- Effective object text X'D202 bddd bddd'

- So three (3) bytes are moved
EXecute instruction: lengths

- R4 in that example holds *machine* length

- If R4 holds *actual* length, then how do we make R4 the machine length (one less)?
  - Any one of these:
    - `S R4,=F'1'` (or `SH R4,=H'1'`) (?)
    - `BCTR R4,0`
    - `LA R4,255(,R4)`
    - `AHI R4,-1` (Recommended!)

EXecute instruction: uses

- Often, the target instruction is SS format, like `MVC`, `CLC`, `TR` or `TRT`
  - Only target instructions not allowed are EX & EXRL

- **NOP** (i.e., `BC 0`) can be EXecuted
  - Use mask of `X'F0'` for unconditional branch
  - Use other mask for program-specified condition
  - Target of `BC 15,...` will always branch, regardless of EX `R1` field
    - However, bits 12-15 of the target can be modified (e.g., `BCR R2` field)

- Example:
  - `EX 0,Target_SVC`
    - Allows shared code (Test and Production) to use different SVCs
Variable number of operand bytes - Take 2

Q: How would we store character strings from 1 to 567 bytes long?

A1: Update instruction in storage (Bad!)
  ▶ Won't work anyway: max length is 256

A2: Use IC and STC in a loop?
  ▶ Even slower

A3: Use EXECute instruction? (Not bad...)
  ▶ Loop moving 256 byte chunks and then an EXECuted move at the end (used in old days)

A4: Use Move Long!

Move Long instruction

- MVCL R₁, R₂
  ▶ MVCL 4, 6 - object text X'0E46'

Operands designate even-odd register pairs:
  ▶ Even register: operand address
  ▶ Odd register (even+1): operand length
    - Source length register has pad character in high order byte
    - Maximum length is 16MB (24 remaining bits of the odd registers)
Move Long instruction (continued)

- All 4 registers may be modified
- Sets condition code
- R0 (implying the pair R0 and R1) is valid
  - Yes, R0 can contain an address!
- Clear a block of storage:
  - LM 0,3,=A(Block,L'Block,0,0)
  - MVCL 0,2 X'00' Pad char in R3

Compare Logical Long instruction

- CLCL R1, R2
  - CLCL 4, 6 - object text X'0F46'
- Same register setup as MVCL
- All 4 registers may be modified - data in storage is NOT modified
- Shorter operand padded with pad character
- Condition code is set
CLCL example

Example of CLCL usage

- LM 2,3,=A(String1,L'String1')
  Target addr, length
- LM 0,1,=A(String2,L'String2')
  Source addr, length
- ICM 1,B'1000',=C' '  Pad byte
- CLCL 2,0
- BE Equal_strings

Extended Move and Compare Long

- Move Long Extended (MVCLE)
- Move Long Unicode (MVCLU)
- Compare Logical Long Extended (CLCLE)
- Compare Logical Long Unicode (CLCLU)

- Lengths can be greater than 16MB
- Pad character formed from 2nd operand
- Unicode: 2 bytes per step
- CC set to 3 if operation is incomplete
Extended Move and Compare Long - examples

```
Compare   CLCLE 2,0,X'40' blank pad
BO        Compare   CC3 test
BE        Equal_strings
```

```
CompUni  CLCLU 2,0,X'020'
BO        CompUni   CC3 test
BE        Equal_strings
```

Move with Optional Specifications

- **MVCOS** \( D_1(B_1), D_2(B_2), R_3 \)
  - Set GPR0 to zero
  - Set R3 operand to **TRUE** length

- Moves 0 - 4096 bytes
  - If true length greater than 4096, then 4096 bytes moved and condition code 3 is set
  - Otherwise, true length bytes moved and condition code 0 is set
Translation

- Q: How to ensure that character data is in upper case?
- A1: Use the IC/STC code earlier (slide 20) with a new table
- A2: Use TRanslate instruction!

TRanslate instruction

- TR D1(L1,B1),D2(B2) SS format
- Operand 1 is source \textit{and} target
- Operand 2 is address of translate table
  - Usually 256 bytes - depends on data

- TR STR,Table
- STR DC C'Hello, World!'
- Table DC C'

(See next page)
TABLE addresses a 256 byte table where each data byte is the desired output byte for that offset. For example, this table would translate lower case EBCDIC to upper case EBCDIC.

```
CAPTABLE DS 0CL256 0 1 2 3 4 5 6 7 8 9 A B C D E F
DC XL16'000102030405060708090A0B0C0D0E0F' 00-0F
DC XL16'101112131415161718191A1B1C1D1E0F' 10-1F
DC XL16'202122232425262728292A2B2C2D2E2F' 20-2F
DC XL16'303132333435363738393A3B3C3D3E3F' 30-3F
DC XL16'404142434445464748494A4B4C4D4E4F' 40-4F
DC XL16'505152535455565758595A5B5C5D5E5F' 50-5F
DC XL16'606162636465666768696A6B6C6D6E6F' 60-6F
DC XL16'707172737475767778797A7B7C7D7E7F' 70-7F
DC XL16'80C1C2C3C4C5C6C7C8C98A8B8C8D8E8F' 80-8F
DC XL16'90D1D2D3D4D5D6D7D8D99A9B9C9D9E9F' 90-9F
DC XL16'AAE1E2E3E4E5E6E8E9EAAABACADAEAF' A0-AF
DC XL16'B0B1B2B3B4B5B6B7B8B9BABBBCBDBDBEBF' B0-BF
DC XL16'C0C1C2C3C4C5C6C7C8C9CACBCCCCDCECF' C0-CF
DC XL16'D0D1D2D3D4D5D6D7D8D9DADDBDCDDDEDF' D0-DF
DC XL16'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF' E0-EF
DC XL16'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEEFF' F0-FF
```
TRanslate instruction - example

- Translate hex data to printable characters

```
UNPK STRING(L'STRING+1),HEXDATA(L'HEXDATA+1)
* Get data into zoned format
LA R5,L'STRING-1 Load machine length
EX R5,TR_INST Perform translation
...
TR_INST TR STRING(0),TABLE Executed TRANSLATE
...
ORG *-240 Position label
TABLE DS 0X Start of table
ORG *+240 Skip to actual data
DC C'0123456789ABCDEF'
```

Related instructions

- Translate and Test
  - TRT D_1 (L_1, B_1), D_2 (B_2) Left to right
  - TRTR D_1 (L_1, B_1), D_2 (B_2) Right to left

- Operands not modified
- Table - operand 1 byte used as index
  - If table byte is zero, scan continues
  - If non zero, scan stops
    - GR1: Address of operand 1 byte
    - GR2: Test-table byte
Related instructions

- Translate Extended
  - **TRE R₁, R₂**
    - First operand address in register R₁
    - First operand length in register R₁+1
    - Translate table address in register R₂
  - Test byte in GR0
    - Translation stops if it matches source byte
    - Registers updated

TRT instruction - example

- Scan for ASCII (**X’20’**) or EBCDIC (**X’40’**) blanks
  - **SR R₂,R₂** Clear R₂
  - **LA R₁,STRING+L'STRING-1** Set R₁ to last byte
  - **LA R₅,L'STRING-1** Load machine length
  - **EX R₅,TRT_INST** Perform scan
  - **JZ No_Blanks** Nothing found (CC 0)
  - **CHI R₂,X’20’** ASCII blank?
  - **TRT_INST TRT STRING(0),TABLE** Executed TRT
  - **TABLE DC 256X'00'** Define 256 byte table
  - **ORG TABLE+X'20'** Move to offset X'20'
  - **DC X'20'** Set non zero
  - **ORG TABLE+X'40'** Move to offset X'40'
  - **DC X'40'** Set non zero
  - **ORG,** Skip to end of TABLE
Summary

Many useful instructions!

- Bit shifting
- Single byte operands
- Halfword operands
- Multiple byte operands
- Variable length operands
- Character translation