Assembler University 206: Powerful New z/Architecture Instructions That Don't Require AMODE(64), Part 1

SHARE 117 in Orlando, Session 9312

Avri J. Adleman, IBM
adleman@us.ibm.com

(Presented by Dan Greiner, IBM)
Monday, 8 August 2011 – 4:30 p.m.

Topics

- Extended displacements
  - Many instructions allow for increased range of base register
- Reduced and enhanced memory access
  - Load, Store, and Insert Immediate Instructions
  - Boolean Immediate Instructions
  - Halfword-register operations
  - Reversed operand access
- Register comparison and testing
  - Registers, storage, swap, sign conversion
- Testing register operands Under Mask: register halfword-immediate
- Arithmetic instructions: 64-bit arithmetic, carry/borrow processing
- High-word instructions ("more registers")

Terminology: all machine generations

Byte	8 bits
Halfword	2 Bytes (16 Bits)
Word (Fullword)	4 Bytes (32 Bits)
Doubleword	8 Bytes (64 Bits)
Quadword	**16 Bytes (128 Bits)**

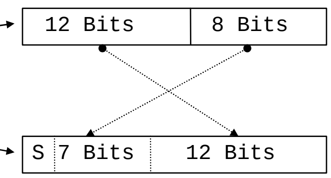
- **Notation:** *64-bit based* **[***32-bit based***]**
  - 64-bit based (Doubleword)
  - 32-bit based (Fullword)
- **Positions:**
  - "*High Order*" refers to the low numbered bits
  - "*Low Order*" refers to the high numbered bits

Extended displacements

- Traditional 12- bit displacements
  - Maximum **+4,095** bytes from origin (base address)
  - Previously, all instructions that use base-displacement addressing
    - Range limits supported by HLASM
      - e.g. **USING (FROM,TO),*register list***
- Extended 20-bit signed displacements
  - **±524,288** (512K) bytes from origin (base address)
    - 8 additional bits appended to the left of 12 bit displacement
      - Illustrated on next slide
    - HLASM range limits apply only to "short" displacements
  - Some old, many new instructions support 20 bit displacements
    - Initial z/Architecture instructions that had reserved fields in instruction format
      - Examples: LG, OG, …
    - New analogues for certain ESA/390 instructions
      - Mnemonics suffixed with "Y"
      - Examples: LY, MVIY, …
    - Consult Principles of Operation; most are very easy to use

Extended displacement: operation

- Signed 20-bit value
  - Internal image
  - Effective value
- Assembler resolution:
  - Priority is to the smallest positive displacement



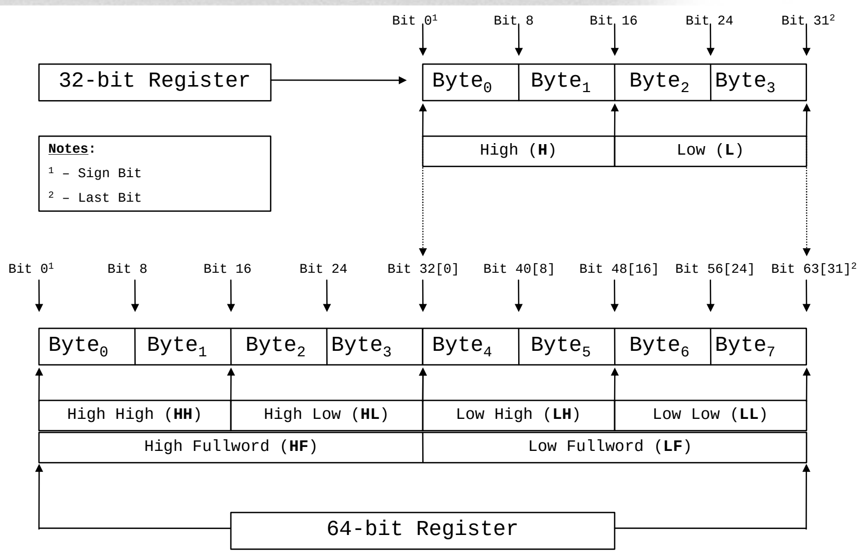
″Traditional″

(Base Register R10)

Extended

```
TEST3    CSECT ,
TEST3    AMODE 31
TEST3    RMODE ANY
         J     START
PROGRAM  DC    CL8'TEST3'
START    DS    0H
         BAKR  14,0
         BASR  11,0
         USING *,11,10
         LA    11,0(,11)
         LR    10,11
         AHI   10,4096
         LAY   2,FARX
         LAY   3,PROGRAM (neg. offset!)
         PR    ,
         DS    XL4096
FARX     DC    CL4'XYZA'
         END   ,
```


Register layout and notation for register-immediate instructions



Instruction mnemonic usage

Mnemonic	Name	Instruction Examples	Additional Remarks
`LL????`	Load Logical	LLGT, LLGC, LLGH, …	Loads specific bytes of a register, fills remainder with zeroes.
`??G??`	64-bit (“Grande”) Register	LGR, AG, LTGR, …	Applies to full 64-Bit register as target or target and source; may widen value with or without sign propagation.
`??F??`	Fullword (“traditional 32-bit register”)	LGF, LGFR, ALGF, …	Applies to 32-bit word as source; value is widened when target is a 64-bit register.
`??T??`	Thirty-One Bit	LLGTR, LLGT	Applies to source as the lower 31 bits: bit 33[1] to bit 63[31]
`??H??`	Halfword (2 bytes)	LGH, AGH, ...	Applies to a halfword (a pair of specified bytes) of a 64 bit register.
`??H??`	High word of a 64-bit register	LMH, STMH	Applies to the high word, bits 0 to 31, of a 64 bit register

`????LL,`
`????LH,`
`????HL,`
`????HH`
	Low-Low
	Low-High
	High-Low
	High-High
		TMLL, LLIHH, …	Specfied halfwords of a 64-bit register, or low and high halves of a 64-bit register

`II????`	Insert-Immediate	IILL, IILH, …	Load specific bytes of a register, leaving remainder alone.

Store/Load (Multiple) high halves of registers

- Store/Load High Half of “*Grande*” Registers
  - Only high word’s 32 bits saved
- Format RSY (extended displacement)
  - **STMH R_1,R_3,D_2(B_2)**

EB	R_1	R_3	B_2	DL_2...	DH_2	26

  - **LMH R_1,R_3,D_2(B_2)**

EB	R_1	R_3	B_2	DL_2...	DH_2	96

- Analogous to STM and LM
  - Acts on range of registers
    - Use multiple-type instruction with $R_1 = R_3$
    - Also LOAD HIGH (LFH) and STORE HIGH (STFH) for loading/storing single high register.

Store/Load 64-bit registers

- STG and LG
  - Store and Load single 64-bit register
  - Analogous to ST (STY) and L (LY)
  - Format RXY:
    - **STG R_1,D_2(X_2,B_2)**
    - **LG R_1,D_2(X_2,B_2)**

E3	R_1	X_2	B_2	DL_2…	DH_2	**24**
E3	R_1	X_2	B_2	DL_2…	DH_2	**04**

- STMG and LMG
  - Store and Load multiple 64-bit registers
  - Analogous to STM (STMY) and LM (LMY)
  - Format RSY:
    - **STMG R_1,R_3,D_2(B_2)**
    - **LMG R_1,R_3,D_2(B_2)**

EB	R_1	R_3	B_2	DL_2…	DH_2	**24**
EB	R_1	R_3	B_2	DL_2…	DH_2	**04**

Load Multiple Disjoint

- **LMD R_1,R_3,D_2(B_2),D_4(B_4)**
  - Format SS:

EF	R_1	R_3	B_2	D_2	B_4	D_4

  - Loads range of full 64-bit registers
  - Uses two different locations
    - High half registers loaded from Arg_2
    - Low half registers loaded from Arg_4
    - Equivalent to doing a LMH and LM in one instruction!

```
*        Example of LMD
         STMH R2,R5,HIREGS
         STM  R2,R5,LOWREGS
         . . .
         LMD  R2,R5,HIREGS,LOWREGS
         . . .
HIREGS   DS 4F    Save High Half
LOWREGS  DS 4F    Save Low Half
```

- Allows AMODE=64 code to load saved "*Grande*" registers from two different save areas (high and low words)
  - Prevents register corruption on needed addresses
- Notes:
  - For performance, use sparingly:
    - Use LMH and LM or LMG if possible
  - There is **no** "*Store Multiple Disjoint*"

Load and Store Pair to/from Quadword

- Load Register Pair from Storage
  - **LPQ R_1,D_2(X_2,B_2)** [RXY-Format]
  - R_1 represents an even/odd 64-bit register pair
  - $D_2(X_2,B_2)$ addresses 16 bytes of storage
    - Must be aligned on a quadword boundary.
  - Process similar to:
    - LG R_1, $D_2(X_2,B_2)$ and LG R_1+1, $D_2+8(X_2,B_2)$
- Store Register Pair into Storage
  - **STPQ R_1,D_2(X_2,B_2)** [RXY-Format]
  - R_1 represents an even/odd 64-bit register pair
  - $D_2(X_2,B_2)$ addresses 16 bytes of storage
    - Must be aligned on a quadword boundary.
  - Process similar to:
    - STG R_1, $D_2(X_2,B_2)$ and STG R_1+1, $D_2+8(X_2,B_2)$

Data-reversing instructions

- Load and Store Reversed
  - Destination bytes are set in reverse order of the source
    - Source bytes from left to right set destination bytes right to left
  - Both source and destination use the same number of bytes
    - Possibly only the low order bytes in a destination register may be used
    - No sign bit propagation
    - Unused bytes in destination register are untouched
  - The bit order in the bytes remains unchanged
- Load Reversed instructions
  - Register to Register: LRVR, LRVGR
  - Storage to Register: LRVH, LRV, LRVG
- Store Reversed instructions: STRVH, STRV, STRVG
  - Register-to-storage form only

Reverse instructions: examples

```
*             c(R2) = X'ABCDEF12'
              LRVR   R3,R2        Register to Register Reverse
*             c(R3) = X'12EFCDAB'
```

```
*             c(R4) = X'01020304' (BEFORE)
              LRVH   R4,HALFWORD   Storage to Register Reverse
*             c(R4) = X'0102D2C1' (AFTER)
HALFWORD      DC     XL2'C1D2'
```

```
*             c(G5) = X'0011223344556677'
              STRVG  G5,DBLWORD    Register to Storage Reverse
*             c(DBLWORD) = XL8'7766554433221100'
DBLWORD       DS     D
```


Register comparison and testing

- Register Comparison with possible widening
  - Register to Register (Similar to CR and CLR)
    - CGR, CGFR, CLGR, *CLGFR*
  - Register to Storage (Similar to C and CL)
    - CY, CG, CGF, CLY, CLG, *CLGF*
  - Compare and Swap (Similar to CS and CDS)
    - CSY, CDSY, CSG, CDSG
  - Compare Logical Characters (Similar to CLM)
    - CLMY, CLMH
- Register Testing with possible widening
  - Load and Test (Similar to LTR)
    - LTGR, LTGFR
- Register Sign Conversion with possible widening
  - Load Complement (Similar to LCR)
    - LCGR, LCGFR
  - Load Positive (Similar to LPR)
    - LPGR, LPGFR
  - Load Negative (Similar to LNR)
    - LNGR, LNGFR

For "Compare Logical Grande with Fullword," widening is with zeroes

All other "Grande with Fullword" compare and/or test, widening is with sign extension

Test Under Mask for register operands

- Test bit settings in registers
  - Similar to the TM instruction
  - **Except**:
    - Test bits in a register (R_1) directly, not storage
    - Mask field maps a halfword (I_2), not a byte
    - Condition code for **mixed!!** (***Different from TM!***)
      - Left most bit tested is zero sets CC = 1
      - Left most bit tested is one sets CC = 2
- Each instruction acts on a specific halfword
  - Four different instructions
    - **TMHH R_1,I_2**
      - Test Under Mask High High (bits 0 to 15)
    - **TMHL R_1,I_2**
      - Test Under Mask High Low (bits 16 to 31)
    - **TMLH R_1,I_2** or **TMH R_1,I_2**
      - Test Under Mask Low High (bits 32[0] to 47[15])
    - **TMLL R_1,I_2** or **TML R_1,I_2**
      - Test Under Mask Low Low (bits 48[16] to 63[31])

Hwd_1	Hwd_2	Hwd_3	Hwd_4

Test Under Mask in registers: examples

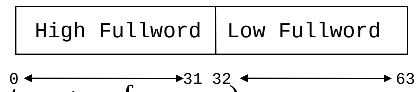
```
* Example #1:
        TMHH R1,X'8000'
        JO   BRANCH
* R1 =  X'F000000000000000'     will branch
* R1 =  X'7000000000000000'     will not branch

* Example #2:
        TMLH R1,X'F000'
        BRC  8,ONES        CC = 0 (JO)
        BRC  4,MIXED1      CC = 1
        BRC  2,MIXED2      CC = 2
        BRC  1,ZEROES      CC = 3 (JZ)
* R1 = X'00000000F0000000'        will branch to ONES
* R1 = X'0000000070000000'    will branch to MIXED1
* R1 = X'0000000080000000'    will branch to MIXED2
* R1 = X'0000000000000000'    will branch to ZEROES

* Example #3: (Set the Condition Code to 2)
        LGHI R1,2
        TMLL R1,X'0003'        Leftmost tested bit = 1
```

Fullword register-immediate instructions (1)

- Similar to Halfword Immediate Instructions
  - 64 bit registers considered as two fullwords
    - **`xxHF`** (high): Bits 0 to 31
    - **`xxLF`** (low): Bits 32 to 63



- Use them to eliminate literals (and storage references)
  - **`IIxF`** – Insert Fullword Immediate High or Low
    - Places fullword into high or low fullword of register
    - Remainder of register is unchanged
    - Condition Code is unchanged
  - **`LLxF`** – Load Logical Immediate High or Low
    - Places fullword into high or low fullword of register
    - Remainder of register is set to 0
    - Condition Code is unchanged

Fullword register-immediate instructions (2)

- Load Immediate
  - Sign bit extended (if necessary for 64-bit operation)
  - Condition code remains unchanged
  - LGFI (64 bits) and LFI (32 bits)
- Arithmetic Immediate
  - Sign bit extended (if necessary for 64-bit operation)
  - Condition code set arithmetically
  - Arithmetic: AGFI and AFI
  - Comparison: CGFI and CFI
- Logical Immediate
  - No sign extension, zero filled (if necessary for 64-bit operation)
  - Condition code set logically
  - Arithmetic: ALGFI, ALFI, SLGFI and SLFI
  - Comparison: CLGFI and CLFI

Load Logical instructions

- Load Logical
  - Loads specified part of a 32 or 64 bit target register
    - Source comes from register, storage or immediate operand
  - Remainder of the target register is **zero** filled, not sign extended!
- Instruction Types
  - Byte to 64 bit register
    - LLGC
  - Halfword to 64 bit register
    - LLGH, LLIHH, LLIHL, LLILH, LLILL
  - Fullword to 64 bit register
    - LLGFR, LLGF

Example: **LLGC R1,=X'FF'**

R1 →

00	00	00	00	00	00	00	**FF**

Example: **LLILH R1,X'C1C2'**

R1 →

00	00	00	00	**C1**	**C2**	00	00

Register-processing instructions

- Load and Test in a single instruction!
  - Load register from storage
    - LTG, LT and LTGF
  - Load register from register
    - LTGR, LTGFR
  - Same as Load, except condition code is set
    - 0 – Result is zero
    - 1 – Result is less than zero
    - 2 – Result is greater than zero
    - 3 – Unused
- Fullword-Immediate instructions
  - Six-byte instructions
  - Four-byte immediate operand
  - Similar to Halfword-Immediate

Handy Dandy LLGT and LLGTR

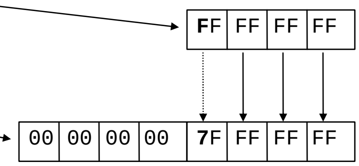
- Load Logical *"Grande"* Thirty One Bits
  - **LLGT R_1,D_2(X_2,B_2)**
    - RXY Format:

E3	R_1	X_2	B_2	DL_2…	DH_2	17

  - **LLGTR R_1,R_2**
    - RRE Format:

B9	17	//	R_1	R_2

- Source (Register or Storage)
  - Fullword, 32 bits (Arg_2)
- Target Register (R_1)
  - Doubleword, 64 bits
    - High word set to all zeroes
    - Low order word copied from source
    - Low order word's high bit 32[0] ***set to 0***



Operand-widening instructions (1)

- Properties
  - Storage or low part of source register to full register
  - No Condition Code set
- From **C**haracter (unsigned byte) without sign extension
  - Storage to register:
    - **LLC** R_1,RX and **LLGC** R_1,RX
  - Register to Register:
    - **LLCR** R_1,R_2 and **LLGCR** R_1,R_2
- From signed **B**yte, with sign extension
  - **LGBR** R_1,R_2 and **LBR** R_1,R_2
- From halfword with sign extension
  - **LGHR** R_1,R_2 and **LHR** R_1,R_2

Insert-Immediate instructions

- Insert Immediate halfwords into a register
  - IIHH, IIHL, IILH and IILL
  - Places halfword into specified register position
  - Remainder of register is unchanged
  - Condition Code is unchanged

Register R1 Before

R1 →

00	**FF**	**00**	**FF**	**00**	**FF**	**00**	**FF**

Example: **IIHL R1,X'ABCD'**

R1 →

00	FF	**AB**	**CD**	00	FF	00	FF

Example: **IILH R1,X'C1C2'**

R1 →

00	FF	00	FF	**C1**	**C2**	00	FF

Boolean-immediate instructions

- Perform Boolean operation on selected register fullword component.
- Properties
  - Only designated halfword or fullword of a "G*rande*" register is operated on
  - Condition code is set as with other boolean operations
- Instructions operating on fullwords
  - And Immediate:
    - **NIHF** R_1,I_2
    - **NILF** R_1,I_2
  - Exclusive OR Immediate:
    - **XIHF** R_1,I_2
    - **XILF** R_1,I_2
  - OR Immediate:
    - **OIHF** R_1,I_2
    - **OILF** R_1,I_2

Boolean-immediate halfword operations

- NI*xx*, OI*xx* (but no XIxx instructions for halfwords!)
  - *xx* = HH, HL, LH or LL
- Performs **halfword** boolean operation into specific register location
- Remainder of register is unchanged
- Sets condition code based on the halfword result

Register R1 Before

R1 →

00	**FF**	**00**	**FF**	**00**	**FF**	**00**	**FF**

Example: **OIHL R1,X'ABCD'**

R1 →

00	FF	**AB**	**FF**	00	FF	00	FF

Example: **NILH R1,X'C1C2'**

R1 →

00	FF	00	FF	**00**	**C2**	00	FF

Load arithmetic instructions: operand widening (2)

- Load full 32 or 64 bit register
  - Source comes from register, storage or immediate operand
  - Sign extension
  - Widening: byte, halfword, or fullword to 64-bit register
- Instruction types
  - Byte to 32- or 64-bit register
    - **LB, LGB**
  - Halfword to 64-bit register
    - **LGHI, LGH**
  - Fullword to 64-bit register
    - **LGF, LGFR**

Example: **LGB R1,=X'A0'**

R1 →

FF	**FF**	**FF**	**FF**	**FF**	**FF**	**FF**	**A0**

Example: **LGF R1,=F'10'**

R1 →

00	**00**	**00**	**00**	**00**	**00**	**00**	**0A**

64-bit arithmetic instructions

- Full 64-bit signed addition and subtraction
  - Analogous to 32-bit arithmetic
    - AG, AGR, SG, SGR, ALG, ALGR, SLG, SLGR
  - Widening from halfword or word to doubleword
    - Sign extension: AGF, AGFR, SGF, SGFR
    - Zero extension: ALGF, ALGFR, SLGF, SLGFR
- Single-register Processing
  - Reduces the need for even/odd register pairs
  - Operand widening in certain cases with sign extension
    - MSG, MSGF, MSGFR, MSGR
    - DSG, DSGF, DSGFR, DSGR ***do*** require register pairs for quotient/remainder!
- Logical Arithmetic on even/odd pairs
  - Allows for 64- or 128-bit *unsigned* product or quotient
  - Unsigned values treated similarly as used with AL, ALR, etc. instructions
    - **ML**, MLG, **MLR**, MLGR
    - **DL**, DLG, DLGR, **DLR** (also require register pairs!)

Note: Instruction(s) in **Bold** are for 32-bit register pairs only

64-bit arithmetic: examples

```
*  Example #1:
             AGF    R1,=F'3'      Note: same as AG R1,=FD'3'
*  Before:   R1 = X'0000000000000001'
*  After:    R1 = X'0000000000000004'

*  Example #2:
             MSGF   R1,=F'3'      Note: same as MSG R1,=FD'3'
*  Before:   R1 = X'0000000000000002'
*  After:    R1 = X'0000000000000006'

*  Example #3:
             DSGF   R2,=F'3'      Note: same as DSG R2,=FD'3'
*  Before:   R2 = ?
*  Before:   R3 = X'0000000000000005' (dividend)
*  After:    R2 = X'0000000000000002' (remainder)
*  After:    R3 = X'0000000000000001' (quotient)
```

Logical arithmetic instructions with Carry and Borrow feature

- New Logical Arithmetic Instructions
  - Performs logical addition or subtraction
    - Similar to the “traditional” AL*x* and SL*x* type instructions
  - Carry or borrow is indicated by the Condition Code
    - Set by previous logical arithmetic statement, as usual
    - Continues to propagate carry or borrow
      - Intermediate instructions must not alter the CC!
- Instructions use 32- or 64-bit registers
  - Addition
    - **ALC**, **ALCR**, ALCG, ALCGR
  - Subtraction
    - **SLB**, **SLBR**, SLBG, SLBGR

Note: Instruction(s) in **Bold** are for 32-bit registers only

Logical Arithmetic Instructions with Carry and Borrow Feature

- Allows easy addition or subtraction of large binary numbers
  - No need to code branches around carry or borrow
    - Condition codes 2 or 3 for add logical
    - Condition code 1 for subtract logical
  - No need to include special instructions for adding or subtracting the carry or borrow
- Process
  - Arithmetic proceeds right to left
  - First instruction is the traditional logical addition or subtraction
    - Preserve the condition code!
  - Remaining instructions are Add With Carry or Subtract With Borrow
    - Propagates the condition code for each successive operation

Logical arithmetic instructions with Carry and Borrow: example

```
* Old Style
     STCK CLOCK
     LM   R2,R3,CLOCK
     LM   R4,R5,FACTOR
     SRDL R2,12

* Addition
     ALR  R3,R5    Add Low
     BC   12,*+8   Carry ?
     AL   R2,ONE   Yes!!!
     ALR  R2,R4    Add High

* Subtraction
     SLR  R3,R5    Subtract Low
     BC   3,*+8    Borrow ?
     SL   R2,ONE   Yes!!!
     SLR  R2,R4    Subtract High

CLOCK     DS   D
FACTOR    DC   FD'nnnnn'
ONE  DC   F'1'
```

```
* New Style
     STCK CLOCK
     LM   R2,R3,CLOCK
     LM   R4,R5,FACTOR
     SRDL R2,12

* Addition
     ALR  R3,R5    Add Low
     ALCR R2,R4    Add High

* Subtraction
     SLR  R3,R5    Subtract Low
     SLBR R2,R4    Subtract High

CLOCK     DS   D
FACTOR    DC   FD'nnnnn'
```


High-word instructions (z196)

- High 32 bits of a 64-bit register
- 16 more 32-bit registers!
  - Add/subtract (signed, logical, immediate)
  - Comparison (signed, logical, immediate)
  - Load/Store (byte, character, halfword, word; register and memory)
  - Logical operations (AND, OR, XOR)
  - Logical shifts
  - Branch Relative on Count
- Many instructions use high- and low-half 32-bit operands
- Add/subtract can be non-destructive

- Examples:
- **AHHLR R_1,R_2,R_3**
  - **R_1 = High-half for sum**
  - **R_2 = High-half operand**
  - **R_3 = Low-half operand**
  - Possible use:
    - Accumulate subtotals in 32-bit low-half of a register
    - Accumulate grand total in 32-bit high-half of the register
- **BRCTH R_1,I_2**
  - Use it for loop counts to free up low-half registers for addressing

High-word instructions: summary

- Mnemonics look complex, but make sense after a while
- Add: **AHHHR, AHHLR, AIH**
  - Logical : **ALHHHR, ALHHLR, ALSIH, ALSIHN** (!)
- Subtract: **SHHHR, SHHLR**
  - Logical: **SLHHHR, SLHHLR**
- Compare: **CHHR, CHLR, CIH**
  - Logical: **CLHHR, CLHLR, CLIH**
- Memory load: **LBH, LHH, LFH**
  - Logical: **LLCH, LLHH**
- Store: **STCH, STHH, STFH**
- Load Immediate: **LLIHF**
- Register Logical load
  - 32-bit: **LHHR, LHLR, LLHFR**
  - 16-bit: **LLHHHR, LLHHLR, LLHLHR**
  - 8-bit: **LLCHHR, LLCHLR, LLCLHR**
- Logical operations **XXXX** R_1,R_2
  - AND: **NHHR, NHLR, NLHR**
  - OR: **OHHR, OHLR, OLHR**
  - XOR: **XHHR, XHLR, XLHR**
- Logical Shifts **xxxx** R_1,R_2,I_3 (!)
  - **SLLHH, SLLLH, SRLHH, SRLLH**