Assembler University 303: A Gentle Introduction to Trimodal Programming on z/Architecture

SHARE 116 in Anaheim, Session 8981

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

(Presented by John Ehrman, IBM)

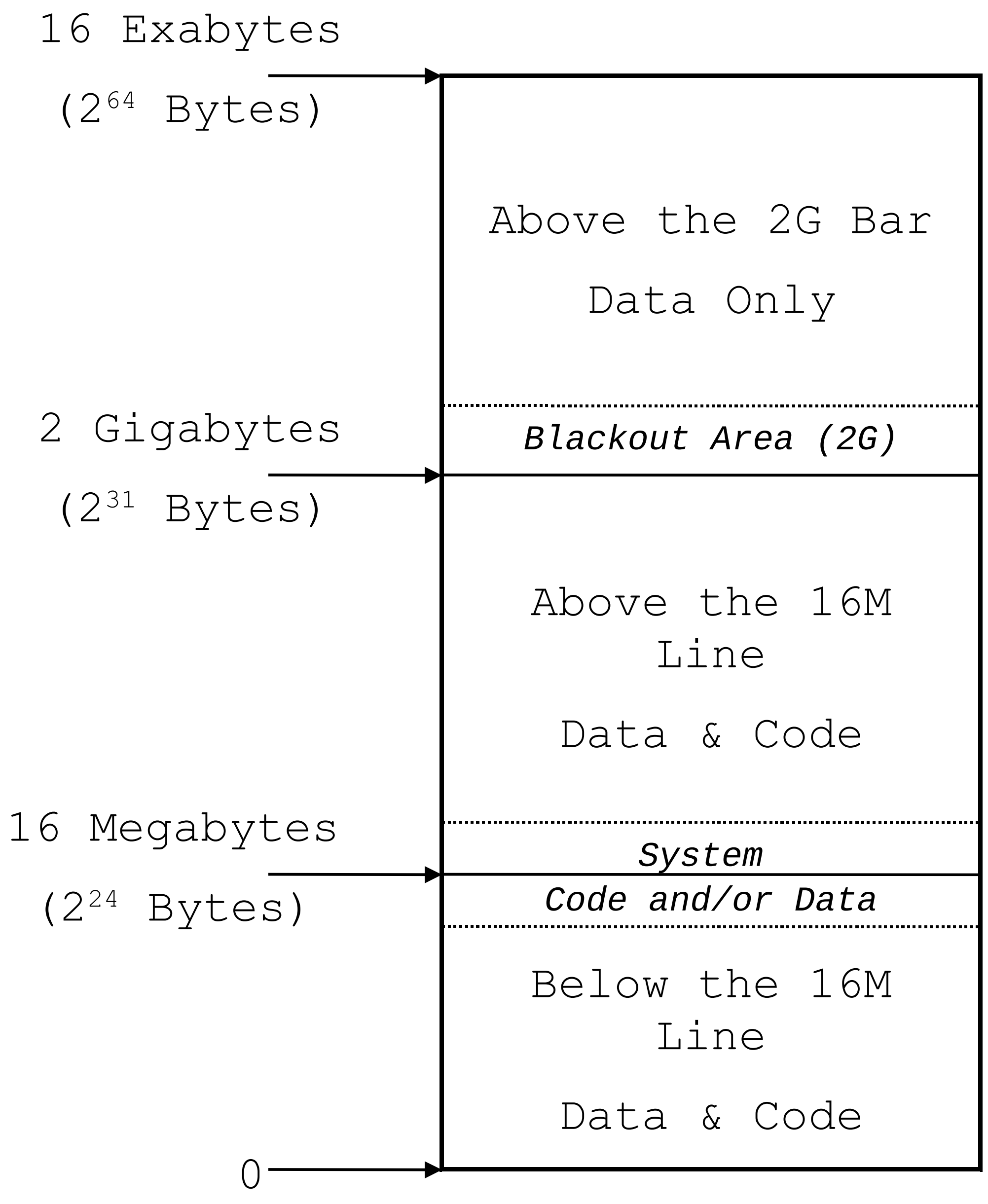
March 3, 2010

Historical perspective

- Classical S360/370
  - (1964) Only 24-bit addressing supported
- 370/XA
  - (1981) Bimodal addressing: 24- or 31-bit addressing
- ESA/390
  - (1988) Dataspaces and Access Registers
- z/Architecture
  - (2001) Trimodal addressing: 24-, 31-, or 64-bit addressing

Memory layout

- Below the 16M Line
  - 24 Bit Addressing
  - Code: RMODE=24
  - Data: GETMAIN LOC=BELOW
- Above the 16M Line
  - 31 Bit Address
  - Code: RMODE=ANY (*may be below the line*)
  - Data: GETMAIN LOC=ANY (*may be below the line*)
- Above the 2G Bar
  - 64 Bit Address
  - Code: None (*as yet*)
  - Data: IARV64



Terminology: all machine generations

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

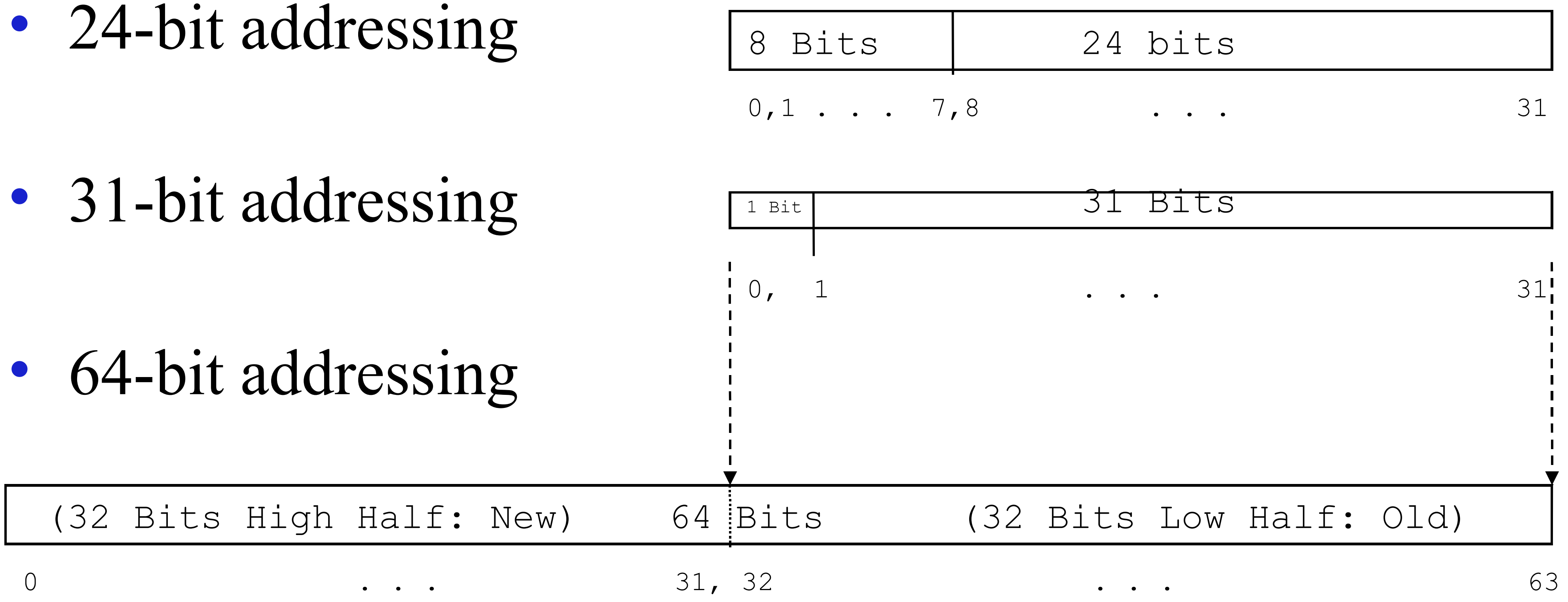
- **Notation:** *64-bit based* **[***32-bit based***]**
  - 64-bit based (Doubleword)
  - 32-bit based (Fullword)
- **Positions:**
  - “*High Order*” refers to the low numbered bits
  - “*Low Order*” refers to the high numbered bits

Registers

- 16 General Purpose Registers
  - In all generations of processors
- Pre-z/Architecture
  - 32 bits in size (the “*Traditional environment*”)
- z/Architecture
  - 64 bits in size: (sometimes called “*Grande*”)
  - Low Order Word: same as with Pre-z/Architecture processors
    - “*Traditional*” Instructions that use 24- and 31-bit addressing
  - High Order Word: z/Architecture Extension
    - New z/Architecture Instructions
      - Modal and modeless
      - 64 bit addressing
    - Ignored by “*traditional*” modeless instructions

Address formats

- 24-bit addressing
- 31-bit addressing
- 64-bit addressing



24-bit addressing in System/360

- General Purpose Registers
  - 32 bits
    - High 8 bits (0 to 7) for user (flags, etc)
      - BALR instruction (ILC, CC, Program Mask)
      - DCB fields
    - Low 24 bits (8 to 31) for addressing
- Special addressing-mode instructions
  - None (none needed!)

24- and 31-bit addressing in 370/XA

- General Purpose Registers
  - 32 Bits
    - High Order bit indicates addressing mode
      - 0 for 24-bit addressing, 1 for 31-bit addressing
    - Bits 1 to 7 depend on addressing mode
      - Part of the address (31-bit addressing)
      - Flags, etc. (24-bit addressing)
- Special addressing-mode instructions
  - BSM (*Branch and Set Mode*)
  - BASSM (*Branch and Save and Set Mode*)
    - More about these two, later

64-, 31-, and 24-bit addressing in z/Architecture

- General Purpose Registers: 64 Bits (Doubleword)
  - 32 Bit Extension (High Order Word)
    - Part of Address, Data, or Unused
  - 32 Bit Original (Low Order Word)
    - Retains Addressing Methodology for 24 -and 31-bit processing
    - Low order Bit 63[31]
      - Considered part of address -or- 64-bit addressing-mode indicator!
- Special addressing-mode instructions
  - Traditional:
    - BASSM, BSM
  - New with z/Architecture:
    - SAM24, SAM31, SAM64 and TAM

PSW description: 2 architecture modes

- <u>ESA/390 mode</u>
- Doubleword (64 Bits)
  - Bit 12 is always 1
  - Bit 31 is always 0
  - Instruction Address:
    - Bits 33 to 63
  - Addressing Mode (A)
    - Bit 32 determines addressing mode
      - 0 in 24-bit mode
      - 1 in 31-bit mode

- <u>z/Architecture mode</u>
- Quadword (128 Bits)
  - Bit 12 is always 0
  - Bit 31 contains the EA mode
  - Instruction Address
    - Bits 64 to 127
  - Addressing mode
    - Bit 31 (EA): Extended Addressing Mode
    - Bit 32 (BA): Basic Addressing Mode

<u>Addressing Modes</u>

	EA(0)	EA(1)
BA(0)	24	*Invalid*
BA(1)	31	64

PSW formats: 2 architecture modes

- ESA/390: Doubleword (64 bits)

0	R	000	T I E	Key	**1**	M W P	AS	CC	Mask	0000	0000

A	Instruction Address

- z/Architecture: Quadword (128 bits)

0	R	000	T I E	Key	**0**	M W P	AS	CC	Mask	0000	000	**E**

B	Zero-Filled (Bits 33 to 63)

Instruction Address (Bits 0 to 31)

Instruction Address (Bits 32 to 63)

Architecture-mode-dependent instructions

- Processed differently based on Architecture Mode:
  - Same code may behave differently in z/Architecture mode vs. non- z/Architecture (ESA/390) mode
- Small (rare) number of cases
  - Examples:
    - BAKR and PR
      - Saves/Restores 64 bit registers
    - ESTA
      - PSW functions
    - BASSM & BSM
      - We will talk more about these two …
- Differences are minimal
- They do what you would expect

Modeless instructions

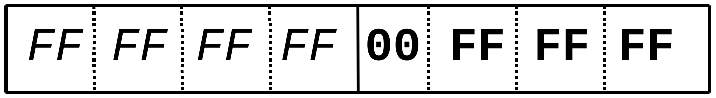
- Independent of architecture mode *and* addressing mode
  - Function is identical
    - Generally non-storage access type instructions
      - Register-register type instructions
      - Size of register access implied by instruction name
- General Purpose Registers
  - Pre-z/Architecture instructions
    - Operate only on low order word (bit 32[0] to bit 63[31])
    - High order word (bits 0 to 31 of 64) ignored
    - Examples: L, LR, A, AR, M, MR, SRDA, …
  - z/Architecture instructions
    - Operate either on 32-bit or all 64-bit registers
    - Examples: LGR (64-64), AGFR (64-32), RLL (32), RLLG (64), …

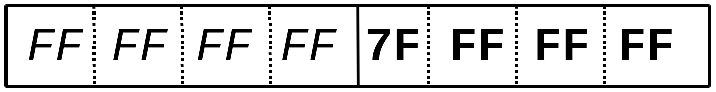
"Regular" modal instructions (1)

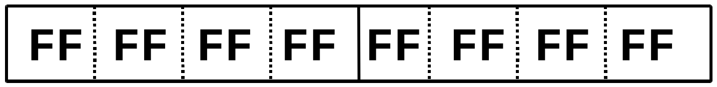
- Addresses function differently based on addressing mode
  - Base and Displacement are no different
  - May be hybrid with modeless
- Very predictable
  - No hidden surprises
- Generally the most commonly used instructions
  - Examples:
    - MVC – both storage operands depend on addressing mode
    - Loads and Stores (hybrids)
      - Arg_1 (register) size is based on instruction name (e.g. L vs LG)
      - Arg_2 (base and displacement) depends on addressing mode

"Regular" modal instructions (2)

- Load Address Types
  - LA, LAE, LAY: $R_1,D_2(X_2,B_2)$
  - Modal Processing – 64-bit register
    - 24-bit (high word is unchanged)
    - Low order word
      - Clears eight bits: 32[0] to 39[7]
      - Retains all other bits
    - 31-bit (high word is unchanged)
      - Low order word
      - Clears one bit: 32[0]
      - Retains all other bits
    - 64-bit
      - Sets full 64 bit register







- Lengths in registers usually interpreted based on addressing modes
  - Examples: CLCLE, MVCLE, TRE, … etc.
  - Some do not, such as MVCL and CLCL

"Irregular" modal instructions

- Function differently based on addressing mode
  - Reference or address storage
    - Base and Displacement
    - Register storage reference
- Have possible *"unpredictable"* side effects or processing
  - Visit: *Principles of Operations*
  - Read the fine print!
- Not many cases; usually, extensive or complex instructions
  - Examples:
    - TRT sets GPR 1 differently depending on addressing mode
    - ESTA (see code 1)
      - If possible use code 4

Mode-switching instructions

- Branch & Set Mode
  - BSM R_1,R_2
    - RR-Format:

0B	R_1	R_2

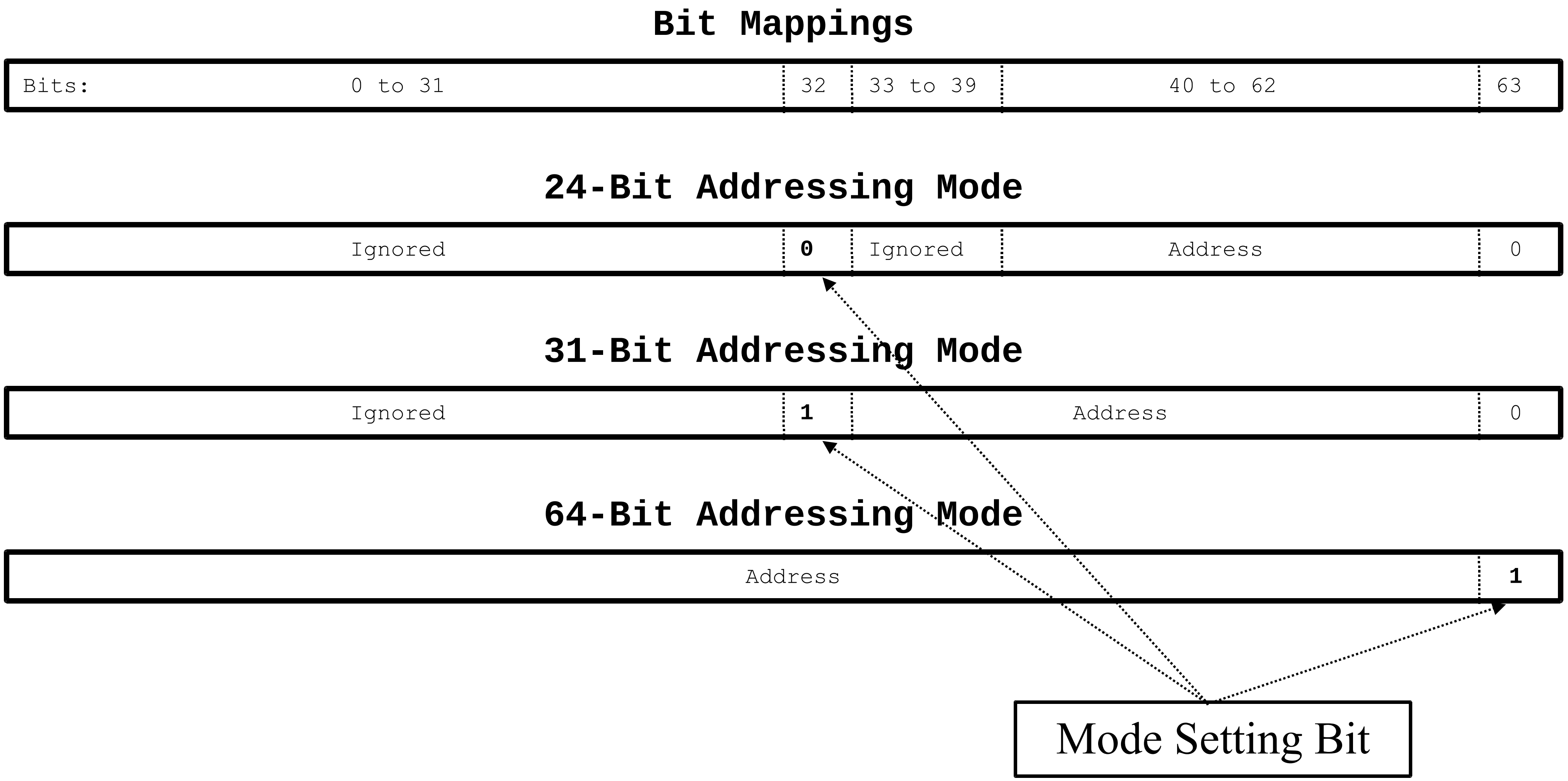
  - R_1 = 32- or 64-bit register
    - $R_1 \neq 0$
      - Receives PSW addressing mode bit only; rest unchanged
    - $R_1 = 0$
      - No Address mode bit saved
  - R_2 = 32- or 64-bit register
    - $R_2 \neq 0$
      - Branch-to address
      - New addressing mode
    - $R_2 = 0$
      - No Branching
      - No Address Mode bit saved

- Branch & Save & Set Mode
  - BASSM R_1,R_2
    - RR-Format:

0C	R_1	R_2

  - R_1 = 32- or 64-bit register
    - R_1 any register number
      - Receives current PSW address (of next instruction) and PSW addressing mode
  - R_2 = 32- or 64-bit register
    - $R_2 \neq 0$
      - Branch-to address
      - New addressing mode
    - $R_2 = 0$
      - No Branching
      - No Address Mode bit saved

Register addressing-mode formats for BSM and BASSM



Bit Mappings

Bits: 0 to 31	32	33 to 39	40 to 62	63

24-Bit Addressing Mode

Ignored	**0**	Ignored	Address	0

31-Bit Addressing Mode

Ignored	**1**	Address	0

64-Bit Addressing Mode

Address	**1**

BASSM & BSM addressing

- Pre z/Architecture
  - Always an even branch address
  - 32-bit Register Only
  - High order bit mode setting
    - 0 for 24-bit addressing
    - 1 for 31-bit addressing
  - Low order bit
    - Part of instruction address
      - 0: Valid
      - 1: Odd instruction address; Invalid!

Note !!!

- z/Architecture 64-bit
  - ***Always*** an even branch address
  - 64-bit register
    - Low-order bit (63)
    - Not used as part of branch address
      - 0 for 24-, 31-bit addressing
      - 1 for 64-bit addressing
    - High/Low order bit 32[0]
      - For 24-, 31-bit addressing determines mode as in pre-z/Architecture
      - For 64-bit mode, part of instruction address

Mode-switching examples: calls within a single assembly

```
* Goto 24-bit mode from any mode
    LA       R15,GOTO24
    BASSM    R14,R15
```

```
*  Entry into 24-bit Mode
GOTO24   DC     0H        Below the line
         . . .
         BSM    0,R14     Return to Caller
```

```
* Goto 31-bit mode from any mode (1)
    L        R15,GOTO31@
    BASSM    R14,R15
* Goto 31-bit mode from any mode (2)
    LARL     R15,GOTO31
    OILH     R15,X'8000'
    BASSM    R14,R15
```

```
*  Entry into 31-bit Mode
GOTO31   DC     0H        Below/Above line
         . . .
         BSM    0,R14     Return to Caller
```

```
* Goto 64-bit mode from any mode
    XGR      R15,R15  For 24/31->64
    LARL     R15,GOTO64
    OILL     R15,X'0001'
    BASSM    R14,R15
```

```
*  Entry into 64-bit Mode
GOTO64   DC     0H        Below/Above line
         . . .
         BSM    0,R14     Return to Caller
```

```
GOTO31@  DC   A(GOTO31+X'80000000')
```

z/Architecture addressing-mode instructions

- SAM*xx:*
- Set Addressing Mode
  - E-Type (2-Byte) format with Opcode X'010x'
    - No registers set/modified; no register preload required
    - Old mode not retained
  - Types
    - SAM24: Switch to 24 bit addressing mode $\rightarrow$ `01 0C`
    - SAM31: Switch to 31 bit addressing mode $\rightarrow$ `01 0D`
    - SAM64: Switch to 64 bit addressing mode $\rightarrow$ `01 0E`
- TAM
  - Test Addressing Mode `01 0B`
  - E-Type (2-byte) format:
  - Sets condition code based on current addressing mode
    - CC=0 – 24-bit addressing (Branch on Zero)
    - CC=1 – 31-bit addressing (Branch on Mixed)
    - CC=2 – Unused
    - CC=3 – 64-bit addressing (Branch on One)
  - No registers set or changed
  - Addressing mode is not switched

```
*  Examples:
         SAM24 , To AMODE(24)
         . . .
         SAM31 , To AMODE(31)
         . . .
         SAM64 , To AMODE(64)
         . . .
         TAM   , Test AMODE
         JZ    IN24
         JO    IN64
*  Running in AMODE(31)
IN31     DS    0H
         . . .
*  Running in AMODE(31)
IN24     DS    0H
         . . .
*  Running in AMODE(64)
IN64     DS    0H
         . . .
```

BALR vs. BASR

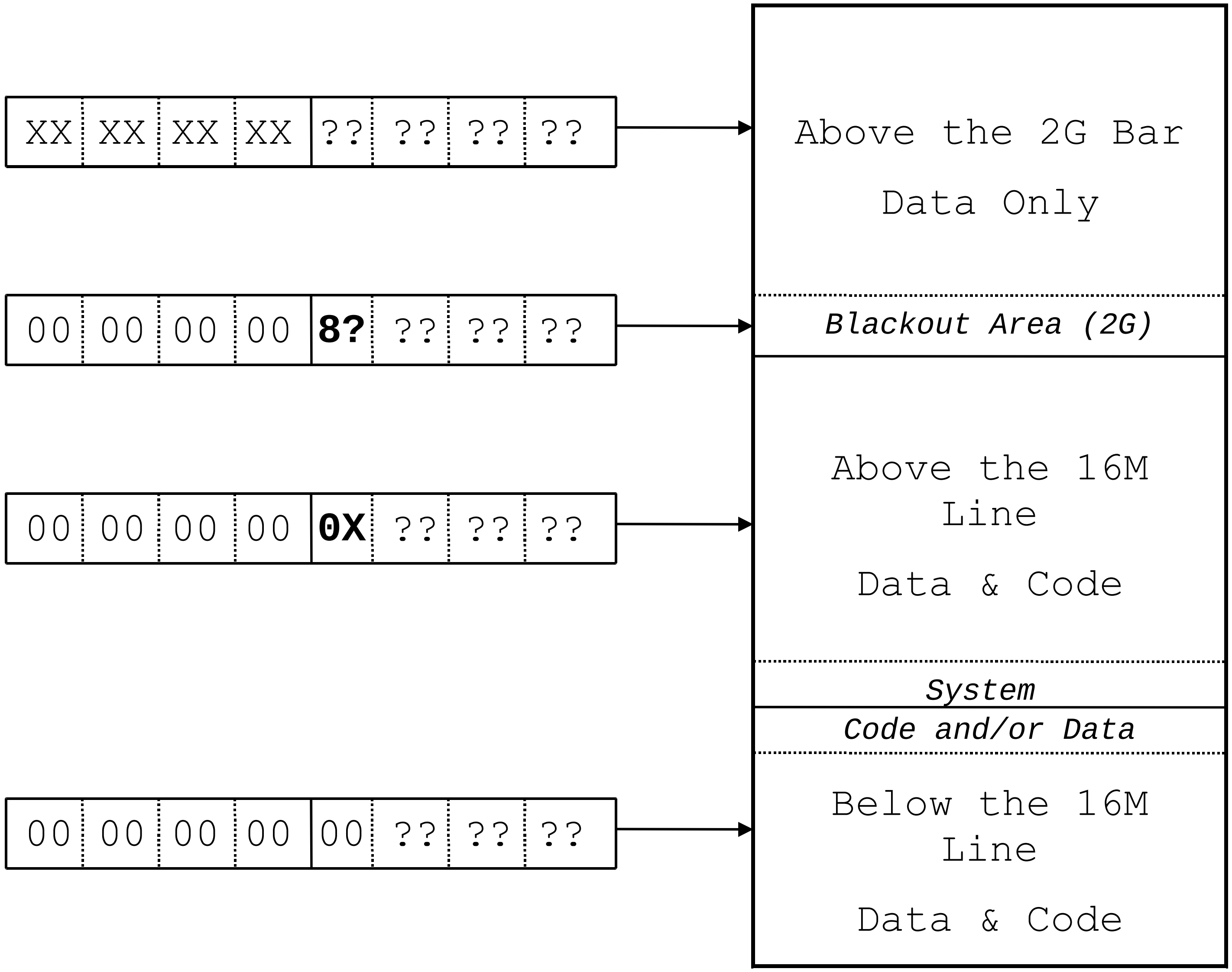
- BALR R_1,R_2
  - Since S/360
  - High order word
    - 24-, 31-bit mode: ignored
    - 64-bit mode: part of address
  - Processing Modes (R_1)
    - 24-bit addressing contains ILC, CC, Program Mask, 24 bit address
    - 31- and 64-bit addressing, identical to BASR
  - Deprecated now
    - Use BASR for branch and link
    - Use IPM instruction for CC and Program Mask

- BASR R_1,R_2
  - Since XA/370 (bimodal)
  - High order word
    - 24-, 31-bit mode: ignored
    - 64-bit mode: part of address
  - Processing Modes (R_1)
    - 24- and 31-bit addressing: contains mode bit and address
    - 64-bit addressing: contains only the address, no mode bit(s)
  - Preferred method of branch and link (or save) without mode switching
    - More consistent

Memory referencing by 64-bit addresses

Legend

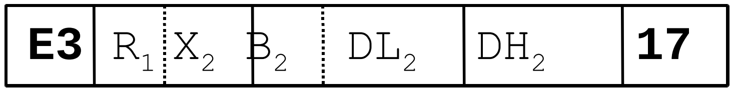
XX	At least one bit is not zero among all XXs
??	Any bit combination
8?	High-order bit is one, all others any combination
0X	High-order bit is zero, seven other bits any combination with at least one bit set to one.
00	All bits zero

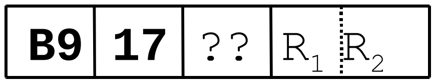


Special considerations

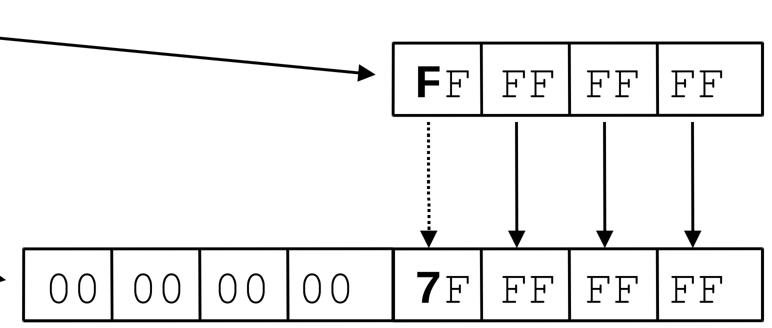
- <u>16M Line</u>
  - System data and code straddles the line
  - Application code or data will not cross over
    - GETMAIN either totally below or above the line
    - Program object segment (class) will either be totally below or above the line
- <u>2G Bar</u>
  - Blackout zone for MVS is first 2G above the 2G bar
    - <u>**Technically, a valid addressable region!!**</u>
    - Applies to a 64-bit address
      - High word is all zeroes
      - Low Order word has address with bit (32[0]) set to 1
    - IARV64 will not allocate storage in blackout zone

The useful LLGT and LLGTR instructions

- Load Logical "*Grande*" Thirty One Bits
  - **`LLGT   R1,D2(X2,B2)`**
    - RXY Format: 

  - **`LLGTR  R1,R2`**
    - RRE Format: 

- 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, High Bit 32[0] set to 0



Example: Call and Return

```
         TITLE     'BAD CASE'
MYPGM    CSECT     ,
MYPGM    AMODE     MY_AMODE
MYPGM    RMODE     MY_RMODE
AMODE    EQU       ... bit setting ...
         . . .

         L         R15,YOURPGM@
         BASR      R14,R15
         . . .
YOURPGM@ DC        A(YOURPGM+AMODE)
         END       ,

YOURPGM  CSECT     ,
YOURPGM  AMODE     YOUR_AMODE
YOURPGM  RMODE     YOUR_RMODE
         . . .
         BSM       0,R14
         END       ,
```

```
         TITLE     'GOOD CASE'
MYPGM    CSECT     ,
MYPGM    AMODE     MY_AMODE
MYPGM    RMODE     MY_RMODE
AMODE    EQU       ... bit setting ...
         . . .
         XGR       R15,R15  <--Important!
         L         R15,YOURPGM@
         BASSM     R14,R15
         . . .
YOURPGM@ DC        A(YOURPGM+AMODE)
         END       ,

YOURPGM  CSECT     ,
YOURPGM  AMODE     YOUR_AMODE
YOURPGM  RMODE     YOUR_RMODE
         . . .
         BSM       0,R14
         END       ,
```

BAD CASE:	Worked OK for *MY_AMODE=YOUR_AMODE* for 24 and 31 but fails for 64
	Fails for *MY_AMODE≠YOUR_AMODE*
GOOD CASE:	Works for all *MY_AMODE* and *YOUR_AMODE* values

Notes: Call and Return

- Make sure CALL and RETURN types match
  - BASSM with BSM
  - BASR with BR
- Be sure alternatives are valid
  - LINK vs. LOAD and CALL
    - LINK: switches address mode as required
    - LOAD and CALL: does not switch addressing mode
- Watch out for addressing-mode bits as part of address
  - May have to clear address mode in register
    - Especially *“odd”* address and AMODE 64

Example: The KILLER bit!

```
         TITLE    'BAD PROGRAM'
MYPGM    CSECT    ,
MYPGM    AMODE    31
MYPGM    RMODE    ANY
         STM      R14,R12,12(R13)
         BASR     R11,0
         USING    *,R11
         . . .
         SAM64    ,
*****************************************
*  The Next Instruction Abends!!!       *
*  (because BASR executed in AMODE(31)) *
*****************************************

         MVC      DATA1,DATA2
         . . .
DATA1    DS       CL10
DATA2    DC       CL10'TESTING'
         . . .
         END      ,
```

```
         TITLE    'GOOD PROGRAM'
MYPGM    CSECT    ,
MYPGM    AMODE    31
MYPGM    RMODE    ANY
         STM      R14,R12,12(R13)
         BASR     R11,0
         USING    *,R11
         . . .
         SAM64    ,
*****************************************
*  The Next Instruction Saves the Day   *
*  (Removes Blackout Area Addressing)   *
*****************************************
         LLGTR    R11,R11
         MVC      DATA1,DATA2
         . . .
DATA1    DS       CL10
DATA2    DC       CL10'TESTING'
         . . .
         END      ,
```

Linkage considerations

- New Instructions
  - Save and Load 64 bit registers
    - STMH, LMH, STG, LG, STMG, LMG, LMD
  - More on this follows …
- Save Areas
  - “Traditional” 72 byte save area
    - 32-bit registers
    - Standard Linkage
  - New Save Area Layout
    - 64-bit registers
    - Standard linkage
    - Transitional

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

EB	R_1	R_3	B_2	DL_2…	DH_2	**26**
EB	R_1	R_3	B_2	DL_2…	DH_2	**96**

- Analogous to STM and LM
  - Acts on range of registers
    - No Store or Load instructions for high half of a single register
    - Use multiple-type instruction with $R_1 = R_3$

Store/Load entire 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!
- 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 **<u>no</u>** "*Store Multiple Disjoint*"

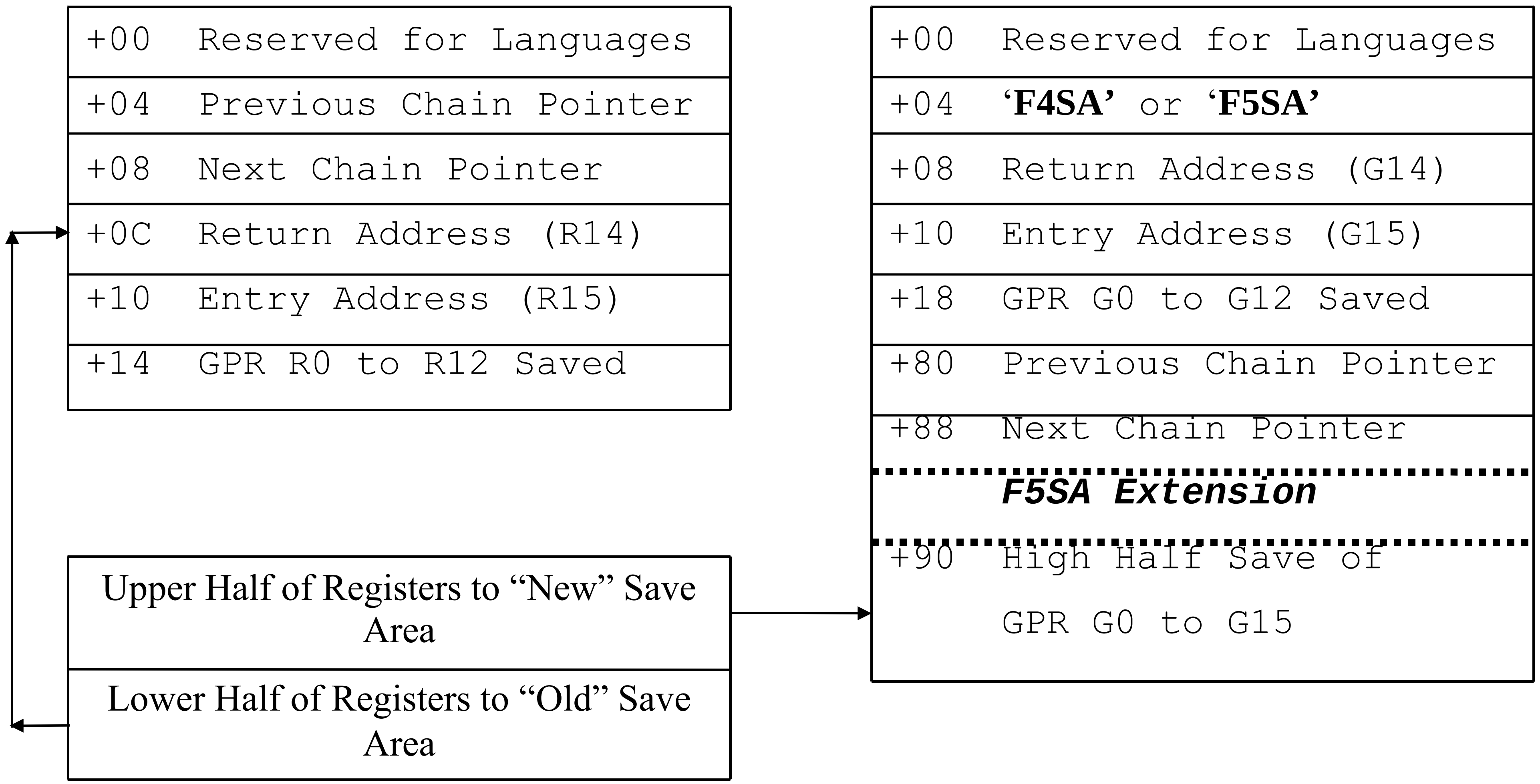
```
*          Example of LMD
           STMH R2,R5,HIREGS
           STM  R2,R5,LOWREGS
           . . .
           LMD  R2,R5,HIREGS,LOWREGS
           . . .
HIREGS     DS 4F    Save High Half
LOWREGS    DS 4F    Save Low Half
```

Save areas: old and new

- IHASAVER macro in SYS1.MACLIB
- Types of save areas:
  - “Traditional” 72 byte save area
    - 32-bit register save
    - Standard Linkage
  - Format 4
    - 64-bit register save
    - Standard linkage
      - Eye catcher “F4SA” at offset X'04'
      - Relocates “previous” and “next” chains to offset 128 (dec) and 136 (dec)
  - Format 5
    - 64-bit register save like format 4
    - 32-bit high register save area appended
      - Used for transition from 32 to 64 bit register processing
    - Standard linkage (like format 4)
      - Eye catcher “F5SA” at offset X'04'
      - Relocates “previous” and “next” chains to offset 128 (dec) and 136 (dec)

Save area layouts

- Traditional
- z/Architecture 64-Bit



+00	Reserved for Languages
+04	Previous Chain Pointer
+08	Next Chain Pointer
+0C	Return Address (R14)
+10	Entry Address (R15)
+14	GPR R0 to R12 Saved

Upper Half of Registers to “New” Save Area
Lower Half of Registers to “Old” Save Area

+00	Reserved for Languages
+04	**‘F4SA’** or **‘F5SA’**
+08	Return Address (G14)
+10	Entry Address (G15)
+18	GPR G0 to G12 Saved
+80	Previous Chain Pointer
+88	Next Chain Pointer
	F5SA Extension
+90	High Half Save of GPR G0 to G15

Sample program

- Two routines
  - **`Sample1`**
    - Runs either in 24- or 31-bit mode
    - Performs I/O to read and write records
    - Driver for a 64-bit-mode processing routine
  - **`RTN64`**
    - Entered in caller’s mode (24 or 31)
      - Uses F5SA save area to save registers
    - Processes records in 64-bit mode
      - Allocates and deletes storage above the 2G bar
        - IARV64
      - Accesses storage above the 2G bar
        - Uses 64 bit registers