

## SEE 3223 Microprocessors

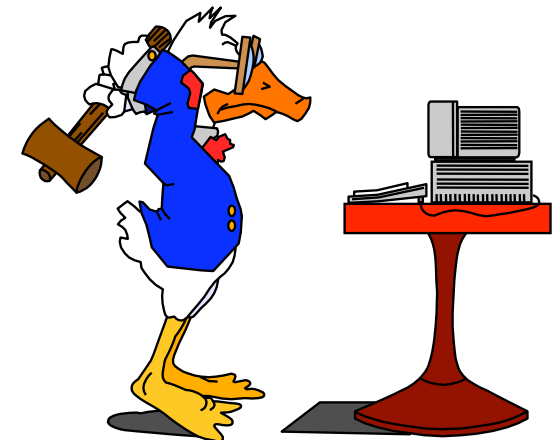
# 5: Data Processing Instructions

Muhammad Mun'im Ahmad Zabidi (munim@utm.my)



# Data Processing Instructions

- Arithmetic operations:
  - ADD, SUB, MULU, MULS, EXT, NEG.
- Logical
  - AND, OR, EOR, NOT
- Shift
  - ASL, ASR, LSL, LSR, ROL, ROR, ROXL, ROXR.
- Bit operations:
  - BCLR, BSET, BCHG, BTST



# Encoding Integers

## Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

## Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

↑  
Sign  
Bit

	Decimal	Hex	Binary
x	15740	3D 7C	00111101 01111100
y	-15740	C2 84	11000010 10000100

### ■ Sign Bit

- For 2's complement, most significant bit indicates sign
  - 0 for nonnegative
  - 1 for negative

# Numeric Ranges

Unsigned Values ( $w$ = number of bits)		
Notation	Value	Binary Pattern
$U_{\min}$	0	000...0
$U_{\max}$	$2^w-1$	111...1

Two's Complement Values ( $w$ = number of bits)		
Notation	Value	Binary Pattern
$T_{\min}$	$-2^{w-1}$	100...0
$T_{\max}$	$2^{w-1}-1$	011...1

Values for  
 $w = 8$

	Decimal	Hex	Binary
$U_{\max}$	255	FF	1111 1111
$T_{\max}$	+127	7F	0111 1111
$T_{\min}$	-128	80	1000 0000
-1	-1	FF	1111 1111
0	0	00	0000 0000

Values for  
 $w = 16$

	Decimal	Hex	Binary
$U_{\max}$	65535	FF FF	1111 1111 1111 1111
$T_{\max}$	+32767	7F FF	0111 1111 1111 1111
$T_{\min}$	-32768	80 00	1000 0000 0000 0000
-1	-1	FF FF	1111 1111 1111 1111
0	0	00 00	0000 0000 0000 0000

# Values for Different Word Sizes

- Observations

$$|T_{\text{Min}}| = T_{\text{Max}} + 1$$

- Asymmetric range

$$U_{\text{Max}} = 2 * T_{\text{Max}} + 1$$

	W			
	8	16	32	64
<b>UMax</b>	255	65,535	4,294,967,295	18,446,744,073,709,551,615
<b>TMax</b>	127	32,767	2,147,483,647	9,223,372,036,854,775,807
<b>TMin</b>	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808

# ADD

- Adds the contents of the source location to the contents of a destination location and stores the result in the destination location.
  - Source: All addressing modes; however, either source or destination must be a data register.
  - Destination: All except immediate, address register direct and program relative.

Effect on CCR Flags	
C	Set if a carry is generated, cleared otherwise
V	Set if an overflow occurred, cleared otherwise
Z	Set if the result is zero. Cleared otherwise.
N	Set if the result is negative. Cleared otherwise.
X	Set the same as the carry bit.

# SUB

- Subtraction: SUB src, dest  
 – [dest] ← [dest] - [src]

SUB.B D0,D1

D1

1	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

D0

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

D1'

1	1	1	0	0	0	1	0
---	---	---	---	---	---	---	---

### Effect on CCR Flags

C	Set if a carry is generated, cleared otherwise
V	Set if an overflow occurred, cleared otherwise
Z	Set if the result is zero. Cleared otherwise.
N	Set if the result is negative. Cleared otherwise.
X	Set the same as the carry bit.

# Effect of Arithmetic Operations on CCR

- Addition:

$$C = \begin{cases} 1, & \text{if carry out from MSB} \\ 0, & \text{otherwise} \end{cases}$$

$$V = \begin{cases} 1, & \text{if operands are of same sign and} \\ & \text{their sum is of the opposite sign} \\ 0, & \text{otherwise} \end{cases}$$

$$V = \overline{a_{n-1}} \cdot \overline{b_{n-1}} \cdot s_{n-1} + a_{n-1} \cdot b_{n-1} \cdot \overline{s_{n-1}}$$

where  $a_{n-1}$ ,  $b_{n-1}$ ,  $s_{n-1}$  are the MSBs of source destination and result, respectively



# Effect of Arithmetic Operations on CCR

- Subtraction:

$$C = \begin{cases} 1, & \text{if NO carry out from MSB} \\ 0, & \text{otherwise} \end{cases}$$

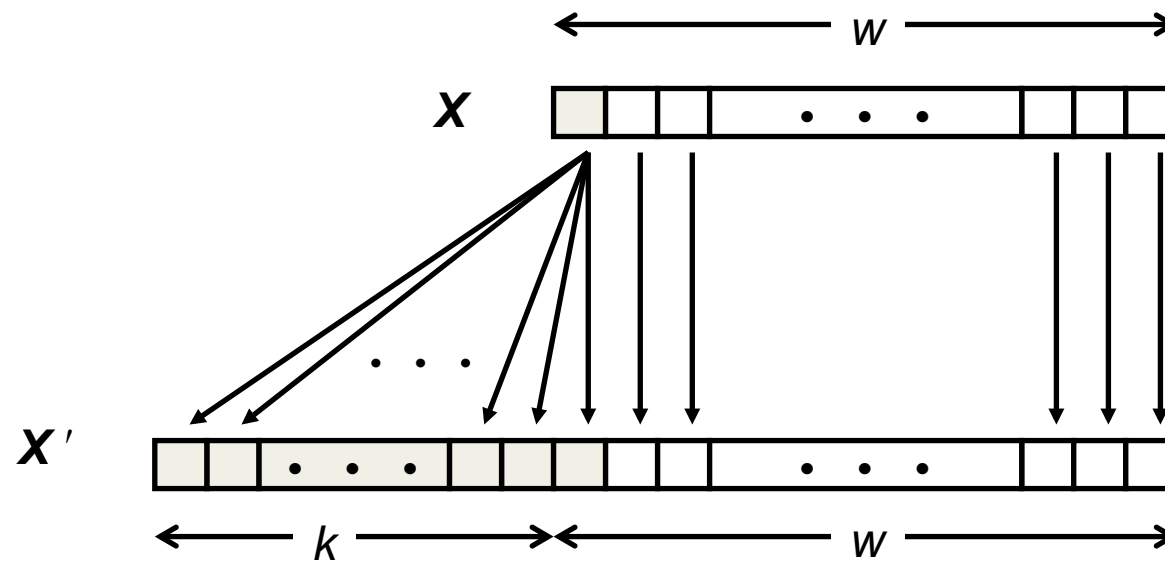
$$V = \begin{cases} 1, & \text{if operands are of opposite sign and} \\ & \text{the result is of same sign as the source} \\ 0, & \text{otherwise} \end{cases}$$

$$V = (a_{n-1} \oplus b_{n-1}) \cdot \overline{(d_{n-1} \oplus a_{n-1})}$$

where  $a_{n-1}$ ,  $b_{n-1}$ ,  $d_{n-1}$  are the MSBs of  
source destination and result, respectively

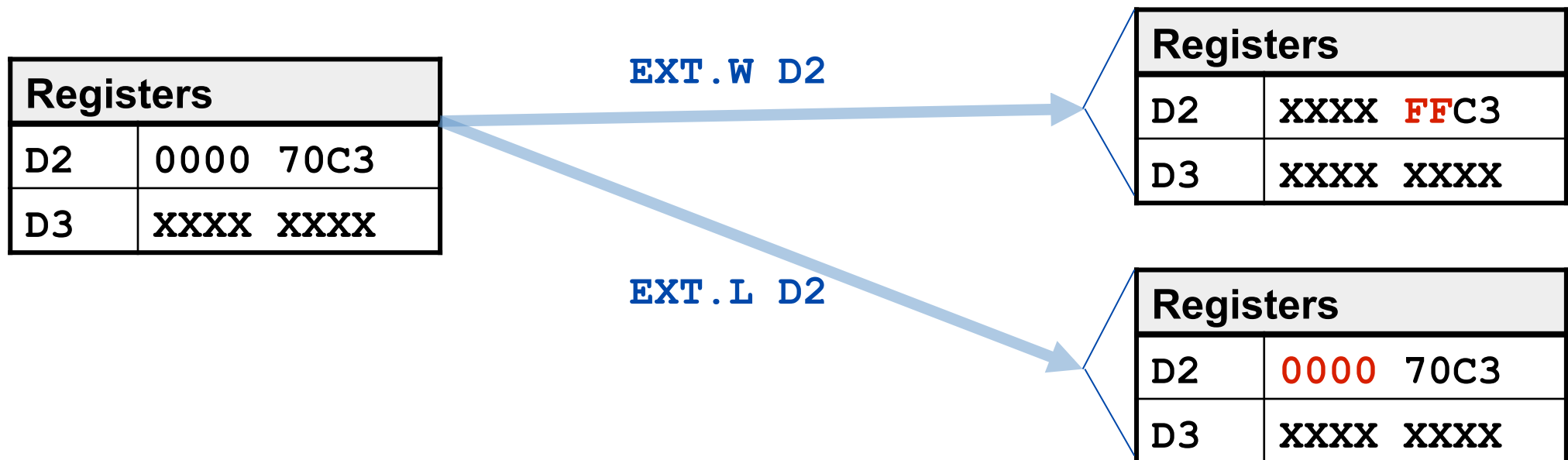
# Sign Extension

- Task:
  - Given  $w$ -bit signed integer  $X$
  - Convert it to  $w+k$ -bit integer with same value
- Rule:
  - Make  $k$  copies of sign bit:



# Sign EXTend Instruction

- Extends the sign bit of the low-order byte or word of a data register:
  - EXT.W sign extends the low order byte to 16 bits;
  - EXT.L sign extends the low order word to 32 bits.



# Example: Adding Different-Sized Numbers

- \* Calculate  $A = B + C - D$
- \* Where B is a longword, C is a word and D is a byte.
- \*

	<b>ORG</b>	<b>\$1000</b>	<b>Program origin</b>
<b>START</b>	<b>MOVE.L</b>	<b>B,D0</b>	Get B to T (running sum)
	<b>MOVE.W</b>	<b>C,D1</b>	Get C
	<b>EXT.L</b>	<b>D1</b>	Convert C to longword
	<b>ADD.L</b>	<b>D1,D0</b>	Add C to T
	<b>MOVE.B</b>	<b>D,D2</b>	Get D
	<b>EXT.W</b>	<b>D2</b>	Convert D to word
	<b>EXT.L</b>	<b>D2</b>	Then convert D to long
	<b>SUB.L</b>	<b>D2,D0</b>	Subtract D from T
	<b>MOVE.L</b>	<b>D0,A</b>	Store T in A
	<b>STOP</b>	<b>#\$2700</b>	Halt processor at end of program
	<b>ORG</b>	<b>\$1000</b>	
<b>A</b>	<b>DS.L</b>	<b>1</b>	
<b>B</b>	<b>DC.L</b>	<b>40</b>	
<b>C</b>	<b>DC.W</b>	<b>-16</b>	
<b>D</b>	<b>DC.B</b>	<b>3</b>	
	<b>END</b>	<b>START</b>	

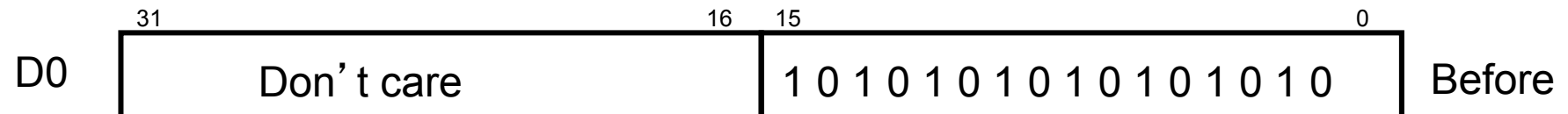
# MULU, MULS Instructions

- MULU performs unsigned multiplication and MULS performs signed multiplication on two's complement numbers.
  - Multiplication is a 16-bit operation that multiplies the low-order 16-bit word in Dn (destination data register) by the 16-bit word at the effective address. The 32-bit results is stored in the full destination data register Dn.
  - Source: All modes except address register direct.
  - Destination: Data register.

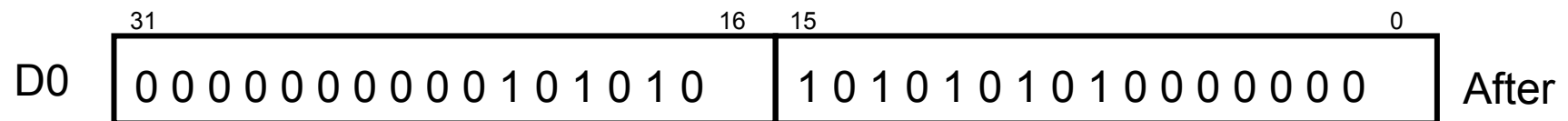
Effect on CCR	
C	Always cleared.
V	Always cleared.
Z	Set if the result is zero. Cleared otherwise.
N	Set if the result is negative. Cleared otherwise.
X	Not affected.

# MULU, MULS Example

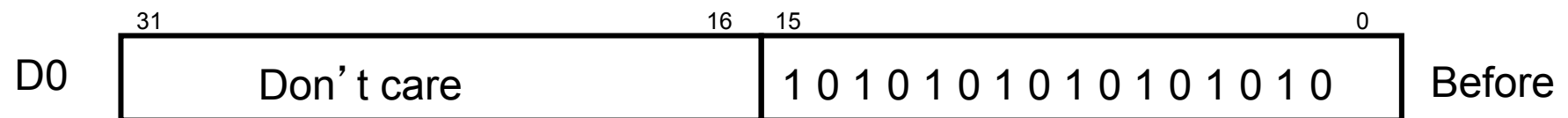
Multiply unsigned:



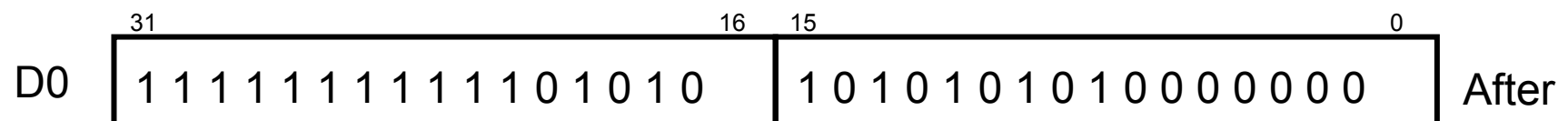
`MULU #%01000000,D0` or `MULU #$40,D0`



Multiply signed:



`MULS #%01000000,D0` or `MULS #$40,D0`



← Sign extension →

# DIVU, DIVS Instructions

- DIVU performs unsigned division, and DIVS performs signed division on two's complement numbers.
  - The 32-bit long word in the data register is divided by the 16-bit word at the effective address.
  - The 16-bit quotient is stored in the lower-order word of the register and the remainder is stored in the upper-order word.
  - Source: All modes except address register direct.
  - Destination: Data register.

- Overflow may occur if quotient does not fit in 16 bits

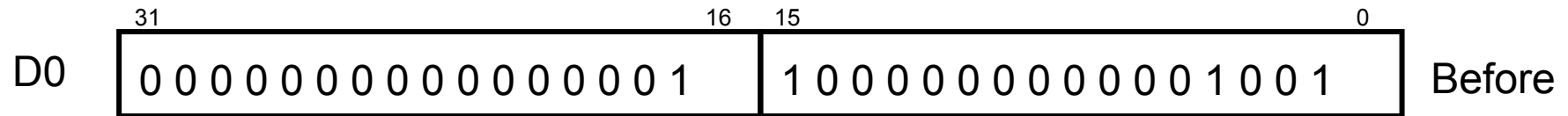
- Trap may occur if divide by zero is attempted

Effect on CCR	
C	Always cleared.
V	Set if division overflow occurred, cleared otherwise. Undefined if divide by zero occurs.
Z	Set if the quotient is zero. Cleared otherwise. Undefined if overflow or divide by zero occurs
N	Set if the quotient is negative. Cleared otherwise. Undefined if overflow or divide by zero occurs
X	Not affected.

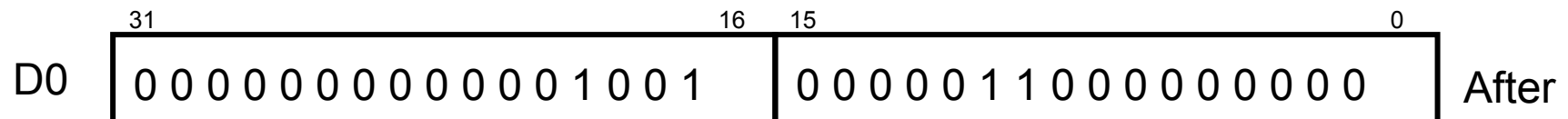
# DIVU, DIVS Example

Divide unsigned:

$D0 = 98309$  divide by 64



`DIVU #01000000,D0` or `DIVU #$40,D0`

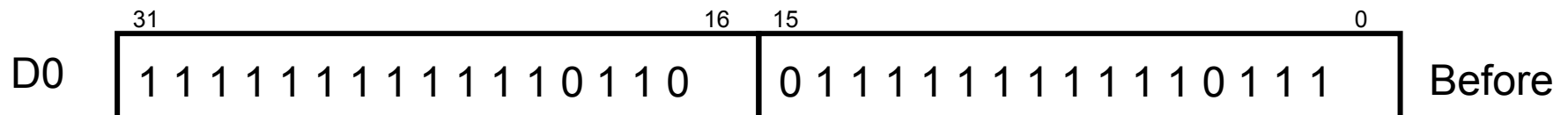


Remainder = 5

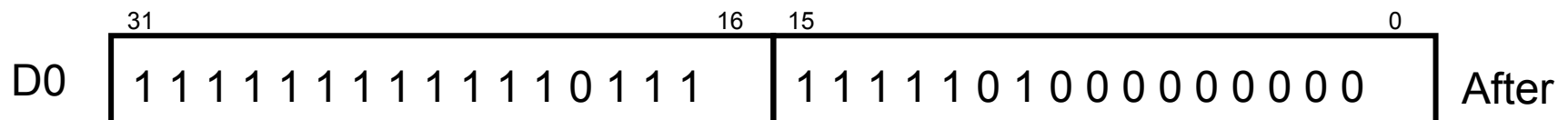
Quotient = 1536

Divide signed:

$D0 = -98309$  divide by 64



`DIVS #01000000,D0` or `DIVS #$40,D0`



Remainder = -5

Quotient = -1536



# EXT before DIVS

- EXT is often used with DIVS, because DIVS requires a 32-bit dividend.
- EXT.L D1 sign-extends the low-order word in D1 to 32 bits by copying D1(15) to bits D1(16:31).

```
MOVE.W  (A0),D0    ; load 16-bit dividend from memory
EXT.L   D0         ; extend to 32 bits
DIVS    #42,D0     ; perform the division
MOVE.W  D0,2(A0)  ; store the quotient
```

# Negate Instruction

- Negation: negative value or 2' s complement

NEG.B D0

D1

1	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

D1'

0	0	0	1	1	0	1	1
---	---	---	---	---	---	---	---

# SWAP instruction

- SWAP instruction exchanges the top word with the lower word of a data register
- Useful to get the remainder of a division operation.

```
*  
* This snippet checks the an unsigned number  
* is divisible by 3  
*  
    CLR.L  D0          ; clear top half of D0  
    MOVE.W (A0),D0    ; load 16-bit dividend from memory  
    DIVU   #3,D0      ; perform the division  
    SWAP  D0          ; bring the remainder to low word  
    CMP.W #0,D0      ; if 0, number was divisible by 3  
    BEQ   YES        ; go to someplace ...
```

# EXG instruction

- EXG instruction exchanges a register with another register



# Logic Instructions

- Logic instructions include:
  - AND            Bit-wise AND
  - OR             Bit-wise OR
  - EOR            Bit-wise Exclusive OR
  - NOT            1's Complement of bits of destination

A	B	A AND B	A OR B	A EOR B	NOT A	NOT B
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Effect on CCR	
C	Always cleared.
V	Always cleared.
Z	Set if the result is zero. Cleared otherwise.
N	Set if the most significant bit of the result is set; cleared otherwise.
X	Not affected.

# Masking

- **Mask:** bit pattern to isolate and manipulate some particular bits
- To set bits, use OR with 1s in the positions of bits to be set.
  - Example: Set bits 1, 6, and 7 in D0:
- To clear bits, use AND with 0s in the positions of bits to be cleared.
  - Example: Clear bits 2 and 5 in D0:

AND.B #%11011011,D0

Original	1	1	1	0	0	1	1	1
Mask	1	1	0	1	1	0	1	1
Result	1	1	0	0	0	0	1	1

OR.B #%11000010,D0

Original	0	1	1	0	1	1	0	0
Mask	1	1	0	0	0	0	1	0
Result	1	1	1	0	1	1	1	0

# Inverting Bits

- To invert only some bits, use EOR with 1s in the positions of bits to be inverted.
  - Example: Invert bits 0 and 1 in D0:
- To invert all bits, use NOT.
  - Examples: Inverts all bits in D0

EOR.B #%11011011,D0

NOT.B D0

Original

1	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

Mask

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Result

1	1	1	0	0	1	1	0
---	---	---	---	---	---	---	---

Original

1	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

Result

0	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---

# Practical Application of Logical Ops

- Example: A subroutine GetChar inputs an ASCII-encoded character from the keyboard, returns in D1 a 7-bit code plus a parity bit in the MSB. The following sequence will get the character and change the received character to lower-case

```
BSR      GetChar
AND.B    #%01111111,D1      Clear MSB
OR.B     #%00100000,D1     Convert char to lowercase
```

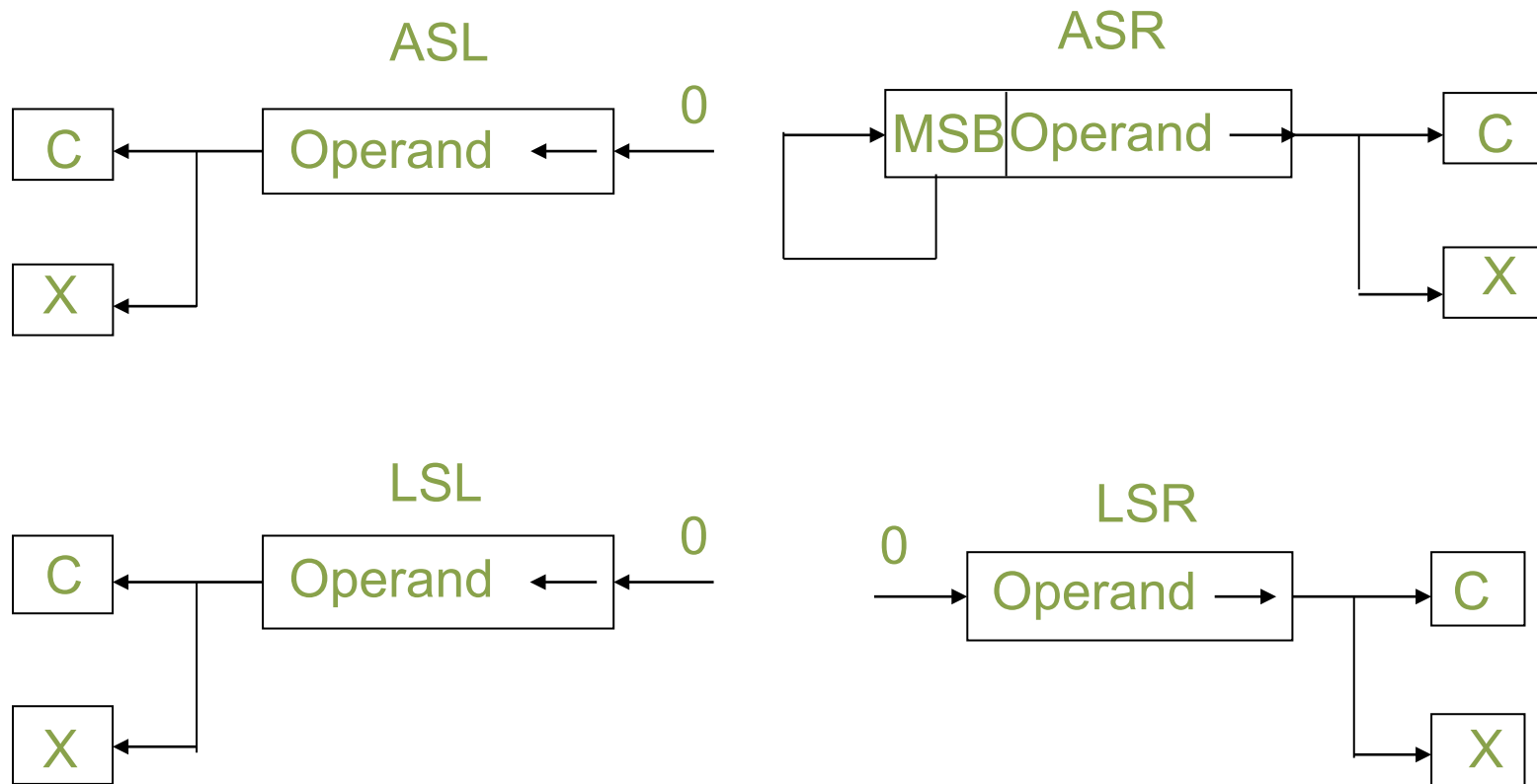
- Note:
  - 'A' = 01000001
  - 'a' = 01100001
- How about lower-to-upper?
- What if the data is not in A-Z range?



# ASCII Table

	000	001	010	011	100	101	110	111
0000	NUL	DLC	SP	0	@	P	.	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

# Shift Operations



# ASR (Arithmetic Shift Left) Instruction

- The arithmetic shift left operation ASL moves the bits of the operand
  - Immediate: in the range 1 to 8
  - Register: by the value in a source data register modulo 64
- As each bit is shifted left, it is stored in the Carry flag of the CCR.
- The vacant spot on the right is filled with a zero.



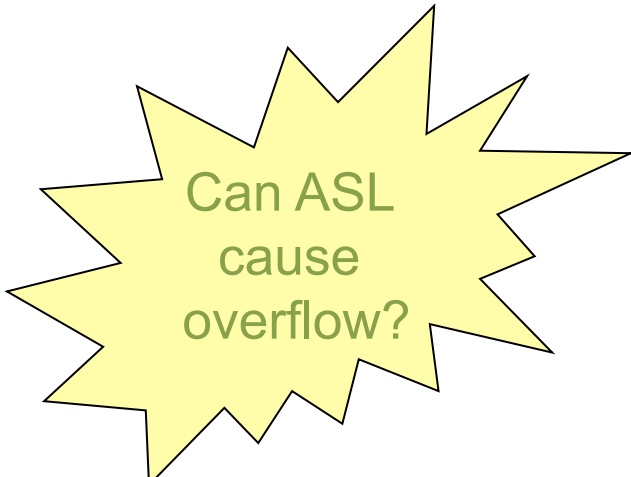
# Why is ASL useful?

- How to multiply D0 by 4 ?
- ASL is the fastest way to perform “multiply by 2’ s power”

6<sub>10</sub>

- What does ASL #n, dest do?
  - [dest] ← [dest] x 2<sup>n</sup>

ASL.B #2,D0

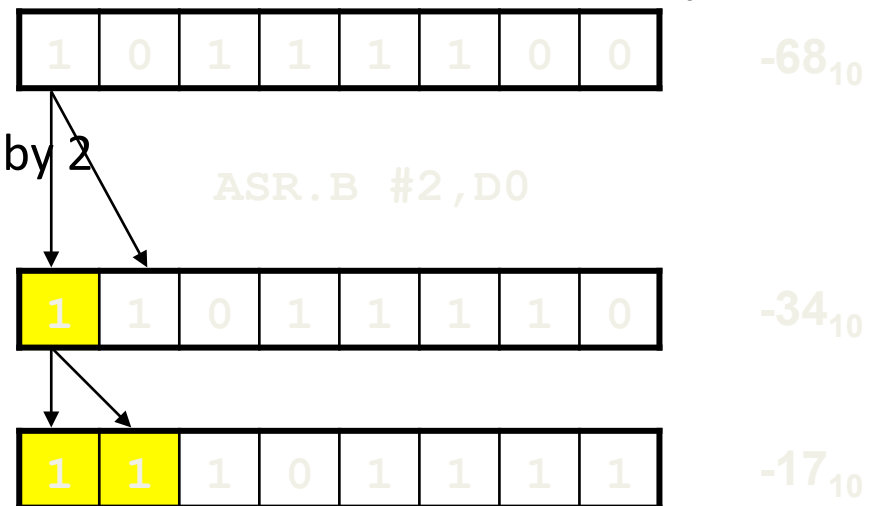
24<sub>10</sub>


Can ASL  
cause  
overflow?

# ASR (Arithmetic Shift Right) Instruction

- Same as ASL, but
  - bits shifted to RIGHT
  - MSB is duplicated back into MSB (Why?)
- ASR.B #1,D0 is equivalent to dividing D0 by 2

- How to divide D0 by 4 ?



Can ASR  
cause  
overflow?

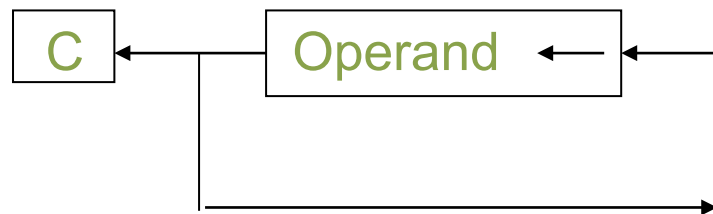
# Logical Shift Instructions

- Two variants:
  - LSL (Logical Shift Left)
  - LSR (Logical Shift Right)
- Shifts the operand the specified number of positions left/right;
  - Immediate: in the range 1 to 8
  - Register: by the value in a source data register modulo 64
- Vacated bit positions are always zero-filled

Effect on CCR	
C	Set according to the last bit shifted out of the operand. Cleared for a shift count of zero.
V	Always cleared.
Z	Set if the result is zero. Cleared otherwise.
N	Set if the result is negative; cleared otherwise.
X	Set according to the last bit shifted out of the operand. Unaffected for a shift count of zero.

# Rotate Operations

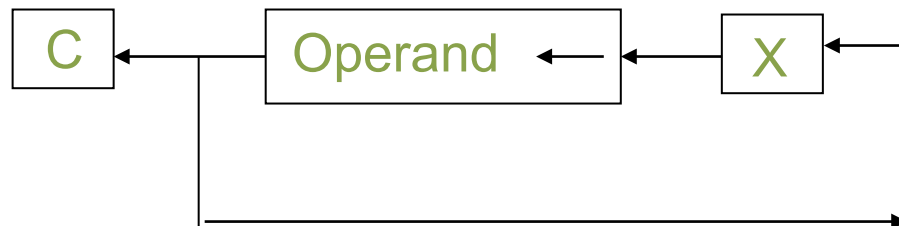
ROL



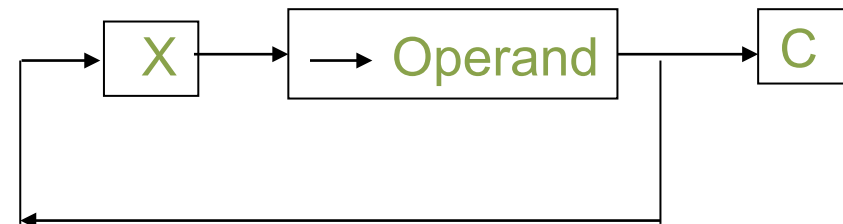
ROR



ROXL



ROXR



# Rotate Instructions

- Two variants:
  - ROL(Rotate Left)
  - ROR(Rotate Right)
- Shifts or rotate the operand the specified number of positions left/right. Bits that move off one end are put back on the opposite end after setting or clearing the C-bit.
- Rotates the operand the specified number of positions left/right;
  - Immediate: in the range 1 to 8
  - Register: by the value in a source data register modulo 64

C	Set according to the last bit rotated out of the operand. Cleared when the rotate count is zero.
V	Always cleared.
Z	Set if the result is zero. Cleared otherwise.
N	Set if the most significant bit is set; cleared otherwise.
X	Not affected



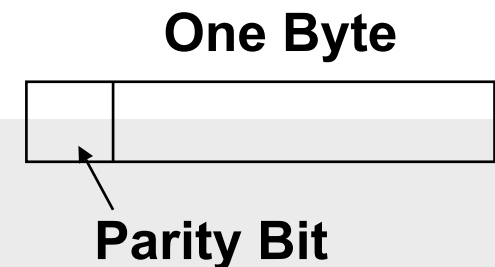
# Rotate with eXtend Instructions

- Two variants:
  - ROXL(Rotate Left with eXtend)
  - ROXR(Rotate Right with eXtend)
- Rotates the operand the specified number of positions left/right including the X-bit.
- Rotates the operand the specified number of positions left/right;
  - Immediate: in the range 1 to 8
  - Register: by the value in a source data register modulo 64

Effect on CCR	
C	Set according to the last bit rotated out of the operand. When the rotate count is zero, set to the value of the extend bit.
V	Always cleared.
Z	Set if the result is zero. Cleared otherwise.
N	Set if the most significant bit is set; cleared otherwise.
X	Set according to the last bit rotated out of the operand. Unaffected for a rotate count of zero.

# Example: Setting Parity Bit of A Byte

- The following program sets the parity bit (msb) of a byte depending on the number of 1's in the byte using rotate.
- If number of ones is odd parity bit is set( = 1), otherwise = 0



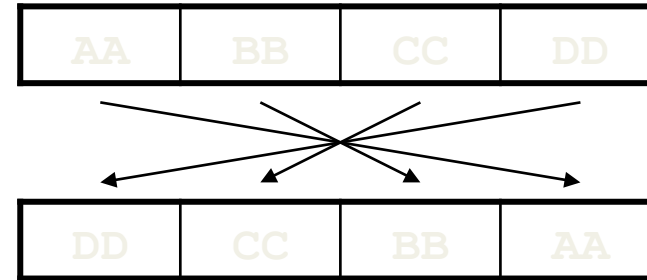
```

* D0 contains the byte of data whose parity bit is to be set
* D1 contains a temporary working copy of D0
* D2 used to count that 7 bits have been tested

        ORG      $400                Program origin
        MOVE     #7,D2                Set the counter to 7
        ANDI.B   #%01111111,D0       Clear the parity bit to start
        MOVE     D0,D1                Make a working copy of D0
Next     ROR.B   #1,D1                Rotate 1 bit right
        BCC     Zero                 If the bit is 1 then
        EOR.B   #%10000000,D0        toggle the parity bit
Zero     SUB.B   #1,D2                Decrement the counter
        BNE     Next                 Check another bit
        STOP    #$2700
        END     $400
  
```

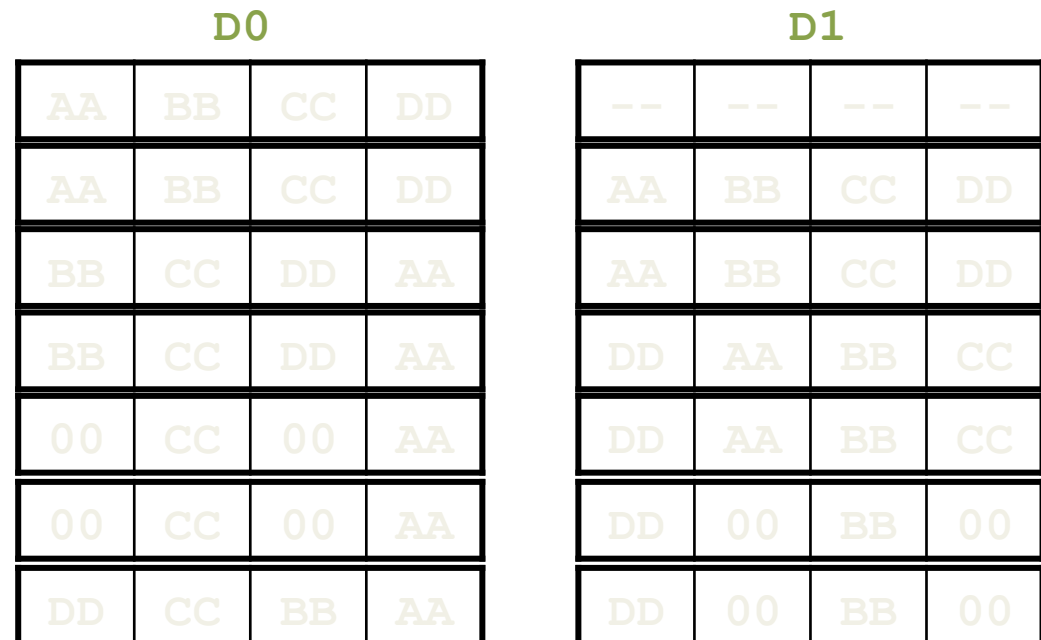
# Big-Endian to Little-Endian Conversion

Graphical Problem Statement



```

MOVE.L  D0,D1
ROL.L   #8,D0
ROR.L   #8,D1
AND.L   #$00FF00FF,D0
AND.L   #$FF00FF00,D1
OR.L    D1,D0
  
```



# Bit Manipulation Instructions

- The 68000 has four instructions that manipulate **single bits**:
  - BSET        Sets the specified bit to 1.
  - BCLR        Sets the specified bit to 0.
  - BCHG        Toggles (inverts) the specified bit.
  - BTST        Tests the value of a bit. If zero, the Z-flag is set.
- The bit number for this operation can be specified in one of two ways:
  - Immediate: e.g. #0, #1, #2, ...
  - Register: The specified data register contains the position of the bit to be manipulated.
- Operations are performed on:
  - 1 bit of a byte if the operand is in memory or
  - 1 bit of a long word if the operand is a data register. Thus:
  - No instruction extension is required.

# Bit Operations

- Some bit operations (not all) can be implemented using logical operations with masks.

```
BCLR    #4,D0          ; same as AND.B #%11101111,D0
BSET    #4,D0          ; same as OR.B  #%00010000,D0
BCHG    #4,D0          ; same as EOR.B #%00010000,D0
BTST    #4,D0          ; almost the same as AND.B #%00010000,D0
                          ; but D0 is not destroyed
                          ; If bit 4 is zero, then the Z-bit of
                          ; CCR is set to 1.
```