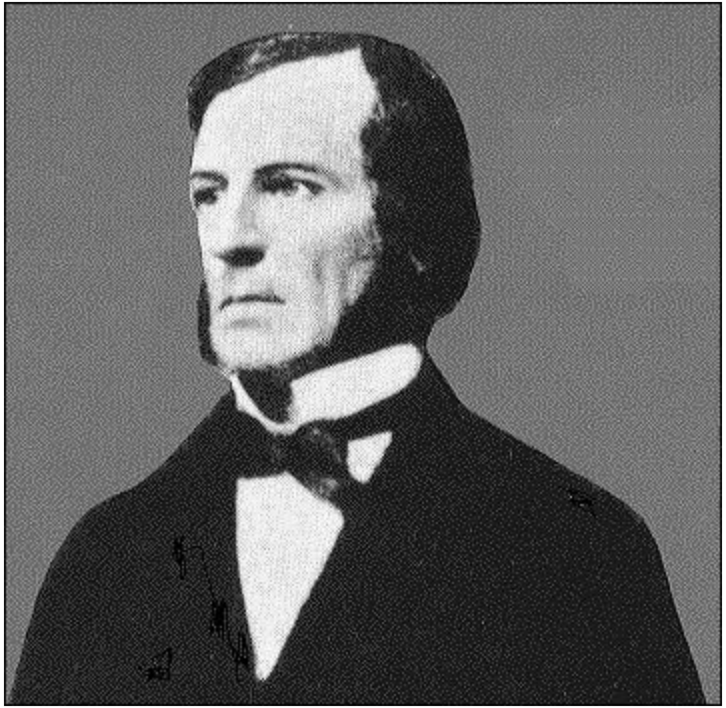
Logical operations

CSC 236

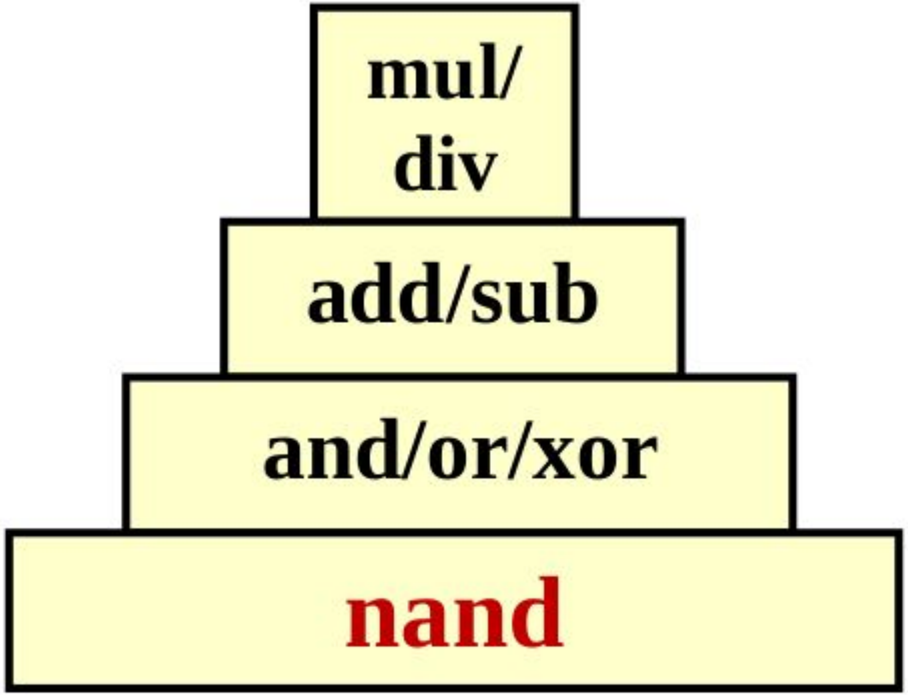
Boolean Algebra

- Developed by George Boole circa 1854
- Basis for computer hardware



Boolean Algebra

- Developed by George Boole circa 1854
- Basis for computer hardware



Boolean truth tables

A	Not A
0	1
1	0

A	B	A and B	A or B	A xor B	$\overline{\text{A and B}}$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	1
1	1	1	1	0	0

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** and **1**
  - **A** and **0**
  - **A** and **A**

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** and **1**
  - **A** and **0**
  - **A** and **A**

A and 1

```
0101      0011
1111      1111
----      ----
0101      0011
```

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** and **1** = **A**
  - **A** and **0**
  - **A** and **A**

A and 1

```
0101      0011
1111      1111
----      ----
0101      0011
```

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** and **1** = **A**
  - **A** and **0** = **0**
  - **A** and **A**

A and 0

```
0101      0011
0000      0000
----      ----
0000      0000
```

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** and **1** = **A**
  - **A** and **0** = **0**
  - **A** and **A** = **A**

A and A

```
0101      0011
0101      0011
----      ----
0101      0011
```

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** or **1**
  - **A** or **0**
  - **A** or **A**

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** or **1** = **1**
  - **A** or **0**
  - **A** or **A**

A or 1

```
0101      0011
1111      1111
----      ----
1111      1111
```

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** or **1** = **1**
  - **A** or **0** = **A**
  - **A** or **A**

A or 0

0101	0011
<u>0000</u>	<u>0000</u>
0101	0011

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** or **1** = **1**
  - **A** or **0** = **A**
  - **A** or **A** = **A**

A or A

```
0101      0011
0101      0011
----      ----
0101      0011
```

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** xor **1**
  - **A** xor **0**
  - **A** xor **A**

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** xor **1** = **not A**
  - **A** xor **0**
  - **A** xor **A**

A xor 1

$$\begin{array}{r} 0101 \\ \underline{1111} \\ 1010 \end{array} \qquad \begin{array}{r} 0011 \\ \underline{1111} \\ 1100 \end{array}$$

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** xor **1** = **not A**
  - **A** xor **0** = **A**
  - **A** xor **A**

A xor 0

```
0101      0011
0000      0000
0101      0011
```

Bit-wise boolean operations

- Let
  - **A** be an arbitrary 4-bit number (base 2)
  - **1** be 1111_2 — all bits are 1
  - **0** be 0000_2 — all bits are 0
- Consider
  - **A** xor **1** = **not A**
  - **A** xor **0** = **A**
  - **A** xor **A** = **0**

A xor A

```
0101      0011
0101      0011
----      ----
0000      0000
```

Bit-wise boolean operations

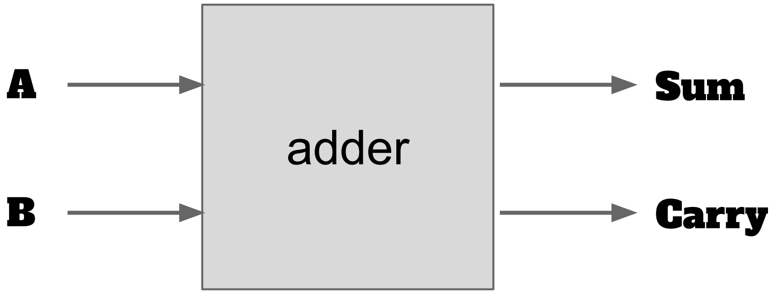
- And
  - **A** and **1** = **A**
  - **A** and **0** = **0**
  - **A** and **A** = **A**
- Or
  - **A** or **1** = **1**
  - **A** or **0** = **A**
  - **A** or **A** = **A**
- Xor
  - **A** xor **1** = **not A**
  - **A** xor **0** = **A**
  - **A** xor **A** = **0**

Bit-wise boolean operations

- And
  - **A** and **1** = **A**
  - **A** and **0** = **0**
  - **A** and **A** = **A**
- Or
  - **A** or **1** = **1**
  - **A** or **0** = **A**
  - **A** or **A** = **A**
- Xor
  - **A** xor **1** = **not A**
  - **A** xor **0** = **A**
  - **A** xor **A** = **0**

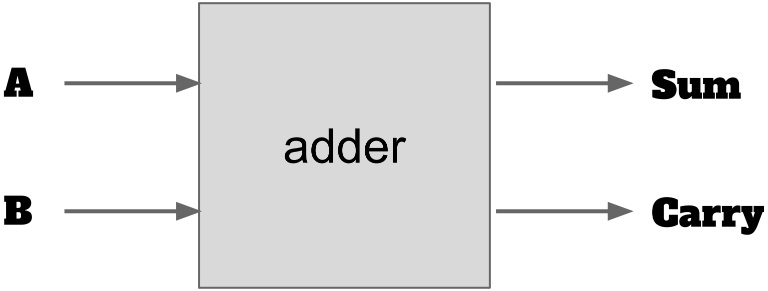
- (**A** xor **1**) xor **1** = **A**

Adder



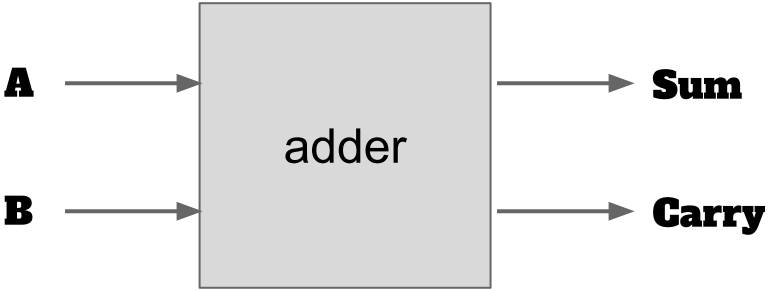
A	B	Sum	Carry
0	0		
0	1		
1	0		
1	1		

Adder



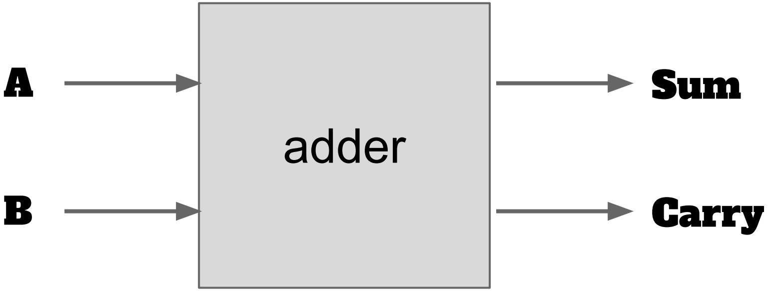
A	B	Sum	Carry
0	0	0	0
0	1		
1	0		
1	1		

Adder



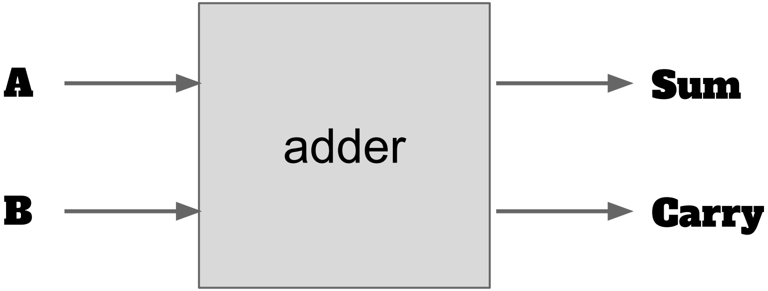
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1		

Adder



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

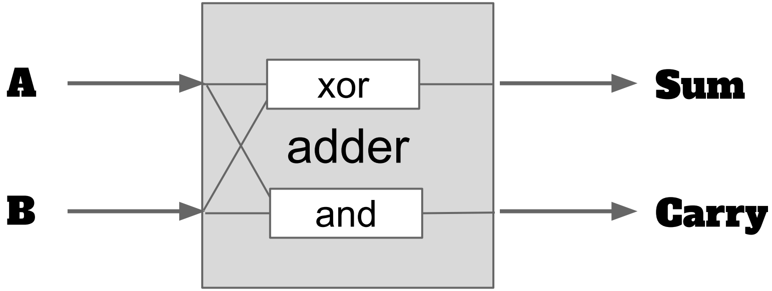
Adder



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

xor and

Adder



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1
		xor	and

8086 boolean instructions

```
not ax           ;invert bits
and ax,bx        ;ax = ax and bx
or  [var],cx     ;var = var or cx
xor al,0FFh      ;al = al xor FFh
```

Shifts

- Logical — unsigned data
- Arithmetic — signed data (two's complement)

- Direction
  - Left
  - Right
- Count
  - How many bits

Left shift

- Instructions
  - `sal dest, 1    ;shift arithmetic left`
  - `shl dest, 1    ;shift logical left`

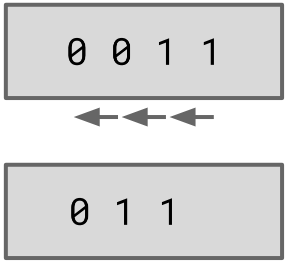
same instruction

0	0	1	1

Left shift

- Instructions
  - `sal dest, 1    ;shift arithmetic left`
  - `shl dest, 1    ;shift logical left`

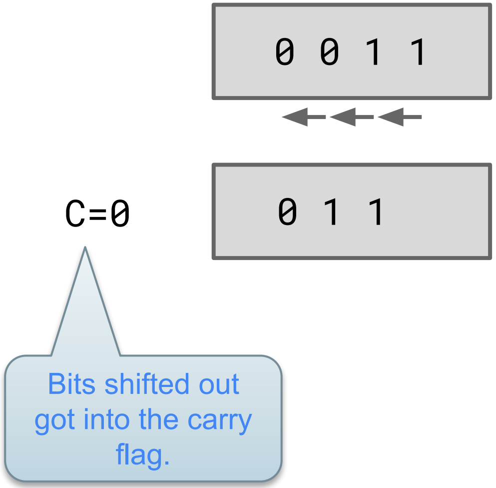
same instruction



Left shift

- Instructions
  - `sal dest, 1     ;shift arithmetic left`
  - `shl dest, 1     ;shift logical left`

same instruction



Left shift

- Instructions
  - `sal dest, 1    ;shift arithmetic left`
  - `shl dest, 1    ;shift logical left`

same instruction

0 0 1 1

C=0   0 1 1 ?

Left shift

- Instructions
  - `sal dest, 1    ;shift arithmetic left`
  - `shl dest, 1    ;shift logical left`

same instruction

```
      0 0 1 1

C=0   0 1 1 0
```

Left shift

- Instructions
  - `sal dest, 1    ;shift arithmetic left`
  - `shl dest, 1    ;shift logical left`

same instruction

0 0 1 1	= 3

C=0

0 1 1 0	= 6

Left shift

- Instructions
  - `sal dest, 1    ;shift arithmetic left`
  - `shl dest, 1    ;shift logical left`

same instruction

```
      | 0 0 1 1 | = 3

C=0   | 0 1 1 0 | = 6
```

Shift left by 1 ⇔ multiply by 2

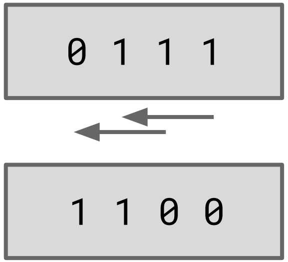
Shifting Multiple Bits

- Instructions
  - `sal dest, cl`
  - `shl dest, cl`

You can shift by more than one bit. But it has to be in cl.

- `sal dest, 2`
- `shl dest, 2`

Can't use an immediate value larger than 1.



C=1

Carry gets the last bit that's shifted out.

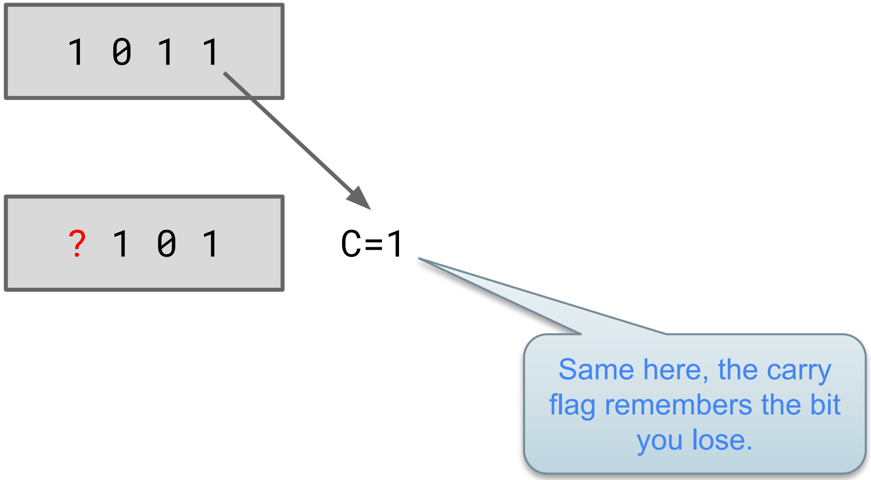
Right shift

- Instructions
  - `sar dest, 1    ;shift arithmetic right`
  - `shr dest, 1    ;shift logical right`

```
0 0 1 1
```

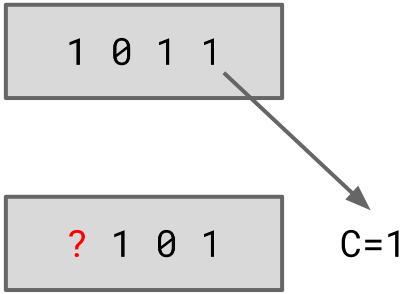
Right shift

- Instructions
  - `sar dest, 1    ;shift arithmetic left`
  - `shr dest, 1    ;shift logical left`



Right shift

- Instructions
  - `sar dest, 1    ;shift arithmetic left`
  - `shr dest, 1    ;shift logical left`

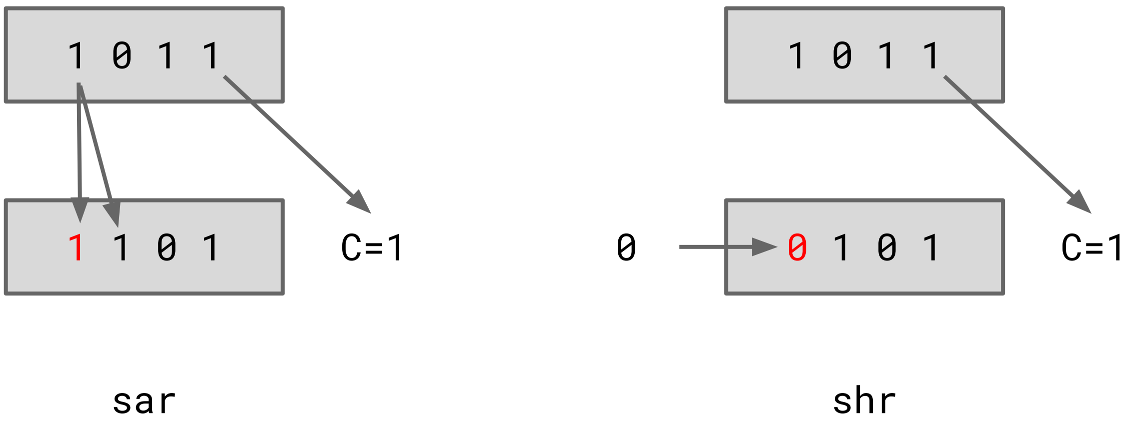


? is either 0 or 1

Arithmetic preserves sign
Logical does not

Right shift

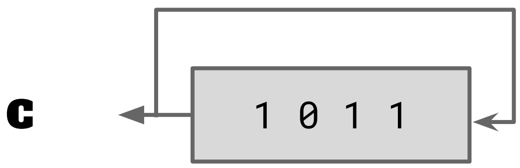
- Instructions
  - `sar dest, 1    ;shift arithmetic right`
  - `shr dest, 1    ;shift logical right`



Rotate

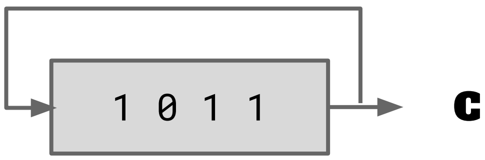
- Rotate left

```
rol dest,1
```



- Rotate right

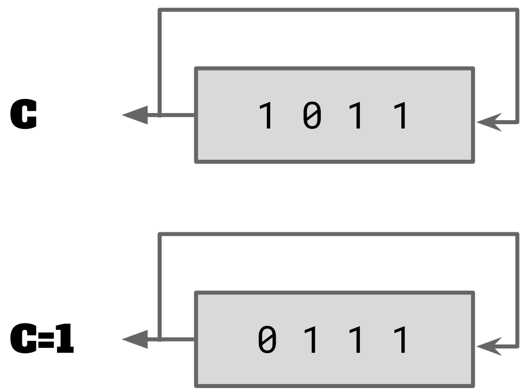
```
ror dest,1
```



Rotate

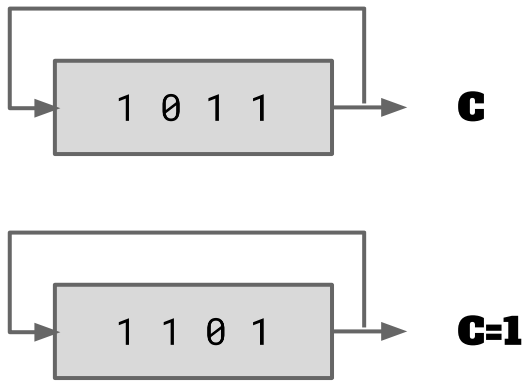
- Rotate left

```
rol dest,1
```



- Rotate right

```
ror dest,1
```



Rotate number of bits (eg, 4) ⇒ original value

Parity

- Simple error detection
- Number of 1s in number (field)
- Odd parity error detection
- Extra bit
  - 1 if even number of 1s in field
  - 0 if odd number of 1s in field
- Field + parity
  - Should give an odd number of 1 bits
  - If not ⇒ error

- Examples - correct
  - `0000:1`
  - `1101:0`
  - `0010:0`
  - `1111:1`
- Received
  - `00000 – error`
  - `00001 – no error detected`
  - `01010 – error`
  - `11100 – no error detected`

Parity

si → 4E 03 07 2A

Imagine this is the memory si is pointing to.

```
    mov     ax, 0   ; count = 0
    mov     bl, [si] ; bl = 0100 1110
    mov     cx, 8   ; 8 bits to process

tst:
    shl     bl, 1   ; shift 1 bit to CF
    jnc     nxt ; do not count 0 bits
    inc     ax  ; do count 1 bits

nxt:
   loop     tst ; loop
```

More efficient solution

- Uses
  - Add with carry
  - rol
- Add with carry
  - `add ax,7 ;ax=ax+7`
  - `adc ax,7 ;ax=ax+7+CF`

- Primary use of adc
  - To add high-magnitude numbers … that don’t fit in a byte or a word.

More efficient solution

- Uses
  - Add with carry
  - rol
- Add with carry
  - `add ax,7 ;ax=ax+7`
  - `adc ax,7 ;ax=ax+7+CF`

- Primary use of adc
  - To add high-magnitude numbers … that don't fit in a byte or a word.
- Example

$$\begin{array}{r} 4\ \ 7 \\ +\ \underline{3\ \ 8} \end{array}$$

More efficient solution

- Uses
  - Add with carry
  - rol
- Add with carry
  - `add ax,7 ;ax=ax+7`
  - `adc ax,7 ;ax=ax+7+CF`

- Primary use of adc
  - To add high-magnitude numbers … that don't fit in a byte or a word.

- Example

$$\begin{array}{r} 1 \\ 4\ 7 \\ +\ \underline{3\ 8} \\ 5 \end{array}$$

More efficient solution

- Uses
  - Add with carry
  - rol
- Add with carry
  - `add ax,7 ;ax=ax+7`
  - `adc ax,7 ;ax=ax+7+CF`

- Primary use of adc
  - To add high-magnitude numbers … that don’t fit in a byte or a word.

- Example

$$\begin{array}{rcc} & 1 & \\ & 4 & 7 \\ + & \underline{3} & \underline{8} \\ & 8 & 5 \end{array}$$

Parity

```
si ──► 4E 03 07 2A
```

```
    mov     ax, 0    ; count = 0
    mov     bl, [si] ; bl = 0100 1110
    mov     cx, 8    ; 8 bits to process

tst:
    shl     bl, 1    ; shift 1 bit to CF
    jnc     nxt ; do not count 0 bits
    inc     ax  ; do count 1 bits

nxt:
   loop     tst ; loop
```

```
    mov     ax, 0       ; count = 0
    mov     cx, 8       ; process 8 bits

tst:
    rol     byte ptr [si], 1 ; 1 bit to CF

    adc     ax, 0       ; ax = ax+0+CF


    loop    tst    ; loop
```

Leave each value the same way we found it.

Less expensive to add up the bits.

XOR Trickery

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

```
xor r2,r1
xor r1,r2
xor r2,r1
```

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

R1	R2
A	B

```
xor r2,r1
xor r1,r2
xor r2,r1
```

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

R1	R2
A	B
A	B⊕A

```
xor r2,r1
xor r1,r2
xor r2,r1
```

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

```
xor r2,r1
xor r1,r2
xor r2,r1
```

R1	R2
A	B
A	B⊕A
A⊕B⊕A	B⊕A

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations
- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

$A \oplus B = B \oplus A$

$(A \oplus B) \oplus C = A \oplus (B \oplus C)$

$A \oplus A = 0$

$A \oplus 0 = A$

```
xor r2,r1
xor r1,r2
xor r2,r1
```

R1	R2
A	B
A	B⊕A
A⊕B⊕A	B⊕A

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

$A \oplus B = B \oplus A$

$(A \oplus B) \oplus C = A \oplus (B \oplus C)$

$A \oplus A = 0$

$A \oplus 0 = A$

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

```
xor r2,r1
xor r1,r2
xor r2,r1
```

R1	R2
A	B
A	$B \oplus A$
$B \oplus A \oplus A$	$B \oplus A$

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations
- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

$A \oplus B = B \oplus A$

$(A \oplus B) \oplus C = A \oplus (B \oplus C)$

$A \oplus A = 0$

$A \oplus 0 = A$

```
xor r2,r1
xor r1,r2
xor r2,r1
```

R1	R2
A	B
A	B⊕A
B⊕A⊕A	B⊕A

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations
- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

$A \oplus B = B \oplus A$

$(A \oplus B) \oplus C = A \oplus (B \oplus C)$

$A \oplus A = 0$

$A \oplus 0 = A$

```
xor r2,r1
xor r1,r2
xor r2,r1
```

R1	R2
A	B
A	B⊕A
B⊕0	B⊕A

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

```
xor r2,r1
xor r1,r2
xor r2,r1
```

R1	R2
A	B
A	$B \oplus A$
B	$B \oplus A$

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

Instruction	R1	R2
	A	B
`xor r2,r1`	A	$B \oplus A$
`xor r1,r2`	B	$B \oplus A$
`xor r2,r1`	B	$B \oplus A \oplus B$

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

	R1	R2
	A	B
xor r2,r1	A	B⊕A
xor r1,r2	B	B⊕A
xor r2,r1	B	A⊕B⊕B

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

	R1	R2
	A	B
`xor r2,r1`	A	$B \oplus A$
`xor r1,r2`	B	$B \oplus A$
`xor r2,r1`	B	A

XOR

- Swap 2 registers
  - 3 instructions
  - 1 memory location
  - 2 memory operations

```
mov temp,r1
mov r1,r2
mov r2,temp
```

- With XOR
  - 3 instructions
  - 0 memory locations
  - 0 memory operations

```
xor r2,r1
xor r1,r2
xor r2,r1
```

R1	R2
A	B
A	$B \oplus A$
B	$B \oplus A$
B	A

This should work for any type of data (signed, unsigned, char, float, etc).

Intermediate results may look like garbage, but the final results should be correct.