Bit Vectors and Sets

Prof. Darrell Long

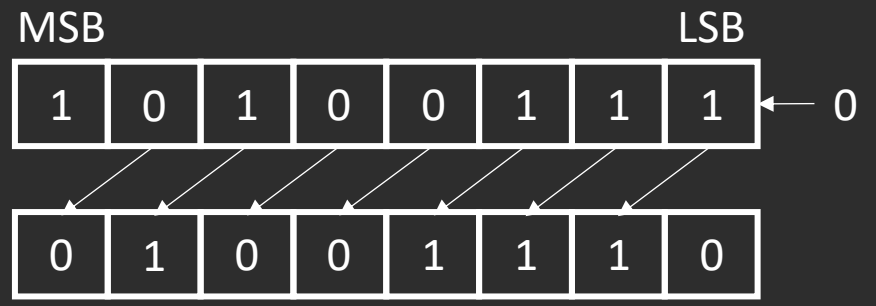
CSE 13S

Units of Information

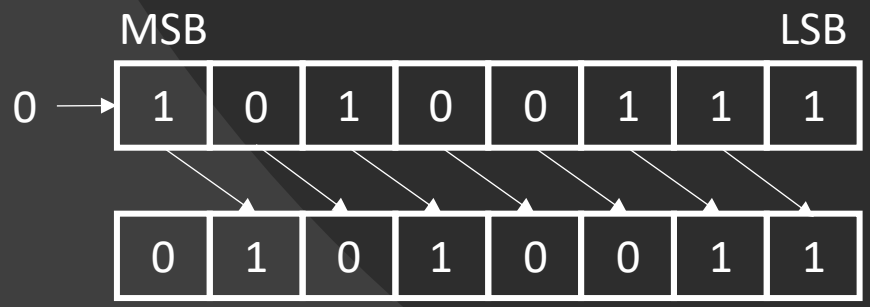
Unit	Size in Bits	Value	Notes
Bit	1	0/1	Smallest
Nibble	4	Hex digit	
Byte	8	ASCII	Smallest addressable
Half word	16		
Word	32		Native size, register length
Long Word	64		Native size, register length

Logical Shift

- Logical shift left: zeroes are shifted in on the right.

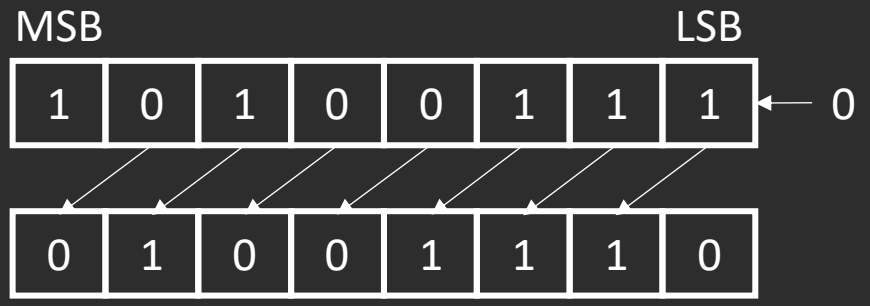


- Logical shift right: zeroes are shifted in on the left.

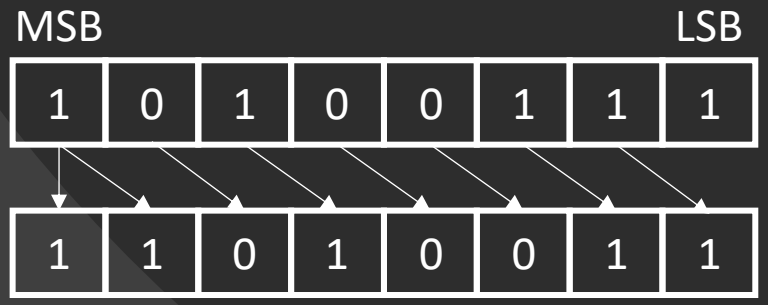


Arithmetic Shift

- Arithmetic shift left: zeroes are shifted in on the right.



- Arithmetic shift right: sign bits are shifted in on the left.



Bitwise Operations in C

- `&` – AND
- `|` – OR
- `~` – NOT
- `^` – XOR
- `<<` – Left shift
- `>>` – Right shift

C's Vexatious Right Shift (>>)

- The result of `v1 >> v2` is `v1` right-shifted `v2` bits.
- If `v1` is unsigned, or is signed with a non-negative value…
  - `(v1 >> v2) ==` (integral `v2` part of the quotient `v1 / 2^v2`)
- If `v1` is signed with a negative value…
  - `(v1 >> v2) ==` implementation defined
    - We don't know for sure.

C's Vexatious Left Shift (<<)

- The result of `v1 << v2` is `v1` left-shifted `v2` bits.
  - Zeroes are filled.
- If `v1` is unsigned…
  - `(v1 << v2) == (v1 * 2^v2) % n`
    - `n` is the maximum value of the resulting type + 1.
- If `v1` is signed with a non-negative value…
  - `(v1 << v2) == (v1 * 2^v2)`
    - If `(v1 * 2^v2)` is representable in the resulting type
- Else, the behavior is undefined.

AND (&) Truth Table

&	0	1
0	0	0
1	0	1

OR (|) Truth Table

|	**0**	**1**
0	0	1
1	1	1

XOR (^) Truth Table

^	**0**	**1**
0	0	1
1	1	0

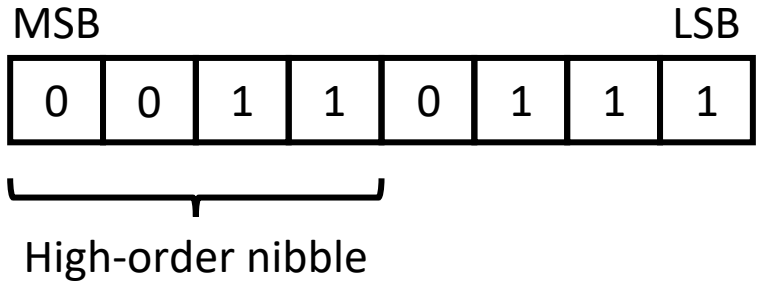
NOT (~) Truth Table

~	
0	1
1	0

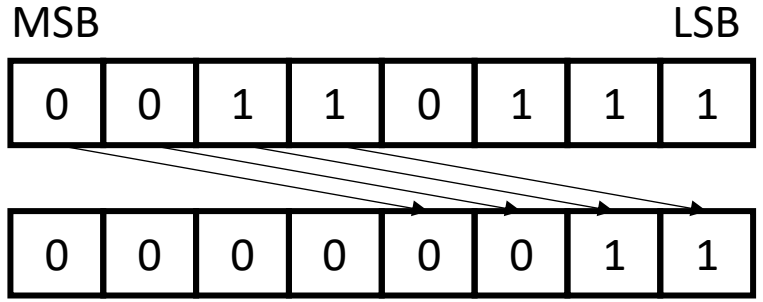
Getting A High-Order Nibble

1. A high-order nibble in a byte means the most significant 4 bits.
2. Bit-shift right 4 times so that the high-order nibble takes the place of the low-order nibble
3. AND with `0x0F`

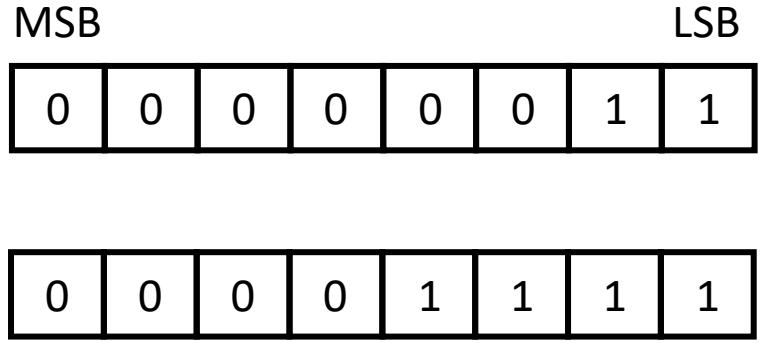
1.



2.



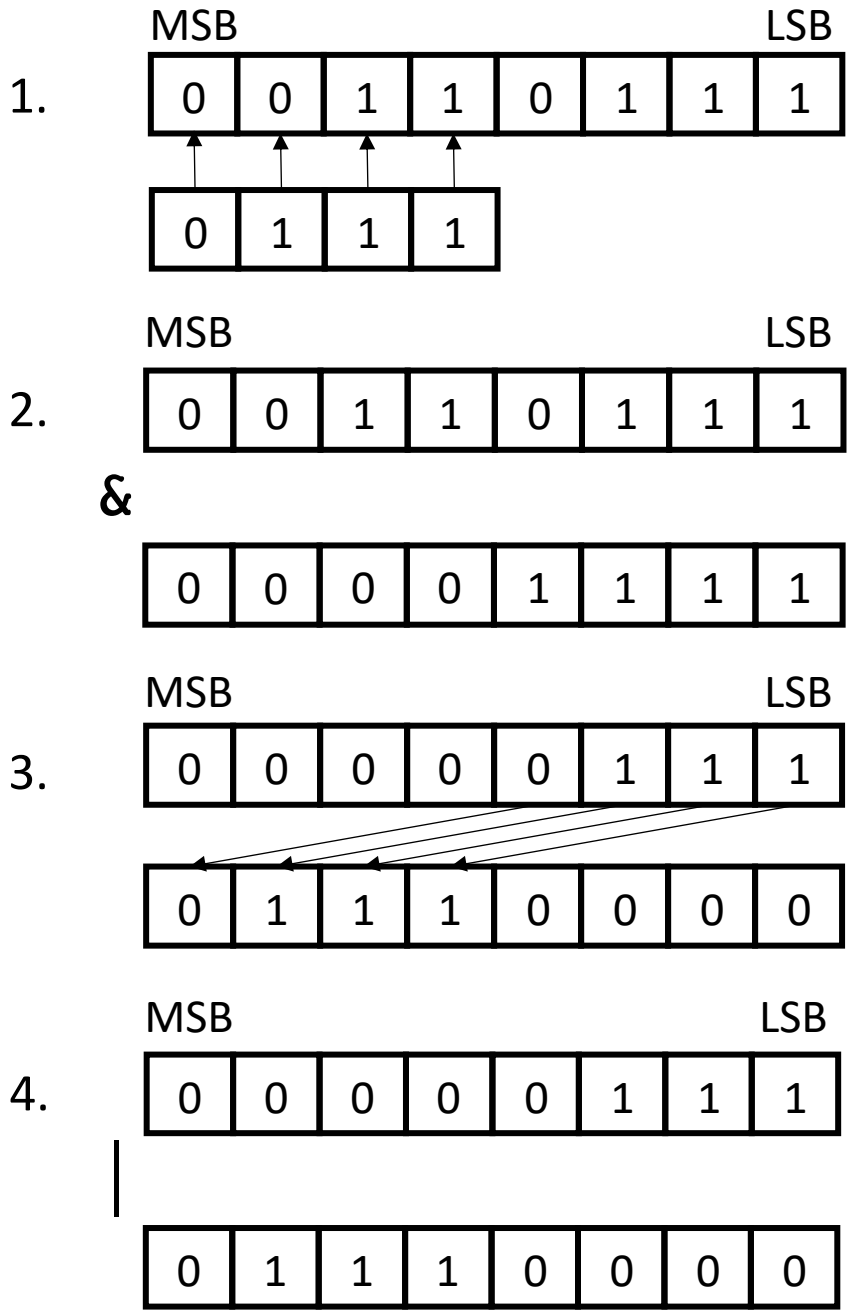
3.



7 February 2020

Setting A High-Order Nibble

1. We want to place a nibble into the higher-order bits of a byte
2. AND byte with `0x0F`
3. Bit-shift nibble left 4 times
4. OR byte with bit-shifted nibble



Sets

- Well-defined unordered collections that are characterized by the elements they contain.
- Sets are equivalent if and only if they have exactly the same elements.
- Basic relation in set theory is membership.
- Operations:
  - Intersection: $A \cap B$
  - Union: $A \cup B$
  - Difference: $A - B = A \cap \bar{B}$
  - Complement: $\bar{A}$ or A^C

dogs = {  ,  ,  ,  ,  ,  ,  ,  }

Set Operations with Dogs

- Let's first define a universal set of dogs:

```
dogs = { shih tzu, poodle, pitbull, lab,
                corgi, chihuahua, rottweiler, beagle }
```

good_dogs = {  ,  ,  ,  }

bad_dogs = {  ,  ,  }

good_dogs ∩ bad_dogs = {  }

Set Intersection (A ∩ B)

- The set of elements in A and B.
- `good_dogs = { pitbull, lab, corgi, beagle }`
- `bad_dogs = { shih tzu, poodle, pitbull }`
- `good_dogs ∩ bad_dogs = { pitbull }`

overrated_dogs = {  ,  }

cute_dogs = {  ,  }

overrated_dogs U cute_dogs = {  ,  ,  ,  }

Set Union (A ∪ B)

- The set of elements in A or B.
- `overrated_dogs = { shih tzu, poodle }`
- `cute_dogs = { corgi, beagle }`
- `overrated_dogs U cute_dogs = { shih tzu, poodle, corgi, beagle}`

some_dogs = {  ,  ,  ,  }

cute_dogs = {  ,  }

some_dogs – cute_dogs = {  ,  }

Set Difference (A – B)

- The set of elements in A that aren't in B.
- `some_dogs = { shih tzu, poodle, corgi, beagle }`
- `cute_dogs = { corgi, beagle }`
- `some_dogs - cute_dogs = { shih tzu, poodle }`

dogs = {  ,  ,  ,  ,  ,  ,  ,  }

cute_dogs = {   }

cute_dogs^C = {  ,  ,  ,  ,  ,  }

Set Complement ($\bar{A}$ or A^C)

- The set of elements not in A.
- Also defined as U – A, where U is the universal set.
- `dogs = { shih tzu, poodle, pitbull, lab, corgi, chihuahua, rottweiler, beagle }`
- `cute_dogs = { corgi, beagle }`
- `cute_dogs`^{C} `= { shih tzu, poodle, pitbull, lab, chihuahua, rottweiler }`

dogs = {  ,  ,  ,  ,  ,  ,  ,  }

some_dogs = {  ,  ,  ,  ,  }

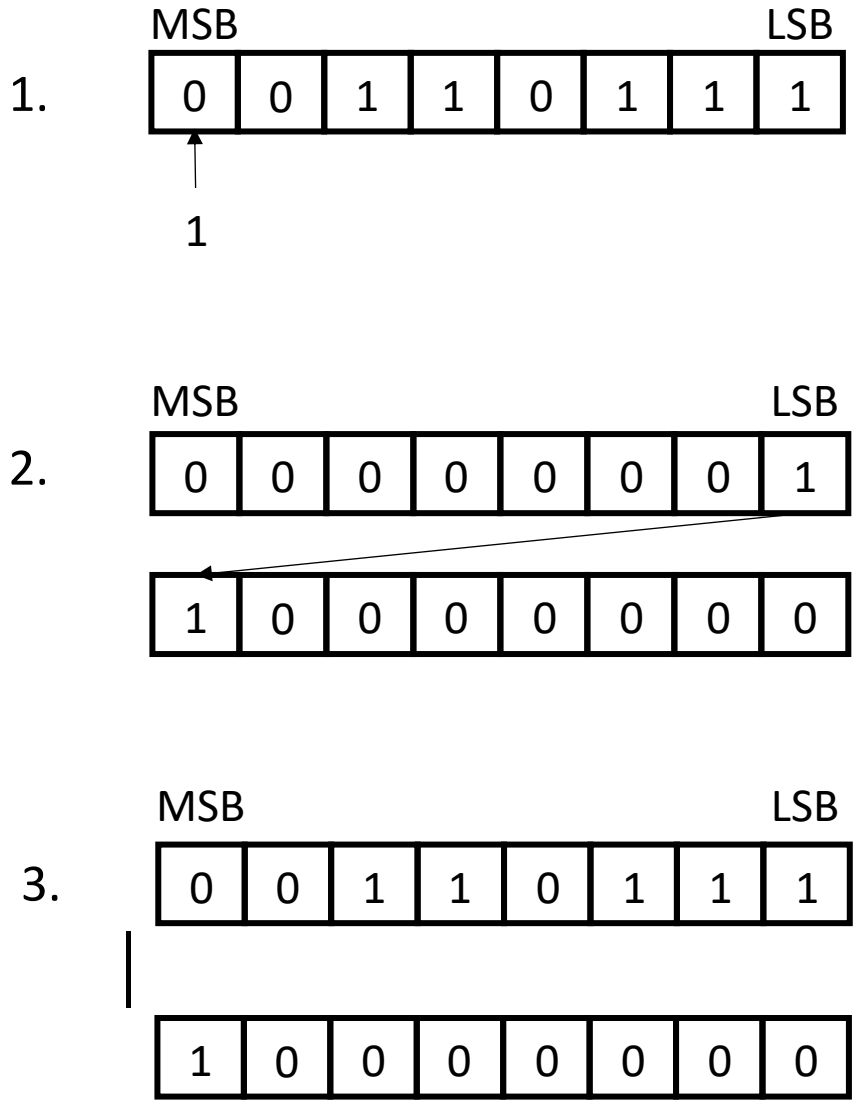
bits = { 1 , 0 , 0 , 1 , 1 , 0 , 1 , 1 }

Representing Sets with Bits

- Sets can be represented with bits.
- A 0 indicates that the element is not a member of the set.
- A 1 indicates that the element is a member of the set.

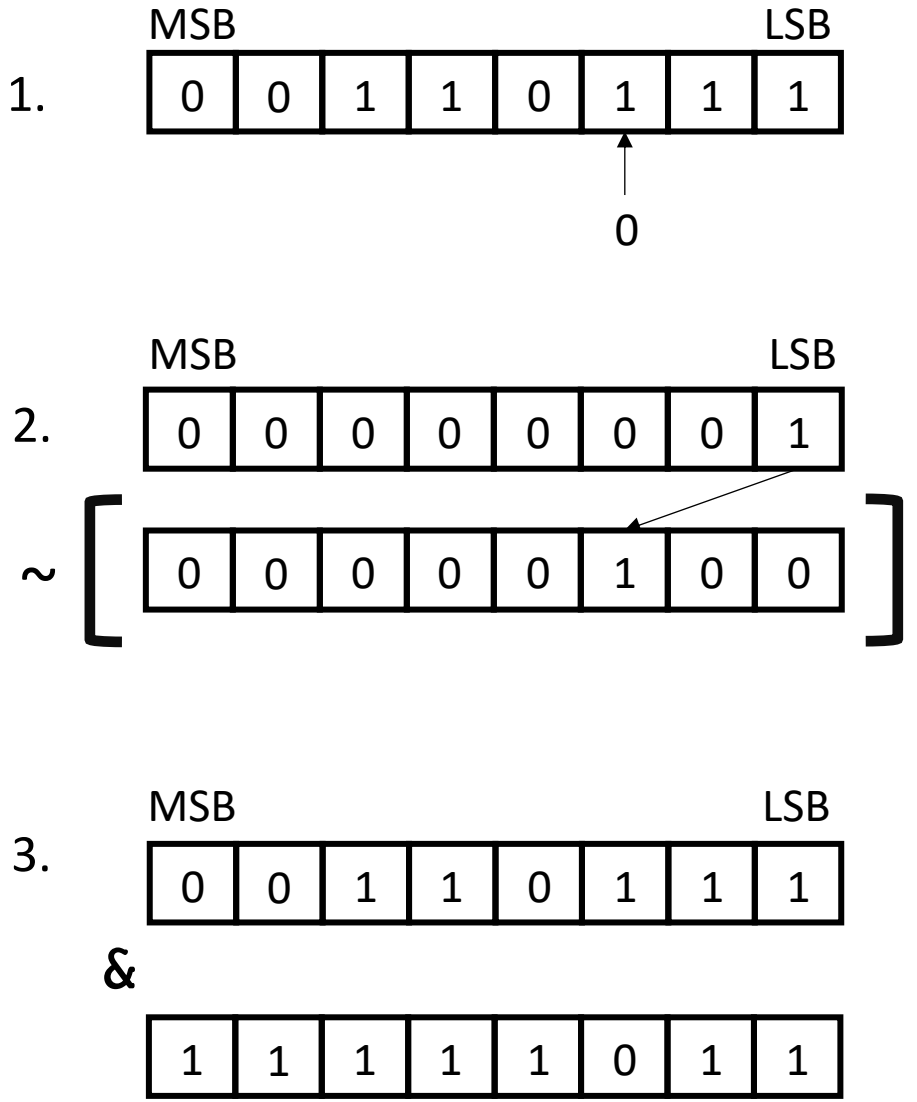
Setting A Bit

1. We want to set the bit at index 7 in a byte.
2. Take another byte with the bit at index 7 set
   - Can do this by shifting `0x1` left 7 times.
3. OR the bytes together to set the bit.



Clearing A Bit

1. We want to clear the bit at index 2 in a byte.
2. Take another byte with all bits set *except* the bit at index 2.
   - Can do this by shifting `0x1` left 2 times and taking the bitwise NOT of the result.
3. AND the bytes together to clear the bit.



Getting A Bit

1. We want to get, or return, the value of the bit at index 4 in a byte.
2. Take another byte with the bit at index 4 set.
   - Can do this by left-shifting `0x1` 4 times.
3. AND the bytes together to mask every bit except the bit at index 4.
4. Right-shift the AND-ed result 4 times to get the value.