



Bonus Lecture: Bit Manipulation in C

EECS 388 – Fall 2022

© Prof. Mohammad Alian

- Binary to Hex to Decimal conversion!
- 1 hex symbol is 4 binary bits
- 1 hex symbol represents decimal values from 0 to 15
 - 0 – F
- 0xAB is 8 bits (1 bytes)
- 0xA is 4 bits (not one bytes!)

- Two's Complement Numbers
e.g., a 4-bit signed variable:

Positive Values		Negative Values	
Decimal	Binary	Decimal	Binary
7	<u>0</u> 000	-1	<u>1</u> 111
6	<u>0</u> 001	-2	<u>1</u> 110
5	<u>0</u> 010	-3	<u>1</u> 101
4	<u>0</u> 011	-4	<u>1</u> 100
3	<u>0</u> 100	-5	<u>1</u> 011
2	<u>0</u> 101	-6	<u>1</u> 010
1	<u>0</u> 110	-7	<u>1</u> 001
0	<u>0</u> 111		

Bitwise Operation Facts

XOR

- $\text{var} \wedge 0\text{s} =$
- $\text{var} \wedge 1\text{s} =$
- $\text{var} \wedge \text{var} =$

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

AND ✓

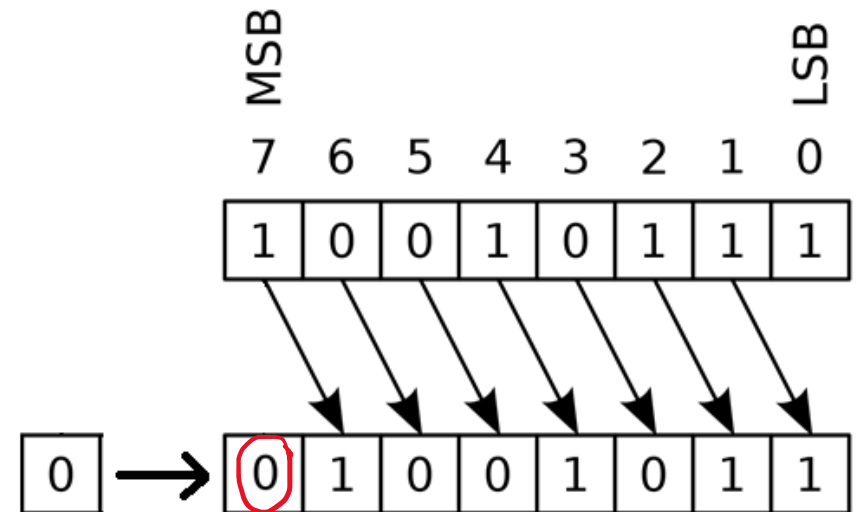
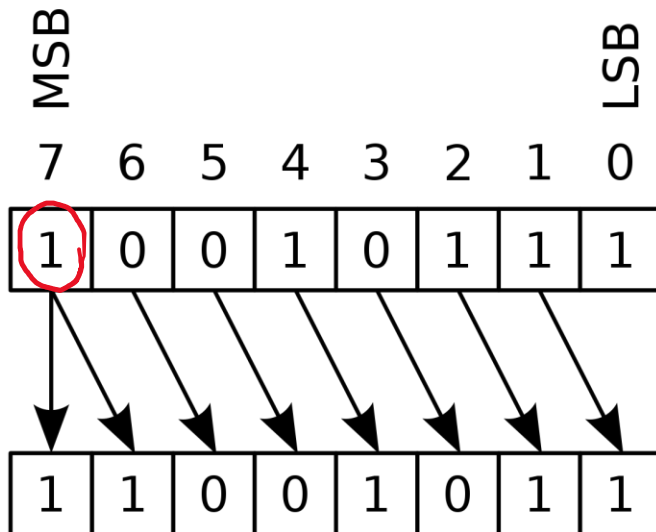
- $\text{var} \& 0\text{s} =$
- $\text{var} \& 1\text{s} =$
- $\text{var} \& \text{var} =$

OR ✓

- $\text{var} | 0\text{s} =$
- $\text{var} | 1\text{s} =$
- $\text{var} | \text{var} =$

• Arithmetic vs. logical shift

Arithmetic



Get bit: return 0 if the bit “i” is zero, otherwise return 1

```
uint8_t getBit(uint8_t num, int i) {
    return ((num & (1 << i)) != 0);
}
```

Handwritten annotations on the code:

- A red wavy line under `uint8_t num` with a blue circle around `32` and a blue arrow pointing to `num`.
- A blue circle around `int i` with a red double arrow pointing down.
- A blue circle around `1 << i` with a red arrow pointing up.
- A blue circle around the closing brace of the function.
- A blue arrow pointing from the function signature to the function body.
- A blue arrow pointing from the function body to the closing brace.
- Handwritten text to the right: `if (i > 31) return -1;`

$\leftarrow \text{getBit}(0x01, 1)$
 \downarrow
 \downarrow
 $0b00000001$
 $0b00000001 \ll 1 \rightarrow 0b00000010$
 $\ll 1$
 $0b00000010$

Set bit: set bit "i" to 1

```
uint8_t setBit(uint8_t num, int i) {  
    return num | (1 << i);  
}
```

ith
↓
0b 0000 1 000

reset Bit (num, i)

return num \otimes (1 << i);
↑↑ ↑↑
Op Mask

0b 1111 0 111 ←
↑
ith

Reset/clear bit: set bit “i” to 0

```
uint8_t clearBit(uint8_t num, int i) {  
    uint8_t mask = ~(1 << i);  
    return num & mask;  
}
```


Update bit: set bit “i” to 0 if flag is 0, otherwise set it to 1

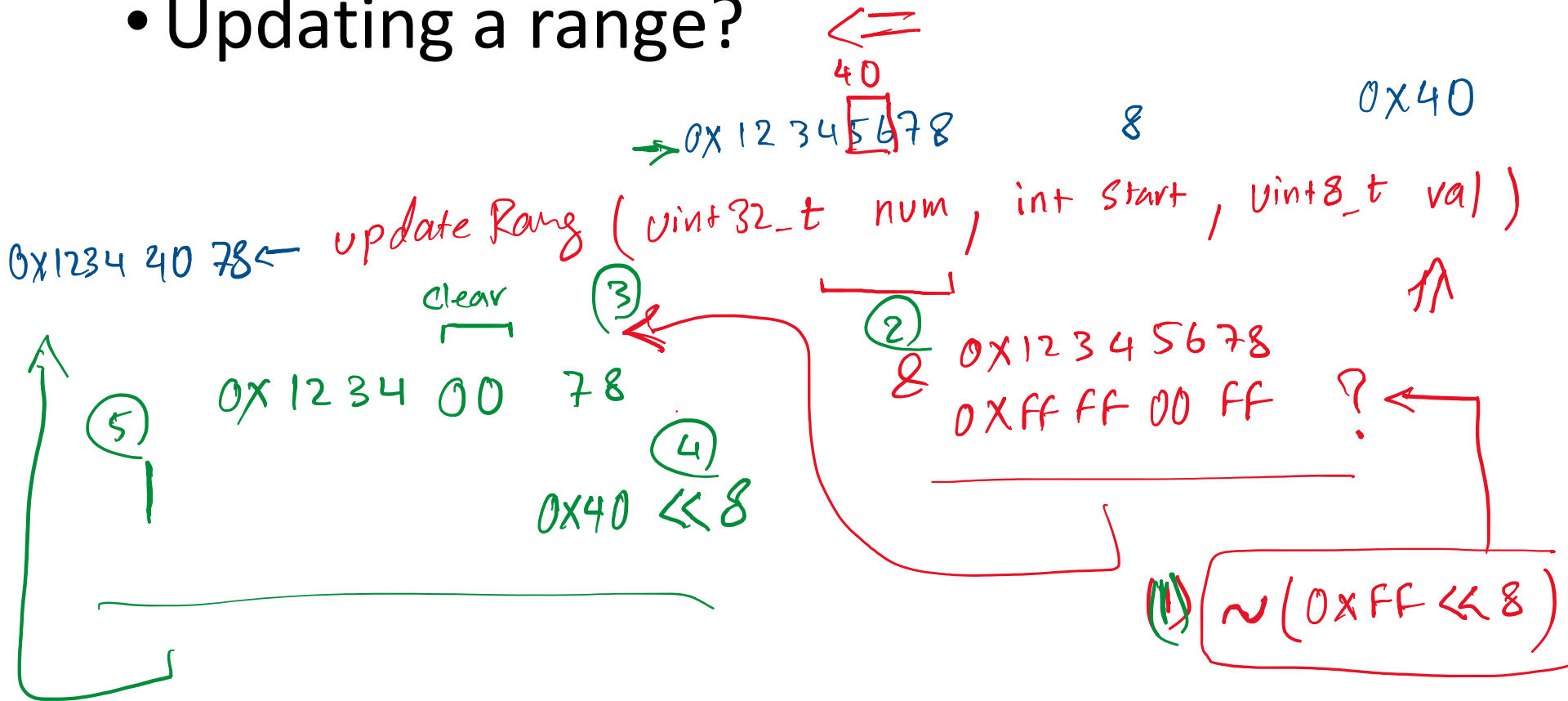
```
uint8_t updateBit(uint8_t num, int i, int flag) {  
    uint8_t value = flag ? 0x01: 0x0;   
    uint8_t mask = ~(1 << i);  
    return (num & mask) | (value << i);  
}
```

reset bit i *set it to either 0 or 1*

- Setting a range?

- Clearing a range?

• Updating a range?



- Returning a range?