
# ARM Microcontroller Internship Programme

**Trainer:** *Kiran Jaybhave*

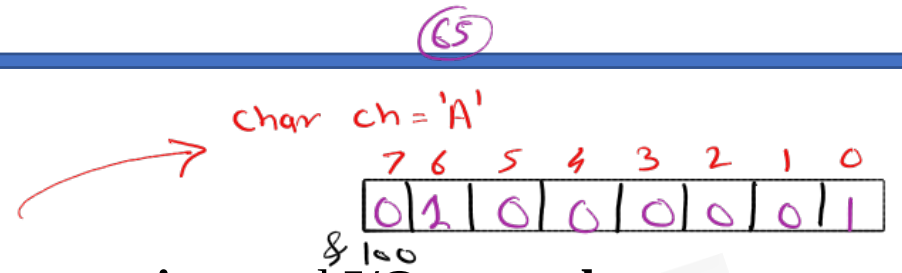
## Sunbeam – Industrial Training Programme

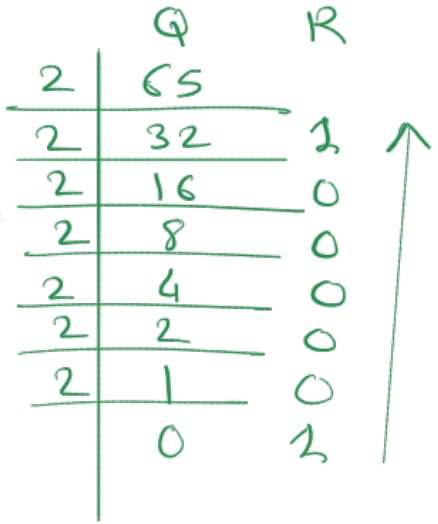
## Sunbeam Pune (Hinjawadi)

- **Prerequisites**
  - **C Programming:**
    - Proficiency in C syntax, pointers, memory management, structures, bit manipulation, and preprocessor directives
  - **Basic Electronics:**
    - Understanding of digital electronics, analog circuits, voltage levels, timing diagrams, and basic circuit analysis

# Bit Manipulation

- **What is Bit Manipulation?**
  - Operations performed directly on **bits** (0s and 1s).
  - Used heavily in **embedded systems**, **register programming**, and **I/O control**.




- **Why Bit Manipulation is Important?**
  - Used for **GPIO control**, **interrupt flags**, **status registers**.
  - Essential for **memory-mapped I/O** and **hardware-level programming**.
  - Helps reduce code size and improves speed.

- # Common Bit Operations

1. **Set a bit** → Make a bit = 1
   value |= (1 << n);
2. **Clear a bit** → Make a bit = 0
   value &= ~(1 << n);
3. **Toggle a bit** → Flip bit (0→1 or 1→0)
   value ^= (1 << n);
4. **Check a bit** → Test if bit = 1
   if (value & (1 << n))
5. **Masking** → Isolate selected bits
   value & mask
6. **Shifting** → Move bits left or right
   value << n, value >> n

# Print Num in Binary

-10 ⇒ True
10 ⇒ True

0 ⇒ False

num = 10

num ⇒ | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

& 100

mask = 0 0 0 0 1 0 0 0
VP & 0 0 0 0 1 0 1 0

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ⇒ 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ⇒ 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ⇒ 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ⇒ 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ⇒ 8 ⇒ 1 |

```
void print_bin (void *vp, int size)
{
    mask = 0x80;
    while ( mask )
    {
        if ( mask & *(char*)vp )
            pf (1)
        else
            pf (0)
        mask = mask >> 1;
    }
}
```

```
void *vp = 100;
size     = 1;
```

⇒

```
char mask = 0x80;
```

| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

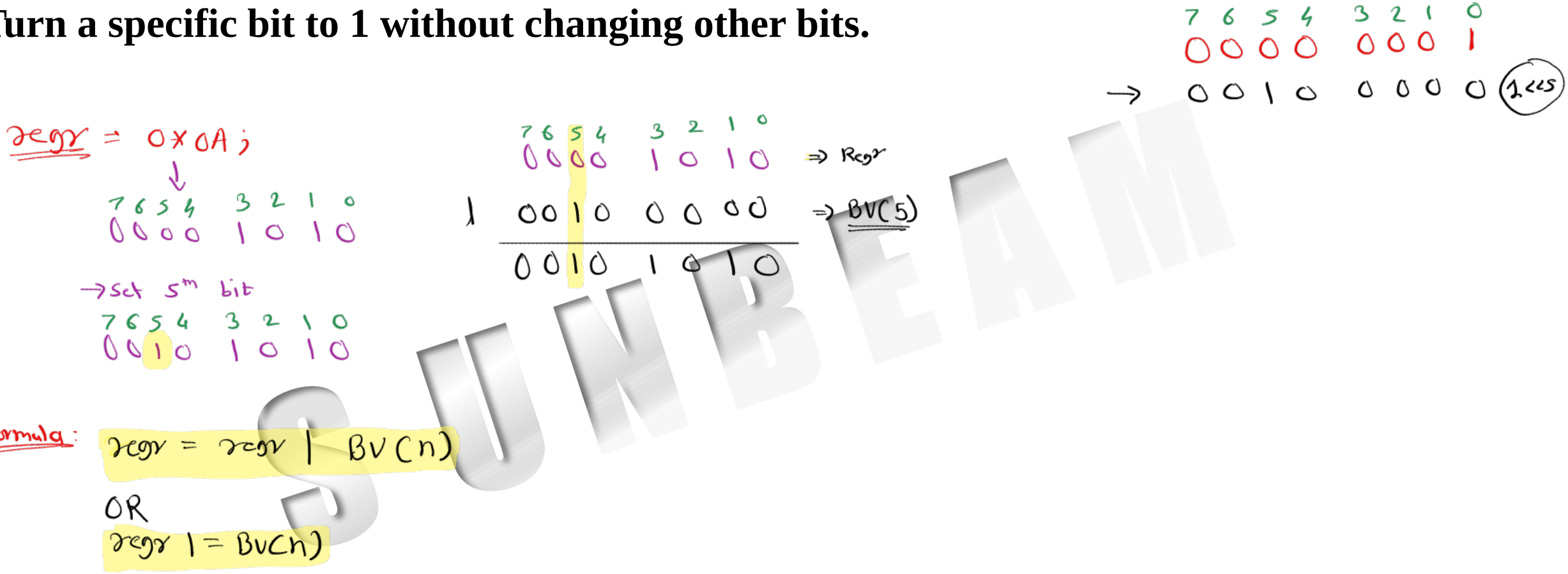
int regr = 0XFAFA;

| 00 | 00 | FA | FA |
|---|---|---|---|
| 0000 0000 | 0000 0000 | 1111 1010 | 1111 1010 |
| mask = 0X80 | mask = 0X80 | mask = 0X80 | mask = 0X80 |
| 103 | 102 | 101 | 100 |

# Set a bit

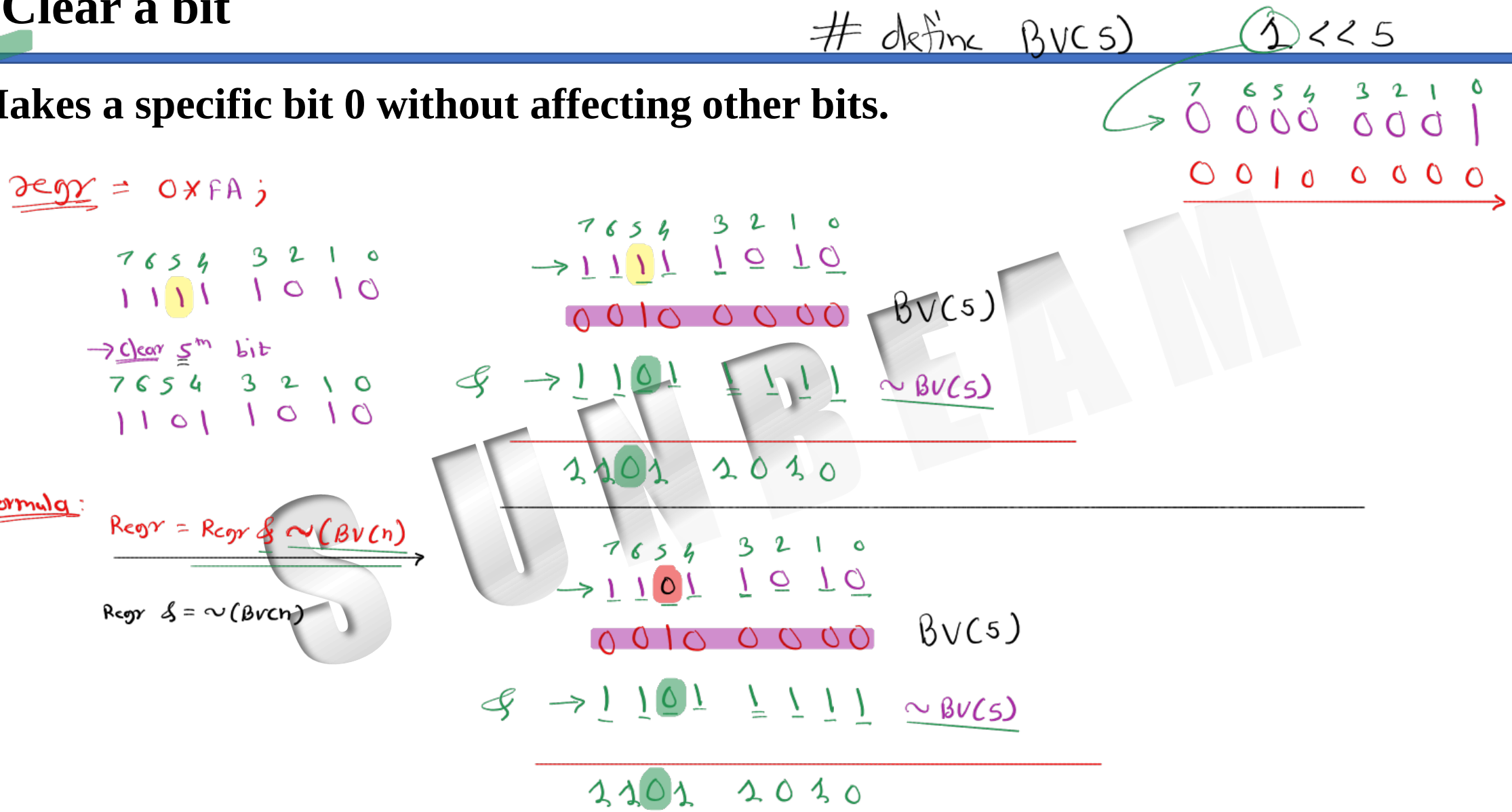
#define BV(n) 1 << n

- **Turn a specific bit to 1 without changing other bits.**



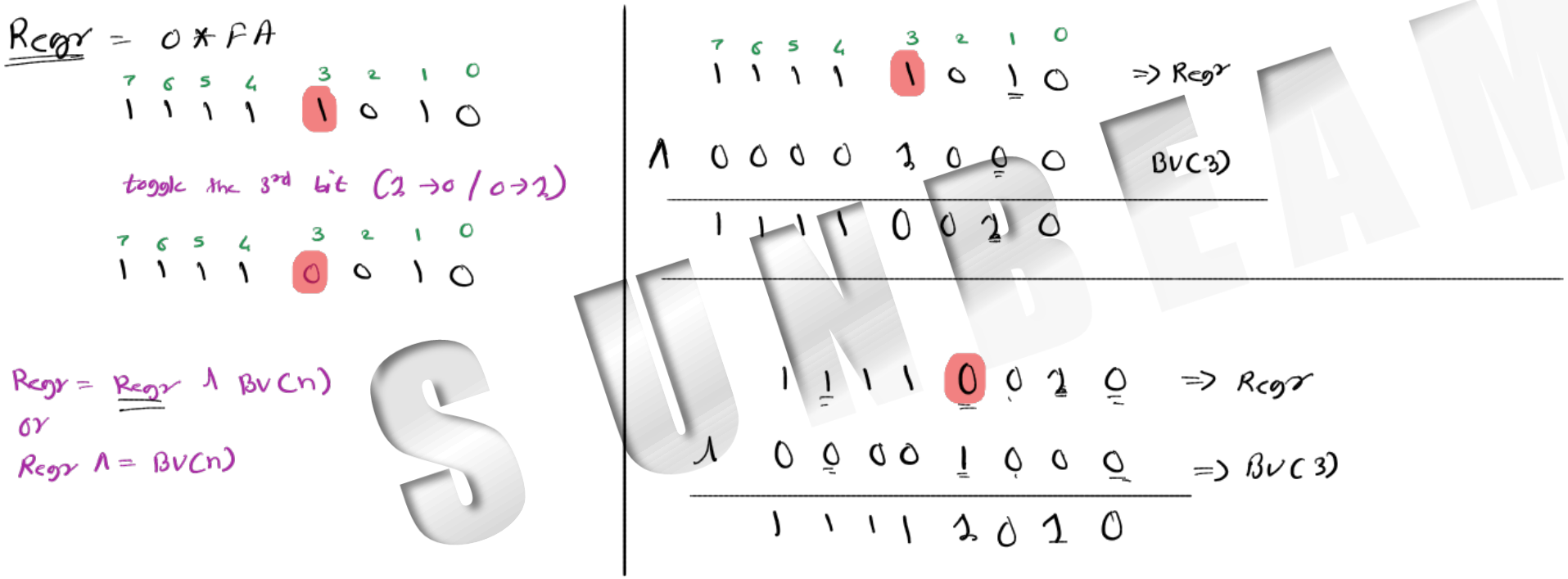
# Clear a bit

- **Makes a specific bit 0 without affecting other bits.**



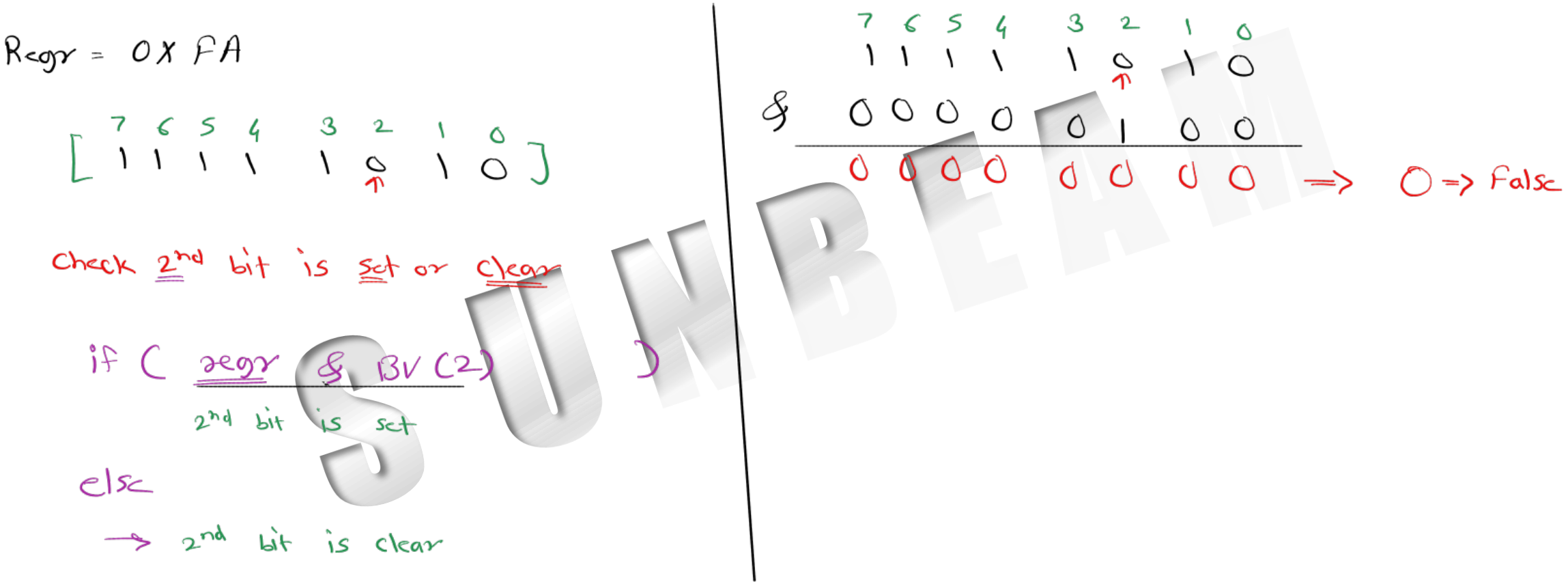
# ➢ Toggle a bit

- **Flips the bit value (0 becomes 1, and 1 becomes 0).**



# ➢ Check a bit

- **Tests whether a specific bit is 1 or 0.**



# ➢ Masking

- **Extracts only the required bits and hides the rest.**

# ➢ Left Shifting

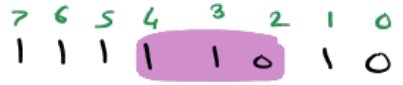
- **Moves bits to the left, multiplying the value by powers of two.**

# ➢ Right Shifting

- **Moves bits to the right, dividing the value by powers of two.**

# ➢ Read value from bit $n^{th}$ to $n^{th}$ of register

Regr = 0 x FA




→ extract value of bit min 2, 3, 4 max

(1 1 0) = 6

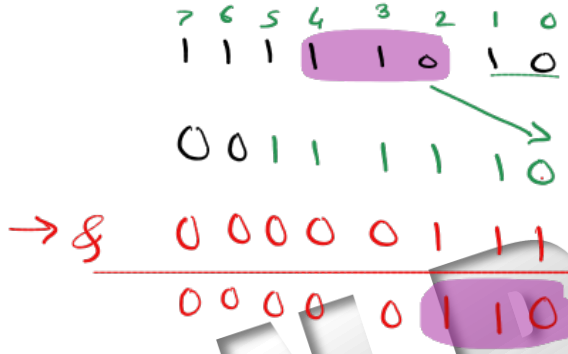
Vale = (Regr >> min) & mask

mask = (max - min) + 1

= (4 - 2) + 1

⇒ 2 + 1

⇒ 3 (1 1 1)

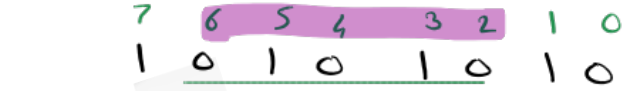



⇒ Regr >> 2

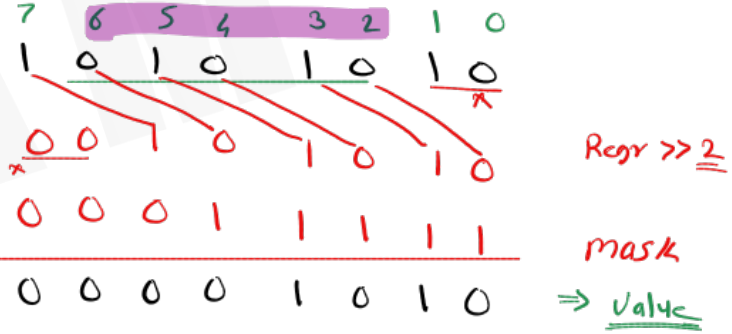
⇒ mask

⇒ Value

Regr = 0x A A

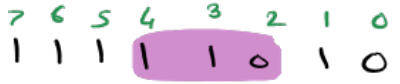



→ Read the bit (2) 3, 4, 5, 6



# ➢ Write value from bit $n^{th}$ to $n^{th}$ of register

Regr = 0 x FA

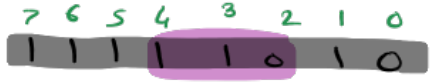



→ extract value of bit 2, 3, 4

(000) = 0

Formula :

Regr = Regr & ~(BV(2) | BV(3) | BV(4))




0 0 0 0 0 1 0 0 => BV(2)

0 0 0 0 1 0 0 0 => BV(3)

0 0 0 1 0 0 0 0 => BV(4)

0 0 0 1 1 1 0 0 (BV(2) | BV(3) | BV(4))

1 1 1 0 0 0 1 1 ~(BV(2) | BV(3) | BV(4))

& 1 1 1 0 0 0 1 0 ←

# *Polling to bit (Check bit continuously)

Regr = [ 0 0 0 0 **0** 0 0 0 ]  (7 6 5 4 **3** 2 1 0)

```
while ((Regr & BV(3)) == 0 )
;
```

→

```
  0 0 0 0  0 0 0 0
& 0 0 0 0  1 0 0 0
-----------------
  0 0 0 0  0 0 0 0   → 0
```

Regr = [ 0 0 0 0 **1** 0 0 0 ]  (7 6 5 4 **3** 2 1 0)

```
while ( (Regr & BV(3)) == 1 )
;
```

→

```
  0 0 0 0  1 0 0 0
& 0 0 0 0  1 0 0 0
-----------------
  0 0 0 0  1 0 0 0   → Non-zero (loop)
```

⇒ 0 (Stop the loop)


# Thank you!

Kiran Jaybhave

email – kiran.jaybhave@sunbeaminfo.com