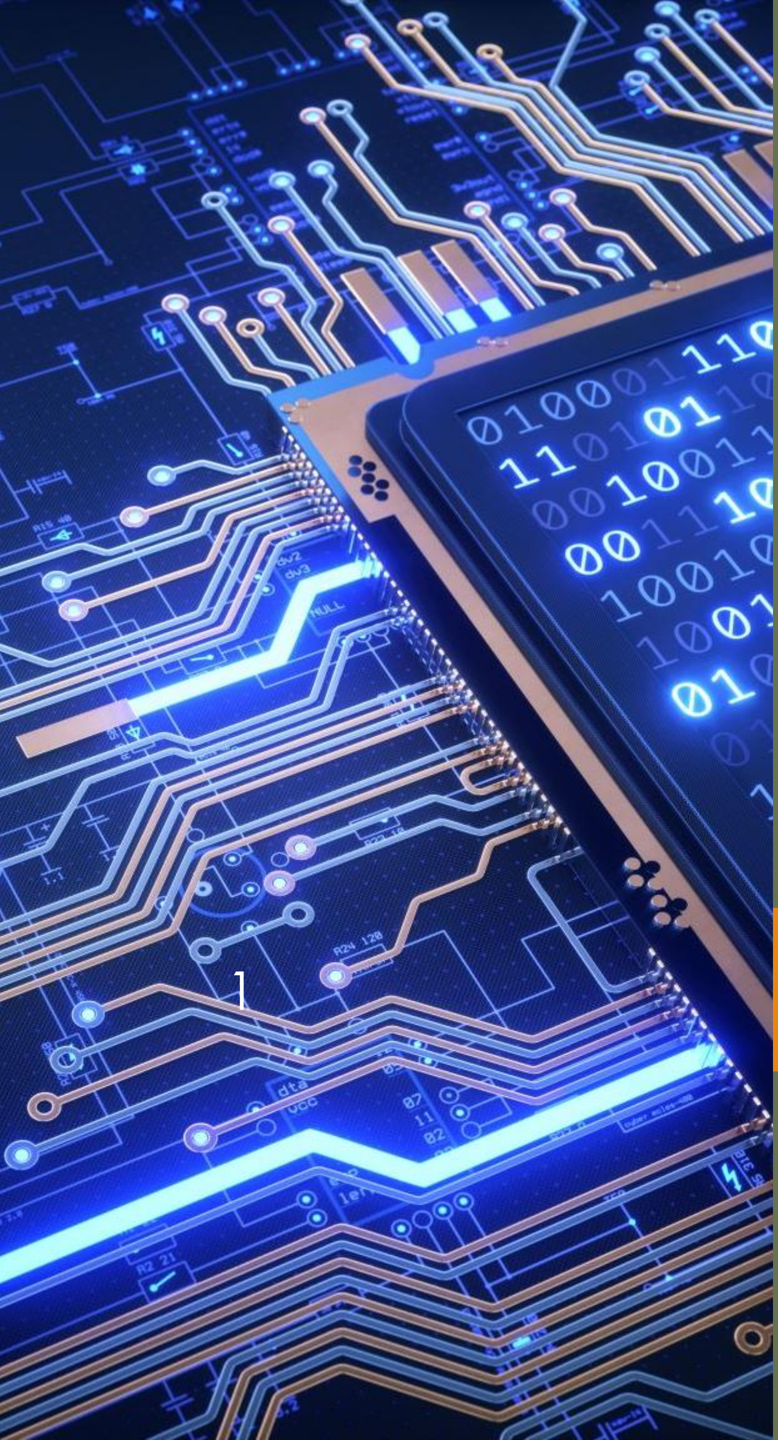


Embedded Controller programming for Real Time Systems: ECE-40097

Lesson 1

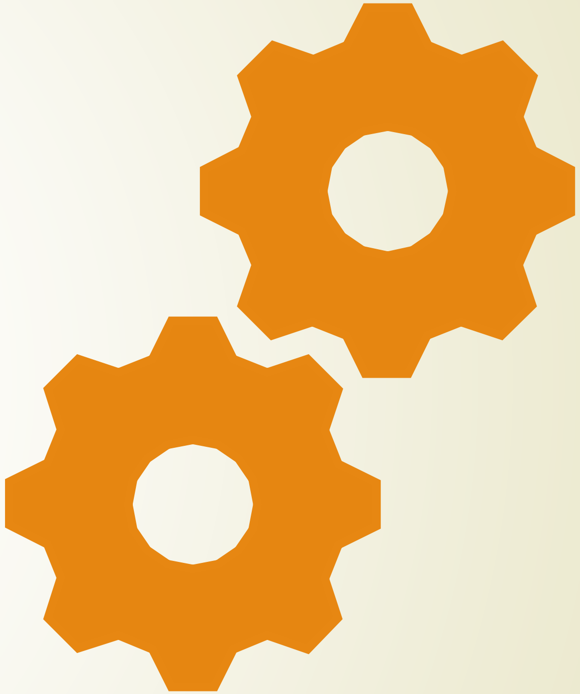
Vijay Kumar

Main Topics

- Real Time Embedded System
- Components of Embedded systems
- Types of microcontroller
- C data Types
- Bitwise Operation in C
- Signed/Unsigned Numbers
- Different Types of Flag

Real Time Embedded System

- Embedded system is embedded within the large systems or part of Echo system
- Embedded system is designed to perform dedicated function
    - E.g. Factory automation
- Real time systems are designed to provide guaranteed worst-case response time to critical events
- Real time system designed as an embedded component
    - Is called real time embedded system
- Real time embedded systems are found in every facets of everyday lives
    - And they are proliferating (Especially with IoT)



Non-Real Time Embedded System

- Non real time systems are not time bounded
- Not time sensitive
- Tasks completion take priority over time constraints

Embedded Software

- Even with most powerful hardware, real time embedded software plays critical role
  - E.g. Sensors in IoT echo system
- Complexity of software grows with complexity of embedded hardware
  - Multi core Processor
  - System with multiple back-end communications
- More than one controllers are common for typical embedded systems
  - One for real time data processing and other one for applications

Types of Real Time Embedded System

- Any system which responds in real time within specified time constraints.
  - Hard real time system
  - Soft real time system
  - Firm real time system (sometimes)

Hard Real Time Embedded System

- If the time constraint is strict then it is called Hard real time system
  - Any deviation is not acceptable and will lead to system failure
  - There is no value to a computation if it is late and the effects of a late computation may be catastrophic
  - A hard-real-time system is one where all activities must be completed on time
  - Examples are mission critical system, cellular systems, Automation, Air traffic controller

Soft Real Time Embedded System

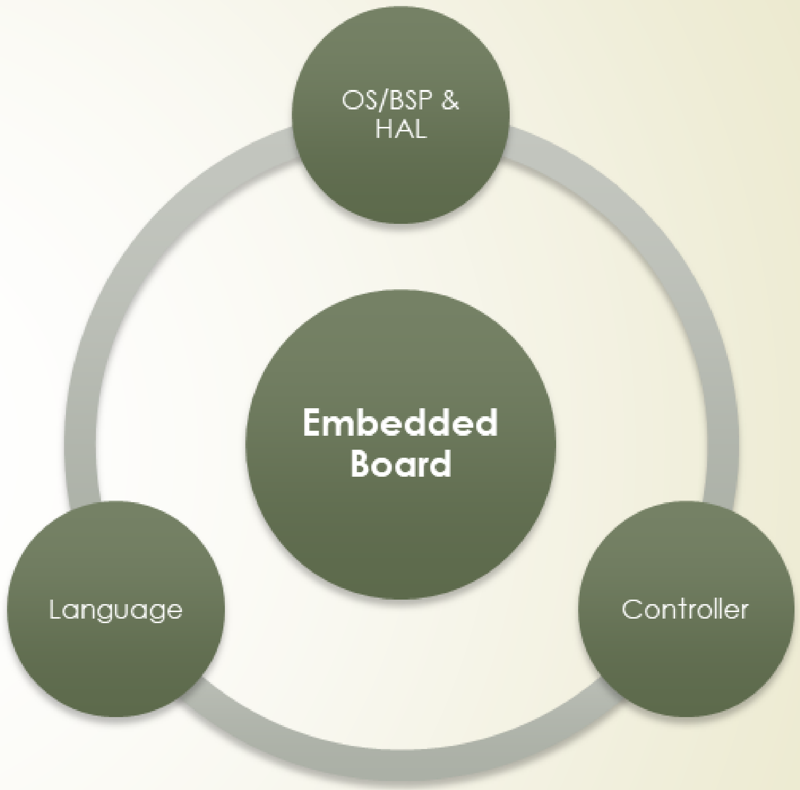
- If the time constraints could be tolerated, then it is called soft real time system
  - Computation values diminishes according to its tardiness
  - Deviation leads to degraded performance but still acceptable
  - Deadlines may be missed but the number and frequency typically comply with QOS
    - E.g. Temp sensors, home appliances

Firm Real Time Embedded System

- It is in-between Hard and Soft Real time system
- Slight deviation is acceptable without any failure
  - E.g. video Streaming

Embedded System

- Three (among many others) major components of embedded system are
  - Controller
    - **Main focus of this course**
  - Operating System/HAL
    - Will use HAL heavily in this class
  - Language (Depends on OS) – why ?



Other Components

- Other components deal with communication, external interface and storage
  - Display unit
  - Connectivity
    - Wired – Ethernet, USB, Serial, GPIO
    - Wireless – Wi-Fi, BT, BLE, Cellular, Satellite, NFC
  - Networking stack
  - Storage devices
    - Flash, RAM, ROM

Real time OS

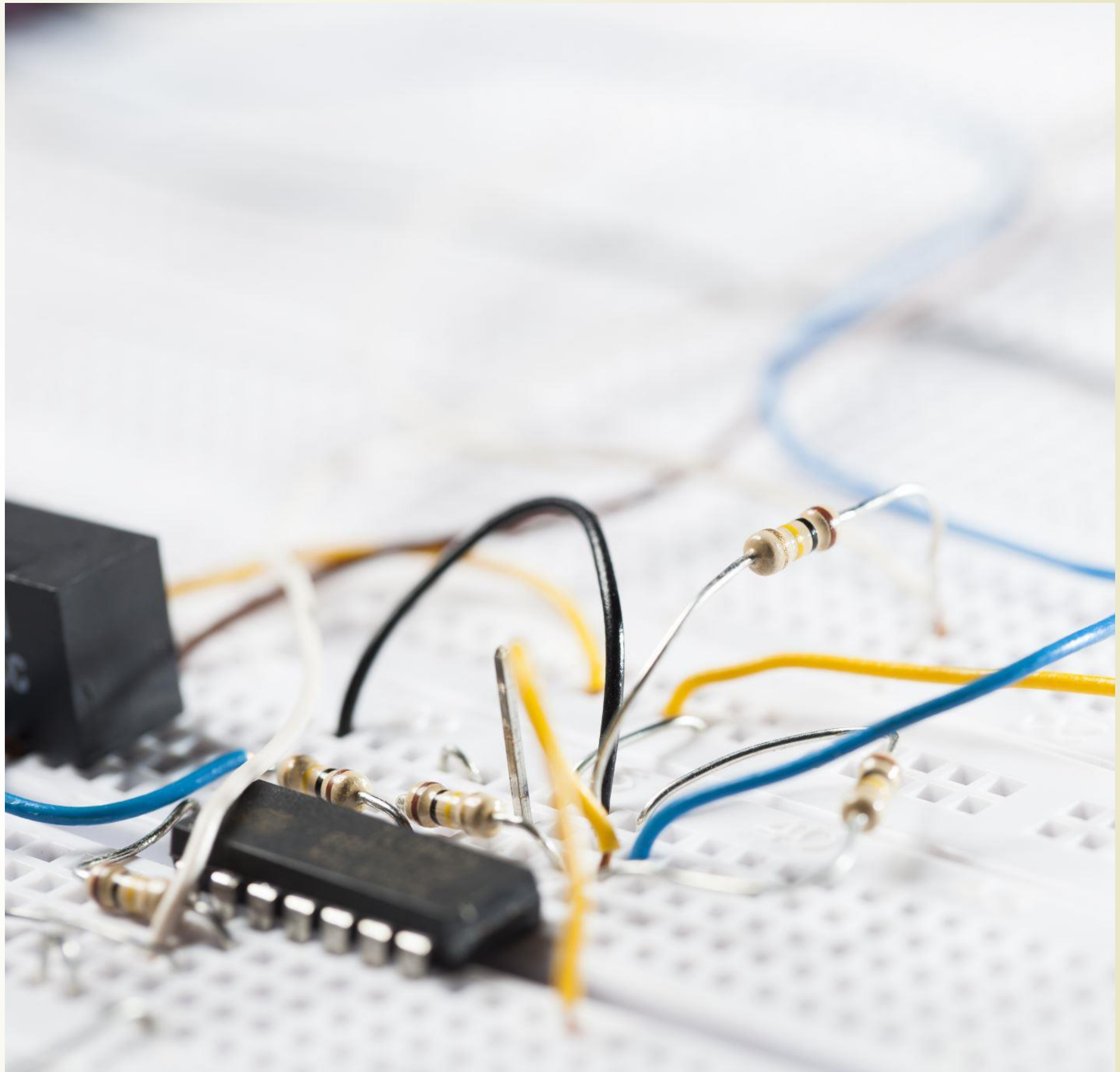
- Choice of operating system (OS) depends on the system applications
  - Real time (Windows Embedded, VxWorks, Linux Embedded, QNX, XINU, Open RTOS ….)
  - Non real time (Windows desktop, Linux desktop, Android ?, IOS ?)
- Language primarily depends on the OS and types of processor
  - C/C++
  - Assembly
    - primarily for IOT type devices and/or Hard Real time Systems
  - Mix of C/C++ and assembly
  - C# or Java or may be Python

Types of Controllers

- Myriad of controllers in the market
- SoC (System on chip)
  - Various functions on a single chip
  - Small form factor and designed for low power consumption
  - Goes in sync with IOTs
- SOB/SBC (System on Board, Single Board computer)
  - Lot of peripherals support (CAN, UART, HDMI, Ethernet, 1553)
  - Might include SoC
  - Advanced embedded system
- Development kit is based on ARM® Cortex®-M4 32-bit RISC core
  - Is it SoC or SOB?

Controller Selection

- Selection of controller depends on many factors
  - Cost
  - Interrupt latency
  - Size
  - Application type (low end or high demanding)
  - Communication support
  - Tools and support availability
    - Support from controller vendors
  - Power consumption



Cortex-xx Family

- ARM Cortex-A family:
  - **A**pplications processors
  - Support OS and high-performance applications
  - Such as ,Tablet, Smartphones

Other Types

- ARM Cortex-R family:
  - **R**eal-time processors with high performance and high reliability
  - Fault Tolerance
  - ➨ Support real-time processing and mission-critical control

- ARM Cortex-M family:
  - **M**icrocontroller
  - Cost-sensitive, support SoC

C Data Types

- Next few slides are very basic but are useful for this course
  - Added here to give you an overview and refresh your memory
- Might have been covered in previous courses

C Data Types

Basic Data Types	Typical size in Memory	Range
Char/unsigned char	1 byte	-128 to 127/ 0- 255
Short int/ unsigned short int	2 bytes	-32,768 to 32,767/0 - 65535
Integer/unsigned int	4 bytes	-2,147,483,648 to 2,147,483,647/ 0 – 4,294,967,295
Long/unsigned long	4 bytes	…..
Long Long/unsigned long long	8 bytes	….
Float	4 bytes	…
Double	8 bytes	

Pattern

0xDEADBEEF	Dead Beef
0xBADDCAFE	Bad Cafe
0xFEE1DEAD	Feel Dead
0x8BADF00D	Ate Bad Food
0xBAADF00D	Bad Food
0xDEADC0DE	Dead Code
0xFACEB00C	Facebook
0xDEADD00D	Deade Dude

Bit-Wise Operation in C

- One of the most powerful feature in C
- Logical operators
  - && (and), II (or), ! (not)
- Bit-wise operators
  - & (and), I (or), ^ (x-or), ~ (invert), << (left shift), >> (right shift)
- Used for resetting, setting, toggling, masking bits and testing bits

Binary, Octal, Decimal, Hex

Decimal	Binary	Octal	Hex
0	0000	00	0x0
1	0001	01	0x1
2	0010	02	0x2
3	0011	03	0x3
4	0100	04	0x4
5	0101	05	0x5
6	0110	06	0x6
7	0111	07	0x7
8	1000	010	0x8
9	1001	011	0x9
10	1010	012	0xA
11	1011	013	0xB
12	1100	014	0xC
13	1101	015	0xD
14	1110	016	0xE
15	1111	017	0xF

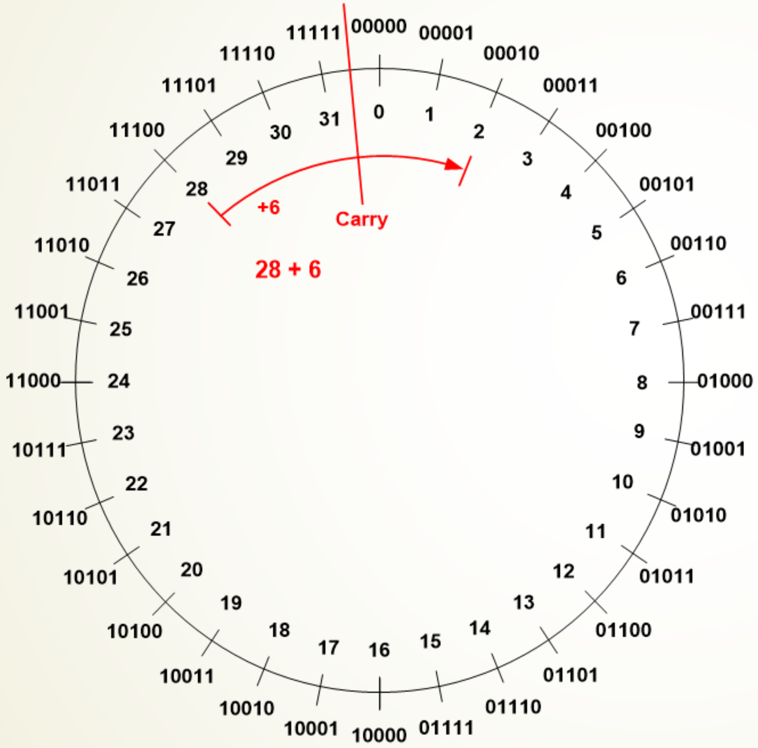
Examples

A	B	AND (A & B)	OR (A | B)	EX-OR (A ^B)	Invert ~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	1
1	1	1	1	0	0

Carry/Borrow Flag for unsigned numbers

- If addition result is larger than the maximum unsigned integer, then Carry occurs
- If subtraction results is negative, then borrow occurs
  - If result is positive, then Carry is Set
- On ARM Cortex-M processors, the carry flag and the borrow flag are physically the same flag bit in the status register ASPR (Application Program Status Register)
  - For an unsigned subtraction, Carry = NOT Borrow

Carry/Borrow Flag for Addition



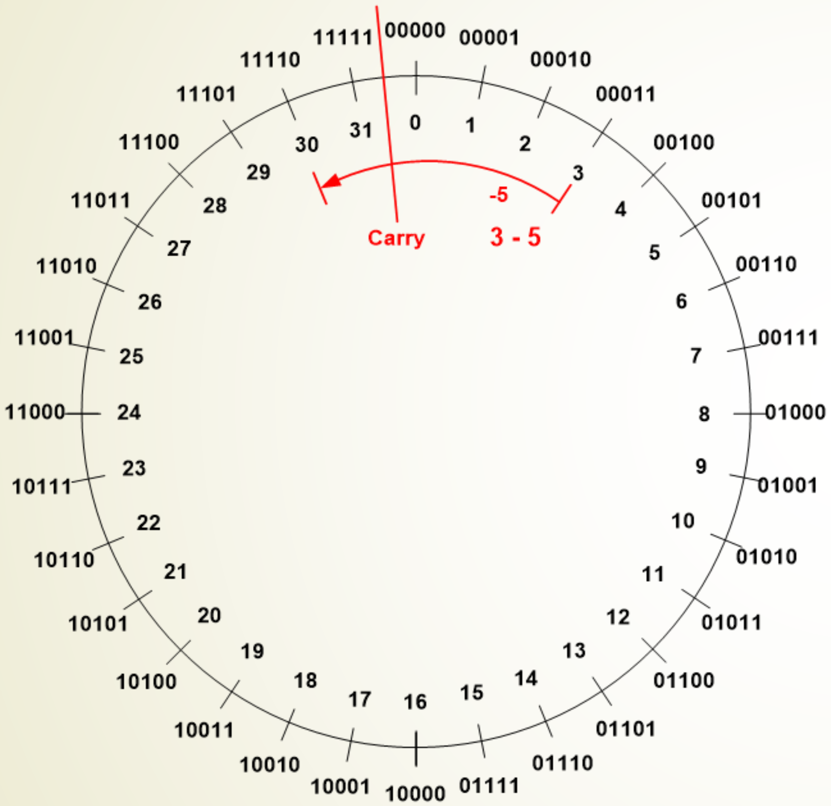
	1 1 1 0 0	28
+	0 0 1 1 0	+ 6
$\boxed{1}$	0 0 0 1 0	2

Extra bit is discarded

5-bit result

- Carry flag = 1, indicating carry has occurred on unsigned addition.
- Carry flag is 1 because the result crosses the boundary between 31 and 0.

Carry/Borrow Flag for Subtraction




```
    0 0 0 1 1       3
-   0 0 1 0 1   -   5
-------------   -----
    1 1 1 1 0      30
```

5-bit result

- Carry flag = 0, indicating borrow has occurred on unsigned subtraction.
- For subtraction, carry = NOT borrow.

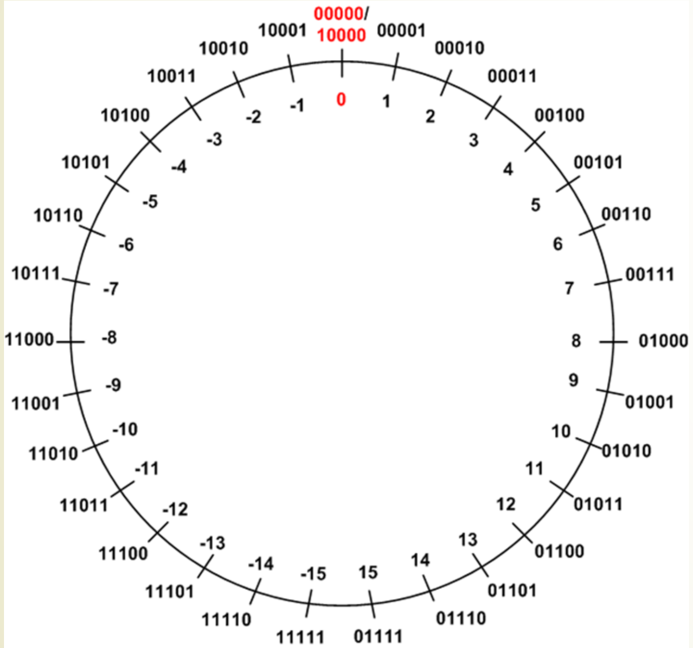
Examples

Action (5 bits, max 32)	Num1	Num2	Carry	Borrow	Comment
Add with carry	28	6	Yes	No	Sum is more than 32
Add but no Carry	10	2	No	No	Sum is less than 32
Subtract with Borrow	3	5	No	Yes	Result is Negative
Subtract with No Borrow	12	2	Yes	No	No borrow, hence carry flag will remain set

Signed Integer presentation

- Different ways to represent negative numbers
  - Signed magnitude
  - 1's complement ()
  - 2's complement ()

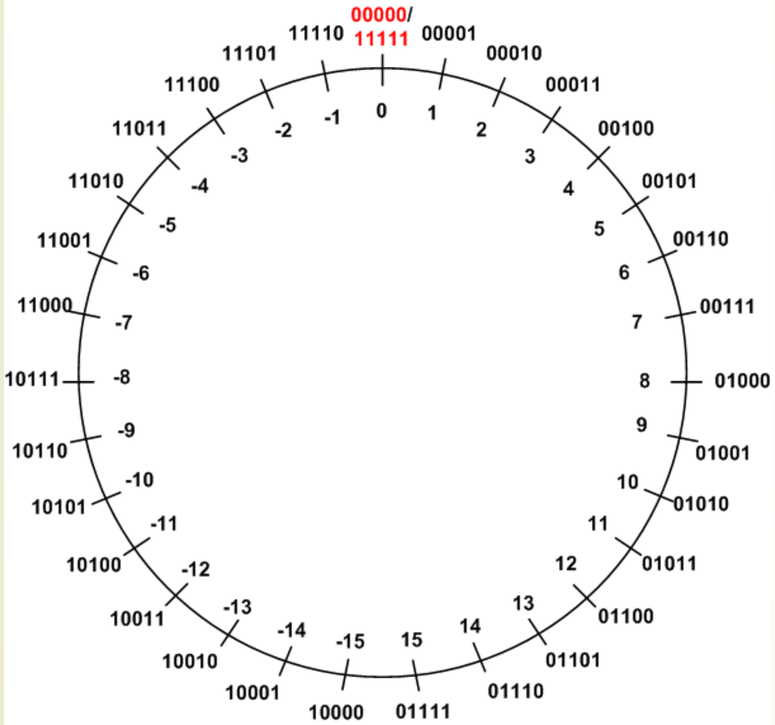
Signed Magnitude



- The most significant bit is the sign.
- The rest bits are magnitude.

- Example: in a 5-bit system
  - $+7_{10} = 00111_2$
  - $-7_{10} = 10111_2$
- Two ways to represent zero
  - $+0_{10} = 00000_2$
  - $-0_{10} = 10000_2$
- Not used in modern systems
  - Hardware complexity
  - Two zeros

1's complement



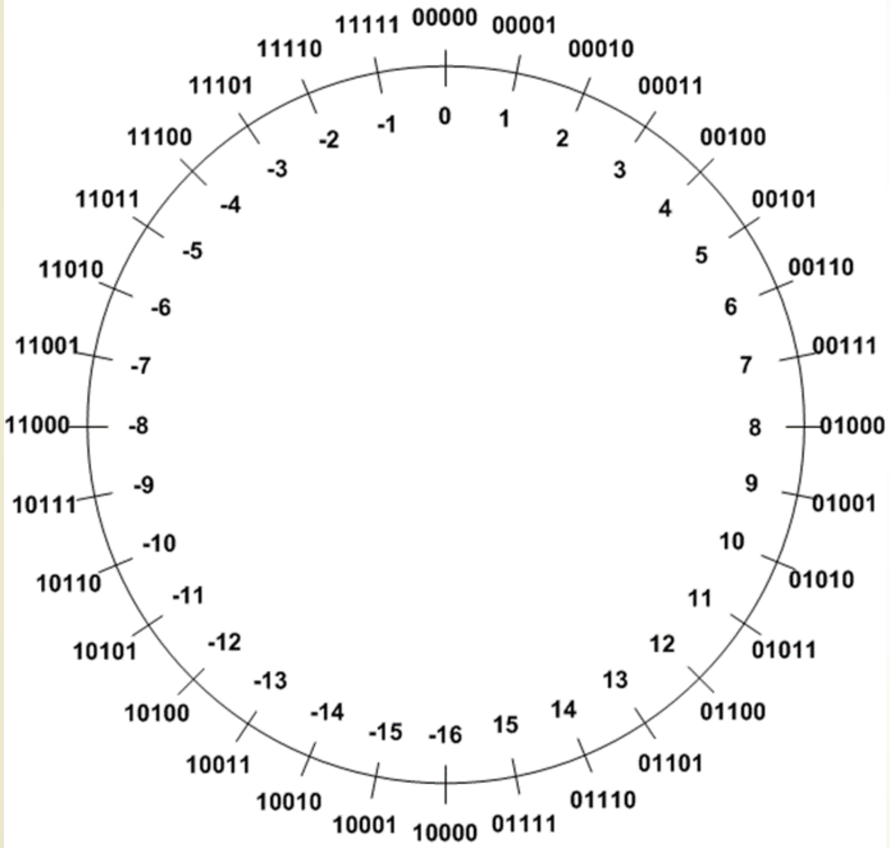
It is bitwise NOT of its positive counterpart.

Example: in a 5-bit system

$+7_{10} = 00111_2$

$-7_{10} = 11000_2$

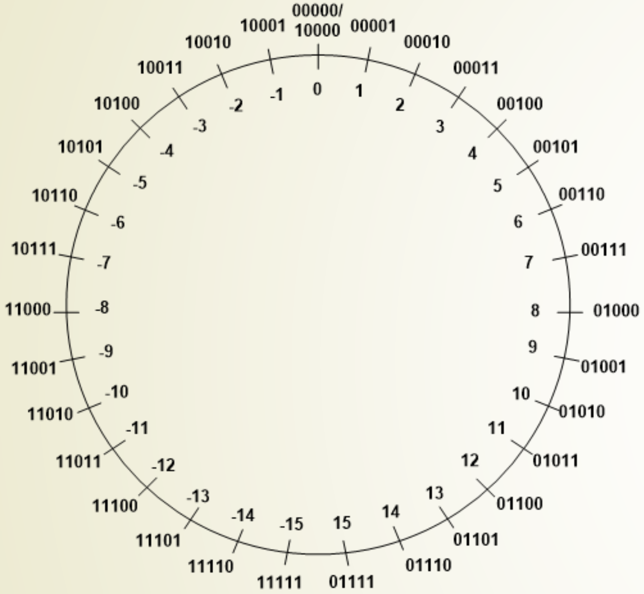
2’s complement



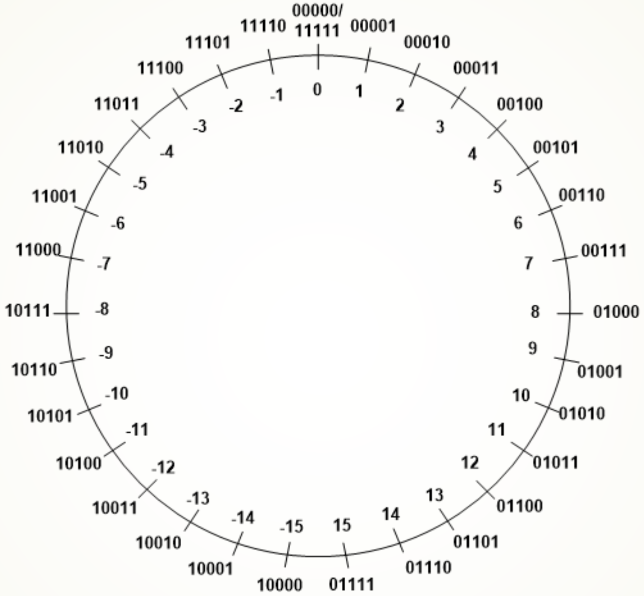
It is bitwise NOT of its positive counterpart plus one.

	Binary	Decimal
Original number	0b00101	5
Step 1: Invert every bit	0b11010	
Step 2: Add 1	+ 0b00001	
Two’s complement	0b11011	--5

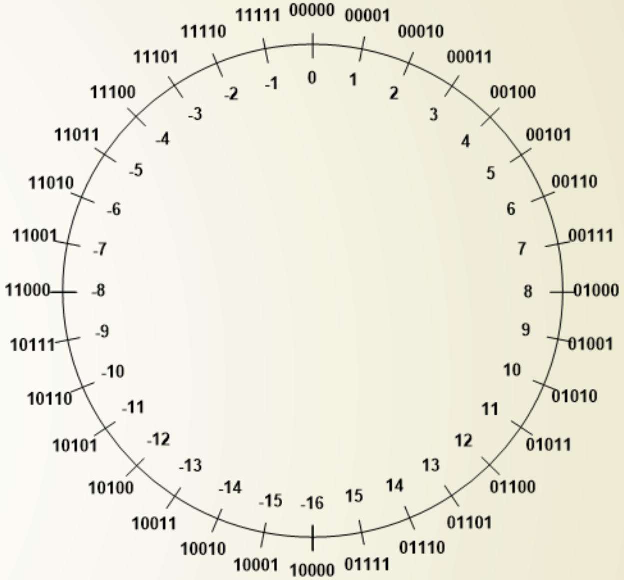
Comparison



Signed magnitude representation
0 = positive
1 = negative



One's complement representation
Negative = invert all bits of a positive

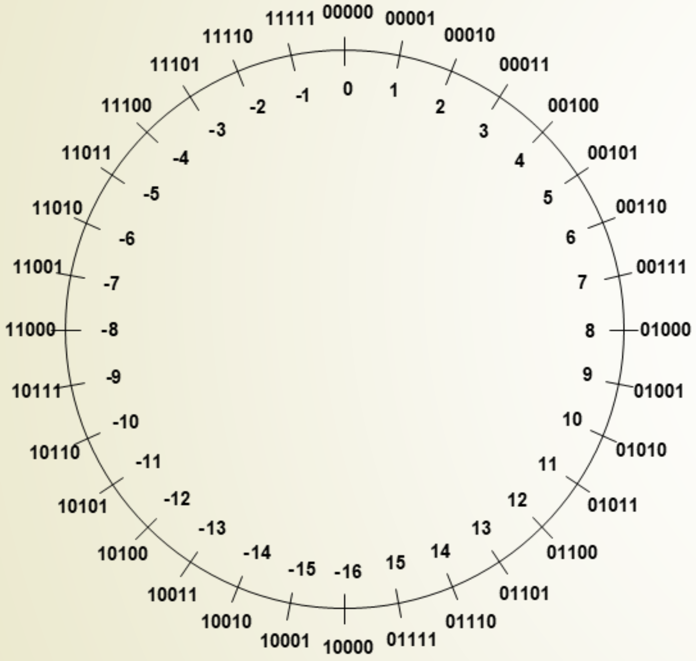


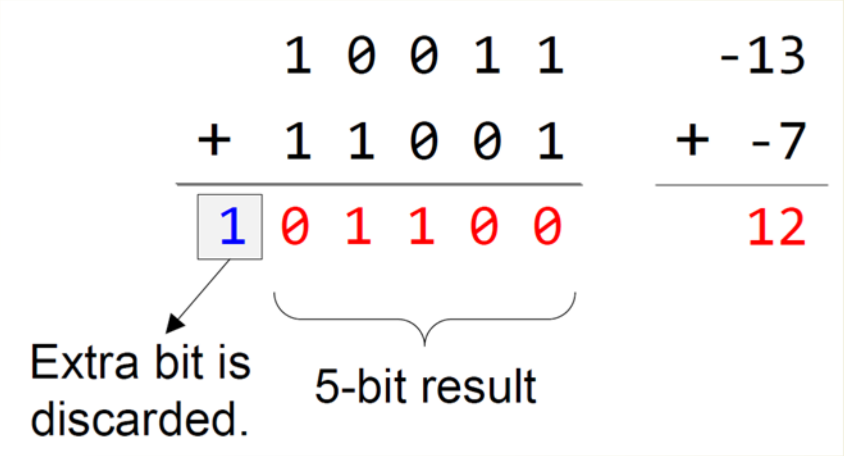
Two's Complement representation
TC = invert all bits, then plus 1

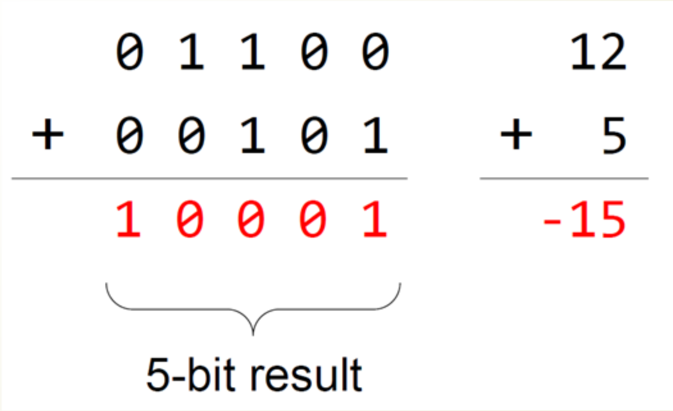
Overflow Flag for Signed Numbers

- When adding signed numbers represented in two's complement, overflow occurs only in two scenarios:
  - adding two positive numbers but getting a non-positive result, or
  - adding two negative numbers but yielding a non-negative result.

Overflow Flag







Subtracting signed numbers

- Similarly, when subtracting signed numbers, overflow occurs in two scenarios
  - subtracting a positive number from a negative number but getting a positive result, or
  - subtracting a negative number from a positive number but producing a negative result

- Overflow cannot occur when adding operands with different signs or when subtracting operands with the same signs.

Summary

- Processor detects overflow
- Indicates when an arithmetic overflow has occurred
  - Means signed 2's complement result would not fit in numbers
- For 5 bits range is: -16 to 15

Examples

Action (5 bits, max 32)	Num1	Num2	Overflow	Comment
Add two positive number	12	5	Yes	Result is negative (-15) as MSB is One.
Add two negative numbers	-13	-7	Yes	Sum is 12, positive
Subtract positive number from negative	-9	6	No	Result is negative
Subtract with no borrow	12	- 2	No	Result is positive

Signed or Unsigned

a = 0b1000011

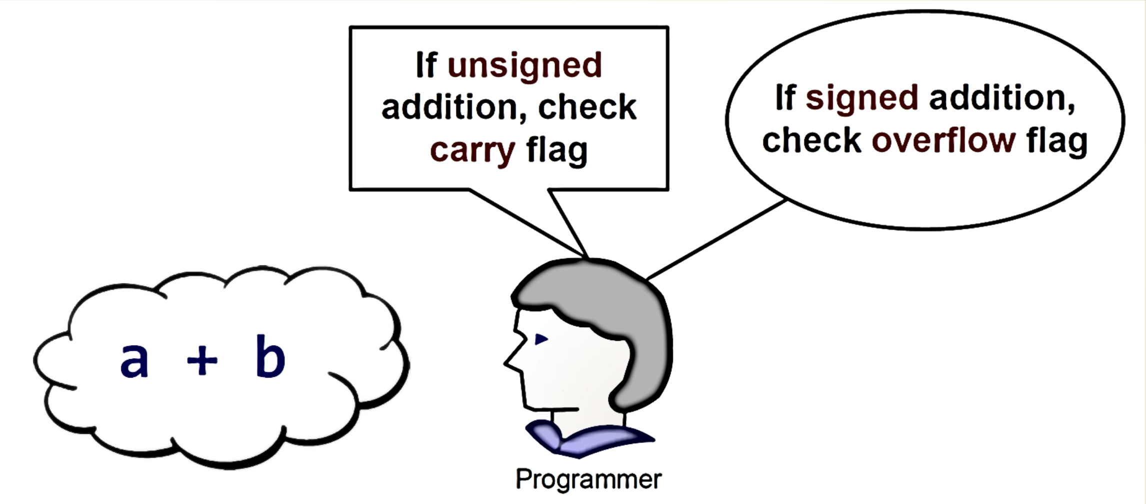
b = 0b1000011

$c = a + b$

- Processor does not know the answer at all
- So, hardware sets up both the carry flag and the overflow flag

Programmer Intention

- Whether the carry flag or the overflow flag should be used depends on the programmer's intention.



- When programming in high-level languages such as C, the compiler automatically chooses to use the carry or overflow flag based on how this integer is declared in source code ("int" or "unsigned int").

Example

a = 0b10000

b = 0b10000

c = *a* + *b*

int a,b;

c = a+b;

- Should we check carry or **overflow** ?

uint a,b;

C = a+b;

- Should we check **carry** or overflow ?

Next lesson

- Processor Architecture
- STM32L475V Processor
- Processor Registers
- Code execution
- Why ARM processor
- Processor packaging