Some hints for programming

Hints for Low Level Programming

- General considerations
- Planning
- Preparing for Low Level programming!
  - Mental Schemes
  - Instructions and objectives
- Programming tools

General considerations (1)

- Write the shortest possible program
  - Less resources, easier to read and maintain.
- Write programs easy to understand!
  - Use comments and appropriate names!
  - Document use of resources!
- Document as you work on the program!
- Write programs easy to modify
  - Different tasks may require small modification

General Considerations (2)

- Create your catalog of common tasks
  - Create codes for simple tasks (subroutines, macros, lists….)
  - Create procedures for special combinations of simple tasks (both in plain terms and in code terms)
  - Document alternatives
- Be aware of shortcuts!
  - Evaluate before use!
  - Do not sacrifice readability or flexibility with a shortcut

Programming with micros (1)

- Previous considerations are applicable!
  - Documentation is a must because of reduced number of instructions!
- Know your hardware system!!!
- Know your instruction set!
  - Even programming with C or other high level language may require assembly instructions!
  - Assembly is better for small programs

Programming with micros (2)

- Avoid using unimplemented bits of memory or registers
  - This limits portability!
- If possible use a single resource for a single purpose
- Program for lowest power consumption
  - Evaluate power modes
  - Cycles, etc.

Planning (1)

- Plan for what you want to do, not for what you want to program!
  - Understand and define your goals (and data) clearly
    - Do you have data? How should they be combined? \
    - What information I am looking for? ..
  - Try small examples by hand so you can visualize the goals and the process
  - If possible, try to visualize how can you determine if you fail or if you succeed
- Use plain language while planning, not computer language!
  - Plain language, plain language, plain language,…….

Planning (2)

- Bring old plans to the table!
  - Your new task may not be as new as you think it is!
  - Look for analogies or small modifications!
  - Look for reusable steps and sub tasks
- Top-down or bottom-up?
  - In general, top-down for complex tasks
  - Bottom-up may work fine for not so complex tasks
  - A combination may be a good fit!
  - Build your own library for bottom-up and top-down

Planning (3)

- Be sure you understand relationship between a task and a subtask or sequence of subtasks
  - What does the subtask deliver to the task?
  - What are the inputs for the subtask?
  - Does this subtask relate to other subtasks in the same tree?
- Look for similarities among different subtasks
  - Are "different" subtasks actually the same with different inputs?
  - Or they differ only in the output to be delivered?

Planning (4)

- Do not confuse goals with procedure
  - The procedure is the series of steps used to achieve goals.
  - A goal can be achieved following different procedures
  - Modification in a procedure may help achieving different goals.
- Do not confuse result with interpretation
  - The same result may have different interpretations, each one associated to a different goal

Planning (6): Main Advice

- **Be your toughest critic!!!!!!!**
  - o puede terminar queriendo lograr que el gallo ponga!

Preparing implementation

- Evaluate resources
  - Are they provided or do you have to select
  - If provided, adapt to them
  - What is the programming syntax (computation modelvs. language model)
- Do not start with the programming language until you have identified the tasks in your procedure and evaluated resources.
  - LOOK AGAIN FOR SIMILARITIES!
    - Different subtasks may actually use the same set of instructions
- Relate tasks with instructions or a sequence of instructions
  - Translating the task to a code may actually be a project by itself!

Building your program

- Test each sub task independently
  - Simulate inputs
  - Check desired outputs
- Add one sub task at a time and test
  - Easier to debug
  - Evaluate outputs
- Have examples with known results available

PREPARE YOURSELF TO WORK WITH LOW LEVEL PROGRAMMING

Work your mental scheme (1) - Are you sure you know?

1. Is it true that 5 + 4 = 0? YES, if we work in modular 9 arithmetic

2. In the **real, physical world,** is it possible to add two positive numbers and still get 0 as result?

YES, a car odometer for example. If it has four digits and is showing 9892 miles, it will show 0000 if we run (add) the car 108 miles.
(The odometer works the addition in modular 10000 arithmetic)

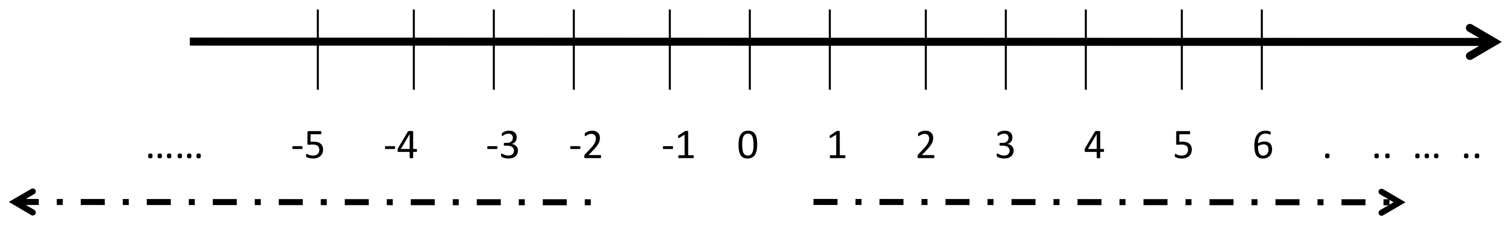
3. An example of an every-day device we use where A-A is never 0 is a … Clock!

4. Are you aware that you will never be able to actually draw a straight line on the ground?

Because Earth is not flat ☺

Work your mental schemes (2) - Change Paradigm

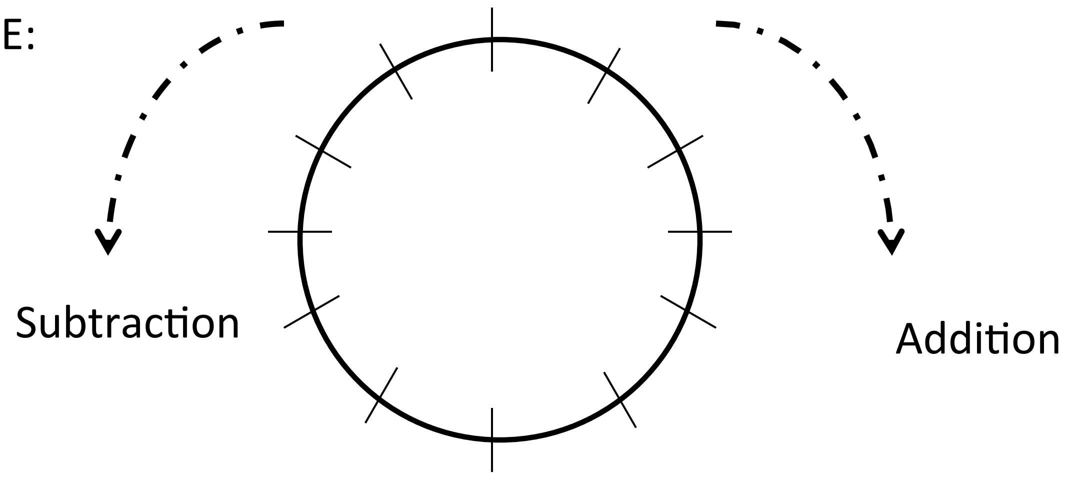
Normal every day scheme: NUMERIC LINE



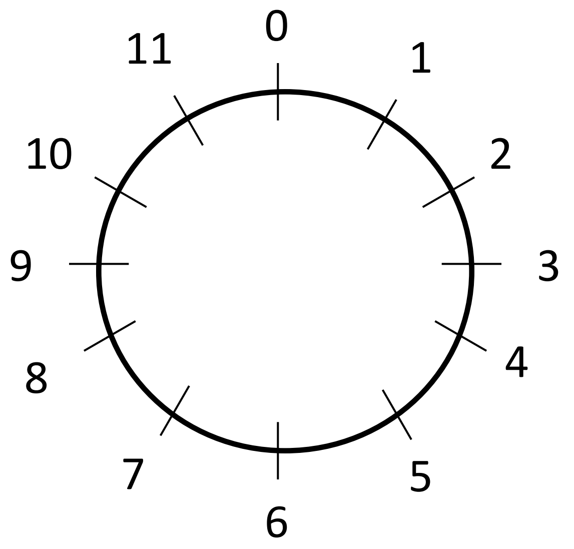
Subtraction of positive numbers
(or addtion of negative numbers)

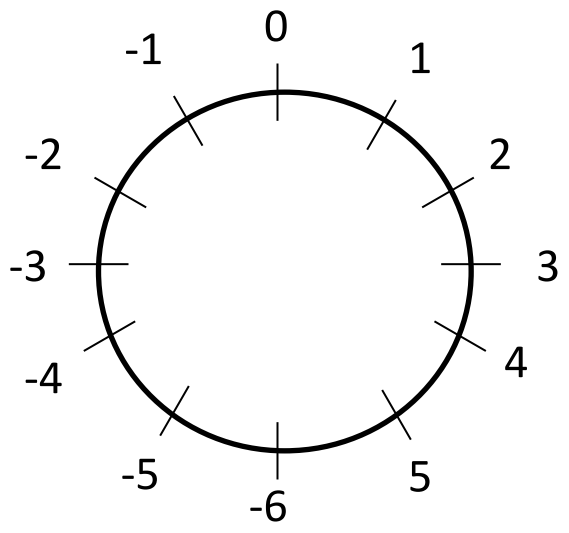
Addition of positive numbers
(or subtraction of negative numbers)

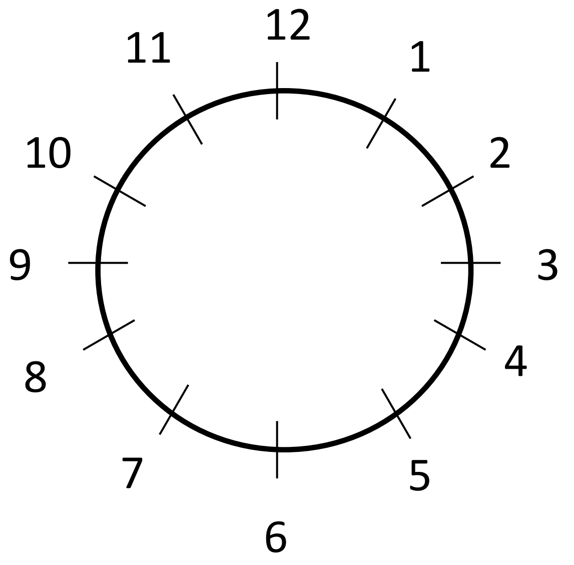
NUMERIC CIRCLE:



Work your mental schemes (3) - Representations -







<u>Differentiate</u> your scheme from the actual operation:
Addition is to go in clockwise direction
Subtraction to go in counter clock direction

5+2 = 7 or 5+2 = -5;   2-2 = 0 or 2-2=12

Work your mental schemes (4) - Analogies -

Use your every day experience, your previous knowledge, etc. to build structures, methods, and so on.

Example) **Experience:** With one digit numbers from 0 to 9; two digits for 0 to 10^2-1; three digits for 0 to 10^3-1.
Terminology associated: least significant digit, most significant digit, carry when sums, borrow when subtraction….

Analogy: With one 8-bit register, 0 to 255; with two 8-byte registers for 0 to 256^2-1; three 8-byte registers for 0 to 256^3-1. Define your least Significant register, your most significant register. Use addition with Carry and subtraction with borrow when adding big numbers ….

Mental Schemes (4)
- Understand operations -

A piece of hardware (adder) does something ……
How I use it and how I interpret results depend on my objectives and mental scheme.

-- An operation is just an operation

-- A result is just a result

- It is the user who makes the difference

Instructions: Know them! (1)

- Understand what is the process involved
  - What they do is hardware dependent and cannot be changed! (This indicate possibilities and limitations)
  - Check for flags!
  - Chech for “hidden” effects.
- Evaluate special cases!
  - Example: testing one bit versus testing a set of bits
  - Example: adding special numbers, specific additions

Instructions: Know them (2) - Interpret operations -

Work with instructions and play on interpretations to build your mental schemes.

$$\begin{tikzcd} & \text{2A} \\ \text{A+A means ...} \arrow[ur] \arrow[r] \arrow[dr] & \text{Shift bits to left, msb to carry, and lsb=0} \\ & \text{Use Carry Flag to determine value of most significant bit.} \end{tikzcd}$$

Let A= b7 b6 b5 b4 b3 b2 b1 b0. What can I say about A + 11110000b?

a) If b4=1, the operation clears b4 and produces a Carry Flag = 1
b) If b4 = 0, then CarryFlag=1 if and only if there is a 1 in the set b7 b6 b5.

Assignment: Can you design an algorithm to clear b4 using addition and Testing the carry flag? (you can copy and retrieve contents too)

Instructions: Know them (3)
- Learn new things -

- Learn from examples how they are used
  – Check for comments and documentation!
  – Build your own catalog with new insights
- Relate tasks with instructions
  – Instructions and sequence of instructions
  – Look for different sequences
- Start with small steps

Tools

- Use debugger to check your understanding
- Check with different assemblers
  - Line assembling http://mspgcc.sourceforge.net/assemble.html
- When in C, look also at the assembly translation.
  - Different C compilers may generate different assembly lines for the same C command!
  - Try to optimize the assembly version of a C program