Branch

- A branch is an instruction in a computer program that can cause a computer to begin executing a different instruction sequence and thus deviate from its default behavior of executing instructions in order.
- Branch (or branching, branched) may also refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction.
- A branch instruction can be either an *unconditional branch*, which always results in branching, or a *conditional branch*, which may or may not cause branching, depending on some condition.
- Branch instructions are used to implement control flow in program loops and conditionals (i.e., executing a particular sequence of instructions only if certain conditions are satisfied).
- Example: 8085; JMP (*unconditional branch, conditional branch*), PCHL, if, else etc.

Subroutine

- In computer programming, a subroutine is a sequence of program instructions that perform a specific task, packaged as a unit.
- This unit can then be used in programs wherever that particular task should be performed.
- Subprograms may be defined within programs, or separately in libraries that can be used by multiple programs.
- In different programming languages, a subroutine may be called a procedure, a function, a routine, a method, or a subprogram. The generic term callable unit is sometimes used.
- The name subprogram suggests a subroutine behaves in much the same way as a computer program that is used as one step in a larger program or another subprogram.
- A subroutine is often coded so that it can be started (called) several times and from several places during one execution of the program, including from other subroutines, and then branch back (return) to the next instruction after the call, once the subroutine's task is done.
- Example: Function CALL, RET.

Advantages

The advantages of breaking a program into subroutines include:

- Decomposing a complex programming task into simpler steps: this is one of the two main tools of structured programming, along with data structures

- Reducing duplicate code within a program
- Enabling reuse of code across multiple programs
- Dividing a large programming task among various programmers, or various stages of a project
- Hiding implementation details from users of the subroutine
- Improving traceability (i.e. most languages offer ways to obtain the call trace which includes the names of the involved subroutines and perhaps even more information such as file names and line numbers); by not decomposing the code into subroutines, debugging would be impaired severely

Disadvantages

- Invoking a subroutine (versus using in-line code) imposes some computational overhead in the call mechanism.
- A subroutine typically requires standard housekeeping code – both at entry to, and exit from, the function (function prologue and epilogue – usually saving general purpose registers and return address as a minimum).