Development tools and software support

Ege Üniversitesi Müh. Fakültesi

Bilgisayar Mühendisliği Bölümü

Gömülü ve Gerçek Zamanlı Sistemler

2013

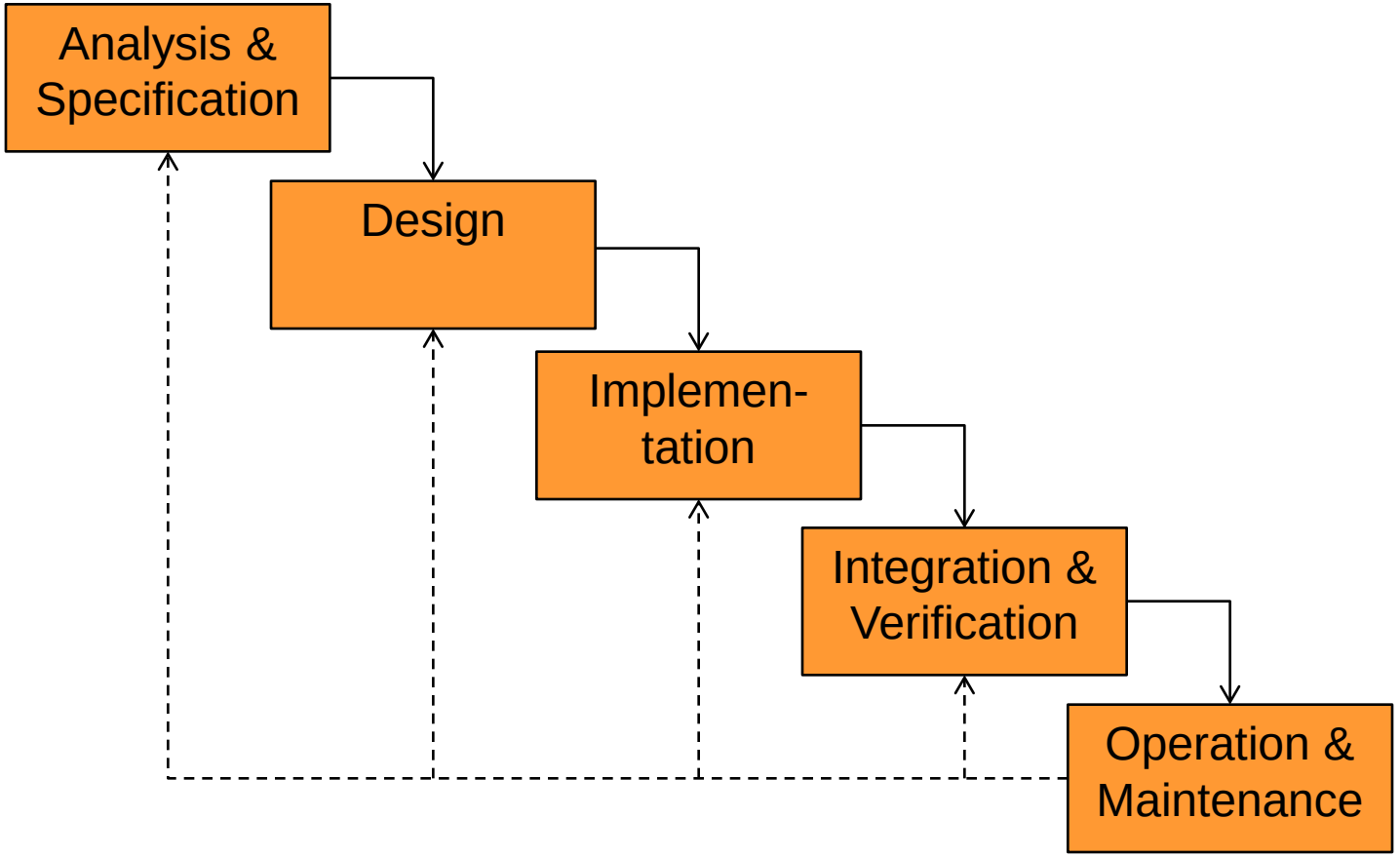
Yar. Doç. Dr. Mustafa Engin

Introduction

- C was introduced in 1972 by Dennis Richie
- Designed as programming language for UNIX
- Two important dialects:
  - K&R:
    - Kernighan & Richie style
    - Language launch
    - Obsolete
  - ANSI C:
    - Current and most significant dialect
- Characteristics
  - High level language
    - Abstract processor view
  - High availability
  - Close to hardware
  - Control and data structures

Development Process I

- Water fall model
  - A simple model



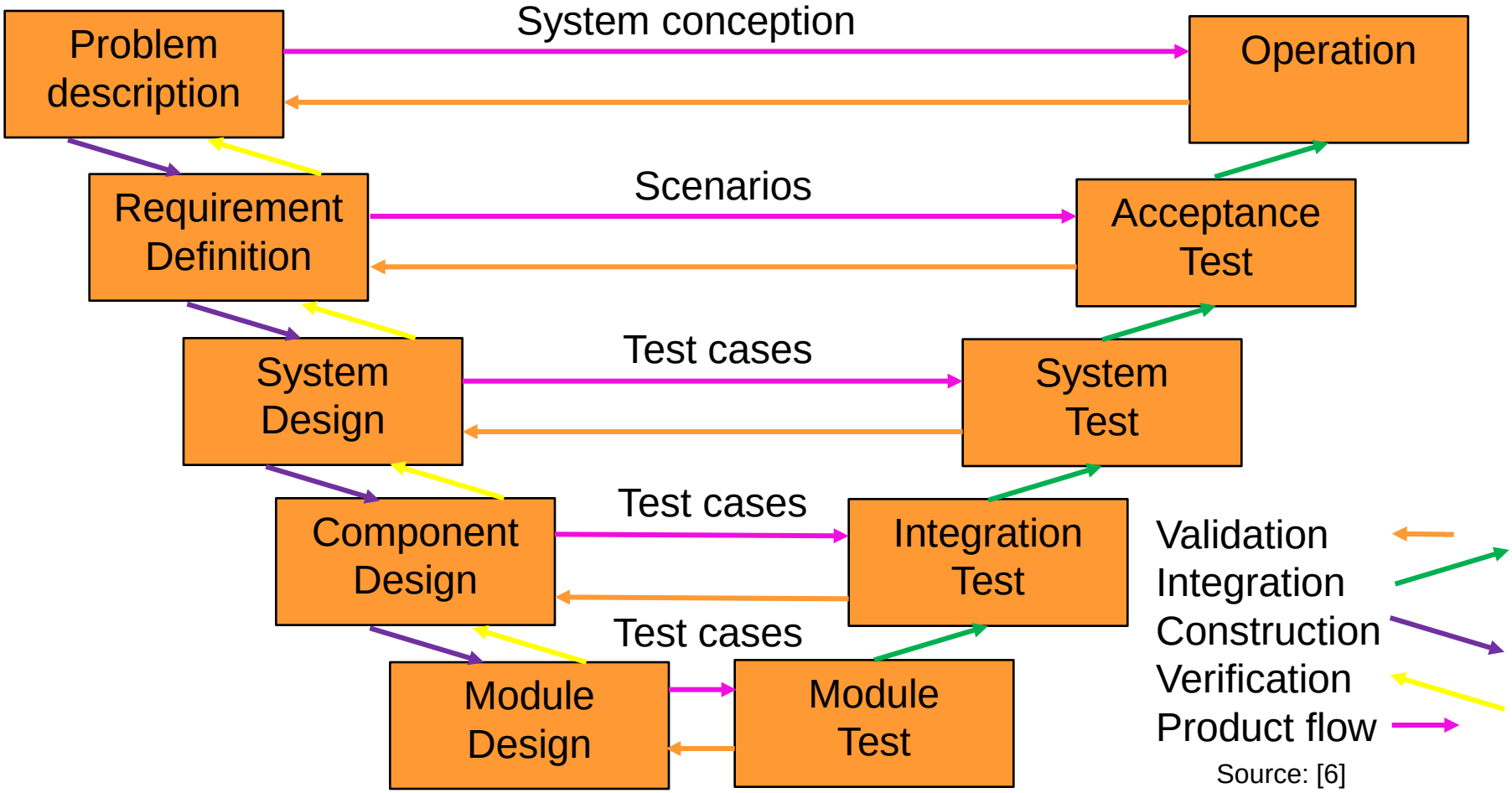
Development Process II

- Analysis & Specification
  - Problem analysis with constraints
  - Requirements Specification
- Design
  - System architecture
  - Hardware/Software Partitioning
- Implementation
  - Module realization
  - Subsystem (module) tests
- Integration & Verification
  - Module Composition
  - System tests
- Operation & Maintenance
  - System launch
  - Functional extension

Main focus

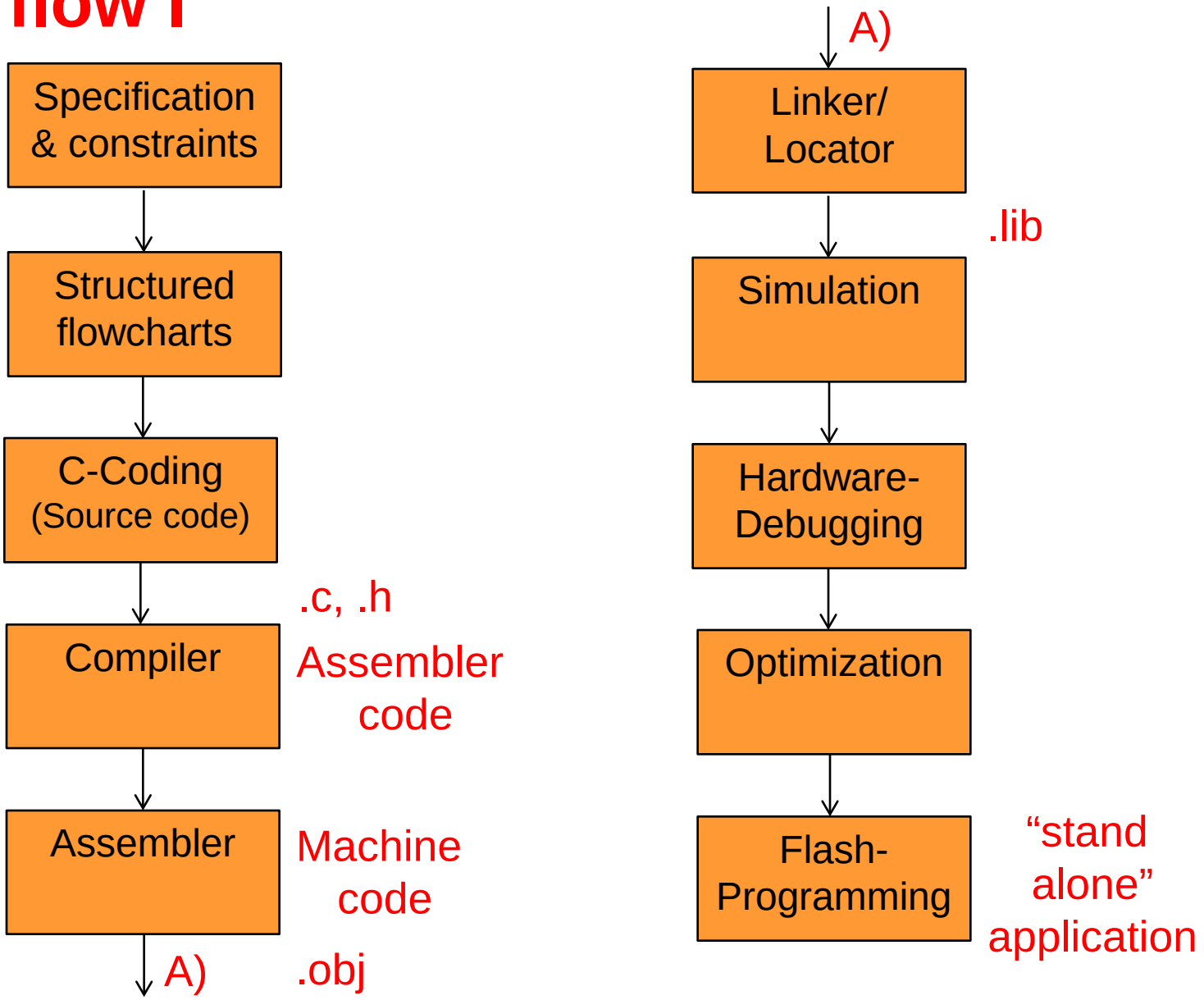
Development Process III: V-Model

- V-Model: Approach model - in German “Vorgehens-Modell”



Source: [6]

Design flow I



Design flow II

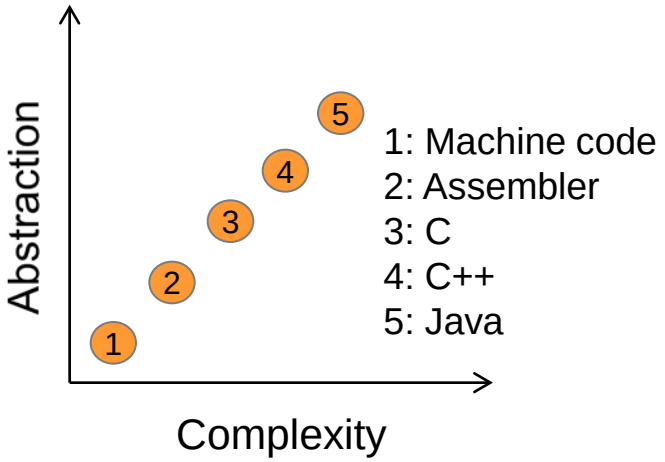
- Compiler
- Translation of high level instructions to Assembler or Machine code
  - Machine code (Native code): sequence of CPU instructions (binary data)
- Assembler
  - Mnemonics: 1:1 representation of machine code
- Linker/Locator
  - Module binding and attaching physical address
- Optimization
  - Optimizing tool for speed or area
- Simulator
  - Source code testing on host PC
- Debugging
  - Source code testing on target
  - Levels: source level, machine code

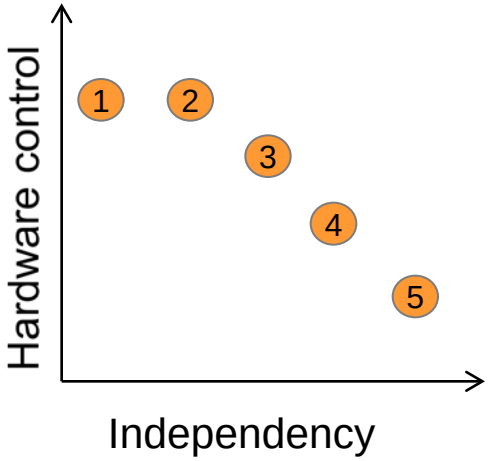
Design flow III

- IDE – Integrated Development Environment
  - Complete development frame including translation and test tools
- Files:
  - Source code: .c:
    - Function implementation
    - Definition global variables
  - Header files: .h
    - Declarations: functions, data types
  - .obj: object file
  - .lib: library file

C versus Assembler

- C
  - Pros
    - Higher abstraction
    - Portable
    - Better readable
    - Early syntax check
  - Cons
    - Less performance than Assembler
- Assembler
  - Pros
    - Closer hardware control
    - Commonly faster and less resources
  - Cons
    - More prone to errors. Extensive testing required.
- ⇨ Focus: C language + StellarisWare®





Embedded C

- C language is general purpose
  - Developing programs for data processing or embedded systems
  - Embedded software often must have immediate hardware access to internal peripherals
- For the different requirements of the applications
  - An add on or a reduction of the general C language is necessary
  - This is called “Embedded C”
- This has consequences to the main advantage of a high level language: the Portability
- Microcontroller-specific categories:
  - Memory: program and data memory
  - Peripherals: Inputs/Outputs, timers etc.
  - Interrupts: sources, priority

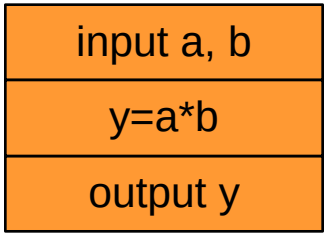
Design elements

- Pseudo code
- Flowchart
- Structured flowcharts (Nassi Shneiderman diagrams)

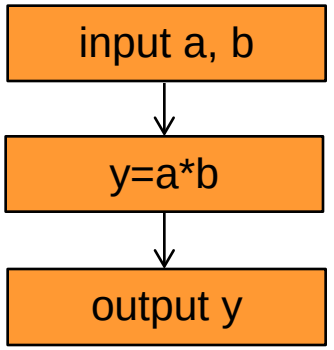
Pseudo code

```
y=a*b
```

Structured flowchart



Flowchart



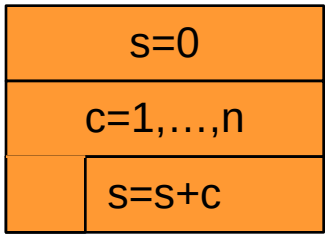
Source: [3]

Design elements: loops

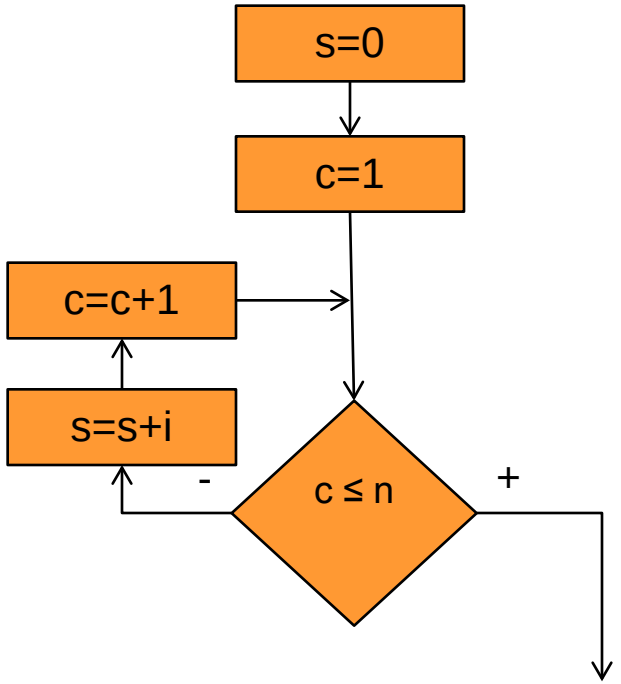
Pseudo code

```
s=0
for c=1,…,n
  s=s+c
end
```

Structured flowchart



Flowchart



+ : true
- : false

Source: [3]

Design elements: loops

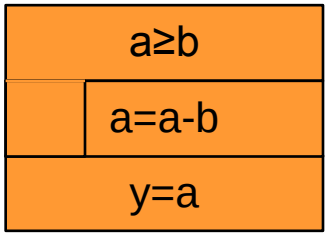
Pseudo code

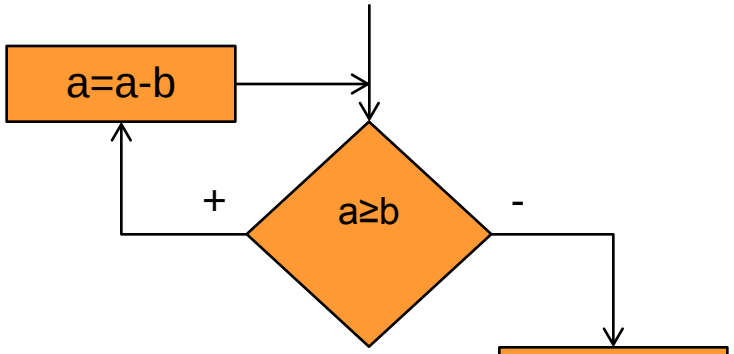
Structured flowchart

Flowchart

Example: y mod b

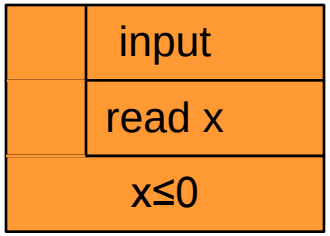
```
while a≥b
  a=a-b
end
y=a
```

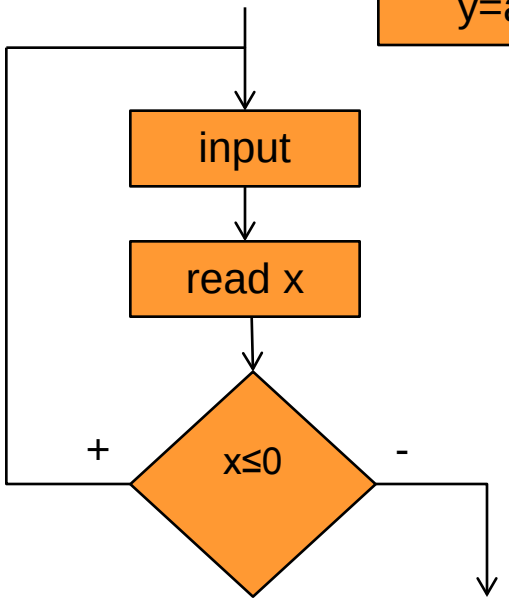




Example: input x>0

```
do
  input
  read x
while x≤0
```





Alternative: repeat…until

Source: [3]

Design elements: conditions

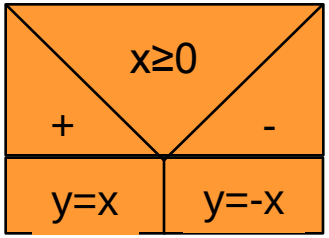
Pseudo code

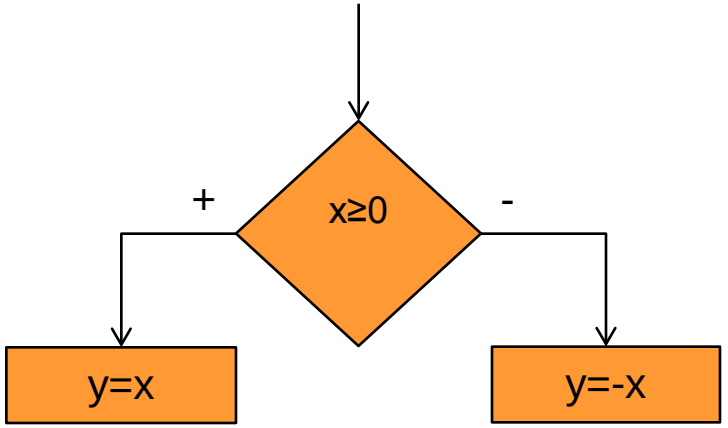
Structured flowchart

Flowchart

Example: y= x, if x≥0
-x, otherwise

```
if x≥0
  y=x
else
  y=-x
end
```

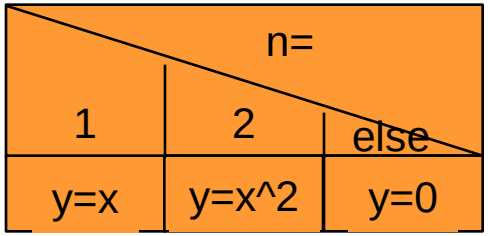


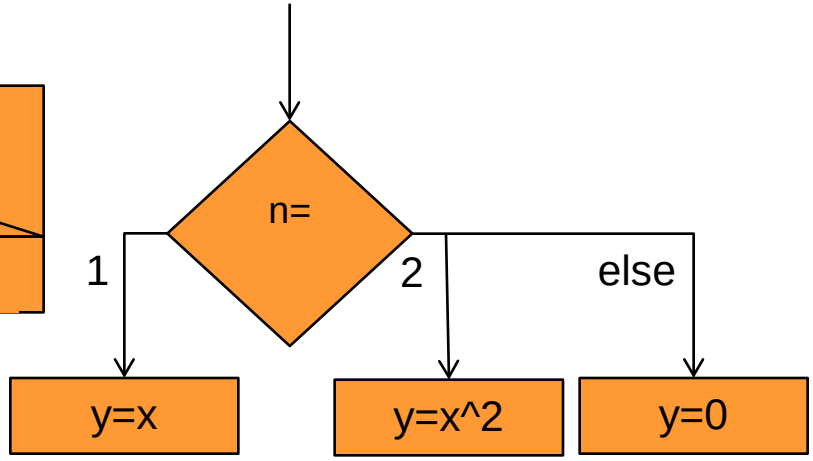


Example: y= x, if n=1
x^2, if n=2,
0, otherwise

```
switch n=
  case 1: y=x
  case 2: y^2
  else    y=0
```

Alternative:
if…else





Source: [3]

Pre-processor

- C uses an upstream pre-processor
- Characteristics
  - C independent
  - Starting with #
  - Instruction ends without semicolon
  - Erases comments
- simple macros
  - #include <name.h> (header file)
    - Default path for name.h
  - #define
    - Example: #define GPIO_PORTG_DATA_BITS_R ((volatile unsigned long *) 0x40026000)
  - #pragma
    - Pragma directives control the Compiler for implementation specific operations.

Data types

- Integer
  - Char: ASCII, 8 bits
  - Short: 8 bits
  - Int: 16 bits
  - Long: 32 bits
  - Note: compiler/architecture specific identifiers
- Floating point types
  - Float: 32 bits
  - Double: 64 bits
  - Long double: 80 bits
- Arrays:
  - Example: int adc_val = {1,2,3,4,5};

Operations

Operator	Description	Example
+, -, *, /, %	addition, subtraction multiplication, division, Modulo	int y=a+b;
+, -	Sign	int y=-a;
++, --	increment, decrement	int y=a++;
&, |	bitwise and, or	int y= a & 0x01;
~	bit negation	int y=~a;
^	bit exclusive or	int y=a^0x01;
<<, >>	bit shift left, right	int y=a<<4;
&&, ||	logic or, and	int y= a || 0x01;
!	logic not	int y=(!(a==b))
==, !=	equal, not equal	if (y==0)
<, >, <=, >=	comparison	if (y<0)
=, +=, &=, …		int y=+5;
*, &	pointer, address	int y=*a;

Instructions

Instructions	Description	Example

if (cond)
instr
else
instr
	conditional statements	if (a==0)
		y=0;
		else
		y=1;

switch (n) {
case cond1: instr1;
case cond2: instr2;
default: instr3; }
	switch statements
	multiway branches
		switch (n) {
		case 1: y=x;
		case 2: y=x^2;
		default: y=0; }

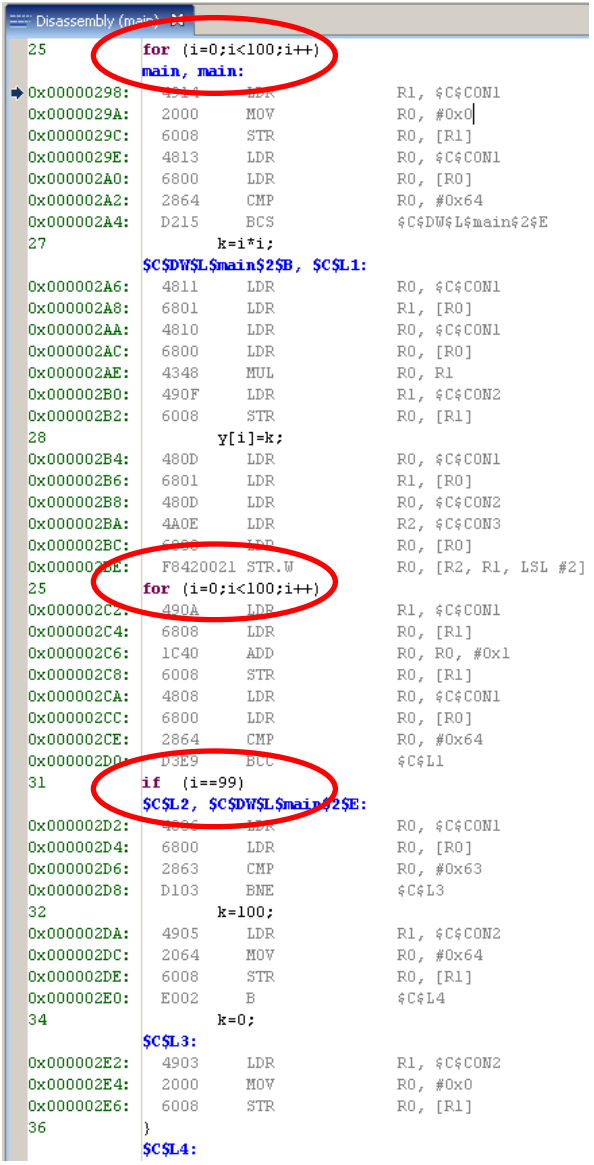
for (init; cond; incr) instr.	number of iteration is known	for (i=0;i<5;i++) {y=y+1;}
while (cond) instr	pre-test loop	while (a>0) y++;

do
intr
while (cond)
	post-test loop	do
		y++;
		while (a<0)

{instr1; instr2; instr3; …}	block	{a++; b--;}

Example: Disassembly

- Basic instructions
  - For …
  - If … else …



Functions

- function
  - “Stand alone” block
  - Called by other blocks
  - Getting parameters (void: no parameter (obsolete))
  - Deliver max. one result (void: no result)
- master function: main()
  - Output
    - Void main(): no return
    - Int main(): integer value back (via return)
  - Input
    - Main (int argc): argc – number input parameter plus one
    - Main (int argc, char *argv[]): argv – pointer array of parameter

C Program Frame

```
/* Simple C Frame
        Project:
        Description
        Date:
        Revision:
        …
*/
#include <xxx.h>
int main (int argc, char* argv[] ) {
            instructions;
            {} // blocks

            return 0;
            }
```

Data Storage

- Variables characteristics
  - Local variables
    - Automatic variable generation by block entrance and deletion after leaving
    - Local: stack
      - Example:
        void func() {
        int localvar = 10;}
    - Local: register
      - Example:
        void func() {
        register int localvar = 10;}
  - Static and global variables
    - Static: defined inside a block, the value is conserved
    - Global: definition and initialisation outside functions
      - Extern: indicates that the global variable is still defined in a other file
  - Const: constant value, cannot be changed
  - Volatile: prevents optimisation

Questions and Exercises

1. Explain the terms Compiler, Assembler, Linker.
2. Design a water fall model.
3. Design a V model.
4. Draw a structured flowchart for an arithmetic mean with n=5.
5. Explain the design flow for the arithmetic mean.