DS 221
Introduction to Scalable Systems
3:1

Classes: Tuesday, Thursday 11:30-1:00

Matthew Jacob

mjt@iisc.ac.in

Syllabus (MJT)

Architecture: computer organization, single-core optimizations including exploiting cache hierarchy and vectorization, parallel architectures including multi-core, shared memory, distributed memory and GPU architectures

Algorithms and Data Structures: algorithmic analysis, overview of trees and graphs, algorithmic strategies, concurrent data structures

Parallelization Principles: motivation, challenges, metrics, parallelization steps, data distribution, PRAM model

Parallel Programming Models and Languages: OpenMP, MPI, CUDA;

Distributed Computing: Commodity cluster and cloud computing; Distributed Programming: MapReduce/Hadoop model.

Reference (MJT)

Bryant, O'Hallaron. Computer Systems – A Programmer's Perspective, Pearson Education Limited 2016, 3rd Global Edition

Culler, Singh. Parallel Computing Architecture. A Hardware/Software Approach

Quinn. Parallel Computing. Theory and Practice

Sahni. Data Structures, Algorithms, and Applications in C++

Grama, Gupta, Karypis, Kumar. Introduction to Parallel Computing

Pacheco. An Introduction to Parallel Programming

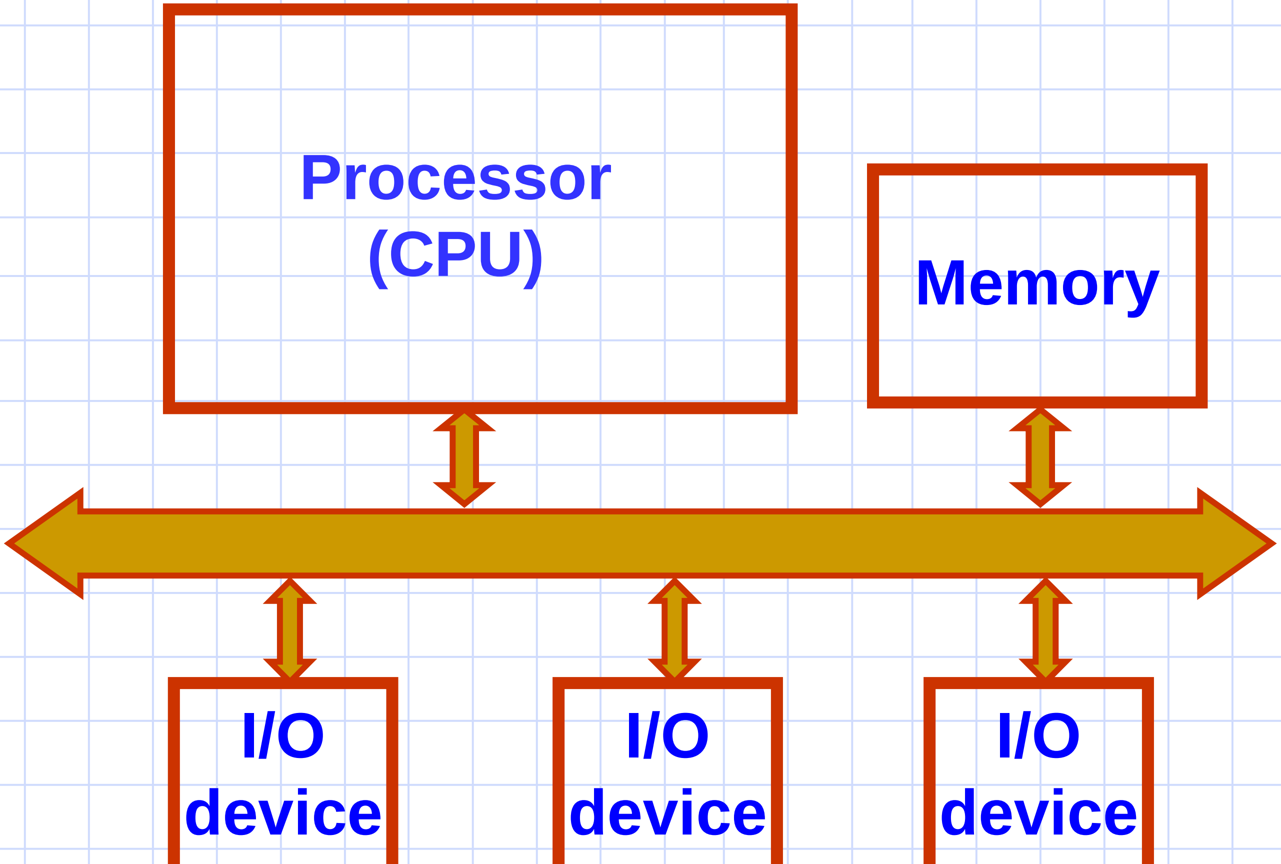
Hwang, Dongarra, Fox. Distributed and Cloud Computing: From Parallel Processing to the Internet of Things

Lin, Dyer. Data-Intensive Text Processing with MapReduce

Course Work (MJT)

Quiz (in class)	10 marks
Assignments (3)	10 marks
Final exam	5 marks

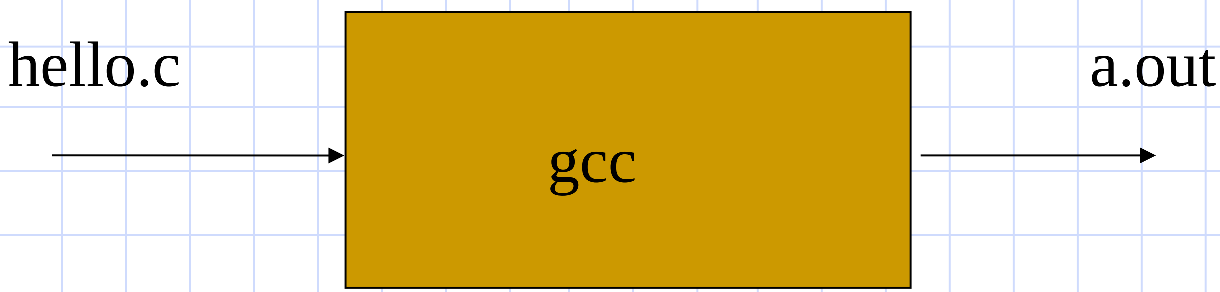
Basic Computer Organization



What is a Computer Program?

- Description of algorithms and data structures
- Could be done in any language, even a natural language like English
- Programming language: standard notation for writing programs
- Examples: C, Java, assembly language, machine language
- Need for program translators
  - Example: gcc

% gcc hello.c



`source file'

`object file'

EXECUTABLE

Contents of a.out file?

- Program “code” (machine instructions)
- Data values
- Other information that is required for
    - execution
    - relocation
    - debugging
- When you execute a program, its instructions and data are brought into the Memory of the computer system

Program Data: Different Kinds

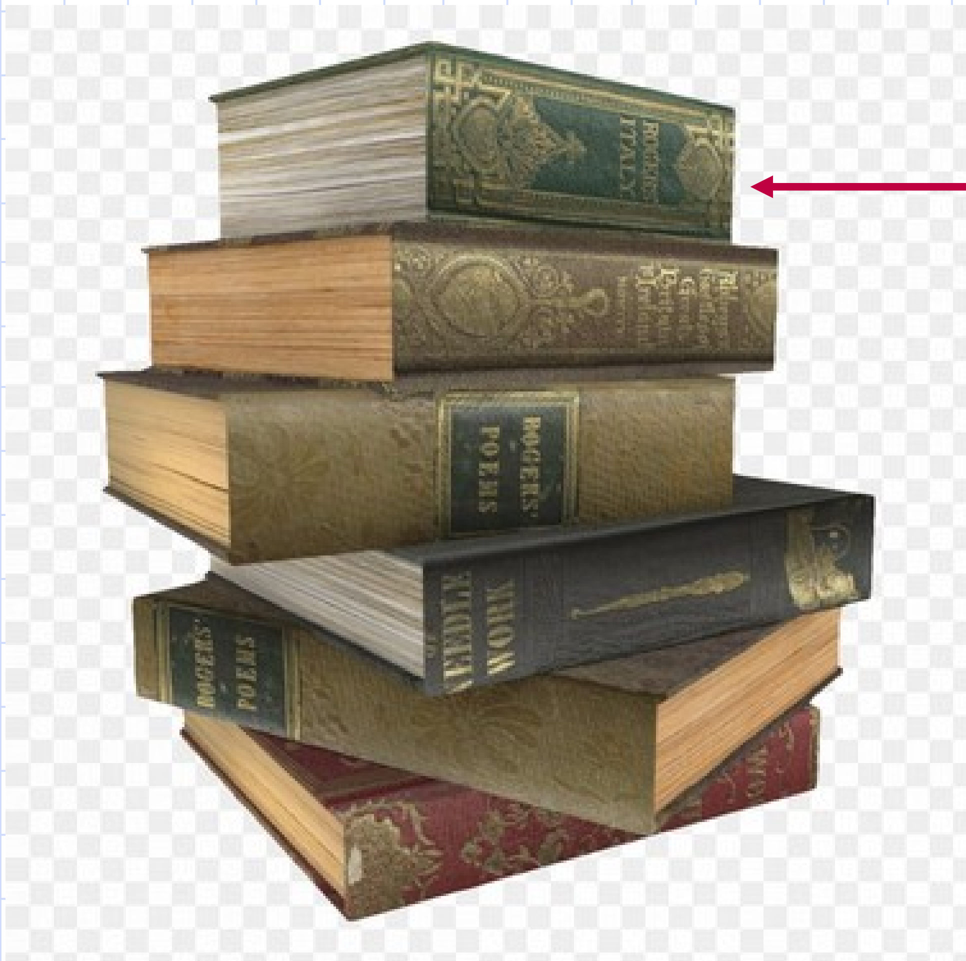
- Constant vs variable
- Basic vs structured
- Of different types
  - Character
  - Integer (unsigned, signed)
  - Real
  - Others (boolean, complex, ...)

Of Different Lifetimes

1. Lifetime = Execution time of program
   - Initialized/uninitialized data
   - Must be indicated in executable file
   - The space for all of this data can be assigned when program execution starts (Static Allocation)
2. Lifetime = Time between explicit creation of data & explicit deletion of data
   - Dynamic memory allocation
   - malloc, free
   - The space for this data is managed dynamically when the malloc/free is executed (“Heap” allocation)
3. Lifetime = During execution of a function (i.e., time between function call and return)
   - Local variables, parameters of the function
   - The space for this data is assigned when the function is called and reclaimed on return from the function (Stack allocation)

Stack allocated: Function Local Variables

Stack? Think of a stack of books

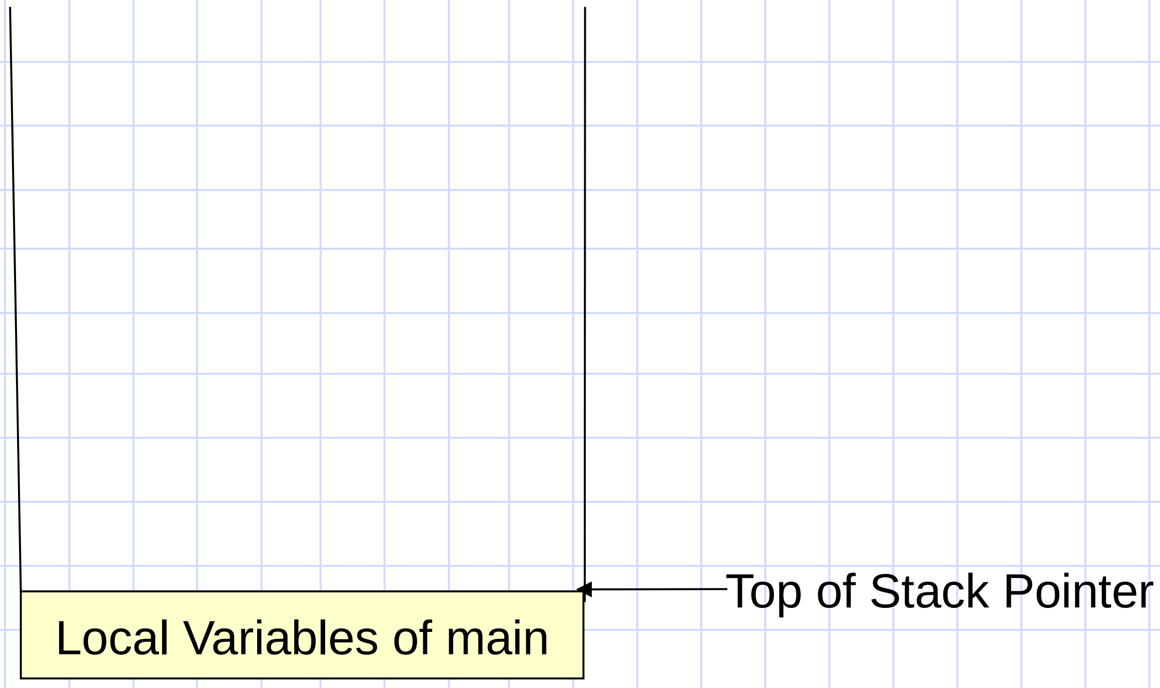


“Top of stack”

https://favpng.com/png_view/a-stack-of-old-books-book-stack-gratis-png/DZ2sxEJa

Stack allocated: Function Local Variables

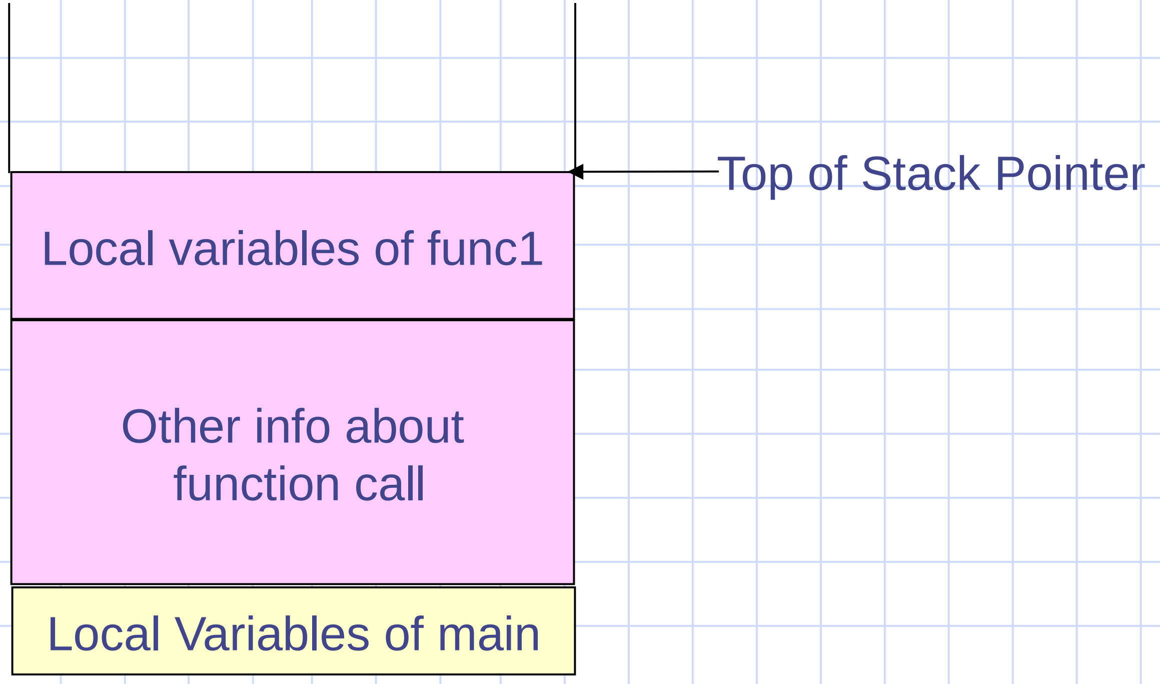
When the program starts executing



What if main( ) then calls function func1( )?

Stack allocated: Function Local Variables.

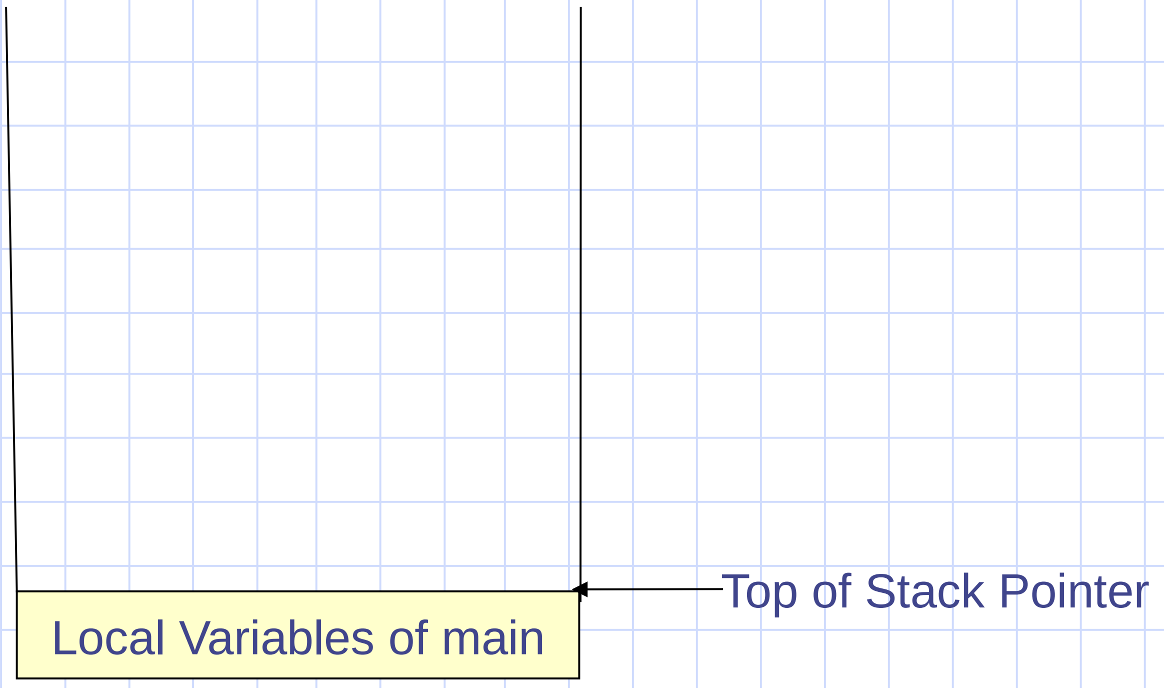
While executing in function func()



What happens on return from the call to func1()?

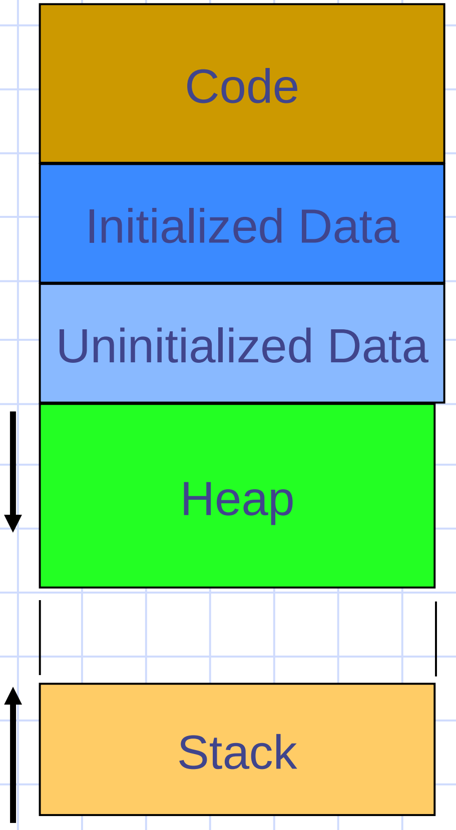
Stack allocated: Function Local Variables..

Executing in main( ) once again



Program and Data

- Code (machine language program)
- Data (initialized and uninitialized)
- Code and Data don't change in size while the program is executing
- Heap (for dynamically allocated data)
- Stack (for function local variables)
- Heap and Stack change in size as program executes



How is Data Represented?

- On a digital computer
- Binary
  - Bit (Notation: b)
  - Byte (Notation: B)
  - Other notation: K, M, G, T, etc
    - K: 2^{10} , M: 2^{20} , G: 2^{30}, etc
- Character data representation: ASCII code
  - 8 bit code
    - 7 bit code with an added parity bit

Integer Data

- Signed vs Unsigned integer
- Representing a signed integer
  - 2s complement representation

The *n* bit quantity

least significant bit

$$x_{n-1}\,x_{n-2}\ldots x_2\,x_1\,x_0$$

represents the signed integer value

$$-x_{n-1}2^{n-1}+\sum_{i=0}^{n-2} x_i 2^i$$

Example: 2s complement

- The signed integer -14_{10} (decimal) is represented as
  - 10010 in 5 bits (i.e., -16 + 2)
  - 110010 in 6 bits (i.e., -32 + 16 + 2)
  - 111...1110010 in 32 bits

Aside: Hexadecimal (base 16)

- Digits 0 1 2 3 4 5 6 7 8 9 A B C D E F

  0000 0001 0010 ... 1101 1110 1111

- Binary sequences can be written more compactly in hexadecimal

1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Example: 2s complement

- The signed integer -14_{10} (decimal) is represented as

  10010 in 5 bits (-16 + 2)

  110010 in 6 bits (-32 + 16 + 2)

  111...1110010 in 32 bits

  1111 1111 1111 ... 0010

  FFFFFFF2

  Usually written as 0xFFFFFFF2