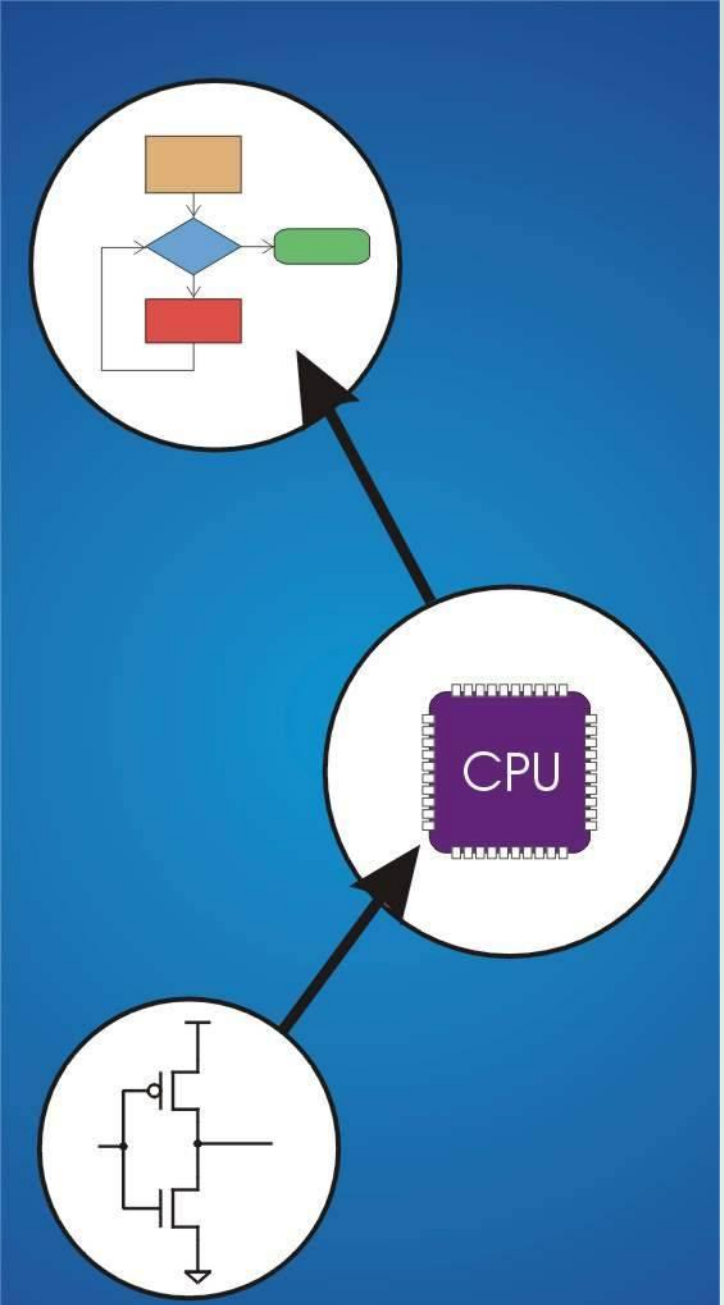


Perspective
Memory Model for Program Execution

Slides by C. Wilcox, Y. Malaiya

Colorado State University

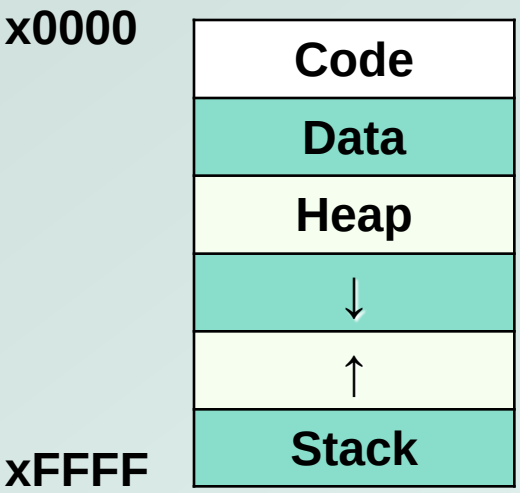
Problem

How do we allocate memory during the execution of a program written in C?

- Programs need memory for code and data such as instructions, global and local variables, etc.
- Need support for a function calling another function, recursion
- Some memory must be allocated dynamically, size and type is unknown at compile time.

Real Solution: Execution Stack

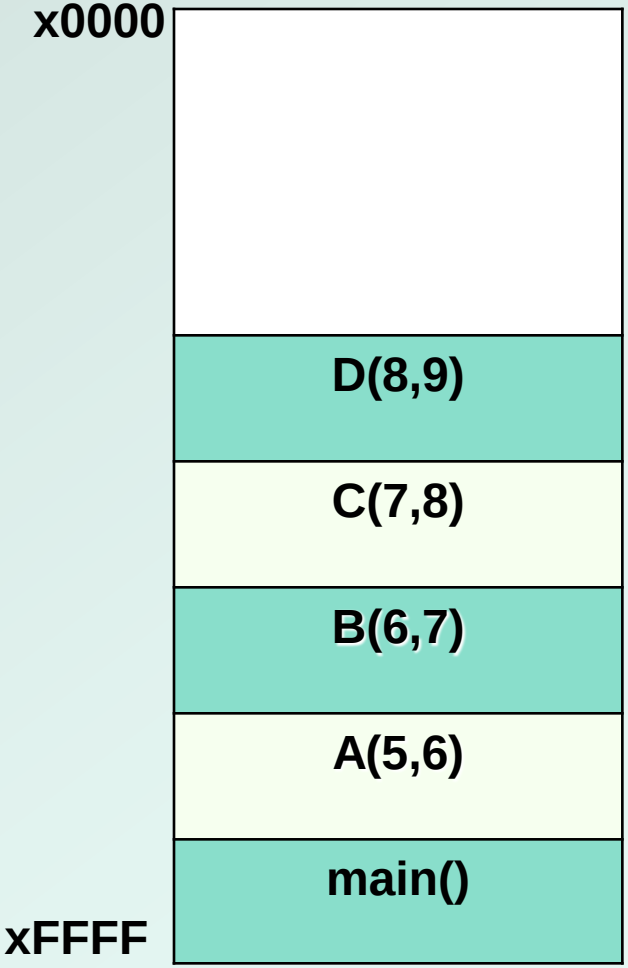
- Instructions are stored in **code segment**
- Global data is stored in **data segment**
- Statically allocated memory (locals) uses **stack**
- Dynamically allocated memory uses **heap**



- Code segment is write protected
- Initialized and uninitialized globals
- Heap can be fragmented
- Stack size is usually limited
- Stack can grow (usual convention is toward smaller addresses)

Execution Stack

- Picture of stack during program execution, same call stack as previous slide:
  - main() calls A(5,6)
  - A(5,6) calls B(6,7)
  - B(6,7) calls C(7,8)
  - C(7,8) calls D(8,9)

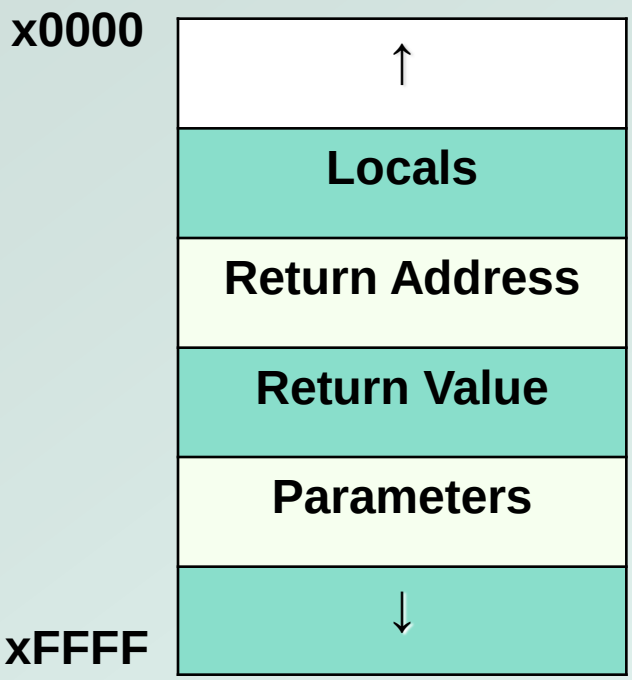


Stack Requirements

- Consider what has to happen in a function call:
  - Caller must pass parameters to the callee.
  - Caller must transfer control to the callee.
  - Callee must allocate space for the return value.
  - Callee must save the return address to caller.
  - Callee requires space for local variables.
  - Callee must return control to the caller.
  - Caller must pop the return value and params.
- Parameters, return value, return address, and locals are stored on the stack.
- The order above determines the responsibility and order of stack operations.

Execution Stack

- Definition: A stack frame or activation record is the memory required for a function call:



- Stack frame below contains the function that called this function.
- Stack frame above contains the functions called from this function.
- Caller pushes parameters, calls callee, pops return value and params.
- Callee allocates return value, pushes the return address, allocates and frees local variables, and stores the return value.

Stack Pointers

- Clearly we need a variable to store the **stack pointer** (SP), LC3 assembly uses R6.
- Stack execution is ubiquitous, so hardware has a stack pointer, sometimes even instructions.
- Problem: stack pointer is difficult to use to access data, since it moves around constantly.
- Solution: allocate another variable called a **frame pointer** (FP), for stack frame, uses R5.
- Where should frame pointer point? Convention sets it at some constant offset relative to locals.

What about Disk & Cache

- Disk (secondary memory)
  - LC-3 simplification: main memory holds everything
  - Actual systems (virtual memory):
    - Some of the stuff is on disk
    - OS manages things so that a process sees a flat virtual address space
    - Main memory appears larger than it actually is.
- Cache:
  - Hardware manages things so that main memory appears faster than it actually is.