EEE3096S: Embedded Systems II

LECTURE 15:

ARM C PROGRAM STRUCTURE

Presented by:

Dr Yaaseen Martin

Electrical Engineering
University of Cape Town

RECAP

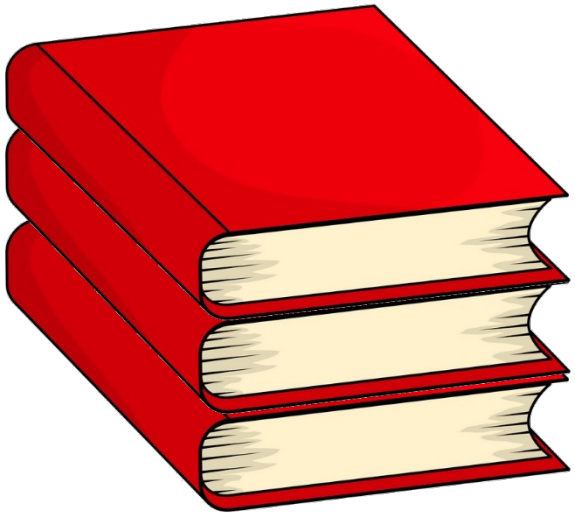
ARM: FUNCTIONS AND FRAMES

- In C, and most other executable formats, function calls are handled with **stack frames**
- These keep track of the call stack
- Special pieces of code are needed to handle these stack frames, specifically:
  - Before a function is called
  - At the start of a function (Prologue)
  - At the end of a function (Epilogue)

For further detail:
See text book (Patterson & Hennessy) Chapter 2, Section 2.8 " Supporting Procedures in Computer Hardware".

Stack Data Type

A stack data type works in pretty much the way you would pile things, like books, one on top of the other to form a stack of items…



A stack data type works in the same way. It has two primary methods that works on it:

push(*x*)	adds the object or value *x* to the top of the stack
x = pop()	removes the object or value at the top of the stack and returns it

Coding a Stack

You can utilize an array, and a *top* index, to construct a stack. i.e. *top* is the index of where the next item to be pushed to the stack will go. Pop uses *top* to indicate the item to pull off the stack and return.

The following code implements a simple stack:

```
// set the size of the stack
#define SIZE 100

int stack [SIZE];
int top = 0;  // index for next item

void push ( int x )  {
   // place an item on top of stack
   if (top<SIZE) {
      stack[top] = x;
      top=top+1;
      }
}
```

```
int pop ()  {
   // remove an item from top of stack
   if (top>0) {
      top = top - 1;
      return stack[top];
      }
}

int empty () {
    // return true if the stack is empty
    return top==0;
}
```

See next slide for full code

Coding a Stack – full program

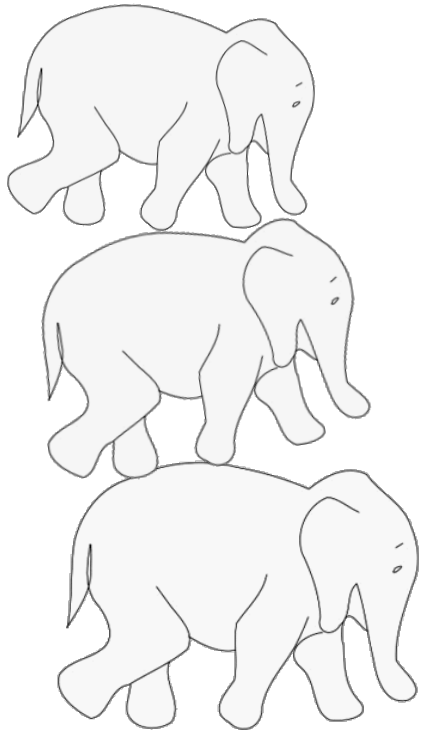
```
/*  The following code implements a simple integer stack in C: */
#include <stdio.h>

// set the size of the stack
#define SIZE 100

int stack [SIZE];
int top = 0;  // first place to put the next item


void push ( int x )  {
   // place an item on top of stack
   if (top<SIZE) {
      stack[top] = x;
      top=top+1;
      }
}

int pop ()  {
   // remove an item from top of stack
   if (top>0) {
      top = top - 1;
      return stack[top];
      }
}

int empty () {
    // return true if the stack is empty
    return top==0;
}

int main ( int argc, char** args )
{
    // main program to test the stack functions
    printf("Hello to stack test\n");
    push(10);
    push(20);
    push(30);
    while (!empty())
        printf("pop: %d\n", pop());
    printf("Program complete\n");
}
```

Example run:

```
Hello to stack test
pop: 30
pop: 20
pop: 10
Program complete
```

WHAT IS A CALL STACK?

- When one function calls another, which calls another function, which calls another…
- The program needs to keep track so that it can return to the original function
- So a stack of stack frames is kept; this stack is called the **call stack**



AN EXAMPLE CALL STACK

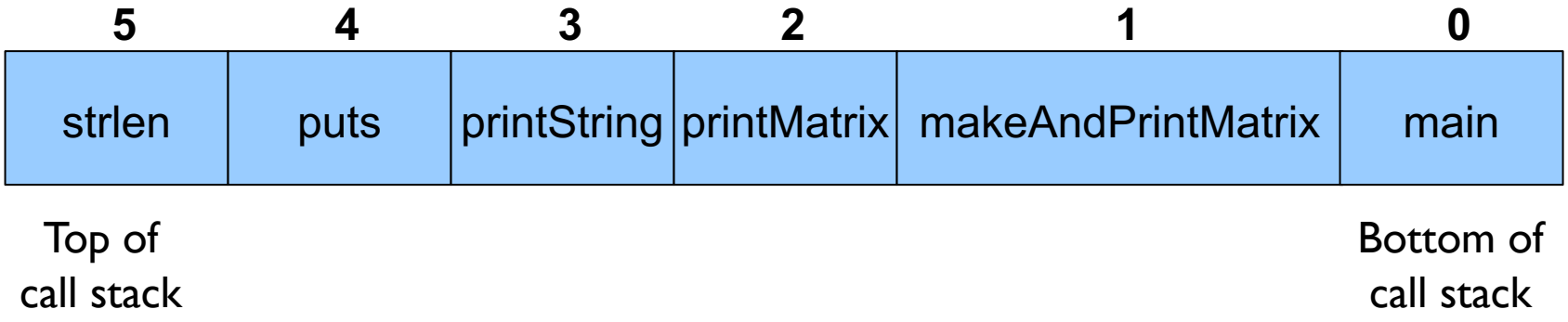
```
#5  0xb7e92243 in strlen () from /lib/i686/cmov/libc.so.6
#4  0xb7e7cfd5 in puts () from /lib/i686/cmov/libc.so.6
#3  0x08048365 in printString (dat=0x0) at frame.c:7
#2  0x0804837e in printMatrix (pc=0xbfded4a0) at frame.c:13
#1  0x080483a2 in makeAndPrintMatrix (mdata=0xbfded4a0…) at frame.c:19
#0  0x080483db in main () at frame.c:27
```

(generated by *gdb* from a crashed program)

e.g.

```
void main () {
  makeAndPrintMatrix(0);
}
```

A graphical representation of this call stack:



C RUNTIME (CRT) ENVIRONMENT

- As we now know, a runtime environment* is the software structures that supports program execution
- For C programs, these are:
  - Start-up code
  - Runtime Libraries (RTLs) **
- These RTLs provide the most basic support to programs (e.g., heap memory management, IO control)
- RTLs simplify the way programs interact with their “world” (i.e. their runtime environment)

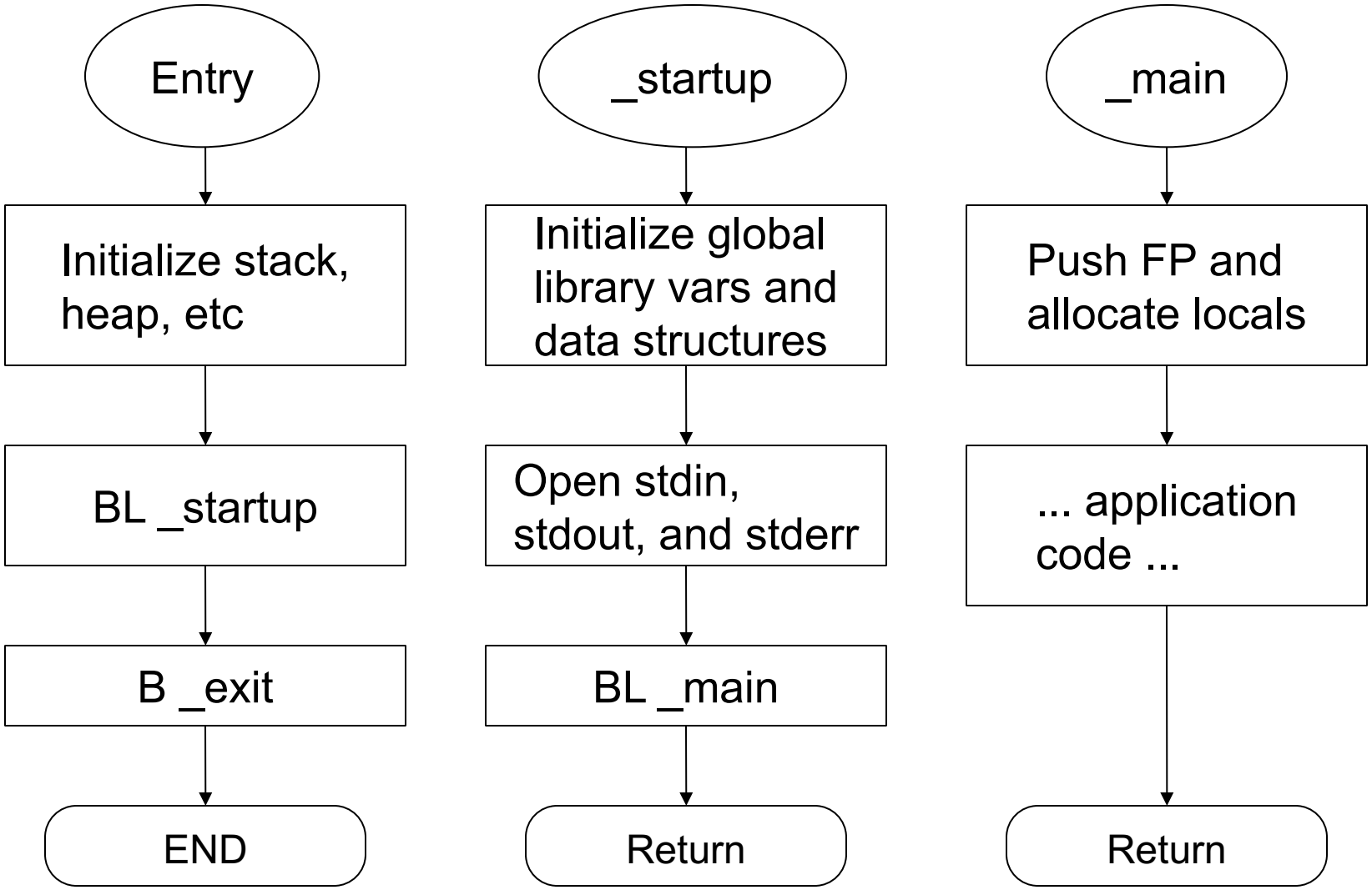
C START-UP (_START OR _STARTUP)

- The start-up code is the code than transfers control to main(). It is the code that is run even **before** the C program's main function starts.
- This code runs at the program **entry point**, which is a platform specific fixed address
- The most simple example would be

  B _main
- In most cases it's more complex than this
- Often, this code is automatically inserted by the C compiler, in module **crt0**

crt = C run time 0 → the first basic starting point

HOW *CRT0* FITS IN

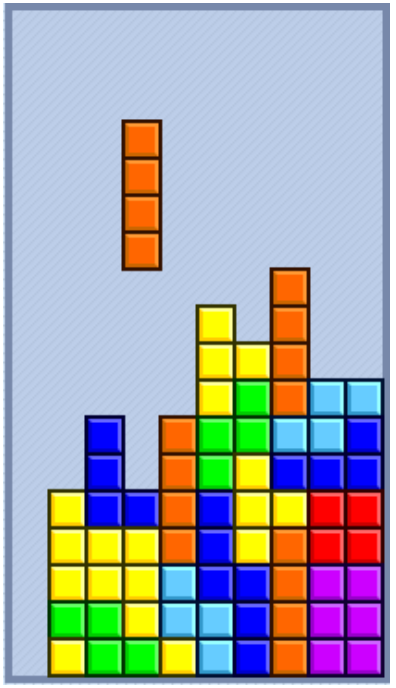


THE C RUNTIME LIBRARY (RTL)

- The RTL provides support functions which simplify the writing of code
- For example:
  - Basic IO functions (like read and write)
  - Floating point arithmetic on platforms which don't have a floating-point unit (**FPU** — a math co-processor)
- RTLs are great, but often **too bulky** for embedded systems, so they are cut down or removed completely (e.g. *printf*, *sin*, *cos*, etc.)

OBJECT PLACEMENT

- The C compiler generates **object code** that needs to be **positioned** in memory and integrated into a suitable **binary format**
- The **linker** is the tool for the job:
  - The linker fixes the location in memory
  - Can override linker with ASM macros (e.g. **ORG**)
- **Linker command file** controls placement of program sections
- Usually set up as a script file or add to **Makefile** for reusability



CMake

Another option that takes Make further; as per the CMake website, it is defined as:

> "CMake is an open-source, cross-platform family of tools designed to build, test and package software." https://cmake.org/

CMake can be considered an abbreviation for "cross-platform make". The C *does not* stand for C or C++, as this tool can be used for many different languages and tools, besides C.

For some quick information, links to online tutorials and manual pages for cmake, see: http://wiki.ee.uct.ac.za/Cmake

EXECUTION ENVIRONMENT MEMORY BUDGETING

- Embedded platforms have **limited** memory
- **Memory budget** = how much memory space do you need to run the program?
- Need to determine how many variables you can declare, max array sizes, max number of functions to call, etc.

MEMORY BUDGETING

Example:

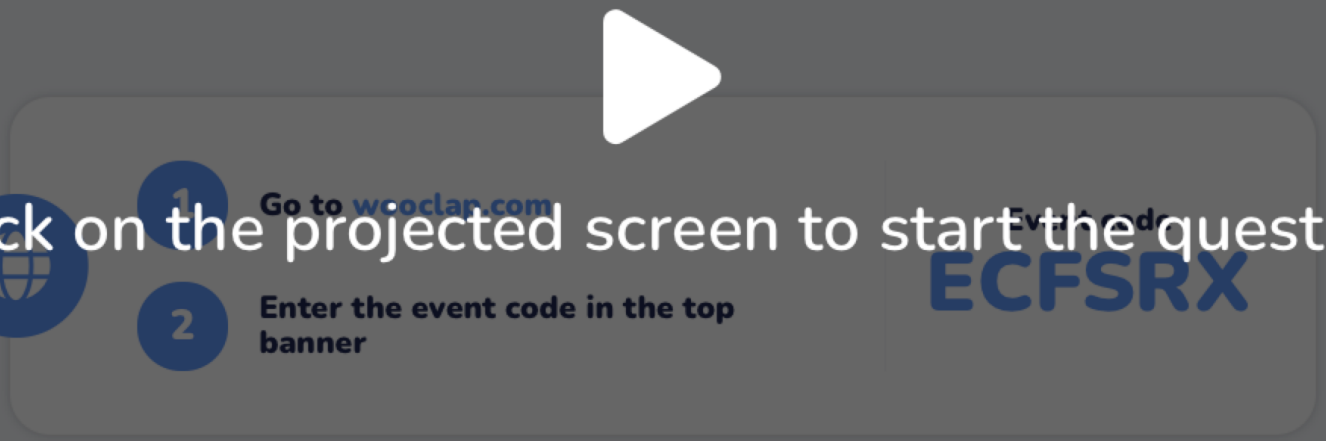
- Using 32-byte call stack, 64-byte general stack
- 6 items/functions on call stack and each function is 4 bytes (assuming 32-bit addresses)
  - Therefore memory required is: 4 bytes x 6 = **24 bytes**
  - Thus you only have **8 bytes** of the call stack left (or space for **two functions**) before a stack overflow* occurs

  *no, not the website
- **Stack overflow**: overflow error that occurs when a program tries to use more memory space in the call stack than has been allocated to that stack

MEMORY BUDGETING

- Another reason for doing a memory budget is for situations where you have a small amount of RAM but a larger amount of **ROM** (**read-only memory** that permanently stores instructions)
- Example:
  - 32 KB SRAM (Static RAM), directly coupled to processor
  - 512 KB external FLASH memory (ROM), with half taken up by functions (8 KB each); needs to be read one block at a time and you can't execute code directly from FLASH
  - Questions to consider:
    - How many of the functions can be in SRAM at once (so that they can be executed)?
    - How deep can the call stack become?
    - Which functions should always be in SRAM?
    - When we load one function, should we load other functions it might call?

How to participate?



Click on the projected screen to start the question

If executing a Branch and Link (BL) instruction, what would be stored into the Link Register?

1. SP
2. PC

Click on the projected screen to start the question

3. PC + 4
4. PC + 32

The exception vector is...

1. a piece of code that is executed when an exception occurs
2. a location in memory where the exceptio[illegible]ndler is stored
3. an exception that has both magnitude and direction
4. an unexpected event that occurs from within the processor

Click on the projected screen to start the question

An Application Binary Interface (ABI) is an interface between two binary program modules.

1. True

Click on the projected screen to start the question

2. False

Serial Peripheral Interface (SPI) is said to be a serial, half-duplex, synchronous communication protocol. True or false?

1 True 0% 0

2 False 0% 0