Feb 06, 2024

Cross-compilation Toolchain

There will always be at least 3 data sheets with each microprocessor:

1. programming manual - processor/chip hardware stuff
   - Things that belong to the CPU
2. Peripheral reference manual - the stuff around the CPU
   - determined by core families
3. Traditional datasheet for the chip
   - pinouts, max electrical nodes, example projects etc.
4. Board manual - if the microprocessor is mounted to one

The C language

- Nowadays, the most commonly-used programming language for embedded systems
- Powerful and easy-to-use to express algorithmic steps
  - Only 32 reserved words
- Considered “high-level” assembly …
  - Low-level control of what the processor does
  - Efficient compilers available almost for every existing architecture
  - High efficiency of the produced code
- … but highly portable
  - From mainframes to microprocessors and micro-controllers

Cross-development Platform

- Microprogrammed embedded systems are devices with limited resources, and are typically not powerful enough to run a compiler, a file system or a development environment

- Cross development is the separation of the system build environment from the target environment

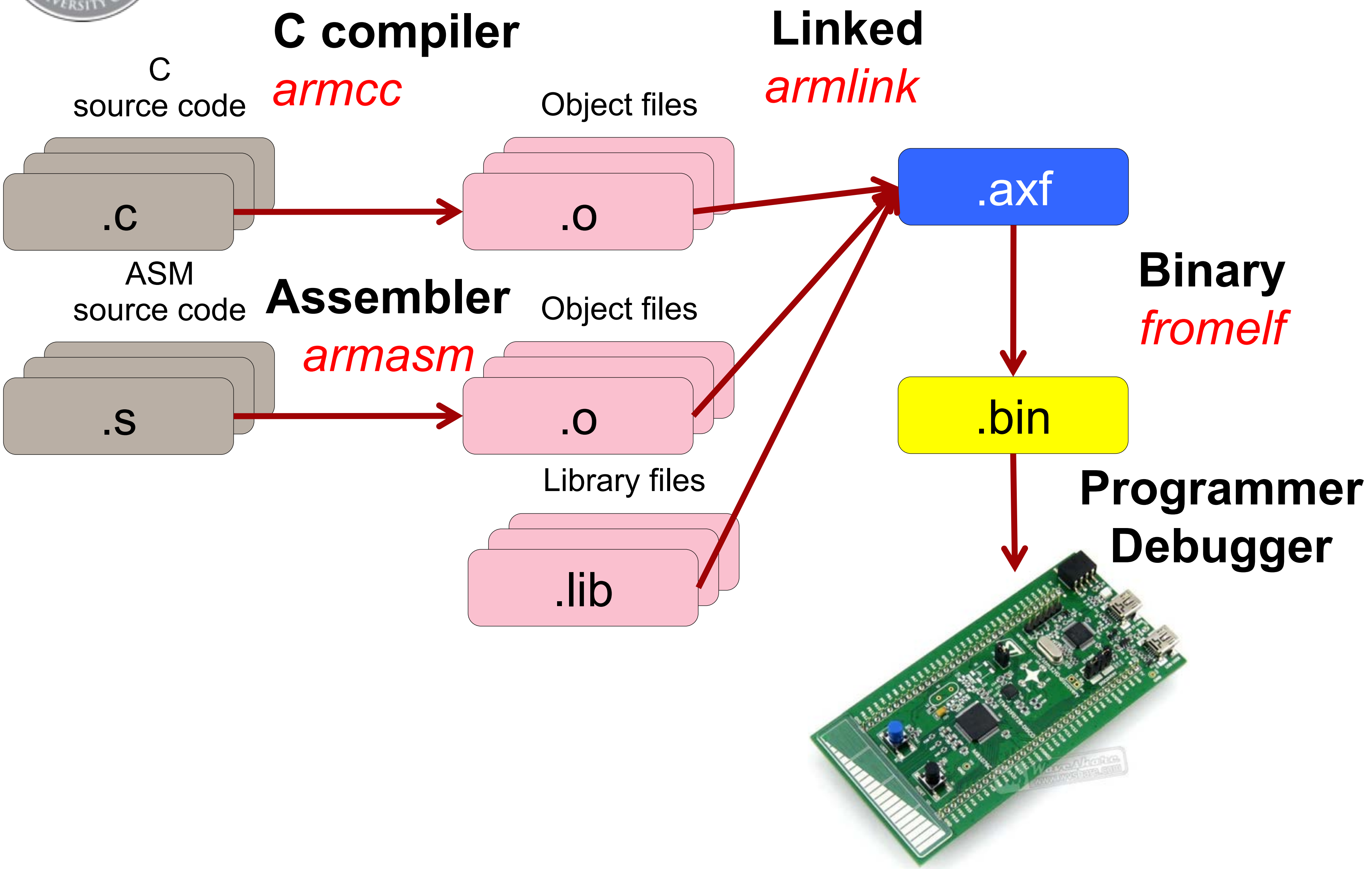
- Benefits
  - Faster compilation of applications
  - Debugging and testing with more resources than available on target embedded system

Cross-toolchain for C language

- The complete flow to generate the final binary for the target platform and its validation for the target micro-programmed embedded systems requires a cross-compilation toolchain
- It is a set of tools running on a host machine, used to

1. Pre-process header files (*#include*) and macros (*#define*) and Compile high-level source code (*.c*) to the target object code (*.o*)
   - Possible to get assembly output as intermediate step (*.s*)
2. Link pre-existing collections or libraries of object files (*.o*) to obtain a final executable object (.elf, *.axf*) with all the necessary object code
3. Pack and format the executable object into a format that can run with the target memory hierarchy and I/O subsystem

Keil C-Based cross-compilation flow



Example: compiling C source code

```
#include <nds.h>
#include <stdio.h>
#define TRUE 1
#define FALSE 0
int main(void)
{
    int i;
    i = 5 * 2;
    printf("5 times 2 is %d.\n", i);
    printf("TRUE is %d.\n", TRUE);
    printf("FALSE is %d.\n", FALSE);
}
```

Handled by the pre-processor

Handled by the compiler

Implemented in C library for the target platform

Memory Stack

Stack Memory Operations

- Stack memory is a memory usage that allows the system memory to be used as temporary data storage.

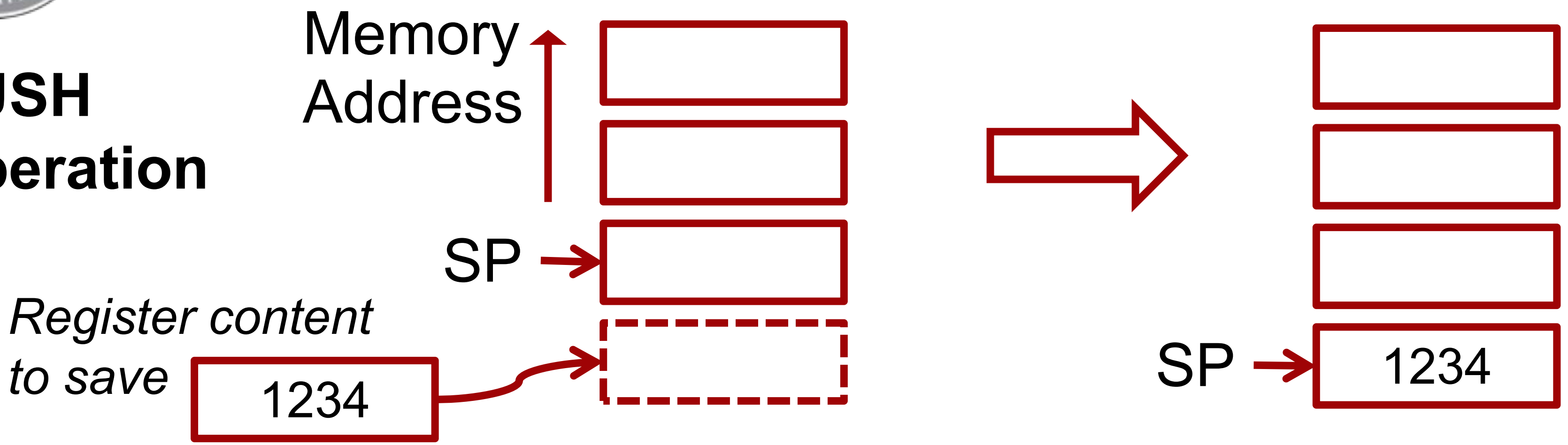
- Particularly useful for register storage.
- Behaves as a first-in last-out buffer.
- Cortex-M uses a “full-descending” stack model.

- Storing register to the stack is called PUSH.
- Restoring register from the stack is called POP.

- SP (R13) register indicates where the current stack memory location is.

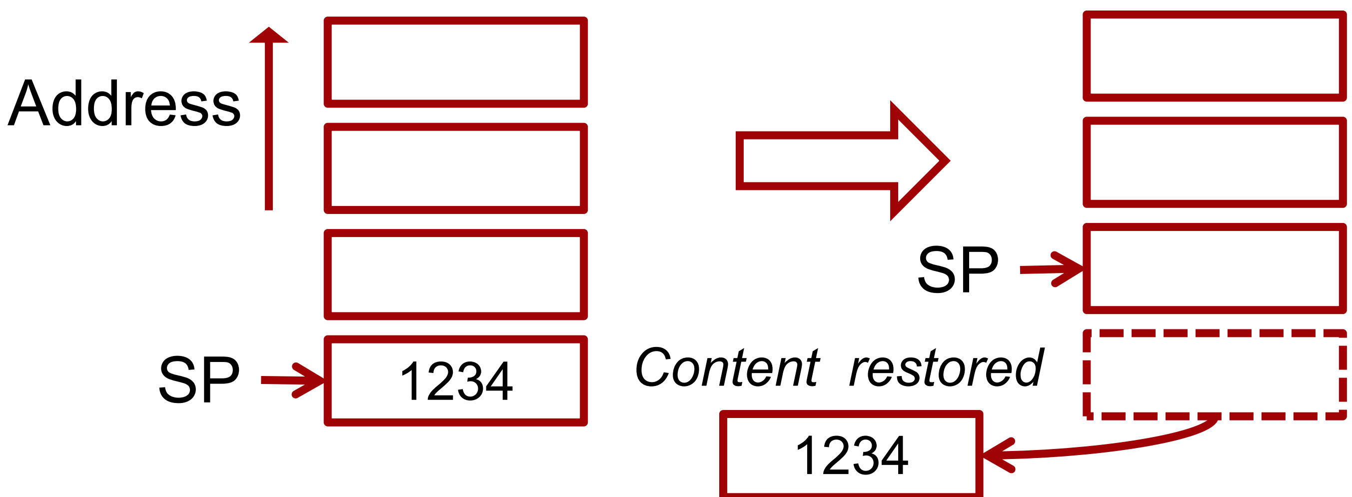
PUSH and POP

PUSH Operation



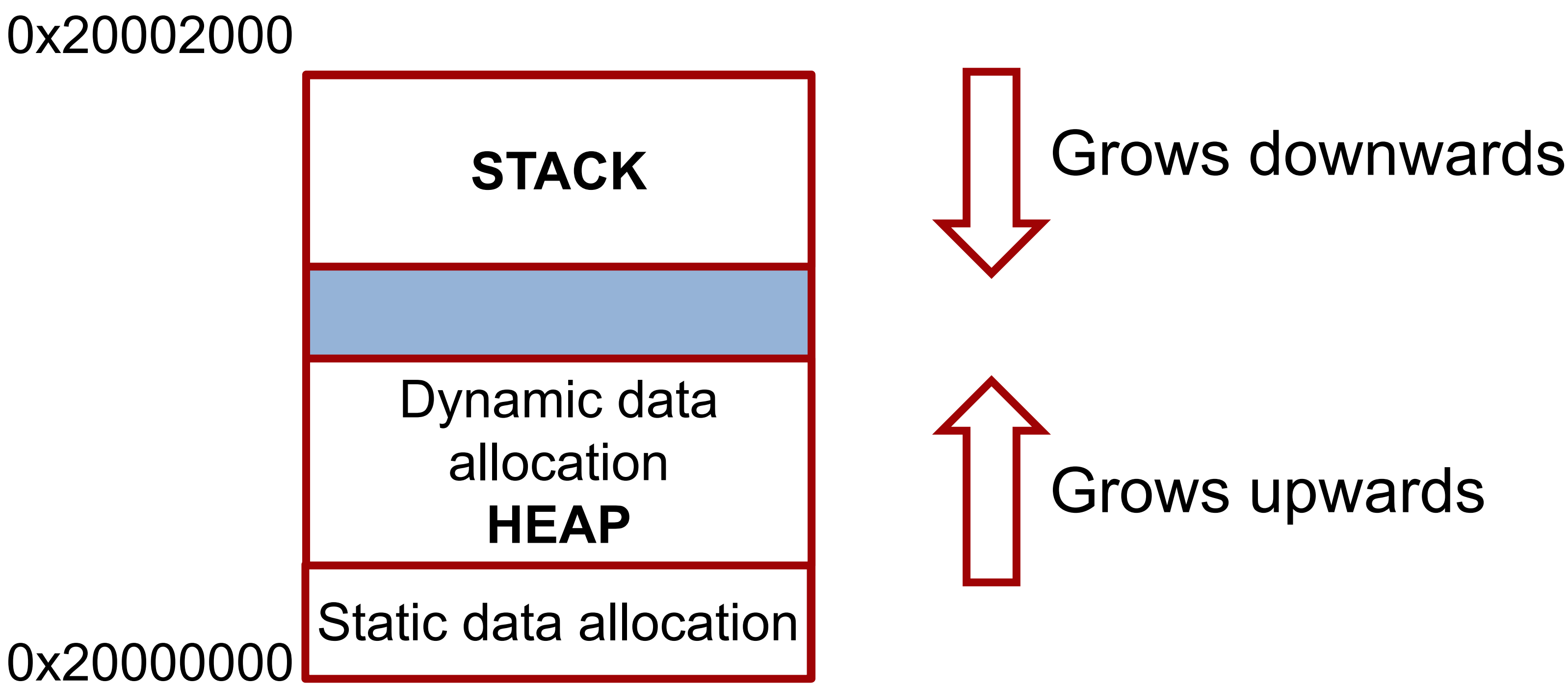
Data processing (Original content gets destroyed)

POP Operation



Why does the Stack grow downwards?

To allow maximum flexibility!



On a general basis, it is dangerous to use dynamic data allocation!

Processor-startup

Reset Sequence