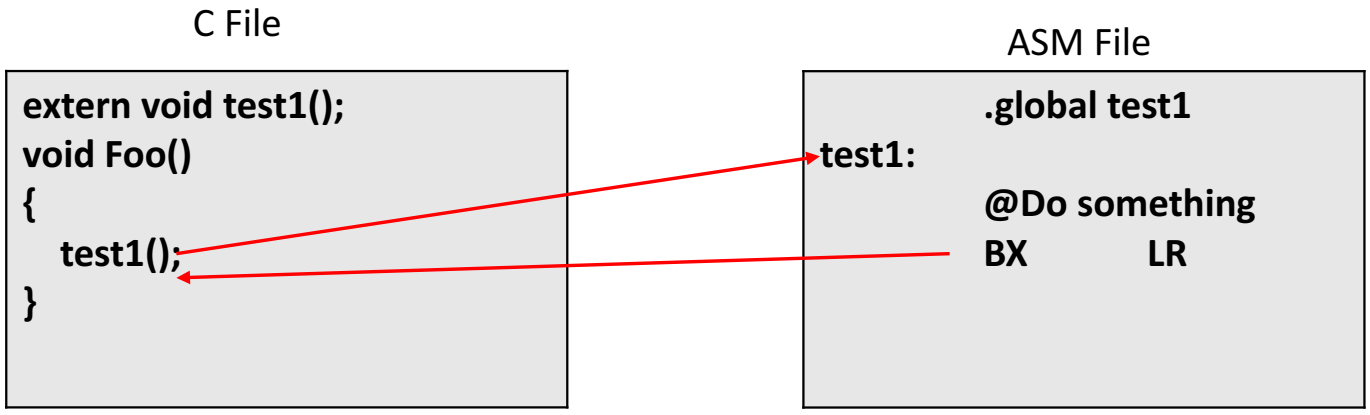
ASM and C Procedures Calls

Outline

- Procedure Call Standard for the ARM Architecture
- How to write a standard subroutine
- Calling convention
- GCC inline assembler

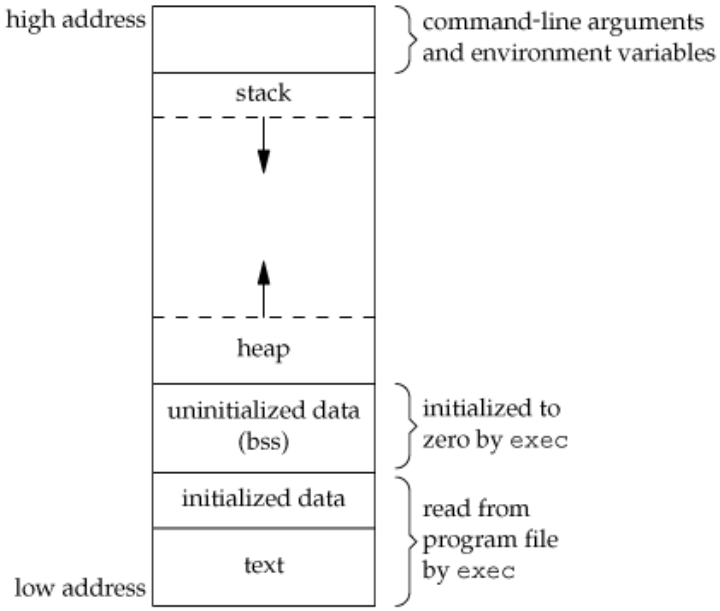
Review Function Calling

- Parameter passing
- Return variable



Program Memory Model

- Code
- Read-only static data
- Writable static data.
- Stack
- Heap



Procedure Call Standard for the ARM Architecture (AAPCS)

- Define how subroutines can be separately written, separately compiled, and separately assembled to work together.
- It describes a contract between a *calling* routine and a *called* routine that defines:
  - Obligations on the caller to create a program state in which the called routine may start to execute.
  - Obligations on the called routine to preserve the program state of the caller across the call.
  - The rights of the called routine to alter the program state of its caller.

Data Type

Type Class	Machine Type	Byte size	Byte alignment	Note
Integral	Unsigned byte	1	1	Character
	Signed byte	1	1	
	Unsigned half-word	2	2	
	Signed half-word	2	2	
	Unsigned word	4	4	
	Signed word	4	4	
	Unsigned double-word	8	8	
	Signed double-word	8	8	
Floating Point	Half precision	2	2	See *§4.1.1, Half-precision Floating Point*.
	Single precision (IEEE 754)	4	4	The encoding of floating point numbers is described in [ARM ARM] chapter C2, *VFP Programmer's Model*, *§2.1.1 Single-precision format*, and *§2.1.2 Double-precision format*.
	Double precision (IEEE 754)	8	8	
Containerized vector	64-bit vector	8	8	See *§4.1.2, Containerized Vectors*.
	128-bit vector	16	8	
Pointer	Data pointer	4	4	Pointer arithmetic should be unsigned.
	Code pointer	4	4	Bit 0 of a code pointer indicates the target instruction set type (0 ARM, 1 Thumb).

AAPCS Register Usage

- Argument registers: R0-R3
- Local scratch registers: R4-R11
- Stack pointer: R13
- Link register: R14
- Program counter: R15

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

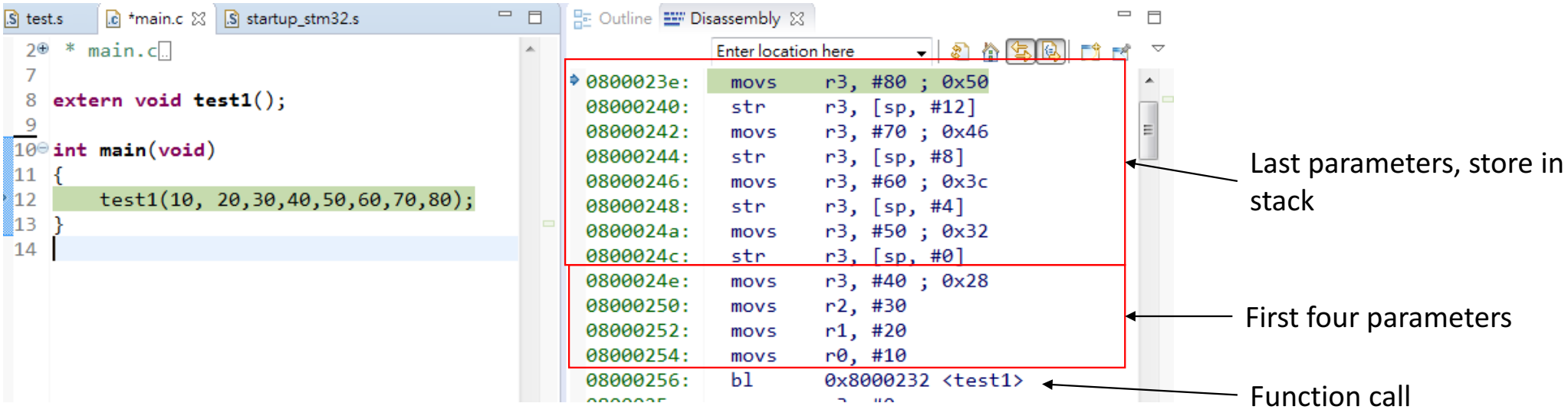
Table 2, Core registers and AAPCS usage

Universal Stack Constraints

- Stack-limit < SP <= stack-base.
  - The stack pointer must lie within the extent of the stack.
- SP mod 4 = 0.
  - The stack must at all times be aligned to a word boundary.
- When a publicly visible function is called, the stack pointer value is 8-byte aligned.

Parameters Passing

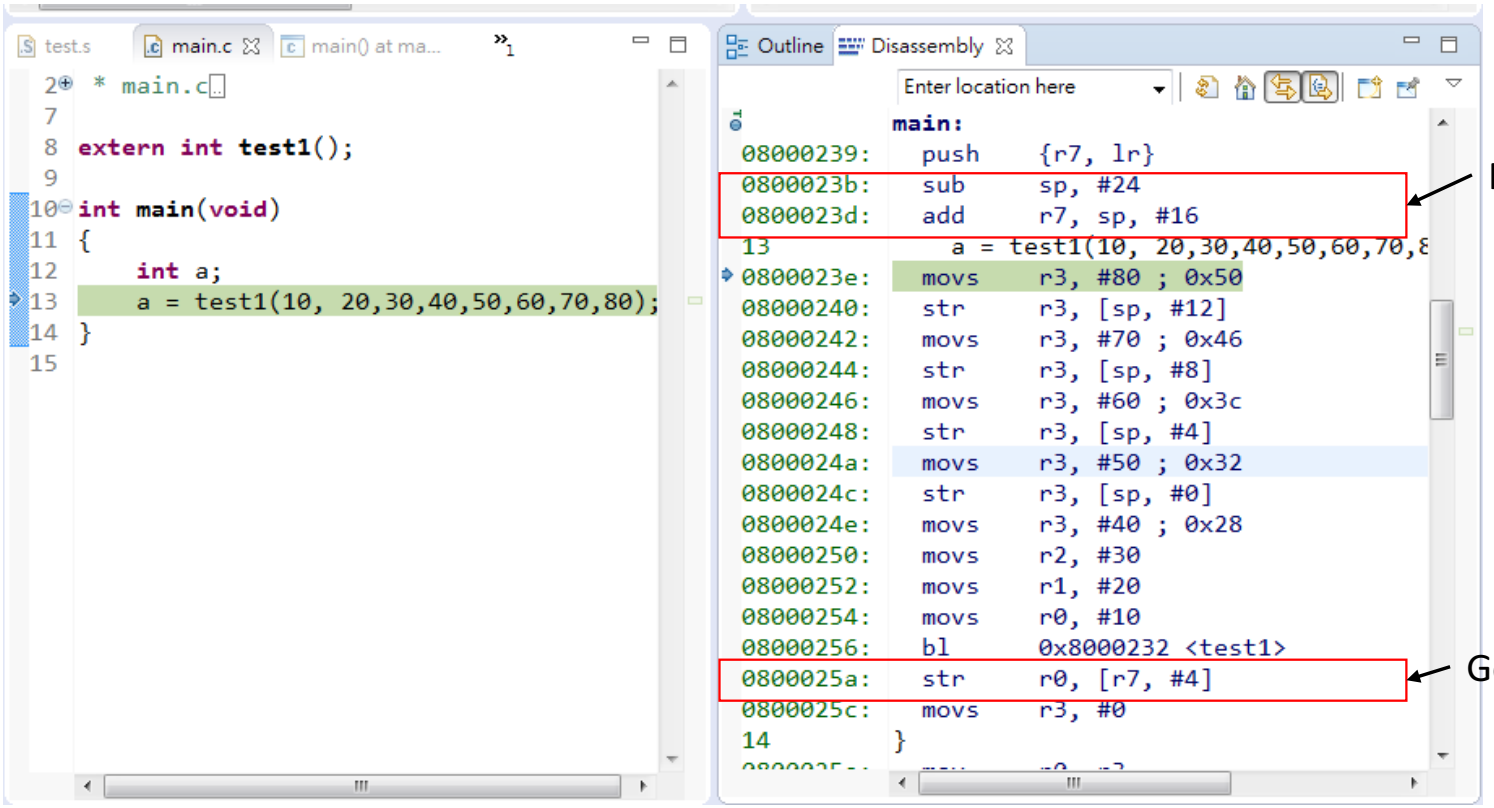
- The subroutine is used r0-r3 for passing first 4 parameters, otherwise place in stack memory.



Return Results

- In normal case the subroutine is returned in R0 register (4bytes)
  - int, float
  - char (last 8 bit of r0)
  - short (last 16 bit of r0)
- 8byte value returned in R0, R1 register
  - long long, double
- A 128-bit containerized vector is returned in r0-r3.
- A Composite Type(struct) not larger than 4 bytes is returned in r0.
- A Composite Type larger than 4 bytes
  - Store in stack memory and pass the address in R0

Result Return Example



Reserve local variable space

Get return value from subroutine

Calling Convention

- In the prologue, push r4 to r11 to the stack, and push the return address in r14, to the stack.
  - (This can be done with a single STM instruction).
- Copy any passed arguments (in r0 to r3) to the local scratch registers (r4 to r11).
- allocate other local variables to the remaining local scratch registers (r4 to r11).
- Do calculations and call other subroutines as necessary using BL, assuming r0 to r3, r12 and r14 will not be preserved.
- Put the result in r0
- In the epilogue, pull r4 to r11 from the stack, and pull the return address to the program counter r15.
  - (This can be done with a single LDM instruction).

Reference: https://en.wikipedia.org/wiki/Calling_convention

Register Usage rule in ASM

- Subroutine must preserve the contents of r4 to r11 and the stack pointer(r13)

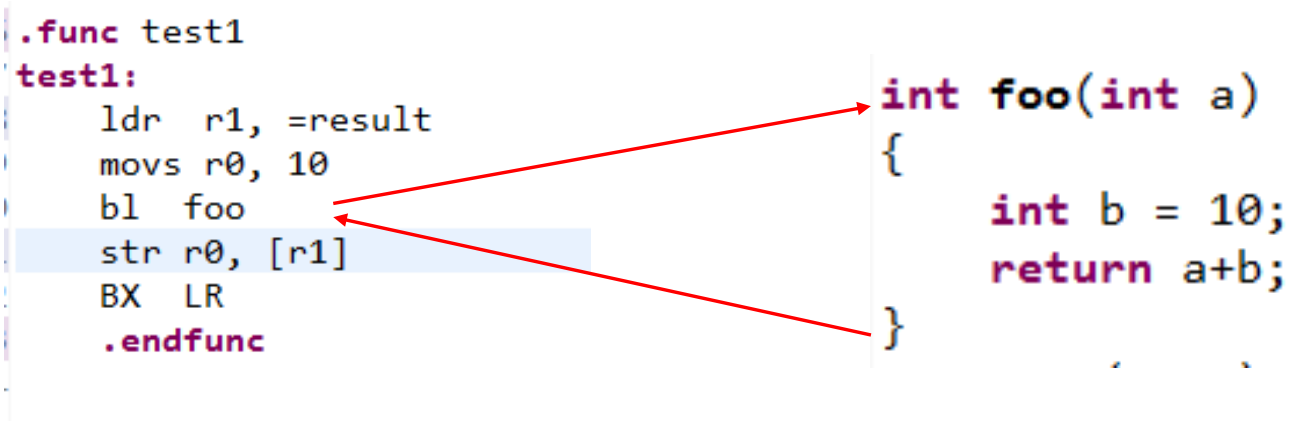
```
        .global foo
foo:
        push {R4-R11, LR}
        //Do something
        pop {R4-R11, PC}
```

Calling ASM function in C

- Define the function symbol as .global in ASM
- Follow the calling convention rules to write the ASM program
- Extern the function symbol and do normal function in C
- Another way: inline assembler

Calling C function in ASM

- Follow the AAPCS to pass function parameter and get return value.



GCC Inline Assembler

- Use for embedded the asm in C language

```
__asm    ("    inst1 op1, op2... \n"
          "    inst2 op1, op2... \n"
          ...
          "    inst op1, op2...\n"
          : output_operands            /* optional */
          : input_operands             /* optional */
          : clobbered_register_list    /* optional */
          );
```

If the assembler code needs to have an input variable and an output variable—for example, divide a variable by 5 in the following code—it can be written as follows:

```
unsigned int DataIn, DataOut; /* variables for input and output */
...
__asm   ("mov   r0, %0\n"
         "mov   r3, #5\n"
         "udiv  r0, r0, r3\n"
         "mov   %1, r0\n"
         :"=r" (DataOut) : "r" (DataIn) : "cc", "r3" );
```

%0 is first operand, %1 is second operand

Tell GCC which register will be modify.

Update condition code flags

Reference

- Procedure Call Standard for the ARM Architecture
  - http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042f/IHI0042F_aapcs.pdf
- ARM ABI慣例概觀
  - https://msdn.microsoft.com/zh-tw/library/dn736986.aspx
- GCC Inline ASM
  - http://www.ethernut.de/en/documents/arm-inline-asm.html
  - http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html
- https://en.wikipedia.org/wiki/Calling_convention#ARM_.28A32.29