µP Tutorial 1

BY ASHRAF SUYYAGH

SLIDES MODIFIED FROM LAST YEAR / ORIGINALLY WRITTEN BY ALEXANDRE COURTEMANCHE

Outline

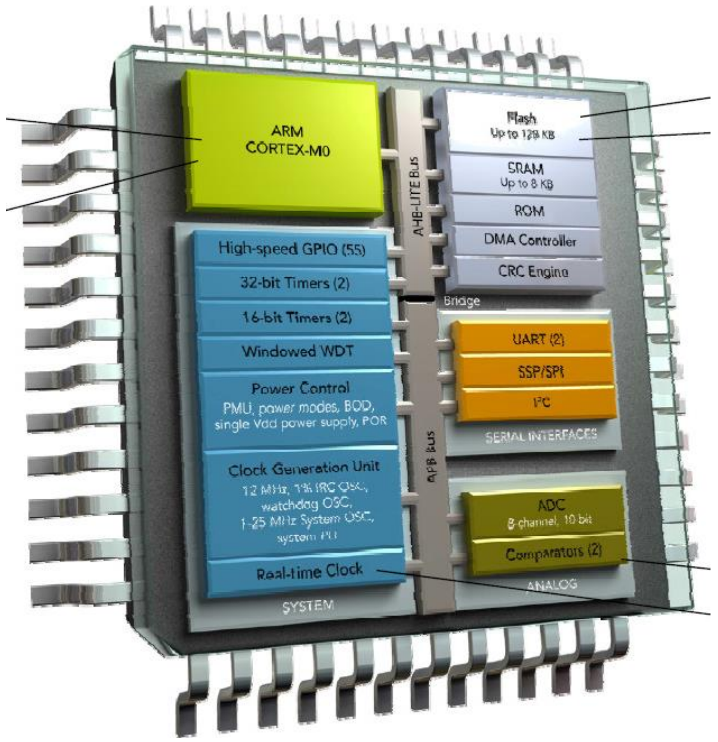
- Introduction to the lab and what is it all about.
- Introduction to the HW platform and supporting software tools
- Understanding the Programming Model of Cortex M4
- Instruction set quick short review (Arithmetic, Logical, Comparison, Movement instructions)
- Example – Writing first assembly program in Keil IDE
- Floating Point Instructions and Enabling the FPU Unit on the µC
- Lab Demos and Report Submissions, Grading

What is a Microcontroller (µC, uC or MCU)?

- Integrated Circuit
- Processor Core
- Memory
- Programmable Peripherals including I/O

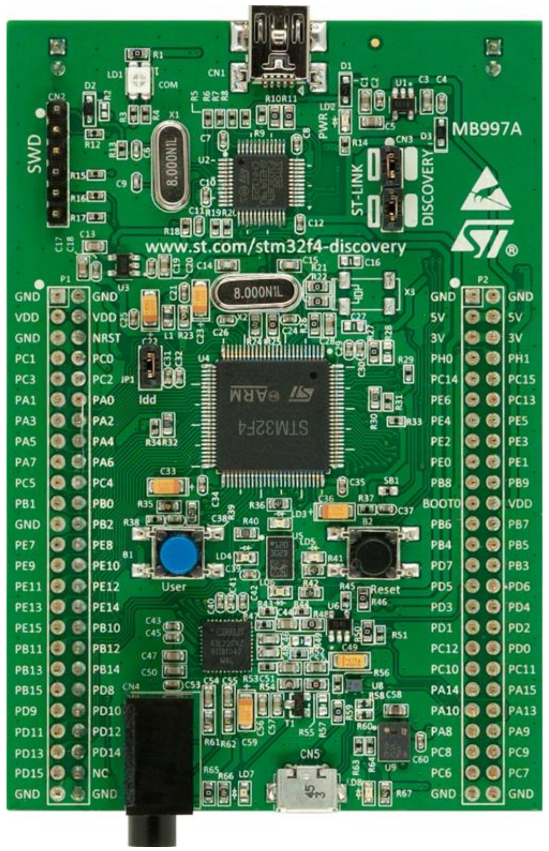
What is ARM?

- **ARM** is a **reduced instruction set computer** (RISC) instruction set architecture (ISA) developed by ARM Holdings.
- ARM does not manufacture its own CPUs. IP Licensing(Texas Instruments, Analog Devices, Atmel, Freescale, Nvidia, Qualcomm, STMicroelectronics and Renesas have all licensed ARM technology).
- Annual shipments estimates of ARM processors are at 7 billion per year [1]. Almost quarter of the total embedded market share is ARM's. ARM leads in 32 bit embedded Market with the most share[2] More News here [http://www.semicast.net/inthenews.html].
- Has three major µC families (profiles): A, R and M



ARM Cortex-M Series

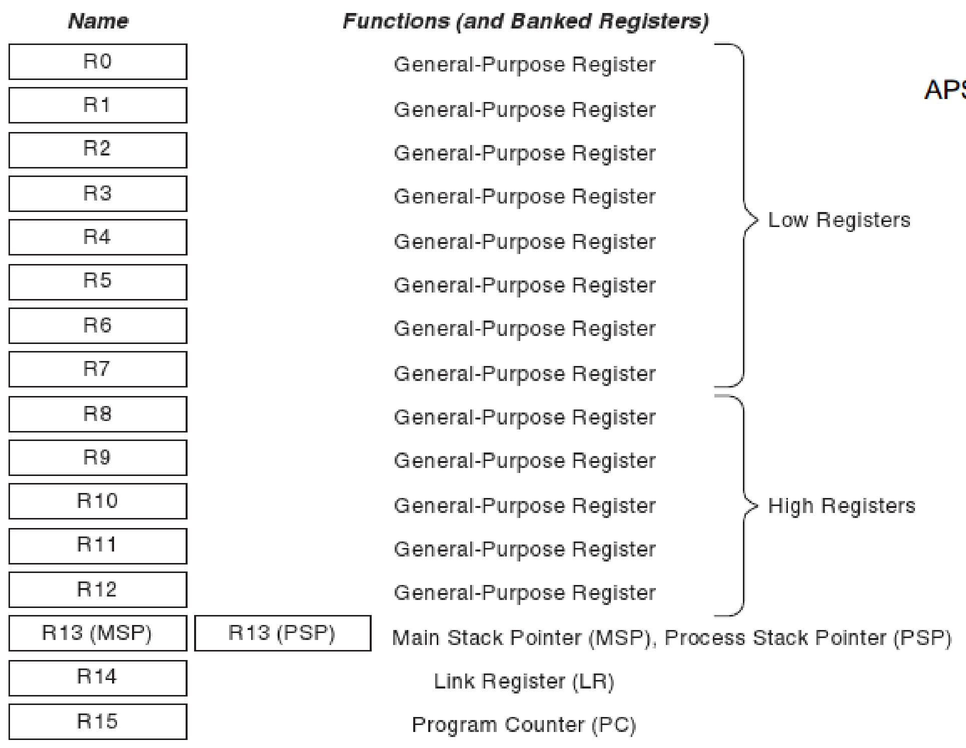
- IP core intended for microcontroller applications
- Cortex-M0, M3, **M4**
- In this lab, we will be using the Discovery F4 Board (Lab2+)
- ARM 32-bit Cortex™-M4 CPU with FPU (STM32F405VG)
- Up to 1 Mbyte of Flash memory
- Up to 140 I/O ports with interrupt capability
- Up to 15 communication interfaces
- Advanced connectivity
- General-purpose DMA
- 3×12-bit A/D converters
- 2×12-bit D/A converters
- And much more…



Assembly Programming

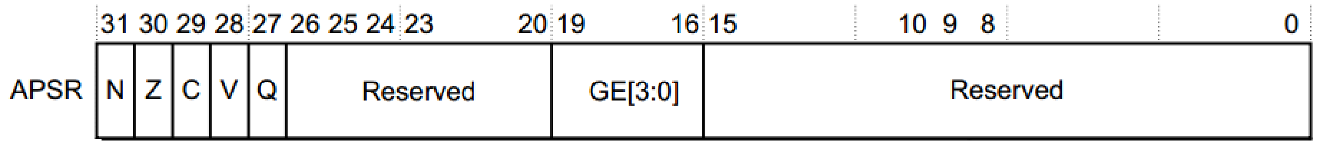
- Low-Level Programming Language
- Each statement corresponds to a single machine instruction
- Specific to a particular architecture (Thumb, Thumb-2, ARMx)
- Assembly is converted to executable machine code by the **assembler**
- *How to learn assembly? Sources?*

Cortex M4 – Registers Overview



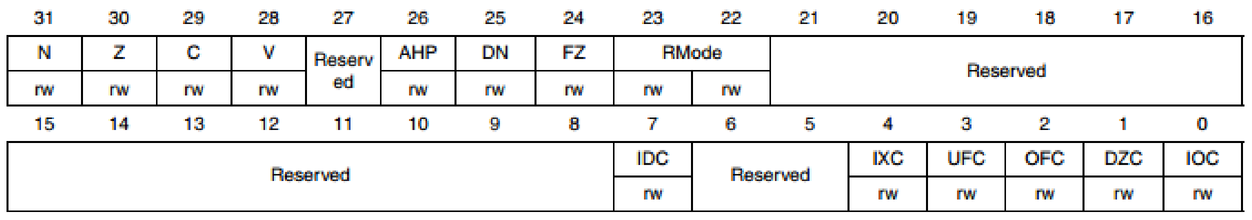
Name		Functions (and Banked Registers)	
R0		General-Purpose Register	Low Registers
R1		General-Purpose Register	
R2		General-Purpose Register	
R3		General-Purpose Register	
R4		General-Purpose Register	
R5		General-Purpose Register	
R6		General-Purpose Register	
R7		General-Purpose Register	
R8		General-Purpose Register	High Registers
R9		General-Purpose Register	
R10		General-Purpose Register	
R11		General-Purpose Register	
R12		General-Purpose Register	
R13 (MSP)	R13 (PSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)	
R14		Link Register (LR)	
R15		Program Counter (PC)	

Registers in the Cortex-M



	31	30	29	28	27	26 25 24 23 20	19 16	15 10 9 8 0
APSR	N	Z	C	V	Q	Reserved	GE[3:0]	Reserved

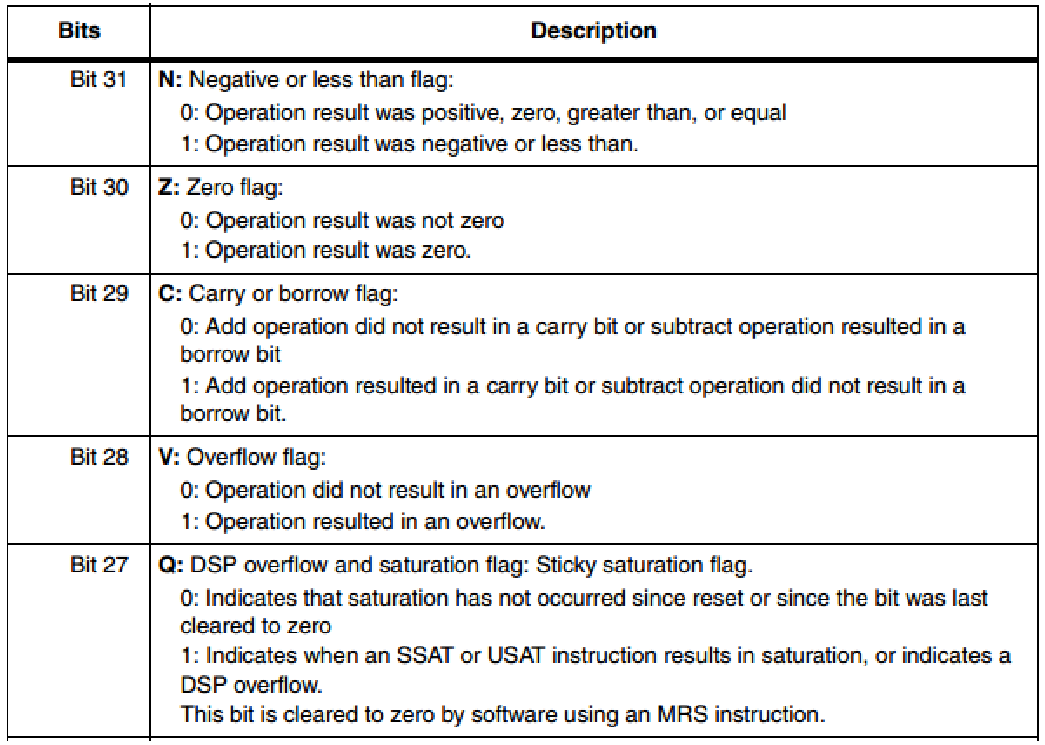
What about FP? Registers? Special Registers



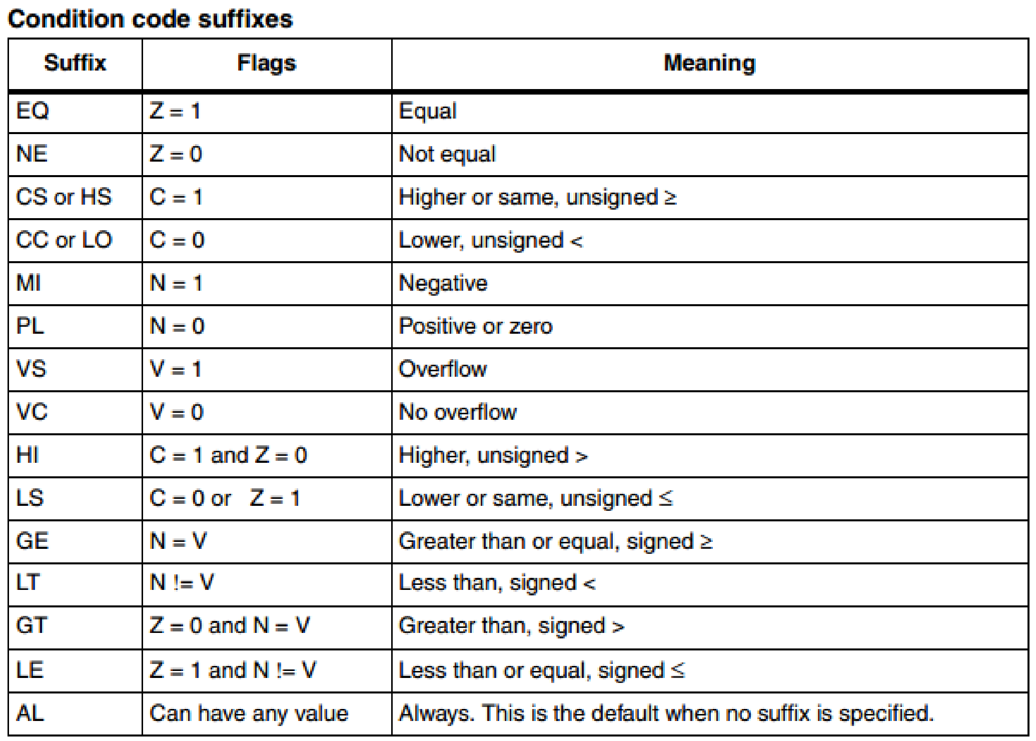
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
N	Z	C	V	Reserved	AHP	DN	FZ	RMode		Reserved					
rw	rw	rw	rw		rw	rw	rw	rw	rw						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								IDC	Reserved		IXC	UFC	OFC	DZC	IOC
								rw			rw	rw	rw	rw	rw

Status Register Bits (APSR)



Bits	Description
Bit 31	**N:** Negative or less than flag: 0: Operation result was positive, zero, greater than, or equal 1: Operation result was negative or less than.
Bit 30	**Z:** Zero flag: 0: Operation result was not zero 1: Operation result was zero.
Bit 29	**C:** Carry or borrow flag: 0: Add operation did not result in a carry bit or subtract operation resulted in a borrow bit 1: Add operation resulted in a carry bit or subtract operation did not result in a borrow bit.
Bit 28	**V:** Overflow flag: 0: Operation did not result in an overflow 1: Operation resulted in an overflow.
Bit 27	**Q:** DSP overflow and saturation flag: Sticky saturation flag. 0: Indicates that saturation has not occurred since reset or since the bit was last cleared to zero 1: Indicates when an SSAT or USAT instruction results in saturation, or indicates a DSP overflow. This bit is cleared to zero by software using an MRS instruction.



Condition code suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned ≥
CC or LO	C = 0	Lower, unsigned <
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned >
LS	C = 0 or Z = 1	Lower or same, unsigned ≤
GE	N = V	Greater than or equal, signed ≥
LT	N != V	Less than, signed <
GT	Z = 0 and N = V	Greater than, signed >
LE	Z = 1 and N != V	Less than or equal, signed ≤
AL	Can have any value	Always. This is the default when no suffix is specified.

Arithmetic Operations

- • **ADD**: operand1 + operand2
  • **SUB**: operand1 - operand2
  • **RSB**: operand2 - operand1
- Syntax:
  • <Operation>{<cond>}{S} Rd, Rn, Operand2
- Examples:
  • ADD r0, r1, r2
  • SUBGT r3, r3, #1
  • RSBLES r4, r5, #5

Logical Operations

- • **AND** operand1 AND operand2
  • **EOR** operand1 EOR operand2
  • **ORR** operand1 OR operand2
  • **BIC** operand1 AND NOT operand2 [i.e bit clear]
- Syntax

  <Operation>{<cond>}{S} Rd, Rn, Operand2
- Examples:

• AND r0, r1, r2

• BICEQ r2, r3, #7

• EORS r1,r3,r0

Comparisons

- Updates the condition flags
- • CMP operand1 - operand2, but result not written
  • CMN operand1 + operand2, but result not written
  • TST operand1 AND operand2, but result not written
  • TEQ operand1 EOR operand2, but result not written
- <Operation>{<cond>} Rn, Operand2
- Examples:

• CMP r0, r1

• TSTEQ r2, #5

Data Movement

What about FP instructions?

-
  - MOV operand2
  - MVN NOT operand2
- Syntax <Operation>{<cond>}{S} Rd, Operand2
- Examples:
  - MOV r0, r1
  - MOVS r2, #10
  - MVNEQ r1,#0

Starting up with Assembly #1

Startup File (Original)

```
 IMPORT SystemInit
 LDR     R0, =SystemInit
 BLX     R0
IMPORT __main
LDR     R0, =__main
BX      R0
```

Assembly File (subroutine)

```
        AREA text, CODE, READONLY
        EXPORT example1
example1
        MOV    R1, #25
        MOV    R2, #75
        MUL    R1, R1, R1
        MLA    R1, R2, R2, R1
        BX LR ; Return
        END
```

Starting up with Assembly #2

Startup File (Modified)

```
 IMPORT example1
 ~~IMPORT SystemInit~~
 ~~LDR     R0, =SystemInit~~
 ~~BLX     R0~~
~~IMPORT __main~~
~~LDR     R0, =__main~~
~~BX      R0~~
LDR     R0, =example1
BX      R0
```

Assembly File (subroutine)

```
        AREA text, CODE, READONLY
        EXPORT example1
example1
        MOV    R1, #25
        MOV    R2, #75
        MUL    R1, R1, R1
        MLA    R1, R2, R2, R1
        BX LR ; Return
        END
```

Starting up with Assembly #3

Startup File (Modified)

```
 IMPORT example2
 ~~IMPORT SystemInit~~
 ~~LDR     R0, =SystemInit~~
 ~~BLX     R0~~
~~IMPORT __main~~
~~LDR     R0, =__main~~
~~BX      R0~~
LDR     R0, =example2
BX      R0
```

Assembly File (subroutine)

```
        AREA text, CODE, READONLY
        EXPORT example2
example2
        VLDR.F32    S1, =2.5
        VLDR.F32    S2, =7.5
        VMUL.F32    S1, S1, S1
        VMLA.F32    S1, S2, S2
        BX LR ; Return
        END
```

Starting up with Assembly #3

```
LDR.W   R0, =0xE000ED88
LDR     R1, [R0]          ; Read present FPU setting
ORR     R1, #(0xF << 20)  ; Set bits to enable FPU
STR     R1, [R0]          ; Store result
DSB     ; Reset pipeline
ISB
```

Startup File (Modified)

```
 IMPORT example2
```

~~IMPORT SystemInit~~

~~LDR   R0, =SystemInit~~

~~BLX   R0~~

~~IMPORT __main~~

~~LDR   R0, =__main~~

~~BX   R0~~

```
LDR     R0, =example2
BX      R0
```

```
example2
        VMOV.F32    S1, #2.5
        VLDR.F32    S2, =7.5
        VMUL.F32    S1, S1, S1
        VMLA.F32    S1, S2, S2
        BX LR ; Return
        END
```

Using the Floating Point Unit

- **Enabling the FPU**
  ; CPACR is located at address 0xE000ED88
- LDR.W   R0, =0xE000ED88
- ; Read CPACR
- LDR   R1, [R0]
- ; Set bits 20-23 to enable CP10 and CP11 coprocessors
- ORR   R1, R1, #(0xF << 20)
- ; Write back the modified value to the CPACR
- STR   R1, [R0]; wait for store to complete
- DSB
-
  ;reset pipeline now the FPU is enabled
- ISB

It is Demo Time :D

- First Assembly Program using Keil
- How to simulate your program
- Basic IDE Features (More covered in next tutorial)

(watch Window, FPSCR, PSR, Performance analyzer, symbols, … etc)

Lets agree on some things, shall we?

- How demos go? What is expected from you? Grading demos? Partner involvement?
- Communicating with TAs, discussion board or emails? When to send an email, when not to?
- Lab Reports. How to write your report? Grading Reports? Fairness?

References

- [1] http://www.ecnmag.com/news/2011/05/arm-passes-x86-and-power-architecture-become-leading-mcu-empu-architecture-2010
- [2] http://www.design-reuse.com/news/27468/arm-processors-annual-shipments-forecast.html?sf2308980=1