MCU Introduction with Architecture & Assembly Review

Lecture 06

Josh Brake
Harvey Mudd College

Outline

- Introduction to the STM32-L432KC
- Review of basic architecture
- Review of assembly programming with focus on ARM-specific differences from RISC-V

Learning Objectives

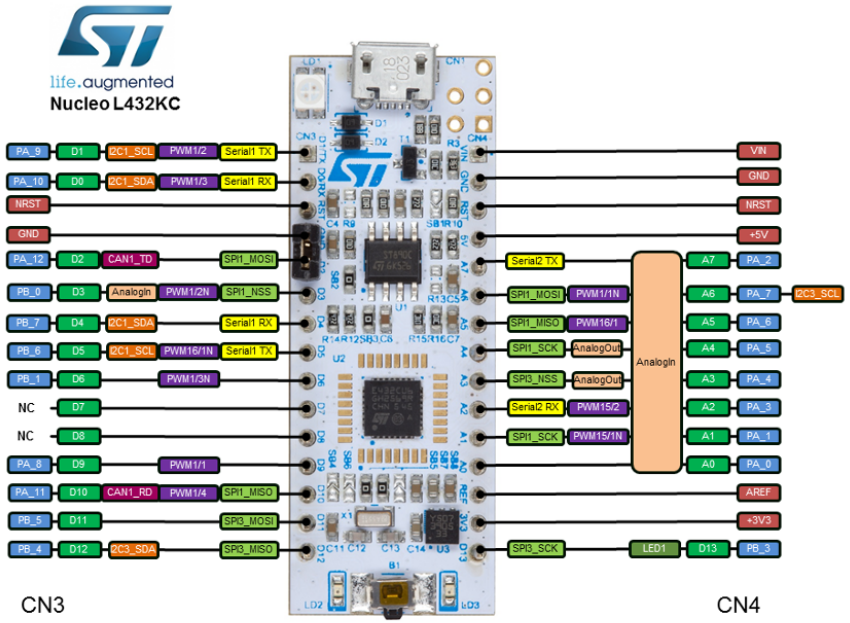
By the end of this lecture you should be able to…

- List the basic details of the architecture of the ARM Cortex-M4 CPU used in our STM32 MCU.
- Recall how to translate C to assembly language and assembly language to machine code.
- Learn the basic syntax of ARM assembly (and differences from RISC-V assembly).

STM32 Nucleo-32 board

Components

- MCU – STM32L432KC
- External flash memory
- 24 MHz crystal oscillator
- On-board ST-LINK debugger/programmer. Virtual COM port and debug port.
- 1 user LED and 1 reset push button
- Arduino Nano V3 form factor



What is an MCU

- MCU = MicroController Unit
- Processor core + peripherals

Figure 1. System architecture

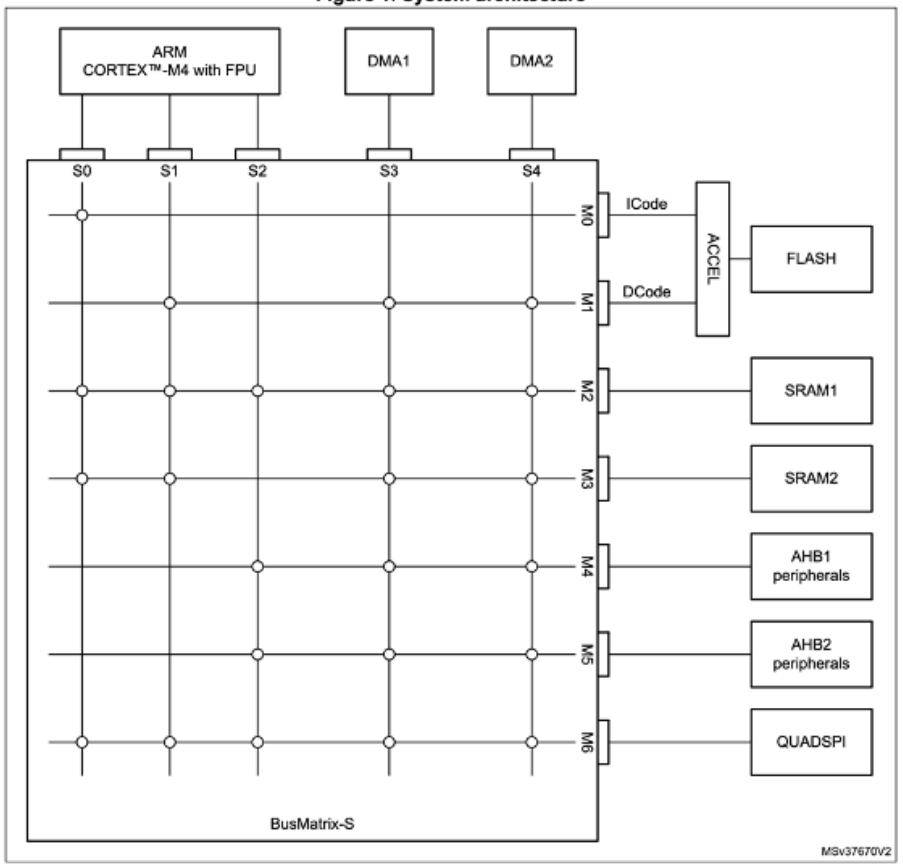
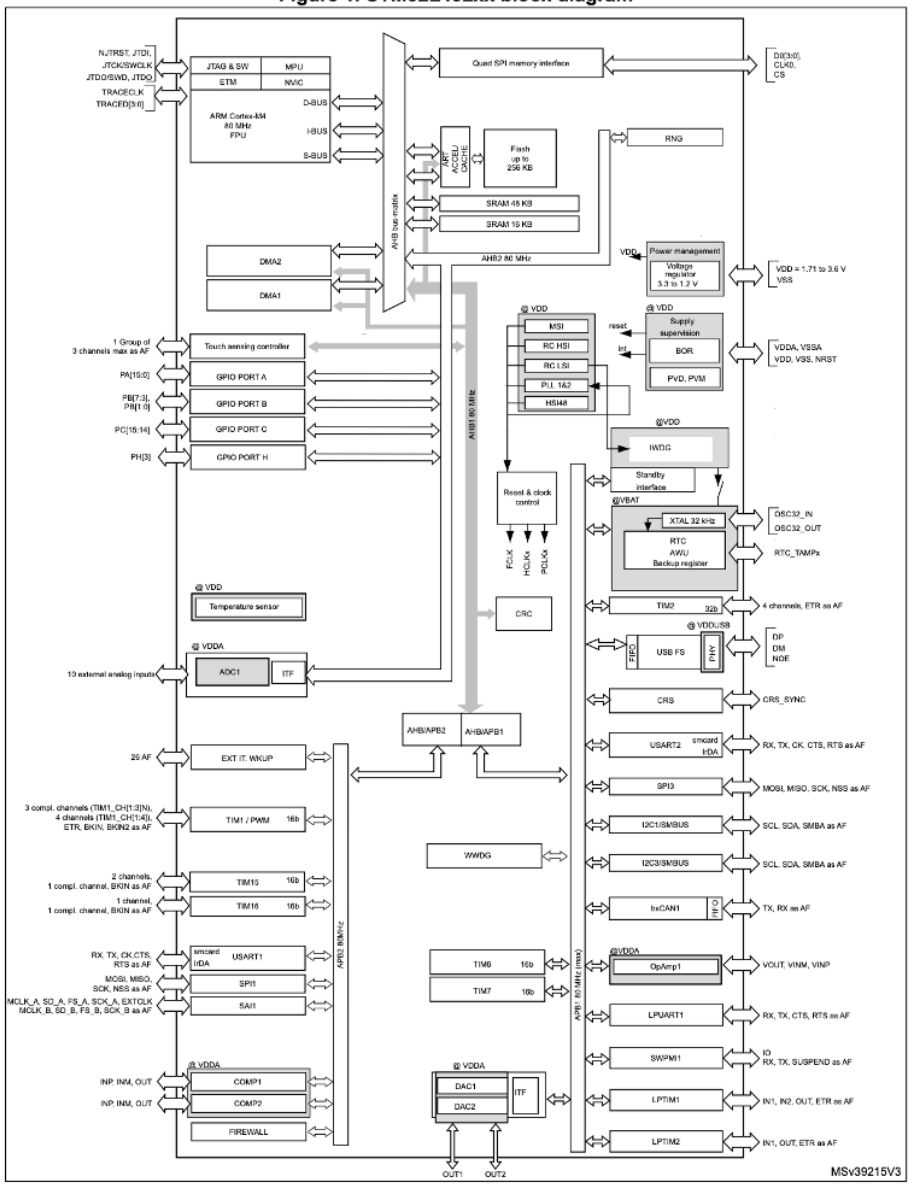


Figure 1. STM32L432xx block diagram



Documentation

Reference Manual: Information about all peripherals and their control registers

Datasheet: System block diagram, Pin functions, electrical characteristics, timing specs

Programmer's Manual: Information about the architecture (e.g., Cortex-M4), supported assembly instructions, registers, memory map, etc.

Questions for a new MCU

- What is the register set?
- What does the memory map look like?
- What addressing modes are used?
- What types of instructions exist?
- What I/O functions are available?

STM32 L432KC Architecture

- STM32 MCU has an ARM Cortex-M4.
- It runs the ARMv7E-M architecture. This is a 32-bit architecture.
- Also supports Thumb-2 execution. This is a set of compressed, 16-bit instructions.
- One special thing to watch with Thumb is conditional execution. With Thumb execution you can only use conditional execution within an “if-then” block which can hold up to four successive instructions.

Architecture Overview

- Instructions are 32-bit
- 16, 32-bit registers R0-R15
- Supports condition codes
- Most instructions operate on two registers and put result in a third.

Microarchitecture Flashback

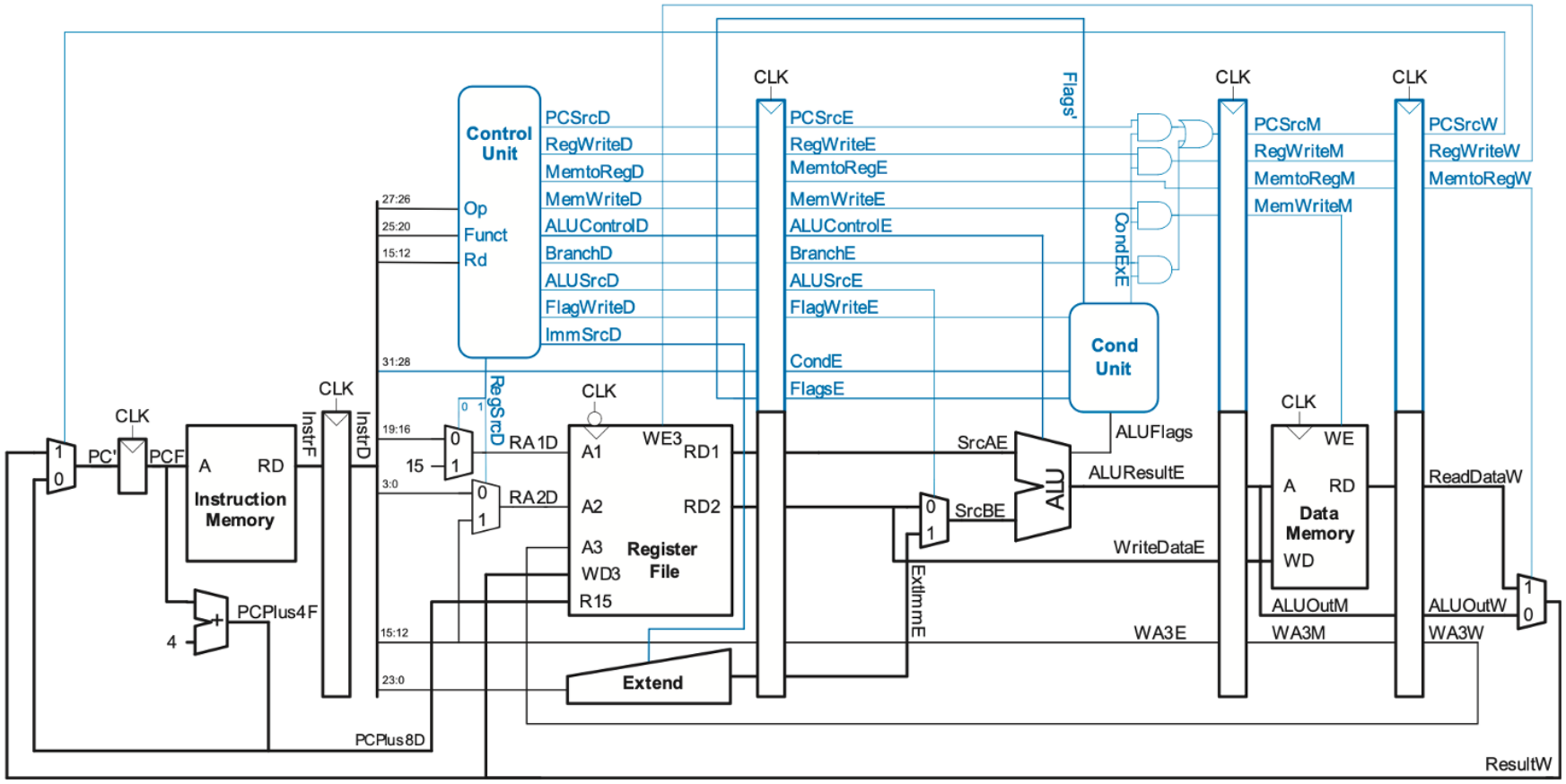


Figure 7.47 **Pipelined processor with control**

Harris and Harris, *Digital Design and Computer Architecture ARM Ed.*, p. 430

Register Set

- R15 is Program Counter (PC)
- R14 is Link Register (LR). Holds return addresses
- R13 by convention used as the stack pointer.
- Four condition codes in current program status register (CPSR)
  - N – negative
  - Z – zero
  - C – carry (unsigned overflow)
  - V – (signed) overflow

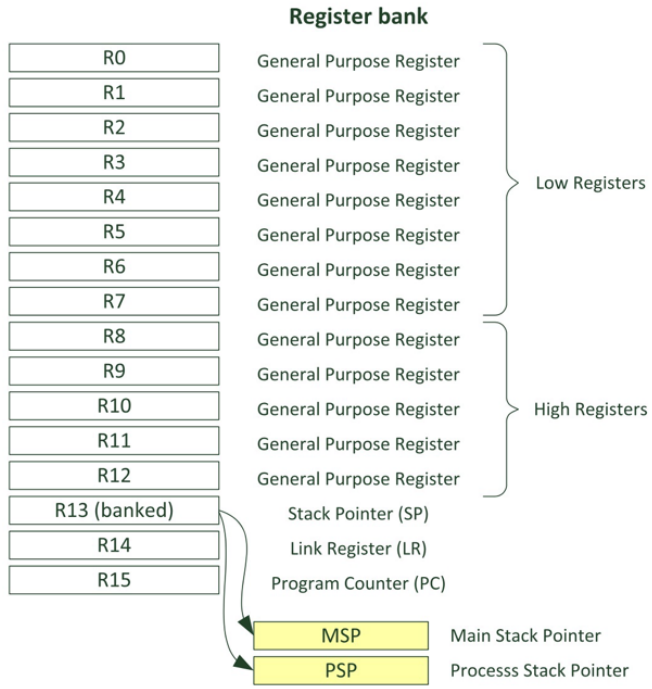


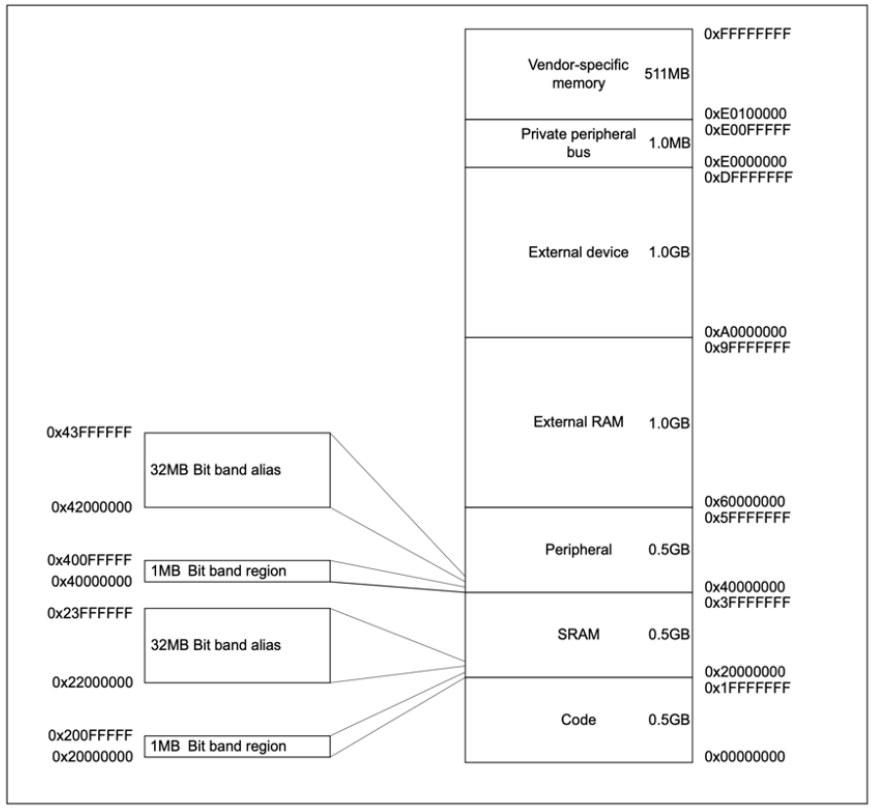
FIGURE 4.3

Registers in the register bank

Memory Map

- Flat 32-bit instruction set
- Addressed in bytes.
- 2^{32} bytes of memory accessible (4 Gigabytes)
- Instructions are always aligned on word (4-byte) in standard ARM and halfword (2-byte) boundaries in Thumb mode.

Figure 8. Memory map

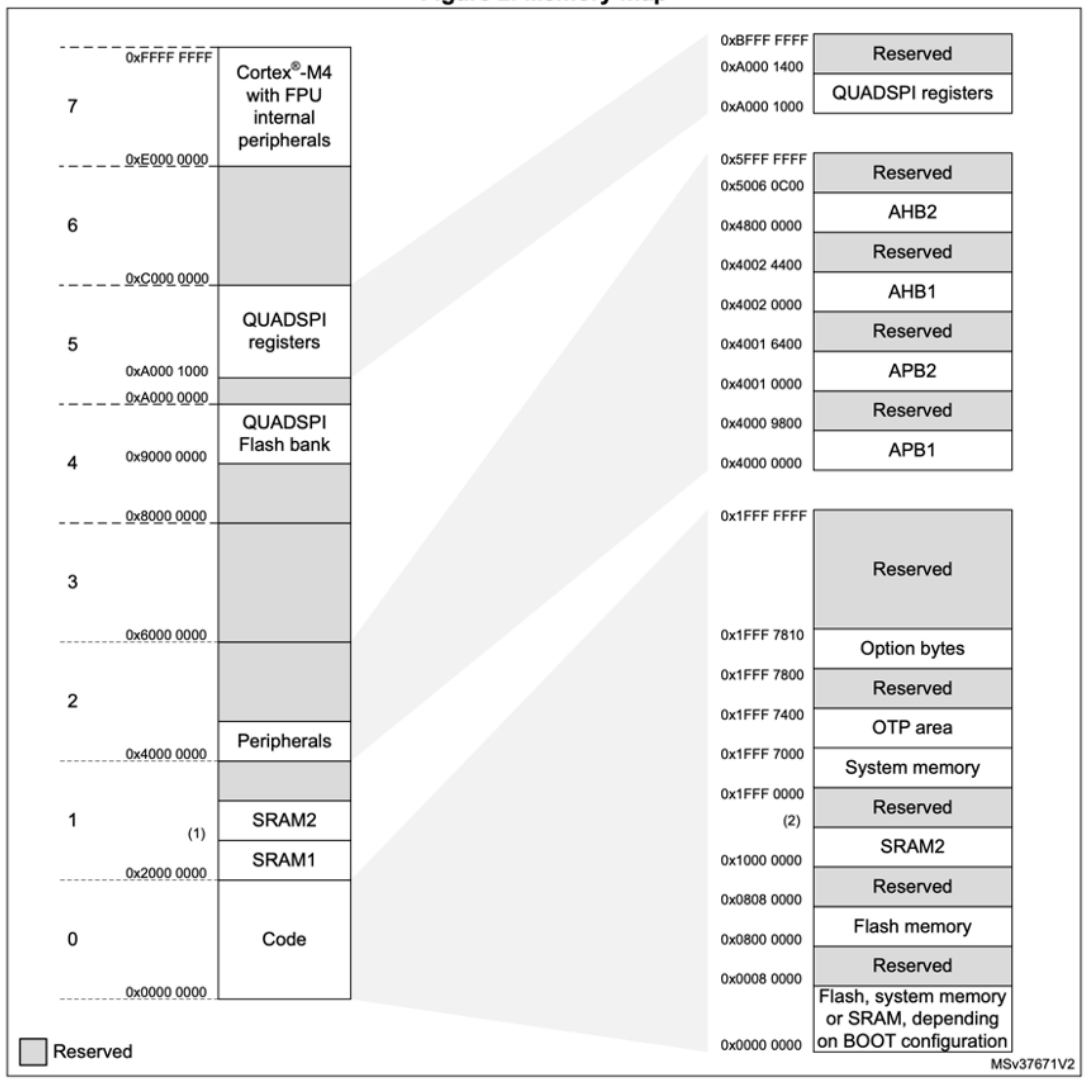


PM0214 Cortex-M4 Programmers Manual

Memory Map - STM32L432KC

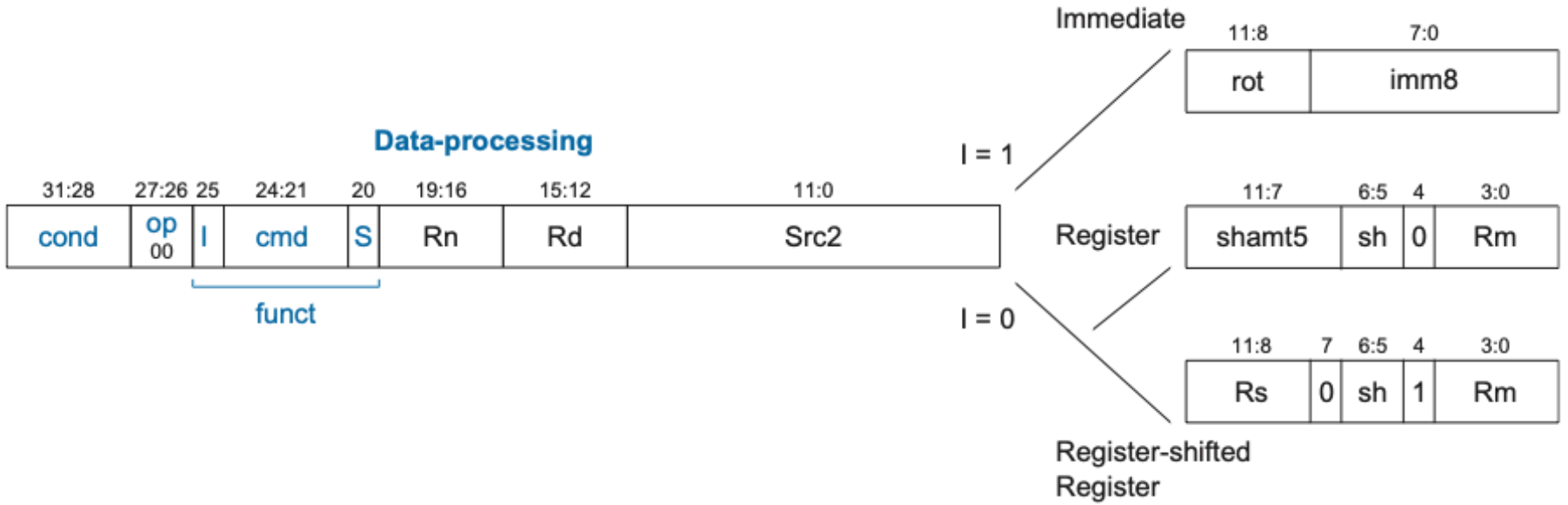
RM0394 p. 67

Figure 2. Memory map



Assembly to Machine Language

```
1 ; c = a & b
2 ; a in R0, b in R1, C in R2
3 AND R2, R0, R1
```



Assembly to Machine Language

```
1 ; c = a & b
2 ; a in R0, b in R1, C in R2
3 AND R2, R0, R1
```

Field	Value
Cond	1110 (unconditional)
I	0 (register mode)
S	0 (doesn’t set condition codes)
Rn	R0 = 0000
Rm	R1 = 0001
Rd	R2 = 0010
Cmd	AND = 0000
Shamt5	sh = 0

Data Processing Instructions

- ADD, SUB, ADDC, SUBC
- AND, ORR, EOR, BIC
- TST, TEQ, CMP
- MOV, MVN, LSL, LSR, ASR, ROR, RRX

cmd	Name	Description	Operation
0000	`AND Rd, Rn, Src2`	Bitwise AND	`Rd ← Rn & Src2`
0001	`EOR Rd, Rn, Src2`	Bitwise XOR	`Rd ← Rn ^ Src2`
0010	`SUB Rd, Rn, Src2`	Subtract	`Rd ← Rn - Src2`
0011	`RSB Rd, Rn, Src2`	Reverse Subtract	`Rd ← Src2 - Rn`
0100	`ADD Rd, Rn, Src2`	Add	`Rd ← Rn+Src2`
0101	`ADC Rd, Rn, Src2`	Add with Carry	`Rd ← Rn+Src2+C`
0110	`SBC Rd, Rn, Src2`	Subtract with Carry	`Rd ← Rn - Src2 -` $\overline{C}$
0111	`RSC Rd, Rn, Src2`	Reverse Sub w/ Carry	`Rd ← Src2 - Rn -` $\overline{C}$
1000 ($S = 1$)	`TST Rd, Rn, Src2`	Test	`Set flags based on Rn & Src2`
1001 ($S = 1$)	`TEQ Rd, Rn, Src2`	Test Equivalence	`Set flags based on Rn ^ Src2`
1010 ($S = 1$)	`CMP Rn, Src2`	Compare	`Set flags based on Rn - Src2`
1011 ($S = 1$)	`CMN Rn, Src2`	Compare Negative	`Set flags based on Rn+Src2`
1100	`ORR Rd, Rn, Src2`	Bitwise OR	`Rd ← Rn | Src2`
1101	Shifts:		
$I = 1$ OR ($\text{instr}_{11:4} = 0$)	`MOV Rd, Src2`	Move	`Rd ← Src2`
$I = 0$ AND ($sh = 00$; $\text{instr}_{11:4} \neq 0$)	`LSL Rd, Rm, Rs/shamt5`	Logical Shift Left	`Rd ← Rm << Src2`
$I = 0$ AND ($sh = 01$)	`LSR Rd, Rm, Rs/shamt5`	Logical Shift Right	`Rd ← Rm >> Src2`
$I = 0$ AND ($sh = 10$)	`ASR Rd, Rm, Rs/shamt5`	Arithmetic Shift Right	`Rd ← Rm>>>Src2`
$I = 0$ AND ($sh = 11$; $\text{instr}_{11:7,\,4} = 0$)	`RRX Rd, Rm, Rs/shamt5`	Rotate Right Extend	`{Rd, C} ← {C, Rd}`
$I = 0$ AND ($sh = 11$; $\text{instr}_{11:7} \neq 0$)	`ROR Rd, Rm, Rs/shamt5`	Rotate Right	`Rd ← Rn ror Src2`
1110	`BIC Rd, Rn, Src2`	Bitwise Clear	`Rd ← Rn & ~Src2`
1111	`MVN Rd, Rn, Src2`	Bitwise NOT	`Rd ← ~Rn`

Data Processing Addressing Modes

- Src 1 (Rn) and destination (Rd) are always a register.
- Src2 (Rm) can be a register or and immediate.
- Registers can be shifted by an immediate or another register.

Condition Codes

- Data processing instructions come with S variant which sets the condition codes based on the result. Don't use these much.
- CMP, TST, TEQ all need the S bit set (but we don't write it in the name)

cond	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\overline{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\overline{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\overline{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\overline{V}$
1000	HI	Unsigned higher	$\overline{Z}C$
1001	LS	Unsigned lower or same	Z OR $\overline{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\overline{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	Z OR $(N \oplus V)$
1110	AL (or none)	Always / unconditional	Ignored

Demystifying CMP, CMN, TEQ, TST

- CMP is SUBS but the result is not written
- CMN is ADDS but the result is not written
- TEQ is EORS but the result is not written
- TST is ANDS but the result is not written

Conditional Execution in ARM Thumb-2

Can use either “if-then” instruction block or a conditional branch

IT blocks overview

- Can support up to four instructions (including the IT instruction)
- Condition codes are not set by instructions in the IT block
- Q: Why do this instead of branching? A: Avoid branching penalties.

IT Block Syntax

The syntax for an if-then block is:

```
1  IT<x><y><z> <cond>
```

- <x>, <y>, <z> are optional and must be either ”T” (then) or “E” (else)
- <cond> is required and must reflect one of the condition codes in the Application Program Status Register (APSR)
- Else conditions must be the opposite of the if conditions.

IT Block Example

C Code Snippet

```
1 if (R4 == R5) {
2   R3 = R1 + R2;
3   R3 /= 2;
4 }
5 else
6 {
7   R3 = R6 + R7;
8 }
```

Assembly Snippet

```
1 CMP R4, R5
2 ITTE EQ
3 ; if R4=R5,R7=R8+R9
4 ADDEQ R7, R8, R9
5 ; if R4=R5,R7/=2
6 ASREQ R7, R7, #1
7 ; if R4!=R5,R7=R10+R11
8 ADDNE R7, R10, R11
```

Branches

- B – branch
- BL – branch and link (saves PC + 2/4 in link register)
- BX/BLX – branch/branch and link + exchange instruction set (go from ARM to Thumb mode or vice versa.

Memory Addressing

- LDR – load register
- STR – store register
- LDRB – load register byte
- STRB – store register byte
- LDRSB – load register signed byte

ARM Assembly Language Programming

C Code Snippet

```c
unsigned char a[32]; // a in R0
unsigned char b; // b in R1
b = a[6];
```

Assembly Snippet

```
LDRB R1,[R0,#6]
```

ARM Assembly Language Programming

C Code Snippet

```c
int a[40]; // a in R0
int b, c; // b in R1, C in R2

b = a[c];
```

Assembly Snippet

```
LDR R1, [R0, R2, LSL #2]
```

Wrap Up

- Microcontrollers are a microprocessor surrounded by peripherals (additional special purpose blocks of hardware for serial communication, general purpose I/O, timing, etc.)
- High level code in C is compiled to assembly code which is then translated into machine code to be stored in the memory of the MCU.
- When learning a new MCU, it’s important to understand the major features of the architecture like the register set, memory map, memory addressing modes, etc.