SELF STUDY

in

MICROCONTROLLERS [15ES4GCMCS]

ARMv7 Architecture

by

Akshit Bhalla

[1BM16EC015]

Under the supervision of

Dr. Suma M N

March 7th, 2018

Department of Electronics and Communication Engineering

B.M.S COLLEGE OF ENGINEERING, Basavanagudi
Bangalore-560019, India

1 Introduction

The hardware board chosen for this project is the Pico i.MX7D, which implements the ARM Cortex-A7 core, which operates at speeds of up to 1.2 GHz, as well as the ARM Cortex-M4 core. The dual-core architecture enables the device to run a rich operating system like Linux on the Cortex-A7 core and an RTOS on the Cortex-M4 core.

The ARM Cortex-M4 and Cortex-A7 range of microcontroller cores are high performance and low power 32-bit RISC processors.

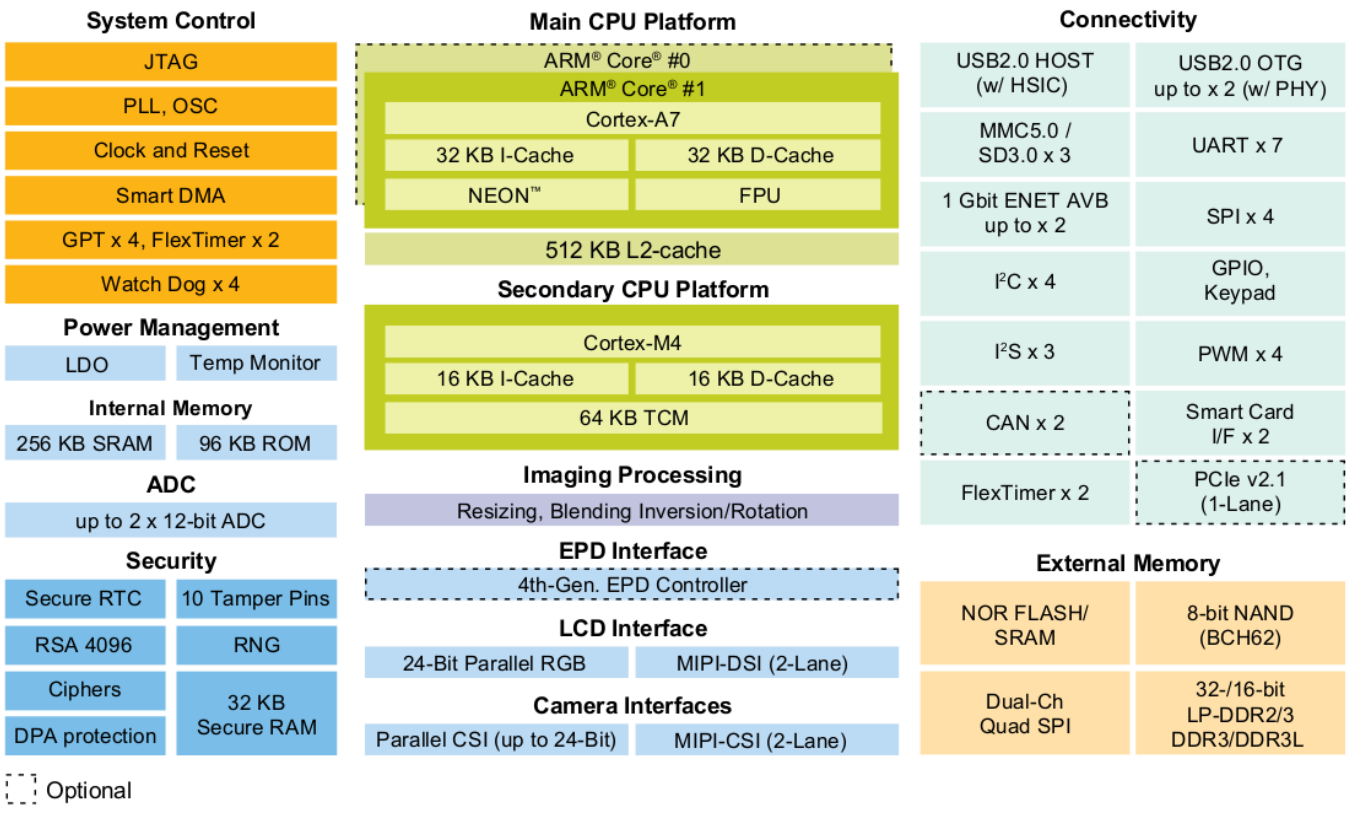


Figure 1: Block Diagram of Pico i.MX7D

1.1 Cortex-A7

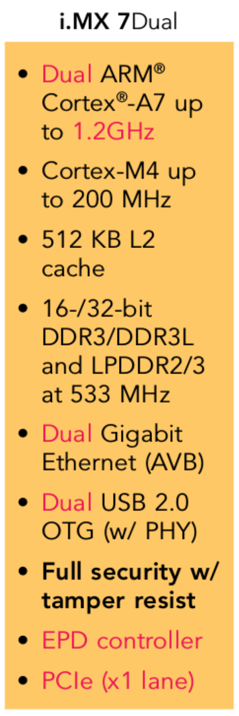
The Application-A profile defines an architecture aimed at high performance processors, supporting a virtual memory system using a Memory Management Unit (MMU) and therefore capable of running fully featured operating systems. Support for the ARM and Thumb instruction sets is provided

- Modified Harvard Architecture (separate, concurrent access to instructions and data).
- Load/Store Architecture.
- Thumb-2 technology as standard. ($discussed\ under\ Cortex-M4$)
- Less than $0.5mm^2$, using $28nm$ process technology.
- Floating Point Unit (FPU)

1.2 Cortex-M4

The Microcontroller-M profile defines an architecture aimed at low cost systems, where low-latency interrupt processing is vital. Cortex-M processors differ from other processors in ARMs range in that they execute only Thumb-2 instructions and do not support the complete ARM instruction set.

- Efficient Harvard architecture 3-stage pipeline core (Fetch-Decode-Execute).
- 32-bit processor, including 32-bit address and data buses.
- 4GB address space and up to 2GB for RAM (either on or off chip).
- A Nested Vectored Interrupt Controller (NVIC) for low-latency interrupt processing.
- Memory Protection Unit (MPU).
- Fixed Memory Map for easy porting of software between systems based on Cortex-M4.



i.MX 7Dual

- Dual ARM® Cortex®-A7 up to 1.2GHz
- Cortex-M4 up to 200 MHz
- 512 KB L2 cache
- 16-/32-bit DDR3/DDR3L and LPDDR2/3 at 533 MHz
- Dual Gigabit Ethernet (AVB)
- Dual USB 2.0 OTG (w/ PHY)
- **Full security w/ tamper resist**
- EPD controller
- PCIe (x1 lane)

Figure 2: Features

Thumb Instruction Set

Thumb instructions are each 16 bits long, and have a corresponding 32-bit ARM instruction that has the same effect. The main reason for using Thumb code is to reduce code density. Because of its improved density, Thumb code tends to cache better than the equivalent ARM code and can reduce the amount of memory required. The complete ARM instruction set may be used for particular code sections which require the highest performance.

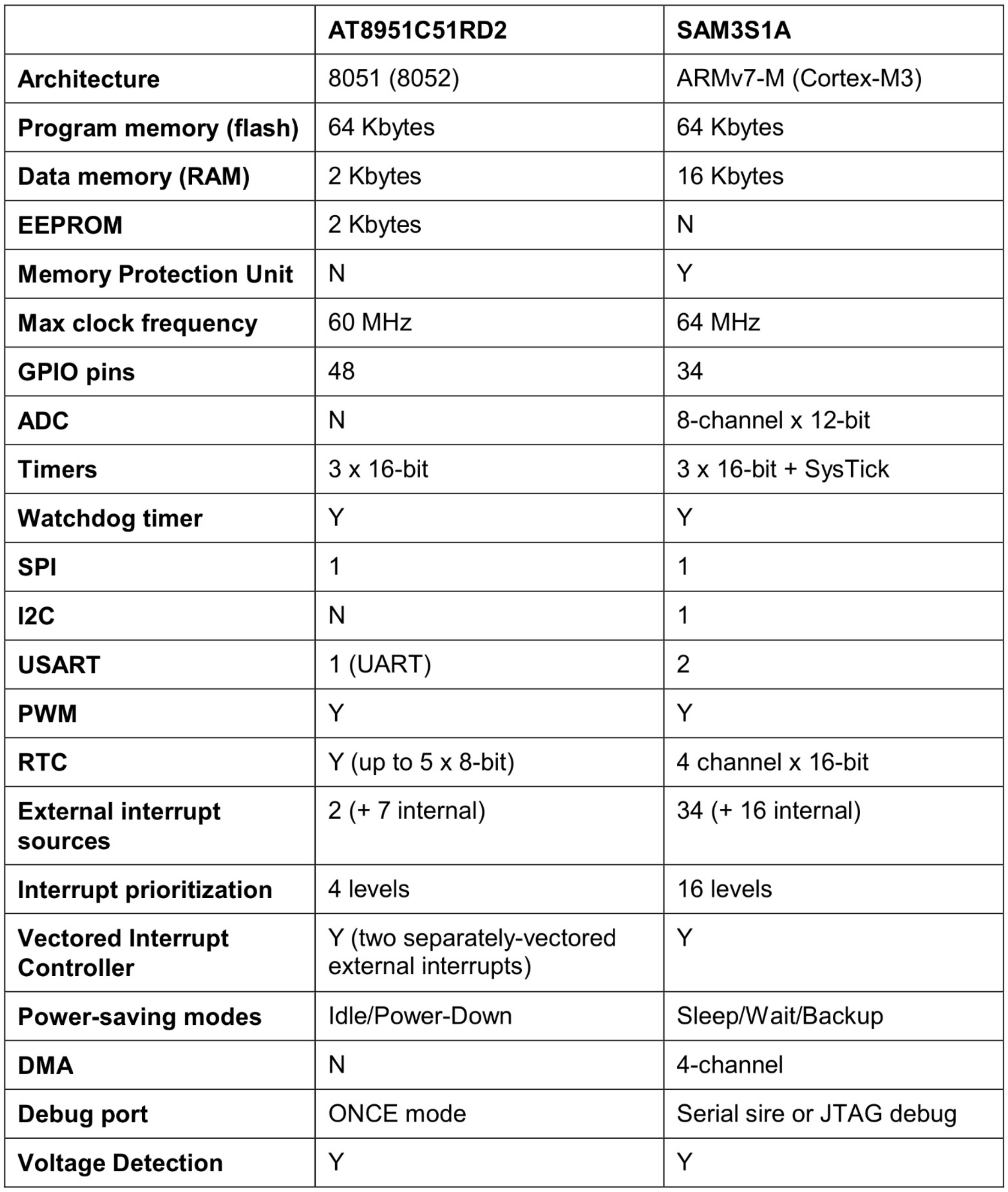
1.3 I/O

1. Communication Ports (I2C, SAI, UART, SPI, SDIO, USB, PCIe).
2. Parallel and MIPI CSI Camera Interfaces.
3. Parallel RGB and MIPI DSI Display Interfaces. The processor integrates an EPD controller that supports E-INK color and monochrome panels with up to 2048 x 1536 resolution at 106 Hz refresh, 4096 x 4096 resolution at 20 Hz refresh and 5-bit grayscale (32-levels per color channel).
4. 2 x 10/100/1000 Mbps Ethernet controllers.
5. The multilevel Cortex-A7 memory system is based on the L1 instruction and data caches, L2 cache, and internal and external memory. The processor supports many types of external memory devices, including DDR3, DDR3L, LPDDR2 and LPDDR3, NOR Flash, NAND Flash (MLC and SLC), QSPI Flash.

2 Comparision with 8051

Direct comparisons between two such different products are difficult. Both are available in many different configurations. The Cortex-M series is arguably more standardized than the 8051, which exists in several different variants (some of which are obsolete).

For the purposes of comparison, I've selected two devices from chip manufacturer Atmel. For the 8051 architecture, we will look at the AT89C51RD2; for the ARMv7-M, the SAM3S1A (Cortex-M3).



	AT8951C51RD2	SAM3S1A
Architecture	8051 (8052)	ARMv7-M (Cortex-M3)
Program memory (flash)	64 Kbytes	64 Kbytes
Data memory (RAM)	2 Kbytes	16 Kbytes
EEPROM	2 Kbytes	N
Memory Protection Unit	N	Y
Max clock frequency	60 MHz	64 MHz
GPIO pins	48	34
ADC	N	8-channel x 12-bit
Timers	3 x 16-bit	3 x 16-bit + SysTick
Watchdog timer	Y	Y
SPI	1	1
I2C	N	1
USART	1 (UART)	2
PWM	Y	Y
RTC	Y (up to 5 x 8-bit)	4 channel x 16-bit
External interrupt sources	2 (+ 7 internal)	34 (+ 16 internal)
Interrupt prioritization	4 levels	16 levels
Vectored Interrupt Controller	Y (two separately-vectored external interrupts)	Y
Power-saving modes	Idle/Power-Down	Sleep/Wait/Backup
DMA	N	4-channel
Debug port	ONCE mode	Serial sire or JTAG debug
Voltage Detection	Y	Y

Figure 3: Comparision Table

2.1 Register Set

1. 8051:
   (a) Two 8-bit registers (ACC and B) which are generally used in arithmetic and logic instructions.
   (b) 16-bit address register, DPTR, which is used exclusively for addressing external memory.
   (c) 4 banks of eight 8-bit registers which are mapped into the first 32 bytes of internal RAM. These are byte and bit addressable. The bank in current use is selected via the RS0 and RS1 bits in the PSW (see below).
2. ARMv7-M
   (a) 16 32-bit registers, R0-R15. R0-R12 are generally available for essentially all instructions.
   (b) R13 is used as the Stack Pointer, R14 as the Link Register (for subroutine and exception return) and R15 as the Program Counter. None of the registers are directly addressable.
   (c) Since the registers are all 32-bit, it supports 32-bit arithmetic and logic operations efficiently.

2.2 Addressing Modes

Name	**Alternative Name**	**ARM Examples**
Register to Register	Register Direct	MOV R0, R1
Absolute	Direct	LDR R0, MEM
Literal	Immediate	MOV R0, #15 ADD R1, R2, #12
Indexed, Base	Register Indirect	LDR R0, [R1]
Pre-Indexed, Base with Displacement	Register Indirect with Offset	LDR R0, [R1, #4]
Pre-Indexed	Register Indirect Pre-Increment	LDR R0, [R1, #4]!
Post-Indexed	Register Indirect Post-Increment	LDR R0, [R1], #4
Double Register Indirect	Register Indirect Register-Indexed	LDR R0, [R1, R2]
Double Register Indirect with Scaling	Register Indirect Indexed-with-Scaling	LDR R0, [R1, r2, LSL #2]
Program Counter Relative		LDR R0, [PC, #offset]

Table 1: Types of Addressing Modes

1. For Thumb Instructions, the destination register cannot be PC (Program Counter) or SP (Stack Pointer).
2. For ARM instructions:
   (a) Destination must be an even-numbered register.
   (b) Destination must not be LR (Link Register).
   (c) In a double-word (64-bit) destination, second destination register must be R(t + 1).

2.3 Status Registers

- The 8051 PSW register contains Carry (CY), Auxiliary Carry (AC) and Overflow (OV) flags (set on results from ALU operations), RS0 and RS1 (used to select one of four register banks in current use) and a parity flag, P (set by the hardware to indicate whether there are an even or odd number of set bits in the accumulator on each instruction cycle).
- The Cortex-M4 Program Status Register (xPSR) is a single 32-bit register with several aliases, each providing a view of a different subset of the contents.
  From the user point of view, the Application Program Status Register (APSR) contains the ALU status flags. For operating system and exception handling use, the Interrupt Program Status Register (IPSR) contains the number of the currently executing interrupt (or zero if none are currently active). The Execution Program Status Register (EPSR) contains bits which reflect execution status and is not directly accessible.

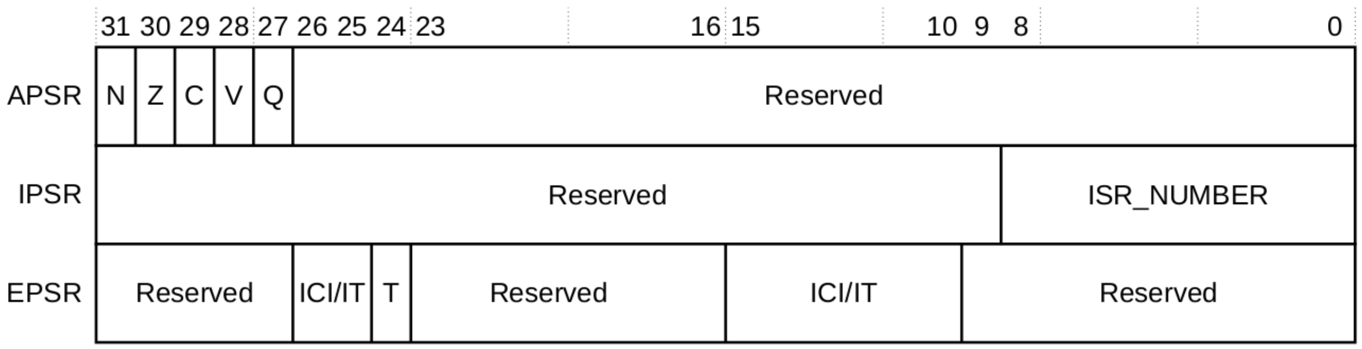


Figure 4: The Program Status Register (xPSR) combines APSR, IPSR and EPSR

2.4 Stack

1. 8051:
   (a) The 8051 supports a Full Ascending stack (byte-sized) using a single stack pointer, SP.
   (b) SP is initialized to 0x07, meaning that the first item pushed on to the stack goes to address 0x08.
   (c) It cannot grow beyond the end of internal RAM at 0xFF. This means that the 8051 only supports 248 bytes of stack space.
   (d) The 8051 stack pointer is typically initialized to the address one byte below the start of the allocated stack area so that the first push uses the first byte of the area (since the stack model is Full Ascending the stack pointer is incremented before the first store).
2. ARMv7-M
   (a) The Cortex-M4 supports a Full Descending stack (word-sized, 32 bits) addressed by the current stack pointer.
   (b) This stack can be located anywhere in RAM. Typically, for best performance, it will be located in internal SRAM. Stack size is limited only by the available RAM space.
   (c) The stack pointer is typically initialized to the word above the top of the allocated stack area. Since the stack model is Full Descending, the stack pointer is decremented before the first store, thus placing the first word on the stack at the top of the allocated region.

3 Programming

The 8051 architecture strictly segregates code and data within the memory map. Further, it requires that data items be assigned to one of the available memory regions.

- **data** Internal on-chip RAM, directly addressed using 8-bit addresses and restricted to 128 bytes or less.
- **idata** Internal on-chip RAM, indirectly accessed using 8-bit addressing. Limited to 256 bytes, the lower half of which overlaps the data region.
- **bdata** Internal on-chip bit-addressable RAM.
- **xdata** External data memory, accessed using 16-bit addressing and restricted to 64Kbytes or less.
- **pdata** One page of external data memory, limited to 256 bytes and accessed using 8-bit addressing.

The ARMv7-M architecture of the Cortex-M4 supports a single, unified address space. All memory access instructions can be used to address items in all regions of memory. This is much more flexible, allowing code and data to be essentially freely allocated across all available memory regions.

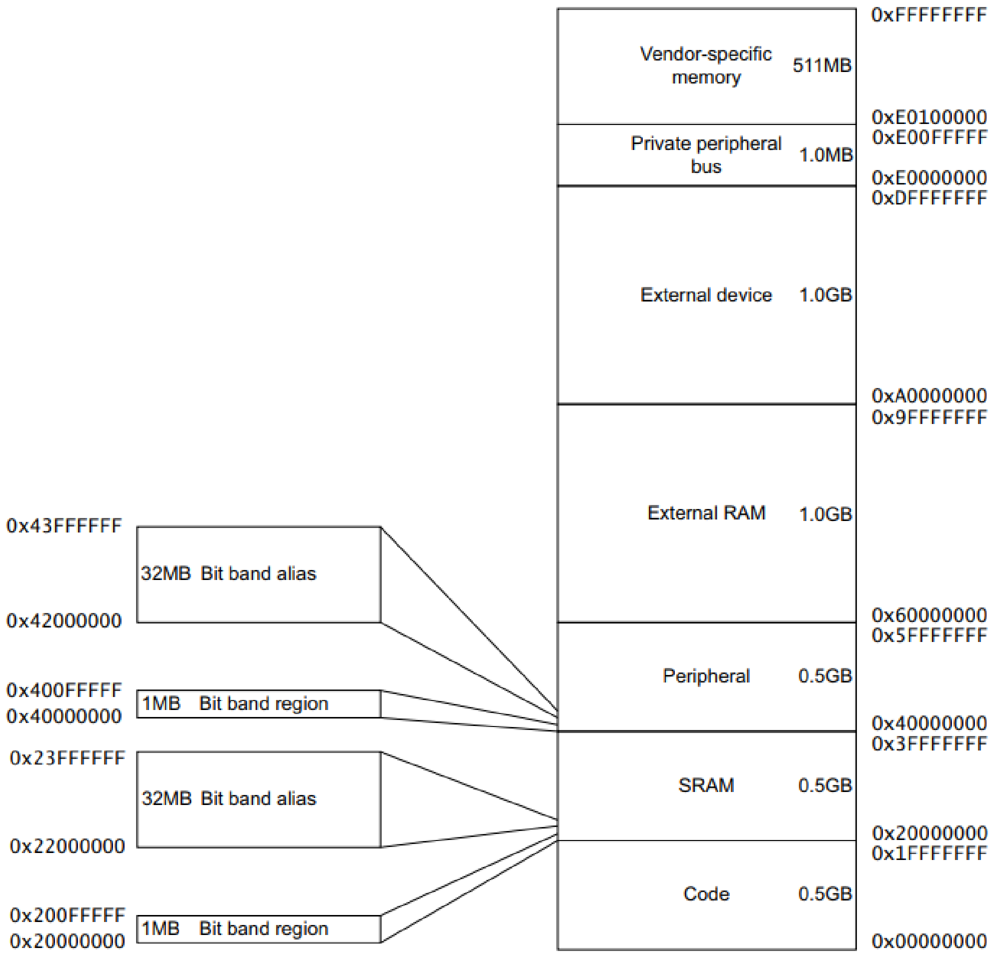
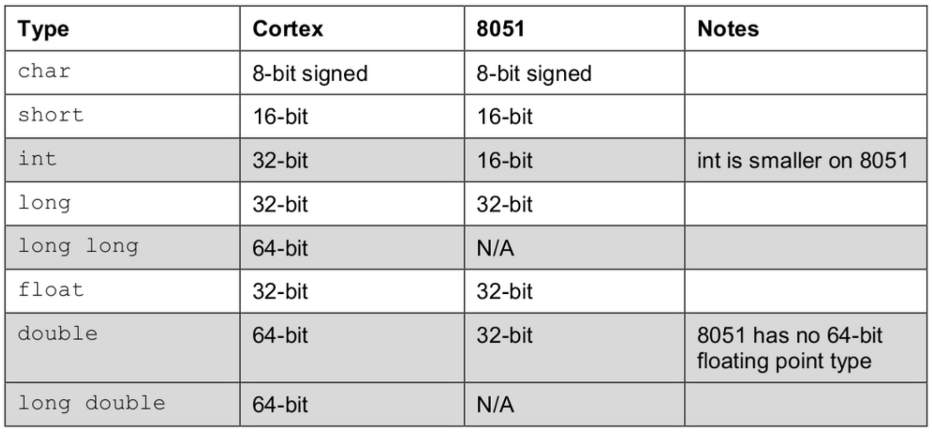


Figure 5: Cortex-M4 Memory Map

3.1 Data Types

The ARMv7-M is a 32-bit architecture and, as such, handles 32-bit types very efficiently. In particular, 8-bit and 16-bit types are less efficiently manipulated, although they will save on data memory if this is an issue.

The sbit keyword is used to declare items which are bit-addressable on 8051. This is unnecessary in the Cortex-M4 as bit-banding provides bit-addressable accesses via standard memory access instructions to aliased addresses.



Type	Cortex	8051	Notes
`char`	8-bit signed	8-bit signed	
`short`	16-bit	16-bit	
`int`	32-bit	16-bit	int is smaller on 8051
`long`	32-bit	32-bit	
`long long`	64-bit	N/A	
`float`	32-bit	32-bit	
`double`	64-bit	32-bit	8051 has no 64-bit floating point type
`long double`	64-bit	N/A	

Figure 6: Data Types

3.2 IDE Availability

There are numerous software options avaialable for development on the ARMv7 platform. Some of the popular IDEs are listed below.

- Keil MDK - ARM
- Visual Studio by Microsoft as IDE, with GNU Tools as compiler/linker
- GNU ARM Eclipse
- ARM Development Studio 5
- Atmel Studio (for Atmel manafactured microcontrollers)

Important : Cortex-A7 and Cortex-M4 are immune to the recently discovered Spectre and Meltdown vulnerabilities found in Intel x86, AMD and various other chips due to the nature in which Speculation Processing is handled in these cores.
`https://www.raspberrypi.org/blog/why-raspberry-pi-isnt-vulnerable-to-spectre-or-meltdown/`

`Sample Code:`

```
1. @ Hello World
    .global _start
    _start:
      MOV R7, #4 @ Syscall to output to screen
      MOV R0,  #1 @ Monitor output stream
      MOV R2, #12 @ String Length
      LDR R1, =message @ Load register with address of string
      SWI 0
    end:
      MOV R7, #1 @ Exit syscall
      SWI 0
    .data @ Signify that what follows is data
    message:
      .ascii "Hello World\n"
```

2. Multiply values (Must store values in Registers)

```
  .global _start
  _start:
  MOV R1, #0x14 @ Put 20 in R1
  MOV R2, #0xA @ Put 10 in R2
  MUL R0, R1, #0xA @ Multiply 20 * 10, store in R0
  MOV R7, #1 @ Exit with a system call
  SWI 0 @ End program and return to terminal
```

3. Compare 2 Values

```
 .global _start
  _start:
    MOV R1, #5
    MOV R2, #10
    CMP R1, R2 @ Compare values setting flags
    BEQ vals_equal @ If equal jump to vals_equal (Zero Flag Set)
    BGT r1_gt @ If R1 is greater jump to r1_gt (Negative Flag Set)
  @ Come here next by default
  r1_lt:
    MOV R0, #2
    B end
  r1_gt:
    MOV R0, #3
    B end
  vals_equal:
    MOV R0, #1
  end:
    MOV R7, #1
    SWI 0
```

References

[1] *Cortex-M4 Technical Reference Manual.* ARM, 1993. https://developer.arm.com/docs/100166/0001

[2] Pico i.MX7D *Datasheet.* NXP, 2017. https://www.nxp.com/docs/en/data-sheet/IMX7DCEC.pdf