Assignment Cover Sheet

COMP2121 | Individual Assignment

Never Stand Still | Faculty of Engineering | School of Computer Science and Engineering

SUBMISSION DETAILS

For submission to (staff name): Sri Parameswaran

Course code: COMP2121	**Course name:** Microprocessors and Interfacing

Assignment details: Assignment 1

ACADEMIC REQUIREMENTS

STUDENT DECLARATION OF ACADEMIC INTEGRITY

STUDENT DETAILS

zID: z5075490	**Date:** 7/04/2018
Family Name: Tawfik	**First Name:** Abanob

Student Signature___BT____________________ **Date** ___7/04/2018__________________

Table of Contents

Overview

ARM microprocessors are a type of “reduced instruction set computer” structure, also known as RISC, which require less cycles in an instruction. Having fewer cycles in instructions equates to less transistors which in turn leads to having cheaper, more power efficient machines. These qualities make ARM desirable for smaller devices which are powered by batteries including smartphones, laptops, tablets etc, and even in some cases a power efficient alternative in super computers.

Pipelining

ARM pipeline uses a three-stage pipelining system which allows operations to perform concurrently. The three-stage system is as followed

1. Fetch - the instruction from memory
2. Decode - the registers required in the instruction
3. Execute – Registers are read from register bank, bit shifting and arithmetic is performed, Registers are written back into register bank

AVR in contrast has a two-stage pipelining system which allow for operations to be performed relatively faster than ARM

1. Fetch – the instruction from memory
2. Execute – Registers are read from register bank, bit shifting and arithmetic is performed, Registers are written back into register bank

Architecture

As mentioned above ARM is a type of RISC architecture. In contrast AVR uses a modified Harvard 8-bit RISC Architecture. ARM architectures have the following design features.

- One cycle execution time - due to the optimised instruction set the architecture has a one clock per instruction
- Large amount of registers – to prevent a large amount of interaction with memory, RISC architectures have a large amount of registers.
- 32-bit architecture, registers from r0-r15 4 bytes (32-bits, 2 words)

AVR Architecture have the following design features.

- 8-bit architecture
- Program memory – flash memory (non-volatile)
- Data memory – SRAM, register file, input output registers.
- Registers – 32 single byte registers r0-r31, (8-bit/ half word)
- Ports – General purpose input output ports
- EEPROM – more permanent data storage, similar to flash memory it can be retained without an electrical supply.

However, both AVR and ARM architects use a load store system which splits all instructions into two parts

1. Access in memory to load or store into the desired register
2. Arithmetic operations which occur between registers

Conditional execution

ARM architecture allows instructions to be executed conditionally, and unlike AVR which allows for conditional execution with branch and rjmp etc. ARM conditional execution can be used with most prompts. This is all done using 4 status registers, N Z C V in the instruction.

N – Negative, if the result instruction was negative
Z – Zero, if the result instruction was 0
C – Carry, if the result in instruction e.g. Addition of 2 integers results in integer larger than 32 bits
V – Overflow, if the result of instruction gives signed result larger than 31 bits
These flags are either 0 or 1 and are set to 1 when condition is met. This allows for optimal conditional executing in ARM. AVR also uses a similar system with status registers.

Hardware support for power saving

ARM processors have multiple methods of power saving which are designed into the hardware. ARM processors are in general slower than other processors which require less power. Due to having a RISC instruction set as mentioned above, this requires less transistors. Larger processes are broken down into small simpler processors which allows more work to be handled by machine code. This allows parts that aren't being used to not use power. ARM processors also save power by going to sleep until an instruction is received. All these features in the RISC architecture alongside allowing parts to sleep unless needed, leads to power saving.

Caching

ARM utilises a cache architecture to increase overall performance. A cache consists of both an instruction cache and data cache. The cache consists of a tag, it's index, a word and a byte. This allows quick access to an instruction and data (instruction stored in word and data stored in byte). Caches are pieces of memory which vary, that can store temporary information.

Hardware support for floating point operations

ARM processors have no built in floating point hardware, instead it uses an external coprocessor known as vector floating point (VFP) alongside the ARM processor. VFP supports single and double precision through the implementation of IEEE floating-point system, with 1 bit for the sign, 8 bits for the exponent and 23 bits for the fractional component.

Memory models

Memory spaces in ARM

ARM microprocessors have only one single large memory space which can vary in size depending on the microprocessor or the usage. ARM processes split the memory space into sections for code, SRAM, peripherals, external RAM, external device, private peripherals and vendor specifics. Each of these specific features have a region in the memory space accessed via bus. This is vastly different to AVR which contains three different memory spaces.

Purpose of memory space in ARM

Since there is only a single memory space in ARM, it is responsible for containing the code, SRAM, peripherals, external RAM, external device, private peripherals and vendor specifics. The memory space in ARM is responsible for everything. In AVR however there are three memory spaces each responsible for a different section

- the data memory – holds general purpose and i/o registers, also stores SRAM
- the program memory - holds the flash memory
- EEPROM – holds EEPROM space (similar to flash memory)

Memory size for the memory space

The maximum memory size for the memory space in ARM is 4gb (32-bit processor).

Registers

Available Registers

ARM architecture contains the following 32 registers, however only 15 are available for user use. The following registers are able to be used by user

- 13 general purpose registers R0-R12
- Stack Pointer
- Link Register

These registers are used in the back end and debugging etc.

- Program Counter

- Application Program Status Registers

General purpose register comparison

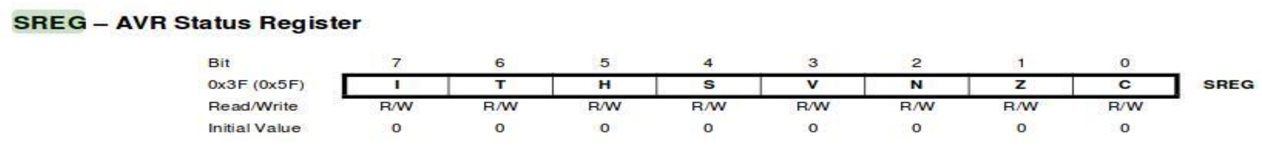
ARM contains 13 general purpose registers, whereas AVR contains 32 general purpose registers. However, ARM registers are also 32-bit registers (2 words) whereas AVR registers are 8-bit (half words). AVR registers have two categories, general registers and i/o registers. AVR has 32 general purpose registers, R0-R31 as well as 64 i/o registers.

ARM equivalent of SREG and differences

ARM's equivalent to SREG is known as the Current Program Status Register (CPSR). A big difference is that the CPSR does not contain the half carry flag, also that the SREG only stores information on application program statuses. THE CPSR not only stores most of these flags but it also contains information on

- The current processor mode
- Processor state
- Endianness state
- Execution state
- Interrupt/disable flags

Below in Figure 1, is a visual representation of SREG



SREG – AVR Status Register

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 1 A visual representation of the status register

Source: http://archive.fabacademy.org/2017/fablabakgec/students/462/week8_Embedded%20Programming/assignment8.html

Below in Figure 2, is a visual representation of CPSR

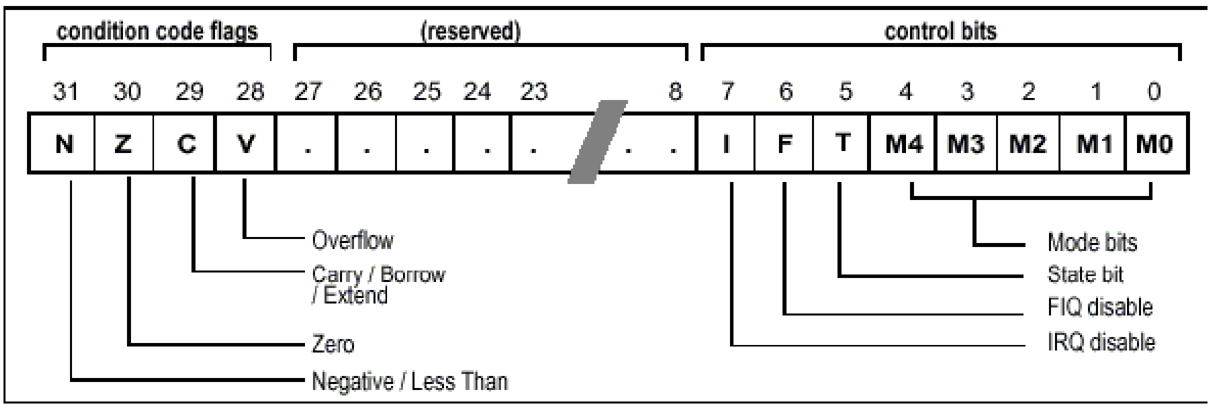


Figure 2 A visual representation of the CPSR

Source: https://stackoverflow.com/questions/21226577/arm-cpu-mode-svc-instruction

ARM interrupt system comparison

AVR interrupt works in the following way

1. Global interrupt bit is enabled by hardware or software
2. The I bit is cleared and the address of the next instruction is pushed onto the stack
3. Interrupt service routine (ISR) is executed, (interruption handler code)
4. The interrupt is handled by the ISR (usually a handle for each interrupt)
5. Return using reti

ARM on the other hand has the following two interrupt methods, fast interrupts and normal interrupts. With two flags in the CPSR, I and F
ARM Regular interrupt request
These are lower priority interrupt request and it is masked before it becomes a fast interrupt request sequence, the request is kept at low priority until the processor acknowledges the interrupt request which is done by the software handler. Similar to AVR interrupts the same procedure is taken on interrupt.
ARM fast interrupt request
These are higher priority interrupts, during a fast interrupt request, all other interrupt requests are masked, meaning an interrupt request cannot interrupt a fast interrupt request handler. This only works one way, and regular interrupt requests don't mask fast interrupt request meaning that fast interrupt request are able to interrupt regular interrupt requests.
Unlike ARM, AVR has no access to a fast interrupt method, however the interrupt protocol is the same between the two architectures, the interrupt is called, values and statuses are saved, then handler is executed, then return back to previous post counter + 1.

Instruction Set

Comparing encoding schemes

ARM instructions have a fixed size of 32-bits. Instructions are encoded as follows

- Bit 31-28 are conditional field bits
- Bits 27-26 are data processing bits
- Bit 25 is immediate bit (if on it means operand is not a register)
- Bit 24-21 has the opcode field which defines what the function does, e.g. 0100 -> ADD
- Bit 20 is the set condition which is optional
- Bit 19-16 are the bits for operand 1
- Bit 15-12 are for the destination
- Bit 11-0 are for operand 2

Below in figure 3, this is clearly illustrated

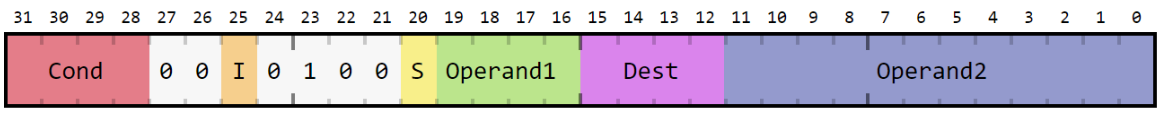


Figure 3 the 32 bits of an instruction

source: https://alisdair.mcdiarmid.org/arm-immediate-value-encoding/

Since operand2 is only 12 bits this limits the range from 0-4095, which is not desired when working with 32-bit systems, so to overcome this, the operand2 is split into two sections illustrated below in figure 4

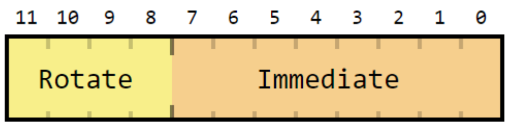


Figure 4 the in-depth view of operand 2

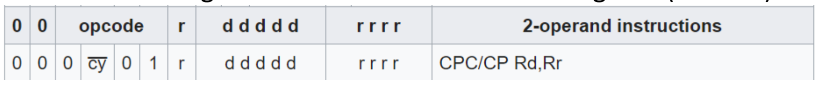
source: https://alisdair.mcdiarmid.org/arm-immediate-value-encoding/

Bit 11-8 are rotational bits, and bits 7-0 are the immediate value, this allows shifting of the immediate value. The 8-bit immediate value is written out and rotated accordingly to provide the desired value. This is a very complex feature to ARM.

In contrast to ARM, AVR instructions are generally 16 bits long, with a few exceptions being 32 bits long such as LDS.
In AVR instructions are encoded in many different formats depending on the instruction called and the bits layout for the instruction encoding is dependent on which instruction, however in general

- 4 bits are used for storing opcode (3 bits in register-immediate operations e.g. subi.)
- 12 bits are used for storing operand and destinations, however this varies on instruction calls

Below in figure 5 the general 2 operand instruction encoding scheme is shown, where ddddd refers to destination register and rrrr refers to source register (r16-r31).

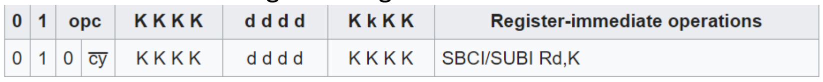


0	0	opcode				r	d d d d d	r r r r	2-operand instructions
0	0	0	$\overline{cy}$	0	1	r	d d d d d	r r r r	CPC/CP Rd,Rr

Figure 5 2-operand instruction encoding scheme

source: https://en.wikipedia.org/wiki/Atmel_AVR_instruction_set#Instruction_encoding

However below in figure 6, the immediate instruction encoding scheme varies different, where the kkkk refers to the unsigned integer and ddddd refers to destination register. Note opcode is 3 bits.

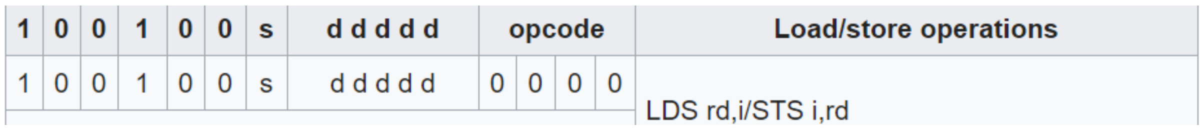


0	1	opc		K K K K	d d d d	K k K K	Register-immediate operations
0	1	0	$\overline{cy}$	K K K K	d d d d	K K K K	SBCI/SUBI Rd,K

Figure 6 immediate instruction encoding scheme

source: https://en.wikipedia.org/wiki/Atmel_AVR_instruction_set#Instruction_encoding

Figure 7 displays the load/store instruction encoding scheme, ddddd refers to destination register and



1	0	0	1	0	0	s	d d d d d	opcode				Load/store operations
1	0	0	1	0	0	s	d d d d d	0	0	0	0	LDS rd,i/STS i,rd

Figure 7 load/store instruction encoding scheme

source: https://en.wikipedia.org/wiki/Atmel_AVR_instruction_set#Instruction_encoding

AVR and ARM instruction encoding schemes vary different, ARM instructions store all the conditional fields, contain more data processing bits and allow for larger operands to be used, whereas AVR have varying encoding schemes specific to instructions and in some cases the opcode is even 3 bits instead of the standard 4-bit opcode encoding. AVR also have a smaller instruction size of 16 bits (only in some functions 32-bits) which differ from the ARM standard of 32-bits.

Function and operation of branching instructions

AVR have branching functions built in, which are listed in table 1 below as follows

Table 1.
ARM established instructions

Instruction	*Operation*
B and BL	Branch and Branch with Link (immediate branches)
BX and BLX	Branch indirect and Branch indirect with link (swaps instruction set to thumb) called on registers

Source: ARM information center (2010)

Predicate functions are called when the T flag is set (when a condition is met) so syntax is generally

B (condition) (label) -> jumps to label. The key difference between B and BL is that when BL returns it stores the return result in the return address lr (the link register). These are immediate branches which use the ARM instruction set, BX and BLX are special branch instructions which swap the processor state to and from ARM->Thumb. Similar to B and BL, BX and BLX differ by the fact that BLX stores the return address in the link register lr.

Comparing instructions used for stack operations

In AVR stack operations are generally as follows, initialise stack using the RAMEND in the following
Ldi r16, low(RAMEND)
OUT SPL, r16
Ldi r16, HIGH(RAMEND)
OUT SPH, r16
And performing stack operations using push and pop where push pushes a register onto stack and pop pops the top register from the stack.
ARM has a different system for stack operations, ARM allows for multiple registers to be pushed and popped from the stack in a single call. ARM also doesn’t require for the stack to be initialised like AVR. AVR can work in full descending or ascending order in the way the stack pushes/pops however in most cases full descending order is the normal. A full descending order operation could be the following
STMFD sp!, {r3-r5} -> store onto stack by push
LDMFD sp!, {r3-r5} -> load from stack by pop
OR
PUSH (r2-r4, lr) ->pushes link register and r2-r4 onto stack
BL somewhereinthecode
somewhereinthecode:
POP (r2-r4, PC) -> pop work registers and program counter from stack

Comparing access of I/O registers

ARM systems use port-mapped I/O instructions to access the functionality of input and output, since the IO is mapped onto the same memory space, only simple load-and-store operations are performed on the registers to interact on an IO port. AVR works differently to ARM in the sense that AVR has a special set of instruction in order to access the I/O registers since the IO registers are separate to the general registers and are mapped in their own space. These instructions are IN and OUT.

Comparing the addressing modes for load/store instructions

The addressing modes which are available for the ARM load and store instructions are

Pre-indexing addressing mode where the address is offset by a register value, for example the line ldr r2, [r0,r1] says load into r2 the address at r0 offset by r1. With pre-indexing offsets are by register values
Post-indexing addressing mode where the address offset is an immediate value, for example the line str r2, [r1], +4 says store into r1 the value in r2, then offset r1 by, register post indexing can also use values store in registers as well.
AVR has multiple addressing modes, however ARM has a lot more addressing mode overall and uses a more complex addressing system. In AVR however the load and store address have a lot simpler choice but cannot perform more complex loads and stores as shown above by off-setting
AVR has access to

1. Immediate addressing – ldi r16, 3 (loads 3 into r16)
2. Direct addressing – sts $FF00, r16, (stores value in address $FF00 into r16)
3. Indirect addressing – ld r16, x (loads the byte in X into r16)
4. Indirect with displacement – ldd r4, Y+1 (load data into r4 from where Y is offset by 1 byte)

AVR has a much simpler addressing mode in comparison with ARM in terms of addressing modes for load/store instruction as ARM allow for multiple offsets and calculations to be done to shit the address in the single line using post/pre-indexing addressing.

Data types

Native data type representation by ARM registers

There are many data types with can be natively represented by ARM's registers, below in table 2 this is shown along with the bits in the data type and the alignment

Table 2.
Data types representable in ARM

Instruction	*Bit Size*	*Alignment*
Character	8	1 (byte aligned)
Short	16	2 (half word aligned)
Integer	32	4 (word aligned)
Long	32	4 (word aligned)
Long long	64	4 (word aligned)
Float	32	4 (word aligned)
Double	64	4 (word aligned)
Long double	64	4 (word aligned)
Pointers	32	4 (word aligned)
Boolean	32	4 (word aligned

Source: ARM information center (1998)

Using the instruction set for 64-bit integer operation

ARM instruction set can be used to do 64-bit integer operations, this is done by using register pairs for operation, one register holds the low bits of the integer, and the other register holds the high bits of the integer. Using this register pair can represent the 64-bit integer in a 32-bit system (similar to lab 1-part A).

Adding 64-bit signed integer in ARM example

The instructions required to add two 64-bit integers would be LDR, ADDS, ADC
Below is the code for adding 64-bit integers in ARM

```
LDR r0, =0x321              ;32 lo bits of integer 1
LDR r1, =0x456              ;32 hi bits of integer 1

LDR r2, =0x123              ;32 lo bits of integer 1
LDR r3, =0x654              ;32 hi bits of integer 2

ADDS r4, r0, r2             ;add the low bits together with the flags (S is for status flags)
ADC r5, r1, r3              ;add along with the carry into the high bits
```

Resultant is in register pair r4:r5.

Conclusion

Main advantages and disadvantages of ARM

ARM has many advantages and few disadvantages listed below in table 3

Table 3.
Advantages and disadvantages of ARM architecture

Advantages	*Disadvantages*
More pipelines allowing much more efficient processing	Harder to learn in comparison to AVR
Power efficient and cost effective due to being RISC architecture and the ability to wait on processes	The simplified instructions may not be enough for much heavier workloads
Lower sized chips and lower power requirements	ARM processors tend to be slower
The register size is 32 bits allowing for better performance	
Inexpensive chips	

Main advantages and disadvantages of AVR

AVR comes with advantages and disadvantages listed in table 4

Table 4.
Advantages and disadvantages of AVR architecture

Advantages	*Disadvantages*
A lot simpler to learn and perform tasks in compared to ARM	8-bit system limits the performance
More specified instruction set	AVR requires a lot more user control on things like memory.
Due to small size better for simple and small applications	AVR processors don't have same performance ability as a 32-bit system
High number of inbuilt i/o registers	
Inexpensive chips	

Which applications are best benefitted by each architecture

ARM being a much more powerful 32-bit architecture, whilst also being lightweight, power efficient and smaller in size leads to high usage in applications such as smart phones, laptops, tablets, televisions, credit cards, alarm systems, thermostats, and even in some cases due to it's power efficiency supercomputers.
AVR is used in smaller and simpler applications where the use of ARM is not required due to it's simplicity, this includes Arduino, for someone learning assembly, smaller projects which don't require the power-saving features and performance of ARM. AVR is a much older system and doesn't see much use due to ARM being a cheap and more powerful alternative with the price of a higher learning curve.

References

Referenced using Harvard referencing style.

ARM Information Center, 2018, The instruction pipeline, [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0084f/ch01s01s01.html
[Accessed 03 April 2018].

Wikipedia, 2018, ARM architecture, [ONLINE] Available from:
https://en.wikipedia.org/wiki/ARM_architecture
[Accessed 03 April 2018].

Chen C. and Novick G. and Shimano K., 2000, What is RISC?. [ONLINE] Available from:
https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/whatis/index.html
[Accessed 03 April 2018].

Wikipedia, 2018, ATMEL AVR, [ONLINE] Available from:
https://en.wikipedia.org/wiki/Atmel_AVR
[Accessed 03 April 2018].

Wikipedia,, 2018, Load/store architecture, [ONLINE] Available from:
https://en.wikipedia.org/wiki/Load/store_architecture
[Accessed 03 April 2018].

David Thomas, 2018, ARM: Introduction to ARM: Conditional Execution, [ONLINE] Available from:
http://www.davespace.co.uk/arm/introduction-to-arm/conditional.html
[Accessed 03 April 2018].

ARM Information Center, 2018, Power-saving features. [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dht0008a/CJAHGJEF.html
[Accessed 03 April 2018].

What makes ARM-based chips relatively power efficient, and what is the trade-off for power consumption?, 2018, Quora, [ONLINE] Available from:
https://www.quora.com/What-makes-ARM-based-chips-relatively-power-efficient-and-what-is-the-trade-off-for-power-consumption
[Accessed 03 April 2018].

ARM Information Center, 2018, About cache architecture. [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0201d/I21752.html
[Accessed 03 April 2018].

ARM Information Center, 2018, Support for floating-point operations. [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0205j/CJAIDCJB.html
[Accessed 03 April 2018].

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0179b/CHDDBAEC.html

ARM Information Center, 2018, Memory map. [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0179b/CHDDBAEC.html
[Accessed 03 April 2018].

AVR tutorials, 2018, AVR Memory Map, [ONLINE] Available from:
http://www.avr-tutorials.com/general/avr-memory-map
[Accessed 03 April 2018].

Dynameo D., 2013, Just a few questions on ARM memory, [ONLINE] Available from:
https://community.arm.com/tools/f/discussions/661/just-a-few-questions-on-arm-memory
[Accessed 03 April 2018].

Arm Limited, 2005, ARM registers, [ONLINE] Available from:
http://www.keil.com/support/man/docs/armasm/armasm_dom1359731128950.htm
[Accessed 03 April 2018].

ARM Information Center, 2018, General-purpose registers, [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0473c/Babdfiih.html
[Accessed 03 April 2018].

AVR Tutorials, 2018, AVR Microcontroller Interrupts, [ONLINE] Available from:
http://www.avr-tutorials.com/interrupts/about-avr-8-bit-microcontrollers-interrupts
[Accessed 04 April 2018].

ARM Information Center, 2018, Interrupts, [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0363g/BEIDDFBB.html
[Accessed 04 April 2018].

Manav m-n, 2018, What is the difference between FIQ and IRQ interrupt system?, [ONLINE] Available from:
https://stackoverflow.com/questions/973933/what-is-the-difference-between-fiq-and-irq-interrupt-system
[Accessed 04 April 2018]

McDiarmid A., 2014, ARM immediate value encoding, [ONLINE] Available from:
https://alisdair.mcdiarmid.org/arm-immediate-value-encoding/
[Accessed 04 April 2018]

ARM Information Center, 2018, Format summary, [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0210c/CACCCHGF.html
[Accessed 05 April 2018].

Guo A., Parameswaran S., 2018, AVR ISA & AVR Programming (I), [ONLINE] Available from:
http://www.cse.unsw.edu.au/~cs2121/LectureNotes/week4_notes.pdf
[Accessed 05 April 2018].

ARM Information Center, 2010, B, BL, BX, BLX, and BXJ, [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489e/Cihfddaf.html
[Accessed 05 April 2018].

ARM Information Center, 1999, Using pointers to access I/O, [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0056d/ch06s09s01.html
[Accessed 05 April 2018].

KMITL, 2018, ARM Cortex M3 &General Purpose Input/Output(GPIO), [ONLINE] Available from:

http://webcache.googleusercontent.com/search?q=cache:Q698uyOUbZ4J:www.ce.kmitl.ac.th/download.php%3FDOWNLOAD_ID%3D6279%26database%3Dsubject_download+&cd=1&hl=en&ct=clnk&gl=au
[Accessed 05 April 2016]

Wikipedia, 2018, Memory-mapped I/O, [ONLINE] Available from:
https://en.wikipedia.org/wiki/Memory-mapped_I/O
[Accessed 03 April 2018].

Clemson cs,2018,ARM Load/Store Instructions, [ONLINE] Available from:
https://people.cs.clemson.edu/~rlowe/cs2310/notes/ln_arm_load_store_plus_multiple_transfers.pdf
[Accessed 03 April 2018].

Atmel, 2009, Addressing models, [ONLINE] Available from:
http://web.csulb.edu/~hill/ee346/Lectures/13-1%20AVR%20Addressing%20Indirect.pdf
[Accessed 03 April 2018].

ARM Information Center, 1998, Basic data types, [ONLINE] Available from:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0041c/ch03s02s02.html
[Accessed 05 April 2018].

Baines R., 2016, Where are ARM processors used?, [ONLINE] Available from:
https://www.quora.com/Where-are-ARM-processors-used
[Accessed 05 April 2018].