# 1

Over the years ARM released newer versions with additional features and improved functionality while keeping the core architecture relativity the same. For version 7 ARM took a forked approach to target specific markets. Cortex-A focusing on creating powerful applications, Cortex-R series for safety-critical real-time processing, and Cortex-M focusing on low power consumption and minimising cost with high efficiency. The benefit of the M4 over the M3 is the M4 has optional FPU and DSP instructions which have reduced system-level complexity, power consumption, materials, and development costs through the capability of high-level programming. Off target a little. 2/5 marks

# 2

The 16bit Thumb instruction set provides additional functionality and can perform most of the standard 32bit ARM instructions. Thumb instructions can be executed through a 32-bit instruction without slowing down execution. The main advantages are additional functionality, condensed code size as well as reduced memory cost to give significant performance improvements. 2/5

# 3

The Cortex-4 has a 3 stage Harvard Architecture and can implement both byte-invariant big-endian and little-endian format for all data memory access. Instruction memory and Private Peripheral Bus must be in little-endian. Endianness can be converted between 32bit, 16bit, and signed 16bit into signed 32bit. 4/5

# 4

The Link Register can store the return address for functions, function addresses, and exceptions. This register can be used as a multipurpose register however it is overridden when needed by the hardware, so this is not recommended. Link Register doesn’t use a stack of registers (?) so there is no pushing or popping involved when changing. Branches instructions such as BL and BLX, are a more stable method of updating the Link Register and the Program Counter as they are done implicitly. 2/5

# 5a

As the Cortex M4 is designed for low power consumption technologies such as FPU and NEON may be unnecessary for algorithms and put a needless burden on power making it perform less efficiently.

Algorithms may not have any use for arithmetical single-precision floating-point. Floating-point arithmetic may also be calculated through a software library so in particular cases it may be more efficient to use software to solve floating-point equations rather than built-in hardware.

Neon technology is aimed at maximising efficiency for processing large amounts of data very quickly such as multimedia. As the Cortex M4 is not solely devoted to such tasks it would certainly put unnecessary strain on the power consumption.

Why is it optional in the hardware?

2/5

# 5b

Floating-point arithmetic may also be calculated through a software library. Depending on the extent of floating-point arithmetic used in a program it may be more efficient to rely on software and save the power that would be allocated to the FPU. 5/5

# 6

The third operand in this instruction is a flexible operand that is capable of being a register with an optional shift. Providing a register (R0) allows for a shift instruction to be placed with a specific number of bits in whichever direction. In this case an arithmetic shift right by 1 bit which divides the integer ‘i’ by 2. The value from the shift is applied before the subtraction but the register is not affected by the shift and the carry flag is updated to the bit shifted out. 10/10

# 7a

The ‘If-Then’ Block is generally more efficient than conditional branches as the number of instructions is often reduced ultimately costs fewer cycles. There are some cases when the IT block would be less efficient, for example, if a condition was to fail at run-time more cycles could be used than an IT instruction. need to explain that. 3/10

# 7b

mylabel: CMP R1,R2 // if ( 5 != 10 ) THEN…

MOVNE R3,R1 // R3 assigned R1 value

ADDSNE R1,R3,R1 // R1 doubled to 10

BNE mylabel // go back to mylabel

mylabel: CMP R1,R2 // if ( 10 != 10 ) ELSE…

ANDEQ R3,R1,#0x03 // R3 = 10 & 3

// 1010 & 0011 = 0010

// **R3 = 2**

# 

The first T in the IT block specifies the condition for the first instruction no, it’s the ‘NE’. You’ll never see an “IE” block. Any following T’s or E’s apply specifies the condition for the second, third or fourth instruction in the block.

The instructions condition must match the IT block condition. The If-then block can process from one up to four instructions. T(Then) applies if the condition is true. E(Else) applies if the condition is false. In the ADDSNE instruction, the ‘**S’** updates the flags. 12/15

# 8a

R0 = 5

10 = 5 \* 21 = 5 << 1

R6 = -1

R0 = -1 + 10 = 9

The value in r0 after the instruction is: 00000009 10/10

# 8b

A logical shift to the left has been applied to the value in r0 (not applied to the register itself) doubling it. This 32-bit value is added to r6(-1) and placing it into r0.

The source C++ for this instruction is: (branchLine \* 2) - 1 10/10

# 8c

R1 has leaves loaded into it for the comparison instruction before branching. 5/5

# 8d

In both cases, the load register instruction offsets the address r4 and retrieves the value stored there and then loads into r0. Where these instructions differ is the addressing mode. 1 is an example of offset addressing where the address r4 is unchanged as a result of the instruction. 2 demonstrates pre-indexed addressing where the offset address is written back to r4 making this method an efficient incrementation for a loop cycling through an array. For offset addressing (1), an addition or subtract function would be required to increment or decrement the array offset which would ultimately bloat code unnecessarily. 10/10