Question Bank for ELEC2142 Theory Test

1. C Language

1. What is the difference between the post-increment (`a++`) and the pre-increment (`++a`) operators in C, in the context of larger expressions?
2. How does pointer arithmetic (addition and subtraction by some integer) translate to real memory values? Use `p = p + 3` as an example (where `p` is declared as a pointer).
3. What is *type casting* in C? When should it be used?
4. What is the difference in C between bitwise operations (`&, |, ~`) and the logical ones (`&&, ||, !`)?
5. Explain when will a C shift-right operation (`>>`) translate into an `lsr` in assembly and when will it translate to an `asr`?

2. Number Systems

6. When specifying constant numbers in ARM assembly, what prefixes are used to denote: *binary*, *octal*, *hexadecimal*, and *decimal* numbers?

3. Programmer's Model

7. Compare the advantages and disadvantages of using the internal registers versus the external memory for data storage.
8. Describe the operation of each stage of the *Fetch-Decode-Execute* cycle.
9. Explain how using a 3-stage (Fetch-Decode-Execute) pipeline helps to optimize the operation of the ARM processor.
10. Explain why instructions that modify the value of the PC (such as branches or `mov pc, lr`) require additional clock cycles to execute than those who don't operate on the PC.
11. How many data registers (excluding the CPSR) are visible to the programmer at any time? What are the hidden data registers and when do they become visible?
12. Name and describe the three main ARM instruction categories. Give an example of an instruction that falls into each one of the categories.
13. What does each of the following register aliases is an abbreviation for? Registers: `a1-a4, v1-v7, fp, ip, sp, lr,` and `pc`.
14. Briefly describe what each of the following registers is used for by convention: `a1-a4, v1-v7, fp, ip, sp, lr,` and `pc`.

4. Data Processing Instructions

15. What do `lsr` and `asr` operations stand for? What is the difference between the two?

16. What numbers constitute as valid immediate values for data processing instructions?
17. What does the `rsb` instruction do? Give an example of an expression that would require the use of `rsb` rather than `sub`.
18. What is the instruction `bic` useful for in the context of bits manipulation using a mask?
19. What is the instruction `orr` useful for in the context of bits manipulation using a mask?
20. What is the instruction `eor` useful for in the context of bits manipulation using a mask?
21. Explain how register multiplication by a constant can be achieved with a series of addition (with shifting) instructions alone.
22. Explain in what circumstances will a multiplication instruction, such as `mul a1, a2, a3`, store the incorrect product in the destination register (`a1`).
23. Explain why the multiplication instruction (`mul`) does not require a specification of signed/unsigned operands, while the long-multiplication instructions (`umull` / `smull`) do.

5. Memory Access Instructions

24. Explain the difference between *pre-indexing* an *post-indexing* in the context of a Load instruction. Give an example for each one.
25. Explain the difference between the instructions `mov v1, v2` and `ldr v1, [v2]`.
26. Describe the operation and result of the ARM architecture when loading a word from a memory address that is not word-aligned.
27. Explain the difference in operation between using a *load-byte* instruction (`ldrb`) and a *loadsigned-byte* (`ldrsb`) instruction.
28. Explain the meaning of *Little Endian* and *Big Endian* addressing modes.
29. Describe the operation of the instruction `stmia`. What four regular store instructions does `stmia v1, {a1-a4}` replaces?
30. Describe the operation of the instruction `stmib`. What four regular store instructions does `stmib v1, {a1-a4}` replaces?
31. Describe the operation of the instruction `stmda`. What four regular store instructions does `stmda v1, {a1-a4}` replaces?
32. Describe the operation of the instruction `stmdb`. What four regular store instructions does `stmdb v1, {a1-a4}` replaces?

6. Control Flow Instructions

33. What will be the values of the condition flags (NZCV) after the execution of a compare instruction, such as `cmp a1, a2`?
34. What will be the values of the condition flags (NZCV) after the execution of a `movs` instruction with shifting, such as `movs a1, a2, lsr #5`?

35. What does *conditional execution of an instruction* mean? How does the processor determines whether to execute a conditional instruction?
36. Explain the operation of a *Jump Table*.
37. Explain why implementing a `switch` statement using a jump table may be more efficient than the alternative implementation as a chain of `if-else`.
38. Define the term *Overflow* in the context of the 32-bit ARM architecture for both signed and unsigned numbers.
39. The condition `LT` (signed less-than) is true when (N != V). Explain the reason for this flag combination.

7. Functions

40. What is the importance of following the *ARM Architecture Procedure Call Standard* (AAPCS)?
41. What does the instruction *branch-and-link* (`bl`) do? How is this useful in the context of function calls?
42. The AAPCS defines the stack as *full descending*. What does this mean? Write the general format of the instructions to PUSH and POP values from the stack.
43. What are the three most common types of data that may be stored in a stack frame?
44. How is the Frame Pointer (`fp`) used in a stack frame? Give an example of when this would be useful.
45. What is meant by *Caller Saves* and *Callee Saves* in the context of registers usage in functions?
46. Which registers fall under the category of *Caller Saves*? Which fall under *Callee Saves*?
47. Where are the arguments stored when calling a function (for any number of expected arguments)?

8. Memory Management

48. A program memory space is commonly divided into four segments: *Code*, *Static*, *Heap*, and *Stack*. What is each used for? Give one type of data as an example for each of the four.
49. What are the *Scope* and *Lifetime* of each of the following types of C variables: *a global variable*, *a local variable*, *a static local variable*.

9. Fixed Point Fractions

50. Define the terms *Accuracy* and *Range* in the context of fixed-point numbers.

10. Floating Point Numbers

51. What are the three fields that make a floating-point number? Describe the purpose of each of the fields.
52. In IEEE 754 floating-point number, what are the representations for *Zero*, *Infinity*, and *NaN*?
53. Describe the spread uniformity of floating-point numbers (i.e. the spacing between any two consecutive numbers).
54. What is the reason for using biased notation, rather than 2s complements one, for the exponent field of a floating-point number?
55. What are *denormalized* floating-point numbers? How are they represented in the IEEE 754 standard?

11. Instructions Encoding/Decoding

56. An immediate number as an operand for a data processing instruction must be constructed by rotating to the right an 8-bit number by an even number of places. Why must the rotation value be even?

12. Pseudo-Instructions and Directives

57. What are *Assembly Directives*? How are they different from *Assembly Instructions*?
58. What are the two main kinds of areas defined by the `AREA` assembly directive? What data types are typical for each of these areas? What are the default read/write permissions associated with each of the two areas?
59. What do the assembly directives `DCD` and `DCB` do?
60. When should the `LTORG` assembly directive be used?
61. What does the assembly directive `EQU` do?
62. What does the assembly directive `IMPORT` do?
63. What does the assembly directive `EXPORT` do?
64. How does the assembler treat the pseudo-instruction `ldr a1, somelabel` ?
65. How does the assembler treat the pseudo-instruction `ldr a1, =somelabel` ?
66. How does the assembler treat the pseudo-instruction `ldr a1, =2500` ?
67. How does the assembler treat the pseudo-instruction `adr a1, somelabel` ?

13. I/O Interfacing

68. Discuss the advantages and disadvantages of *Polling* versus *Interrupts*.

69. What is meant by a *Memory Mapped I/O* architecture?
70. What does the C variable modifier `volatile` mean? Why is it important to use it in the context of I/O device access?

14. Exceptions and Interrupts

71. Define the terms *Exception* and *Interrupt*.
72. What is the difference between a *synchronous exception* and an *asynchronous* one? Give an example for each type.
73. List the steps the ARM processor take as a response to an exception.
74. What two operations does the processor perform for the single instruction `movs pc, lr` when in privileged mode? Why is a single instruction needed, rather than performing the operations in two steps?
75. What is the *Exception Vector Table*? Where is it located in the system memory? What is a typical content of such table?
76. What does a *Software Interrupt* (`SWI`) do? What is a typical use for it in general embedded systems?
77. What are two features provided by the ARM architecture that are unique to the FIQ exception (than other types of exceptions), which minimize the overhead involved in executing the FIQ handler routine?

15. Cache

78. What is *Cache Memory*? How does it help improve the overall system performance?
79. How are the terms *Temporal Locality* and *Spatial Locality* applied in the context of a Cache?
80. Explain the method to calculate the size of the data RAM required for a direct-mapped cache, with index field consisting of *i* bits and byte-offset field consisting of *j* bits (where *i* and *j* are integers).
81. Define the terms *Cache Hit* and *Cache Miss (with replacement)*. What operations are performed in each one of the cases?
82. What is a *Fully Associative Cache*? Give one advantage and one disadvantage of using such cache type.

Answer Bank for ELEC2142 Theory Test

By Tommy Sailing

C Language

1. Post incrementing (a++) uses the current value of the pointer or literal in the operation, before incrementing the pointer/literal. Pre incrementing (++a) performs the increment operation first, then uses the now-incremented pointer/literal in the current operation.
2. Pointer arithmetic translates to memory values as numbers are now treated as lengths of the data types (e.g. char is 1 byte long). For example, if p was a char, p = p + 3 will begin at the memory address defined by p, and then move forward by three bytes.
3. Type casting is converting data of one type (e.g. int, which spans 4 bytes) to another (e.g. short, which spans 2 bytes). Information is lost when typecasting to data types containing less bits. It should be used when the majority of MSBs are zero in a given operation.
4. Bitwise operations compare the *individual* binary bits between two operands. Logical operations act using Boolean logic between two or more *cases* (usually conditional statements). For example, x = 0010 | 1100 is a common operation that will produce a binary number from comparing binary bits, resulting in 1110. The common usage of logical operations includes situations such as: if(x = 5 && y = 2), which will continue only if both conditions (x = 5 and y = 2) are met.
5. A C right shift operation (>>) will translate to an lsr in assembly when the operand is positive. It will translate to an asr in assembly if the operand is negative, filling the remaining bits with 1's (the sign bit of a negative number). With that said, asr has the exact same operation as lsr if the operand is positive.

Number Systems

6. In GCC (GNU compiler collection) the following are used: Binary: 0b, Octal: 0 (leading zero), Hexadecimal: 0x, Decimal: (no prefix). In Keil ARMCC, the following are used: Binary: 2_, Octal: 8_, Hexadecimal: 0x, Decimal: (no prefix).

Programmer's Model

7. Advantages of using internal registers include: very fast performance compared to external memory; allowing operations to be performed (In ARM, operations cannot be performed on data in memory). Disadvantages of using internal registers include: There are only sixteen registers, which is a severe limit of storage space, while storage is abundant in external memory.
8. The Fetch operation obtains an instruction from the memory, the Decode operation determines the actions the processor must take from the instruction, and the Execute operation computes the result.
9. The program counter may fetch the next instruction on the same clock cycle that a decode is occurring, and on the next clock cycle, may execute the first instruction while concurrently decoding the next one and fetching a third. The entire pipeline is productive, as a result.
10. PC-modifying instructions require additional clock cycles because they break the Fetch-Decode-Execute cycle. When the PC register is moved, the processor must flush the pipeline to avoid incorrect operation and begin the fetch-decode-execute cycle at another point in the program, using up valuable clock cycles.

11. There are 16 user-mode data registers available for use. The hidden data registers (with suffixes fiq, svc, abort, irq and undefined) are accessible only to the system during exceptions, when they must enter these modes.
12. The three main ARM instruction categories are: Data Processing – which performs arithmetic and logical operations on operands (eg. add, sub, and), Control Flow – which aids navigation around the program and may or may not be conditional (e.g b, bl), and Load/Store – which loads data from the memory to the registers for operations to occur, or stores processed data back to the memory (e.g. ldr, stmia).
13. - “a1 – a4” is an abbreviation for argument registers.
    - “v1 – v7” is an abbreviation for variable registers.
    - “fp” is an abbreviation for “frame pointer”
    - “ip” is an abbreviation for “intra-procedure-call scratch pointer”
    - “sp” is an abbreviation for “stack pointer”
    - “lr” is an abbreviation for “link register”
    - “pc” is an abbreviation for “program counter”
14. By convention, “a1 – a4” are used to hold data that may be passed between functions and jumps, as well as being used for scratch. “v1 – v7” are used to hold local variables within functions, and should be reinstated to their previous values once a function is complete. “fp” is used to point to a particular address in memory from which values may be offset. “ip” is a useful register to move values between procedures. “sp” is used to denote the address of the beginning of the stack. “lr” is used to hold the return address of the function’s callee. “pc” holds the memory address of the program’s current fetch state.

Data Processing Instructions

15. “lsr” stands for “logical shift right”. “asr” stands for “arithmetic shift right”. Both functions will shift the current number bitwise right by a number of bits specified as an immediate following the command, however “lsr” will always fill the emptied bits with zeroes, while “asr” will fill the emptied bits with the sign bit (equal to the MSB of the shifted number).
16. Valid immediate values for data processing instructions are signed 8 bit numbers.
17. “rsb” is the “reverse subtraction” assembly instruction. An example of its use would be when you need to subtract a register from an immediate:
    ```
    rsb r1,r2,#6     ; r1 = 6-r2
    ```
18. “bic” is useful for masking groups of bits to all 0’s. It can also be useful for masking immediates that are too large – inverting a valid immediate may allow you to specify and immediate larger than what is normally allowed.
19. “orr” is useful for forcing bits to 1’s while leaving 0’s unchanged.
20. “eor” is useful for forcing bits into their complements.
21. Shifting a value left by an immediate ‘n’ is equivalent to multiplying the value by 2^n. Any multiplicand can be broken down into powers of 2^n and added together to give a final result.
22. A “mul” instruction such as `mul a1, a2, a3` will store the incorrect product when the real product of a2 and a3 is greater than 32 bits – the result stored in a1 will only be the bottom-most 32 bits.

23. The long multiplication instructions give results in 64 bits, so all the topmost bits now matter, including the MSB. As the MSB could either be a sign bit or a data bit in the unsigned case, the computer needs to know what to treat it as, so it needs to be marked as such.

Memory Access Instructions

24. Pre-Indexing a Load instruction begins the load operation from the address specified plus the offset. E.g. `ldr r1, [r2, #4]` will load the value at address r2 + 4 into r1. Post-Indexing a Load instruction begins the load operation from the address specified, then updates the address with the offset, so that the next time the operation is incurred, the address will have been offset by the immediate. E.g. `ldr r1, r2, #4` will load the value at address r2 into r1, then update the address of r2 by adding 4 bytes.
25. "mov v1, v2" will load the contents of the register v2 into register v1, whereas "ldr v1, [v2]" will load the value stored in the memory address given by the value of v2 into v1.
26. It will start reading from the byte that the pointer is pointing to, and will read until the end of that aligned section, and loop back around to the beginning of the aligned section. For example, given the word 0xAAEEFFDD, and it starts reading from F, it will read FFDDAAEE.
27. A load-byte instruction (ldrb) will load a single byte into LSB end of the register and fill the remaining three bytes with zeroes. A load-signed-byte instruction (ldrsb) will load a single byte into the MSB end of the register and fill the remaining three bytes with the sign bit of the loaded byte.
28. Little Endian addressing addresses the start of the data at the LSB end of the register, and Big Endian addressing addresses the start of the data at the MSB end of the register. For example 0xF0 in Little Endian is represented as: 11110000, but is represented as 00001111 in Big Endian architectures.
29. "stmia" is the instruction pertaining to "store multiple, increment after", which copies a block of data beginning with the address specified by the first argument. "stmia v1, {a1-a4}" replaces str a1, [v1]; str a2, [v1, #4]; str a3,[v1, #8]; str a4, [v1, #12].
30. "stmib" is the instruction pertaining to "store multiple, increment before", which copies a block of data beginning with the address specified one word after the first argument. "stmib v1, {a1-a4}" replaces str a1, [v1, #4]; str a2, [v1, #8]; str a3,[v1, #12]; str a4,[v1,#16].
31. "stmda" is the instruction pertaining to "store multiple, decrment after", which copies a block of data beginning with the address specified by the first argument and moving upwards in the stack. "stmda v1, {a1-a4}" replaces str a1, [v1]; str a2, [v1, #-4]; str a3,[v1, #-8]; str a4, [v1, #-12].
32. "stmdb" is the instruction pertaining to "store multiple, decrement before", which copies a block of data beginning with the address specified one word before the first argument and moving upwards in the stack. "stmdb v1, {a1-a4}" replaces str a1, [v1, #-4]; str a2, [v1, #-8]; str a3,[v1, #-12]; str a4,[v1,#-16].

Control Flow Instructions

33. After the execution of a compare instruction, the computer will have a result for RESULT = a1 – a2. The N flag equals the MSB of the result, the Z flag equals 1 if the result is zero, the C flag will be set if an unsigned overflow has occurred and the V flag will be set if a signed overflow has occurred.

34. The computer will execute a “sub” operation between the operands and the N flag equals the MSB of the result, the Z flag equals 1 if the result is zero, the C flag will be set if an unsigned overflow has occurred and the V flag will be set if a signed overflow has occurred. It will also save the contents of the SPSR into the CPSR.
35. Conditional Execution is the process of executing an instruction following a “cmp” operation so long as the result of the “cmp” operation agrees with the condition specified. For example if you execute `cmp a1, a2` followed by `bgt function`, the program will only branch to “function” if a1 is greater than a2.
36. A Jump Table is a list of addresses and/or constants which work as a convenient index of numbers that the program may require. Registers may load the addresses or values of these indices to facilitate program control and branching.
37. You only compute the number corresponding to the index you wish to jump to, so less compare functions and branches are required to implement the same function.
38. Overflow occurs in signed numbers when there is a lack of bit fields to store the bits that make up a number. In ARM, if an unsigned number requires more than 32 bits to represent it, an overflow will occur. If a signed number becomes too large it may change the sign bit, resulting in a number of the opposite sign.
39. Signed Less-Than is true when NEGATIVE != OVERFLOW because if an negative has occurred the result of the cmp must be negative, meaning the N flag is high. But if there was an overflow, the N flag would be low and the result is no longer negative!

Functions

40. The AAPCS must be followed to ensure that functions written in assembly can co-exist with functions written in high level languages (as compilers for ARM must all follow the AAPCS as well). This way, different functions cannot ‘break’ the functions that called them, as following AAPCS ensures that the previous state must be saved.
41. ‘bl’ branches to a label and saves the return address into the link register in one step. This is useful so that functions may return to the caller in a standardised method after a mov pc, lr instruction.
42. ‘Full Descending’ means that the stack pointer points to the location in which the last piece of data was stored, and in a push, the stack pointer will be decremented (the stack grows towards lower-numbered addresses). PUSH instructions follow this format: stmdb sp!,{registers} and POP instructions follow this format: ldmia sp!, {registers}.
43. The three most common types of data that may be stored in a stack frame are: saved registers, local variables and arguments to be passed on to functions.
44. In a stack frame, the frame pointer (fp) is used as a fixed point acting as the caller’s stack pointer minus a word.
45. ‘Caller saves’ means that the calling procedure must save any of the registers it needs to use on to the stack prior to branching to a different function. ‘Callee saves’ means that the function most take the responsibility of saving the current-state registers to the stack prior to beginning, and restore them afterwards
46. Registers a1-a4 and lr fall under the Caller Saves category, and v1-v7 fall under the Callee Saves category.

47. When calling a function, the first four arguments to be passed to it are stored in the scratch registers, a1 to a4. The remaining arguments are stored in the beginning of the caller's stack frame.

Memory Management

48. Code is used to store the instructions that make up the program. For example, "add r1, r2, r3" will be translated to binary machine code that will take up 32 bits in the memory's code segment. Static stores static variables, such as string literals. The heap is the total remaining pool of memory available that may be allocated to data the program needs to use. The stack is the ordered area of memory that stores variables and parameters on a per-function basis.
49. A global variable within a program can be seen from any part of that program and exists until the program terminates. A local variable exists only within the function that created it, and will only exist until the function returns. A static local variable is visible only within the function it was created, however it exists for the duration of the program.

Fixed Point Fractions

50. The range is the difference between the highest and lowest number you can represent. The further towards the lsb the point is the larger the range.
    Accuracy is the difference between 2 consecutive numbers that you can represent. The further towards the msb the point is the larger the accuracy.

Floating Point Numbers

51. The 3 fields making up a floating point number are: significand, radix and exponent. The significand is the primary multiplier of the number. The radix (or base) is the number declaring what initial value to multiply the significand by. The exponent is the power that the radix must be taken by to represent numbers larger or smaller than what the significand alone can handle. For example in the number: 1101.11 x 2^22, 1101.11 is the significand, 2 is the radix and 22 is the exponent.
52. Zero is represented in IEEE 754 as 'the 32-bit number is all zeroes'. Infinity is represented as 0 with the significand and 255 as the exponent. Not-a-numbers (NaN) are represented as a nonzero number as the significand and 255 as the exponent.
53. The spacing between numbers depends on the exponent. The larger the exponent the larger the spacing between consecutive numbers. This is because the smallest gap is always the smallest difference between 2 significands *2^exponent.
54. Biasing (where 00000000 is the most negative, increasing all the way to 11111111, which is the most positive number) allows integer compare functions to correctly identify when numbers are greater or less than others (as every consecutive number from start to finish is greater than the last) whereas a two's complement does not have this property, hence some numbers may be incorrectly (integer) compared.
55. Denormalised floating point numbers are nonzero numbers with significands smaller than the smallest normal number (i.e. have leading zeros in the significand). They are represented in IEEE 754 as a zero exponent with a nonzero significand.

Instructions Encoding/Decoding

56. Because we do not have enough bits to represent 0-32, only 0-16 so it was defined that there would be an implied zero on the end, meaning ever number is even.

Pseudo Instructions & Directives

57. Assembly directives are directions telling the assembler what to do but are not translated to machine code at assembly time. Assembly instructions, however, do get translated to machine code at assembly time.
58. The AREA directive most often defines the areas 'CODE' and 'DATA'. 'CODE' will typically store instructions to be translated to machine code, and is read-only by default. 'DATA' will typically store data to be referenced by the code and is read/write enabled by default.
59. DCD allocates a word of data to be stored in memory, aligned on 4 byte boundaries. DCB allocates a byte of data to be stored in memory.
60. LTORG instructs the assembler to assemble the current literal pool immediately.
61. EQU is the directive to instruct the assembler to define constants. An immediate preceded by a label and the EQU directive can now be represented in the code as that label, to aid readability.
62. IMPORT aids modularity by defining a label of an exported function stored in a separate .s file (that is not in the current file), allowing it to be used in the current file upon assembly time.
63. EXPORT aids modularity by defining labels to functions in the current file that may be brought into other assembly files (through use of the IMPORT directive), allowing them to be used in other files upon assembly time.
64. Ldr a1, somelabel will, at assembly time, load the value defined after 'somelabel' into register a1.
65. Ldr a1, =somelabel will, at assembly time, load the address of the value defined after 'somelabel' into register a1.
66. Ldr a1, =2500 will, at assembly time, load the address of the program counter shifted by 2500 – (pc + 8).
67. Adr a1, somelabel will, at assembly time, load the address defined after 'somelabel' computed as "address of somelabel" – (pc + 8).

I/O Interfacing

68. The disadvantages of polling range from taking up a significantly high proportion of CPU time (that otherwise could be spent on other functionality) and having a 'time window' between polls, meaning input/output could be missed if it occurs during a clock cycle in which the CPU is occupied by something else. Interrupts do not interfere with CPU operations until required, and signal the CPU when ready, so it is impossible to miss an input/output. Advantages of polling include not requiring external interrupt hardware, providing performance advantages over interrupts if the CPU has no other functionality to tend to – an interrupt will need to go through the high-latency process of saving the CPSR and registers, etc. Both mechanisms support prioritising.
69. A memory-mapped I/O architecture means that a selection of the memory address space is reserved for reading/writing changes to/from the I/O device controller.
70. The C modifier 'volatile' means that the compiler must not optimise the code (which might occur, for example, if you polled a variable for a change that appears to act in an infinite

loop, but could be modified by a hardware change), ensuring that the variable referenced will always appear 'as-is' in the code. It is necessary for I/O devices to operate as if an optimisation does occur, it might become impossible to read from an I/O device affected by that address.

Exceptions & Interrupts

71. An exception is a hardware or software event not expected nor handleable by a user-mode function, that forces the CPU into a special mode to attend to it. An interrupt is a type of exception generated by external inputs or outputs (or software, in the case of SWI).
72. A synchronous exception is an exception that is expected, i.e. occurs within the user's code. An example is the Software Interrupt. An asynchrounous exception is an unexpected exception that can occur at any time during a CPU's execution. An example is the Fast Interrupt.
73. When responding to an exception, the ARM processor must halt the current operation, then jump to the exception handler function, which will save the current state (copying CPSR to SPSR_mode), set the CPSR bits (mode, T, F or I) to its requirements, save the return address of the interrupted function (pc – 4) to lr_mode and branch to an appropriate exception handler. The handler will perform whatever function it needs to perform, then reinstate the original conditions, as if the exception never happened.
74. 'movs pc, lr' will restore the original CPSR from the SPSR, and will restore pc to the return address. It needs to execute as one instruction because:
    a. If the CPSR was restored first, you lose access to the privileged mode's lr, meaning that you cannot restore pc to the previous function, and your program will be stuck in limbo between user mode and the privileged mode.
    b. If pc was restored first, the SPSR cannot be restored to the CPSR, meaning that the previous "user-mode" function is now running in privileged mode, unbeknownst to it. If this function is a sub-function (requiring lr to return to a previous function), then lr is now lr_mode, meaning that exiting this function will return to the point that the interrupt was called. The function would then infinite loop.
75. The Exception Vector Table is a jump table defining actions to be taken upon occurrence of an exception, occupying the memory addresses 0x00 to 0x1C. The content of the table are often addresses of handler functions – special instructions which the pc will branch to upon reaching an exception.
76. A software interrupt is a synchronous, software-based method of accessing the Supervisor privileged mode. A common use for it in embedded systems is to access hardware addresses, which are often located in memory restricted for access only by supervisor modes – switching to SWI's gives an easy method to access those addresses in the programmer's code.
77. FIQ (Fast Interrupt reQuest) has two unique features making it the fastest exception:
    a. Its address is the last on the Exception Vector Table, meaning that the immediate memory following the FIQ address is free to be used as the FIQ handler function, reducing the need for a branch (wasting a fetch-decode-execute-cycle)
    b. It has its own registers, r8 – r14 available for use, meaning that in order to process data it does not need to save the user mode registers to the stack (and restore them on return), wasting valuable clock cycles.

Cache

78. Cache memory is a form of temporary memory, used to store recently or frequently used data in a layer much closer to the registers, speeding up processing by reducing the amount of continual R/W cycles to the much slower RAM.
79. Temporal Locality is the concept “if we have just used a piece of data, we will want to access it again in a very short time”, therefore keeping this data “nearby” for quicker access. Spatial Locality is the concept “if we access a given word, we’re likely to access other nearby words soon”, keeping nearby data ready for quicker access.
80. 2^I * 2^j
81. Cache Hit means when the processor requests data, the cache contains the address to that data and hence the data may be read. Cache Miss (with replacement) means there is incorrect data in the cache at the appropriate block, so it must be discarded and the desired data must be fetched from the memory (and subsequently placed in that cache block).
82. Fully Associative Cache is a way to organise cache memory without indexes or ‘rows’, so an advantage is any block can go anywhere in the cache, but is disadvantaged because this is at the expense of comparing all tags in the entire cache when a request is served – requiring a comparator circuit.