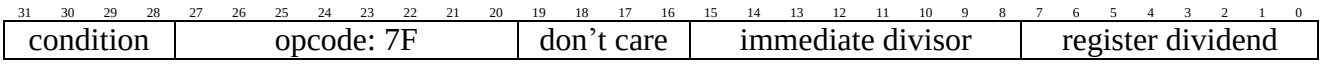
Computer Architectures
Lab 5

1) Define two literal pools with N elements (N is a constant). The literal pools contain 32-bit signed integers. Write a program that computes the sum of every pair of corresponding elements in the two literals pools (i.e., element 0 of the first literal pool is summed to element 0 of the second literal pool, and so on). The results are stored in memory (a suitable block of memory should be defined in a data area with the `space` directive).
The program should check and correct overflow in the results. For example, both 0x70000000 and 0x12345678 are positive numbers. The result of their sum is 0x82345678, which is a negative number. If the program detect this issue after the sum, it calls a software interrupt (by means of the `SWI` instruction) with identification code 0x10. This interrupt is served by setting R6 to 0x7FFFFFFF, which is the largest positive number in 32-bit notation.
Similarly, the sum of two negative numbers, such as 0x800000F0 and 0xF0004538, can give a positive number (0x70004628 in the example). In this case, the program calls another software interrupt with identification code 0x20. The interrupt handler sets R6 to 0x80000000, which is the largest negative number.
The value stored in memory is the sum of the pair of elements (if there is no overflow) or the value of R6 after calling the software interrupt (if there is an overflow in the sum).
Example:

```
literal1 DCD 0x10, 0x70000000, 0xFFFFFFE0, 0x800000F0, 0x100EC023
literal2 DCD 0x200, 0x12345678, 0xE00A1238, 0xF0004538, 0xE9800348
```

Values in memory: `0x210, 0x7FFFFFFF, 0xE00A1218, 0x80000000, 0x98EC36B`
Suggestion: modify the file *template.s* in the same folder of this document.

2) The ARM instruction set does not include any instruction for division. We want to allow an instruction `DIV` that takes a register as dividend, an 8-bit immediate value as divisor and it saves the results of the division in R12. The encoding of our personalized instruction is the following:



More in details:

- as every ARM instruction, our `DIV` can be conditionally executed. In order to execute it always (i.e., the mnemonic suffix is missing or equal to AL), the condition field is set to 0x7
- the operation code that identifies our `DIV` is 0x7F
- the immediate divisor is an unsigned integer
- the dividend is expressed as 0xF?, where ? is the index of the register. For example, 0xF3 identifies R3, 0xFA identifies R10.

If we write `DIV` in the code, the assembler does not recognize it. In order to build the program, we use a `DCD` statement. For example, the following code divides R6 by 5:

```
            MOV R6, #26
DIVr6BY5  DCD  0x77F005F6
```

When the processor fetches the second instruction in the example, it tries to execute the instruction and it generates an undefined exception. Write the handler of this undefined exception. Fields in the instruction should be extracted by applying `BIC` with proper masks. The division may be performed with a simple loop, where the divisor is subtracted from the dividend. The result (i.e., the number of iterations of the loop) is saved in R12.
Suggestion: the handler of the undefined exception should save the register as its first operation:

```
            STMFD      SP!, {R0-R11, LR}
```

R12 is not saved because it will contain the result of the division.
Now the stack contains the value of all registers, so we can access it in order to load the content of the register indicated in our `DIV`:

```
LDR R3, [SP, R2, LSL #2]
```

In the example, R2 contains the index of the register (as extracted from the least significant byte of our `DIV`), and it is used to access the stack. Then R3 contains the value indexed by R2.

3) Build and debug the *sample* project. In particular, open Port 0 of General Purpose Input/Output and check/uncheck bit 30 to interact with the emulated board.