Current Processor Status Register CPSR and Flags:

In addition to the 16 ARM registers discussed in the previous Lab, another register will be introduced which is the Current Processor Status Register (CPSR) which includes various condition code flags and other status and control information.

We are only interested for now with the most four significant bits (upper four bits – bits from 28 to 31) of the status register which contains a set of four condition code flags. The condition code flags are:

Negative (N) Zero (Z)
Carry (C)
Overflow (V)

These flags may be changed as a result of arithmetic and logical operations in the processor, may be tested by all instructions to determine if the instruction is to be executed or not.

The following table represents when the flags changes its value:

N	The Negative flag takes on the value of the most significant bit of a result. Thus when an operation produces a negative result the negative flag is set (=1) and a positive result results in the negative flag being reset (cleared =0). This assumes the values are in standard two's complement form.
C	The Carry flag holds the carry from the most significant bit produced by arithmetic operations or shifts. The carry flag is inverted after a subtraction so that the flag acts as a borrow flag after a subtraction.
Z	The Zero flag is set when an operation produces a zero result. It is reset when an operation produces a non-zero result.
V	The Overflow flag is set when an arithmetic result is greater than can be represented in a register.

S qualifier:

Many instructions can modify the flags, these include arithmetic, logical, comparisons and move instructions. Most of these instructions have S qualifier which instructs the processor to set the condition code flags or not.

Placing an (S) after the mnemonic will cause the flags to be updated. For example there are two versions of the MOV instruction:

MOV R0, #0 will move the value 0 into the register R0 without setting the flags.

MOVS R0, #0 will do the same, move the value 0 into the register R0, but it will also set the condition code flags accordingly, the Zero flag will be set, the Negative flag will be reset and the Carry and overflow flags will not be affected.

Condition Codes

Almost all ARM instructions contain a condition field which allows it to be executed condition dependent on the condition code flags. If the flags indicate that the corresponding condition is true when the instruction starts executing, it executes normally. Otherwise, instruction does nothing and the next instruction is executed normally.

The following table shows a list of the condition codes and their mnemonics. To indicate that the instruction is conditional we simply place the mnemonic for the condition code after the mnemonic for the instruction.

Mnemonic	Condition
CS	Carry Set
EQ	Equal (if zero is set)
VS	Overflow Set
GT	Greater Than
GE	Greater Than or Equal
HI	Higher Than (unsigned numbers)
HS	Higher or Same (unsigned numbers)

Mnemonic	Condition
CL	Carry Clear
NE	Not Equal (if zero is reset)
VC	Overflow Clear
LT	Less Than
LE	Less Than or Equal
LO	Lower Than (unsigned numbers)
LS	Lower or Same (unsigned numbers)

For example, the following instruction will move the value of R1 into R0 register only R0 is greater than R1, R0 will remain unaffected if R0 is less than or equal to R1.

```
MOVGT  R0, R1
```

The S qualifier and the condition code could be used with the same instruction. The instruction will move the value of R1 into R0 register only Ro is greater than R1, and will also update the flags accordingly.

```
MOVGTS  R0, R1
```

Note that the <S> always comes after the <cc> (conditional execution) modification if it is given. Thus the full description of the move instruction would be:

```
MOV <cc> <S> Rd, <op1>
```

The condition codes really makes sense after a comparison instruction CMP.

Comparison and Branching Instructions

The CMP instruction compares the register value with another arithmetic value (whether an immediate value or in another register) and perform a subtraction of the second operand from first operand and update the flags according to the result of subtraction. It is always beneficial to use CMP instruction followed by a branch instruction.

There are different types of branch instructions, we are interested now with b branch instruction that jumps to a target label and performs a change in the normal flow of the program.

Arithmetic Operations:

There are three main instructions in ARM Assembly that were discussed the previous lab, ADD to add two operands, SUB to subtract two operands, and MUL to Multiply two operands, the following will discuss the different forms of writing the instructions and important notes:

General Form
ADD<cc><s> destReg, op1, op2
SUB<cc><s> destReg, op1, op2
MUL<cc><s> destReg, op1, op2

Where:

desReg is the destination register used to store the result in and op1 and op2 are the two operands of the operations as following:
(ADD) => desReg = op1 + op2
(SUB) => desReg = op1 - op2
(MUL) => desReg = op1 * op2

Along with any of the above instructions, there are two optional parts that can be added to the instruction name which are:

1.<cc> => used to apply the instruction if certain condition is applied.

Example:
CMP R0, R1
ADDGT R2, R0, R1

In this example, first we compare the values of R0 and R1. Then ADD instruction is called but with GT which means if R0 is greater than R1, perform the addition and store the results in R2.

2. <S> => specifies that the instruction updates the CPSR and the values of C, N, V and Z flags based on the result of this operation.

Example:
SUBS R0, R0, #1
BNE Label1

In this example, first the R0 is subtracted by 1 and the value is stored back in R0 too. Also, the flags in CPSR are updated because the instruction contains <S> part, so we can use BNE after this operation to branch to Label1 position if the values are not equal.

For the arithmetic instructions, op1 refers to the first operand in the operation and this operand must be register. The second operand op2 is optional part and can match with one of the following cases:

1. op2 is a register
For example: ADD R0, R1, R2
This will add the values of R1 and R2 and store the result into R0 as R0 = R1 + R2

2. op2 is a constant value
For example: ADD R0, R0, #1
Note that we use (#) before the constant value. This will add 1 to R0 and store the result back in R0 as R0 = R0 + 1

3. op2 is omitted
For example: ADD R0, R1
In this case, the contents of R1 is added to R0 and the result is stored in R0 again as R0 = R0 + R1

An important note for MUL instruction, there is a restriction for the operands in MUL instruction in which we cannot assign destination register to be the same as operand 1, also MUL will not accept that 2nd operand is an immediate operand.

For example:
MUL R0, R0, R2

This will produce an error in the assembler because it will cause some conflicts while the instruction being executed. To recover from this error we can use another register to hold the results and move it back to the original one as following:

MUL R1, R0, R2
MOV R0, R1

There is no division instruction in ARM. To perform this operation we treat it as a successive subtraction as discussed in Lab3-Exercise Sheet.