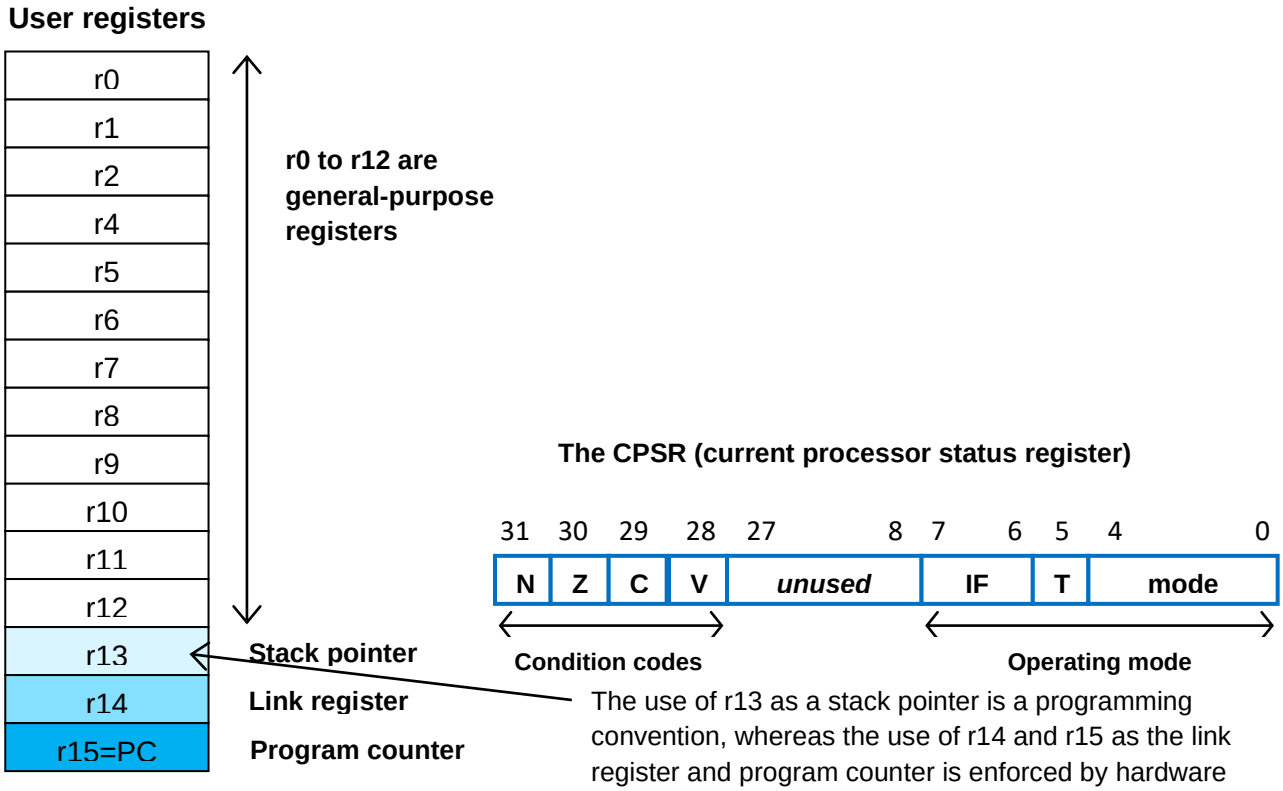
Lab work #2

1. **Objective of this lab:**To explore ARM branch instructions and implement them in ARM uVision5

2. **Preparation**

2.1. ARM's Flow Control Instruction

As mentioned in the previous assignment, ARM has 16 programmer-visible registers and a *Current Program Status Register*, CPSR.
Here is a picture to show the **ARM register set**.



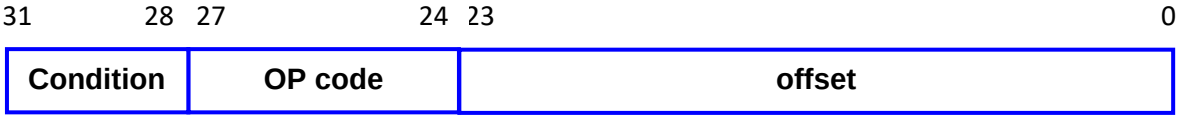

```
R0 to R12 are the general-purpose registers.
R13 is reserved for the programmer to use it as the stack pointer.
R14 is the link register, which stores a subroutine return address.
R15 contains the program counter and is accessible by the programmer.

Condition codes flagsin CPSR:
N - Negative or less than flag
Z - Zero flag
C - Carry or borrow or extended flag
V - Overflow flag
The least-significant 8-bit of the CPSR are the control bits of the
system.
The other bits are reserved.
```

2.2. Setting Condition Code Flags

Some instructions, such as Compare, given by **CMP Rn, Rm** which performs the operation [Rn]-[Rm] have the sole purpose of setting the condition code flags based on the result of the subtraction operation.
The arithmetic and logic instructions affect the condition code flags only if explicitly specified to do so by a bit in the OP-code field. This is indicated by appending the suffix S to the OP-code. For example, the instruction **`ADDS R0,R1,R2`** sets the condition code flags.
However, **`ADD R0,R1,R2`** does not.

2.3. The Encoding Format for Branch Instructions

Conditional branch instructions contain a signed 24-bit offset that is added to the updated contents of the Program Counter to generate the branch target address. Here is the encoding format for the branch instructions:

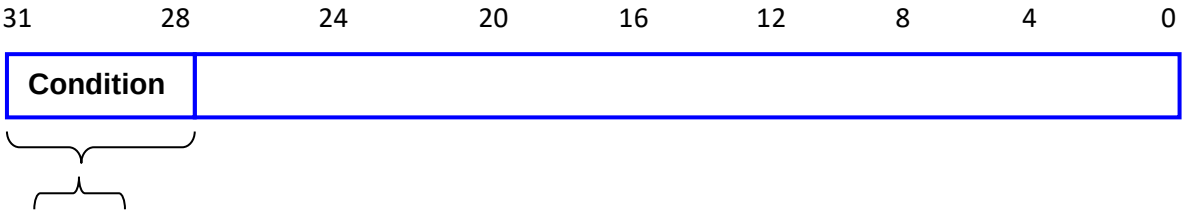


Offset is a signed 24-bit number. It is shifted left two-bit positions (all branch targets are aligned word addresses), signed extended to 32 bits, and added to the updated PC to generate the branch target address. The updated PC points to the instruction that is two words (8 bytes) forward from the branch instruction.
ARM instructions are conditionally executed depending on a condition specified in the instruction. The instruction is executed only if the current state of the processor condition code flag satisfies the condition specified in bits `31-28` of the instruction. Thus, the instructions whose condition does not meet the processor condition code flag are not executed. One of the conditions is used to indicate that the instruction is always executed. Here is a more detailed description.

31-28	27	26	25	24-21	20	19-16	15-12	11-0
Condition	0	0	I	OP code	S	Rn	Rd	Operand 2

- **Rn** = source register operand 1
- **Rd** = destination register
- **31-28: condition code**
  - ALL ARM instructions can be conditionally executed
  - eg: `ADDEQ`
    - add, but only if the previous operation produced a result of zero
    - checks CPSR stored from previous operation.

All the ARM instructions are conditionally executed depending on a condition specified in the instruction (bits 31-28).



CONDITION		FLAGS	Note
0000	EQ	Z==1	Equal
0001	NE	Z==0	Not Equal
0010	HS/CS	C==1	>= (U) / C=1
0011	LO/CC	C==0	< (U) / C=0
0100	MI	N==1	minus (neg)
0101	PL	N==0	plus (pos)
0110	VS	V==1	V set (ovfl)
0111	VC	V==0	V clr
1000	HI	C==1 && Z==0	> (U)
1001	LS	C==0 || Z==1	<= (U)
1010	GE	N==V	>=
1011	LT	N!=V	<
1100	GT	Z==0 && N==V	>
1101	LE	Z==1 || N!=V	<=
1110	AL	always	
1111	NE	never	
(U) = unsigned			

- The instruction is executed only if the current state of the processor condition code flag satisfies the condition specified in bits 31-28 of the instruction.

```
For example:
            CMP R0, #'A'      ; flags are updated according to (R0 - #'A'
            BEQ VowelCount
```

- The instructions whose condition does not meet the processor condition code flag are not executed.
- One of the conditions is used to indicate that the instruction is always executed.

2.4. Branch and Control Instructions

Branch instructions are very useful for selection control and looping control.
Here is a list of the ARM processor's Branch and Control instructions.

Mnemonic	Interpretation	Meaning
B loopA **BAL loopA**	Unconditional Always	Branch to label loopA unconditionally Branch to label loopA always
BEQ target	Equal	Conditionally branch to target, when Z = 1
BNE AAA	Not equal	Branch to label AAA when Z = 0
BMI BBB	Minus	Branch to label AAA when N = 1
BPL CCC	Plus	Branch to label CCC when N = 0
BLT labelAA	Less then	Conditionally branch to label labelAA, N set and V clear or N clear and V set i.e. N != V
BLE labelA	Less or equal	Conditionally branch to label labelA, when less than or equal, Z set or N set and V clear or N clear and V set i.e. Z = 1 or N != V
BGT labelAA	Greater than	Conditionally branch to label labelAA, Z clear and either N set and V set or N clear and V clear i.e. Z = 0 or N = V
BGE labelA	Greater or equal	Conditionally branch to label labelA, when Greater than or equal to zero, Z setor N set and V clear or N clear and V set i.e. Z = 1 or N != V
BL funC		Branch with link (Call) to function funC, return address stored in LR, the register R14
BX LR		Return from function call
BXNE R0		Conditionally branch to address stored in R0
BLX R0		Branch with link (Call) and exchange (Call) to an address stored in R0.

2.5. Comparisons

- The only effect of the comparisons is to update the condition flags. Thus no need to set S bit.
- Operations are:
  - `CMP operand1 - operand2` ; Compare
  - `CMN operand1 + operand2` ; Compare negative
  - `TST operand1 AND operand2` ; Test
  - `TEQoperand1 EOR operand2` ; Test equivalence
- Syntax: `<Operation>{<cond>} Rn, Operand2`

2.6. Examples of Compare Instructions

Mnemonic	Meaning
CMP R2, R9	; update the N, Z, C and V flags
CMN R0, #6400	; update the N, Z, C and V flags
CMPGT SP,R7,LSL #2	; update the N, Z, C and V flags
TSTEQ r2, #5	

2.7. An Example of Using Branch Instructions

```
; The semicolon is used to lead an inline documentation
;
; When you write your program, you could have your info at the top document block
; For Example:  Your Name, Student Number, what the program is for, and what it does etc.
;
; This program will count the length of a string:
;
            AREA   MYCODE, CODE, READONLY
            ENTRY

            ADR    R0, string1   ; Load the address of string1 into the register R0
            MOV    R1, #0        ; Initialize the counter counting the length of string1
loopCount
            LDRB   R2, [R0]      ; Load the character from the address R0 contains
            CMP    R2, #0        ; Compare with 0
            BEQ    STOP          ; If it is zero...remember null terminated...
                                 ; You are done with the string. The length is in R1.
            ADD    R0, #1        ; Otherwise, increment index to the next character
            ADD    R1, #1        ; increment the counter for length
            B      loopCount

STOP        B      STOP

            AREA   MYCODE, CODE, READWRITE
string1
            DCB    "Hello world!",0
            END                  ; End of the program
```

2.8. Another Example

```
; The semicolon is used to lead an inline documentation
;
; When you write your program, you could have your info at the top document block
; For Example:  Your Name, Student Number, what the program is for, and what it does etc.
;
;           See if you can figure out what this program does
;
            AREA  MYCODE, CODE, READWRITE
            LDR   R1, N          ; Load count into R1
            MOV   R0, #0         ; Clear accumulator R0
LOOP
            ADD   R0, R0, R1     ; Add number into R0
            SUBS  R1, R1, #1     ; Decrement loop counter R1
            BGT   LOOP           ; Branch back if not done
            LDR   R3, SUMP       ; Load address of SUM to R3
            STR   R0, [R3]       ; Store SUM
STOP        B     STOP

            AREA  MYCODE, CODE, READWRITE
SUM         DCD   0
SUMP        DCD   SUM
N           DCD   5
            END                  ; End of the program
            END                  ; End of the program
```

3. Lab Assignment

3.1. Assignment#1:

Write an ARM assembly language program **CountVowelsOne.s** to count how many vowels and how many non-vowels are in the following string. **You will hand in the following:**

```
"ARM assembley language is important to learn!",0
```

Recommendations for writing the program:

- Put the string in the memory by using DCB.
- Use R0 to hold the address of a character in the string.
- Use R1 to be the counter for vowels.
- Use R2 to be the counter for non-vowels.
- Build the program, debug if needed.
- Run the program step by step and see how values are changing in the registers.
- Make a screenshot to capture the results in your designated registers.

You will hand in the following:

1. The source code in the file **CountVowelsOne.s**
2. The print out of the screen shot (print screen) to show the program has been successfully built
3. The print out of the screen shot showing the number of vowels in R1 and non-vowels in R2.