CG2028 Assignment Report

Zhuang Jianning - A0214561M
Vikas Harlani - A0214360U

Section 1: Assembly Program Logic

The function **int classification**(**int** N, **int*** points, **int*** label, **int*** sample) classifies a sample point based on its k nearest neighbors. Since the assignment sets k = 1, the sample is simply assigned to the class of the single nearest neighbor.

In the C implementation of classification(), the N squared distances between the test sample and N training data were calculated and stored in an array d[N] using a for loop. A current minimum is initialised to be the first distance in the array d[N] and another for loop compares it with each distance stored in the array to update the current minimum. The current minimum at the end of the for loop will be the closest distance and the class associated with the point which produced the closest distance is the output of the function.

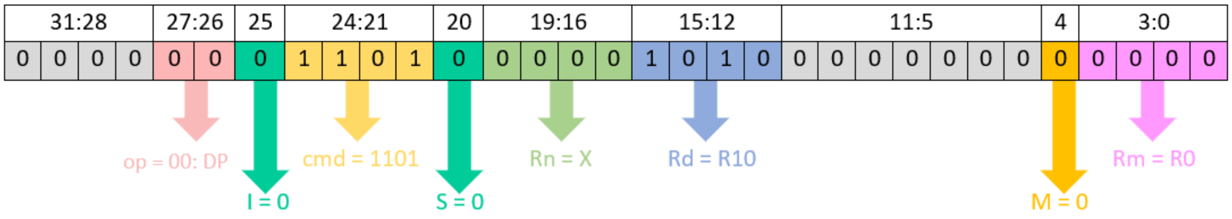
For our assembly code, the overall logic is similar to the C implementation. We still calculate the N squared distances between the test sample and N training data. However, we realised that generating the squared distances and updating the current minimum distance could be done within the same for loop. After the squared distance between a data point and the sample point is calculated, it can immediately be compared to the current minimum to update the minimum distance if necessary. This eliminated the need for us to store the distances inside an array.

To initialise our current minimum distance, we can only do so within the first iteration of the for loop, after generating the squared distance from the first data point to the sample. To do so, we check if the loop counter = N. If True, we jump to the INITIALISE subroutine which sets the minimum distance to be the squared distance between point[0] and sample and the nearest class to be the class of point[0].

For the rest of the iterations, we compare the squared distance generated during that iteration with the current minimum. If it is less than, we jump to the UPDATE subroutine which updates the minimum distance to be the squared distance between the current point and sample and the nearest class to be the class of the current point.

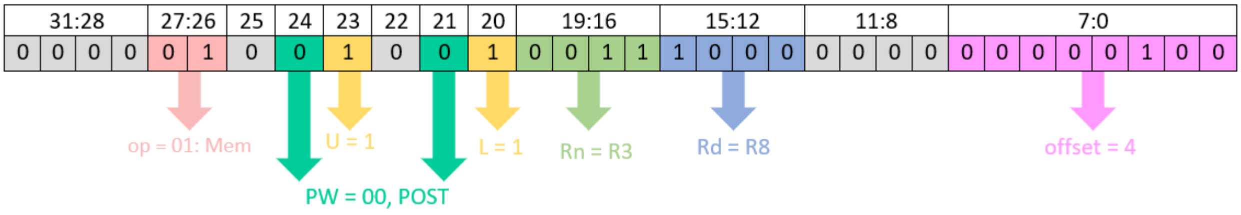
Section 2: Machine Codes

MOV R10, R0: 0b0000,0001,1010,0000,1010,0000,0000,0000 = 0x01A0A000 MOV Rd, Rm



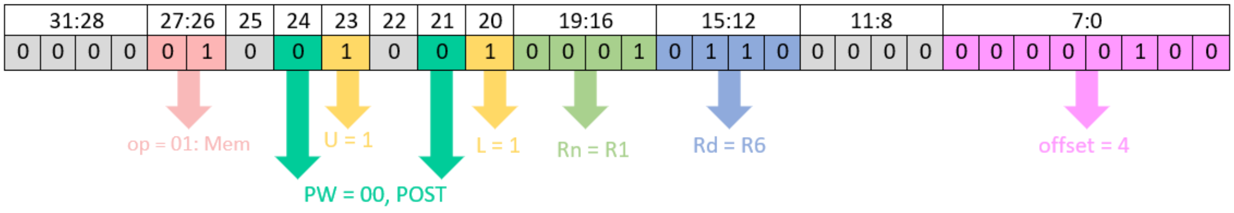
31:28	27:26	25	24:21	20	19:16	15:12	11:5	4	3:0
0 0 0 0	0 0	0	1 1 0 1	0	0 0 0 0	1 0 1 0	0 0 0 0 0 0 0	0	0 0 0 0

LDR R8, [R3], #4: 0b0000,0100,1001,0011,1000,0000,0000,0100 = 0x04938004 **LDR/STR *Rt*, [*Rn*], #offset**



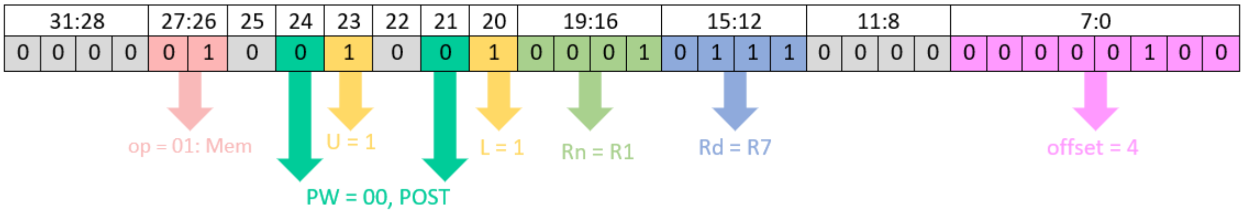
31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
0 0 0 0	0 1	0	0	1	0	0	1	0 0 1 1	1 0 0 0	0 0 0 0	0 0 0 0 0 1 0 0

LDR R6, [R1], #4: 0b0000,0100,1001,0001,0110,0000,0000,0100 = 0x04916004 **LDR/STR *Rt*, [*Rn*], #offset**



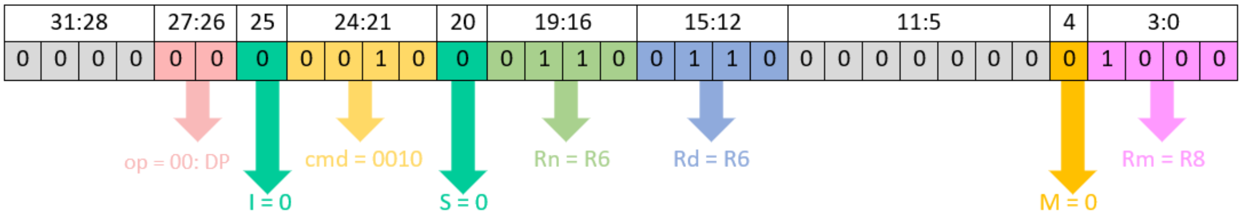
31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
0 0 0 0	0 1	0	0	1	0	0	1	0 0 0 1	0 1 1 0	0 0 0 0	0 0 0 0 0 1 0 0

LDR R7, [R1], #4: 0b0000,0100,1001,0001,0111,0000,0000,0100 = 0x04917004 **LDR/STR *Rt*, [*Rn*], #offset**



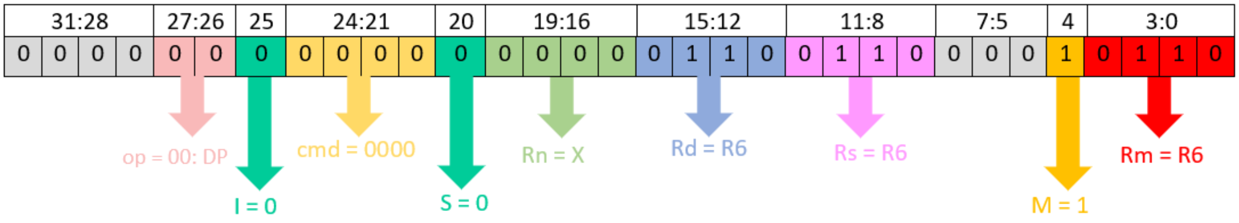
31:28	27:26	25	24	23	22	21	20	19:16	15:12	11:8	7:0
0 0 0 0	0 1	0	0	1	0	0	1	0 0 0 1	0 1 1 1	0 0 0 0	0 0 0 0 0 1 0 0

SUB R6, R6, R8: 0b0000,0000,0100,0110,0110,0000,0000,1000 = 0x00466008 `OP{S} Rd, Rn, Rm`



31:28	27:26	25	24:21	20	19:16	15:12	11:5	4	3:0
0 0 0 0	0 0	0	0 0 1 0	0	0 1 1 0	0 1 1 0	0 0 0 0 0 0 0	0	1 0 0 0

MUL R6, R6, R6: 0b0000,0000,0000,0000,0110,0110,0001,0110 = 0x00006616 `MUL Rd, Rm, Rs`



31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:5	4	3:0
0 0 0 0	0 0	0	0 0 0 0	0	0 0 0 0	0 1 1 0	0 1 1 0	0 0 0	1	0 1 1 0

SUB R7, R7, R9: 0b0000,0000,0100,0111,0111,0000,0000,1001 = 0x00477009

`OP{S} Rd, Rn, Rm`

31:28				27:26		25	24:21				20	19:16				15:12				11:5							4	3:0			
0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1

op = 00: DP

I = 0

cmd = 0010

S = 0

Rn = R7

Rd = R7

M = 0

Rm = R9

MUL R7, R7, R7: 0b0000,0000,0000,0000,0111,0111,0001,0111 = 0x00007717

`MUL Rd, Rm, Rs`

31:28				27:26		25	24:21				20	19:16				15:12				11:8				7:5			4	3:0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0	0	1	0	1	1	1

op = 00: DP

I = 0

cmd = 0000

S = 0

Rn = X

Rd = R7

Rs = R7

M = 1

Rm = R7

ADD R4, R6, R7: 0b0000,0000,1000,0110,0100,0000,0000,0111 = 0x00864007

`OP{S} Rd, Rn, Rm`

31:28				27:26		25	24:21				20	19:16				15:12				11:5							4	3:0			
0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1

op = 00: DP

I = 0

cmd = 0100

S = 0

Rn = R6

Rd = R4

M = 0

Rm = R7

LDR R5, [R2], #4: 0b0000,0100,1001,0010,0101,0000,0000,0100 = 0x04925004

LDR/STR *Rt*, [*Rn*], #offset

31:28				27:26		25	24	23	22	21	20	19:16				15:12				11:8				7:0							
0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0

op = 01: Mem

PW = 00, POST

U = 1

L = 1

Rn = R2

Rd = R5

offset = 4

CMP R10, R0: 0b0000,0001,0101,1010,0000,0000,0000,0000 = 0x015A0000

CMP Rn, Op2

31:28				27:26		25	24:21				20	19:16				15:12				11:5							4	3:0			
0	0	0	0	0	0	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

op = 00: DP

I = 0

cmd = 1010

S = 1

Rn = R10

Rd = X

M = 0

Rm = R0

CMP R4, R11: 0b0000,0001,0101,0100,0000,0000,0000,1011 = 0x0154000B

CMP Rn, Op2

31:28				27:26		25	24:21				20	19:16				15:12				11:5							4	3:0			
0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

op = 00: DP

I = 0

cmd = 1010

S = 1

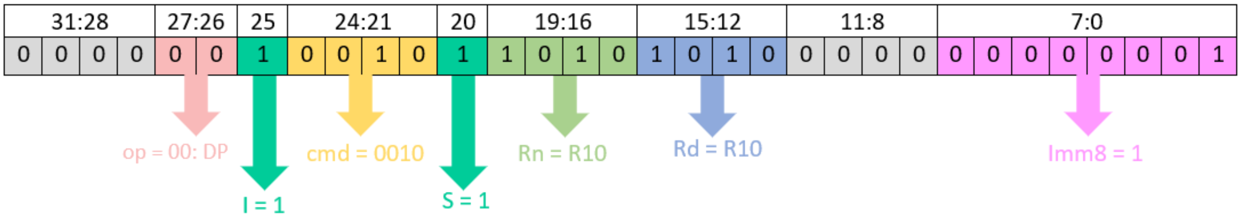
Rn = R4

Rd = X

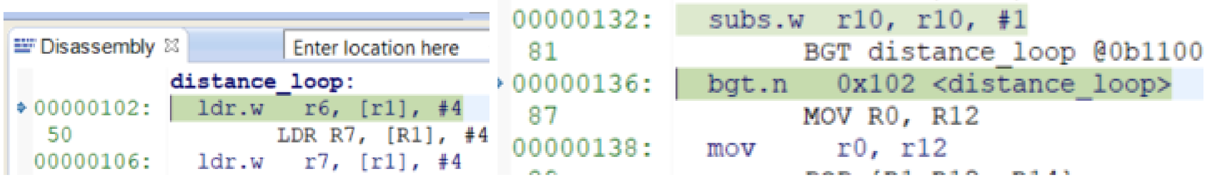
M = 0

Rm = R11

SUBS R10, R10, #1: 0b0000,0010,0101,1010,1010,0000,0000,0001 = 0x025AA001 `<OP Rd, Rn, #imm8>`



31:28				27:26		25	24:21				20	19:16				15:12				11:8				7:0							
0	0	0	0	0	0	1	0	0	1	0	1	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1



```
            distance_loop:
00000102:     ldr.w     r6, [r1], #4
 50                   LDR R7, [R1], #4
00000106:     ldr.w     r7, [r1], #4
```

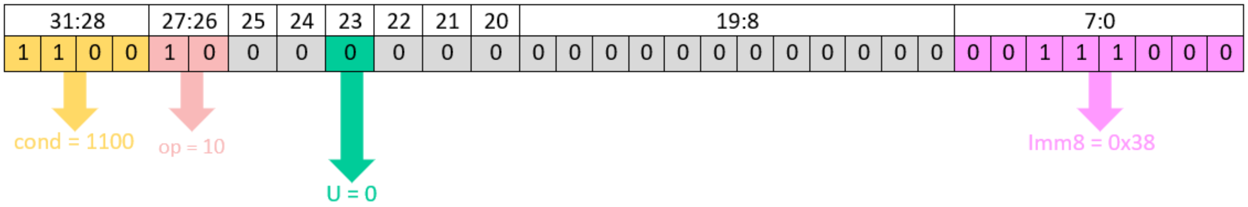
```
00000132:    subs.w   r10, r10, #1
 81                BGT distance_loop @0b1100
00000136:    bgt.n    0x102 <distance_loop>
 87                MOV R0, R12
00000138:    mov      r0, r12
```

PC = 0x136, PC+4 = 0x13A, BTA = 0x102

Offset = 0x102-0x13A = -0x38 = -0b00111000 (-0x38 = 14 instructions before PC+4)

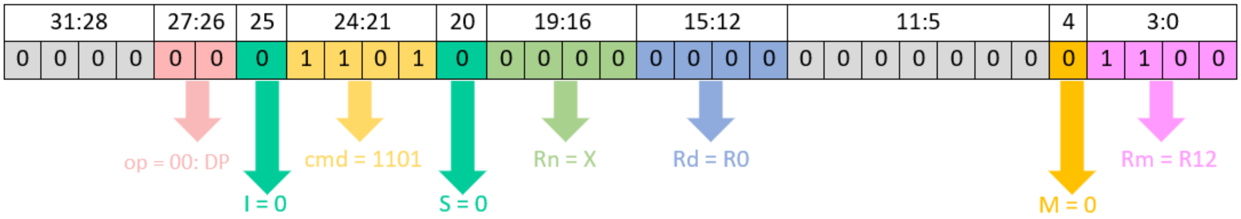
BGT distance_loop: 0b1100,1000,0000,0000,0000,0000,0011,1000 = 0xC8000038

```
B{cond} LABEL
LABEL encoded as #±imm8
```



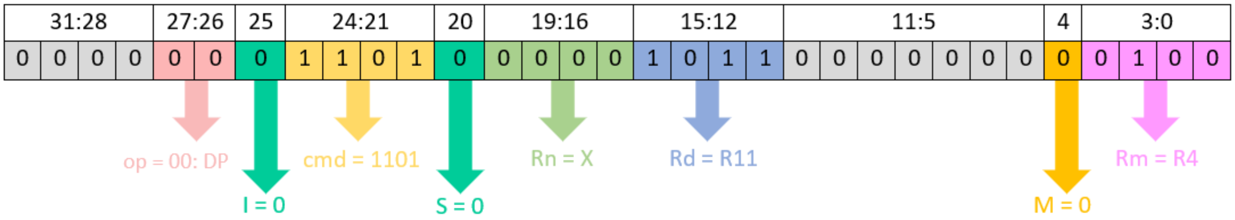
31:28				27:26		25	24	23	22	21	20	19:8												7:0							
1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0

MOV R0, R12: 0b0000,0001,1010,0000,0000,0000,00001100 = 0x01A0000C MOV Rd, Rm



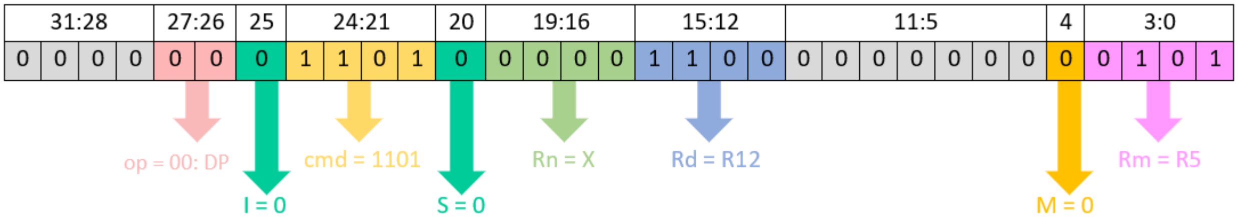
31:28				27:26		25	24:21				20	19:16				15:12				11:5							4	3:0			
0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

MOV R11, R4: 0b0000,0001,1010,0000,1011,0000,0000,0100 = 0x01A0B004 MOV Rd, Rm



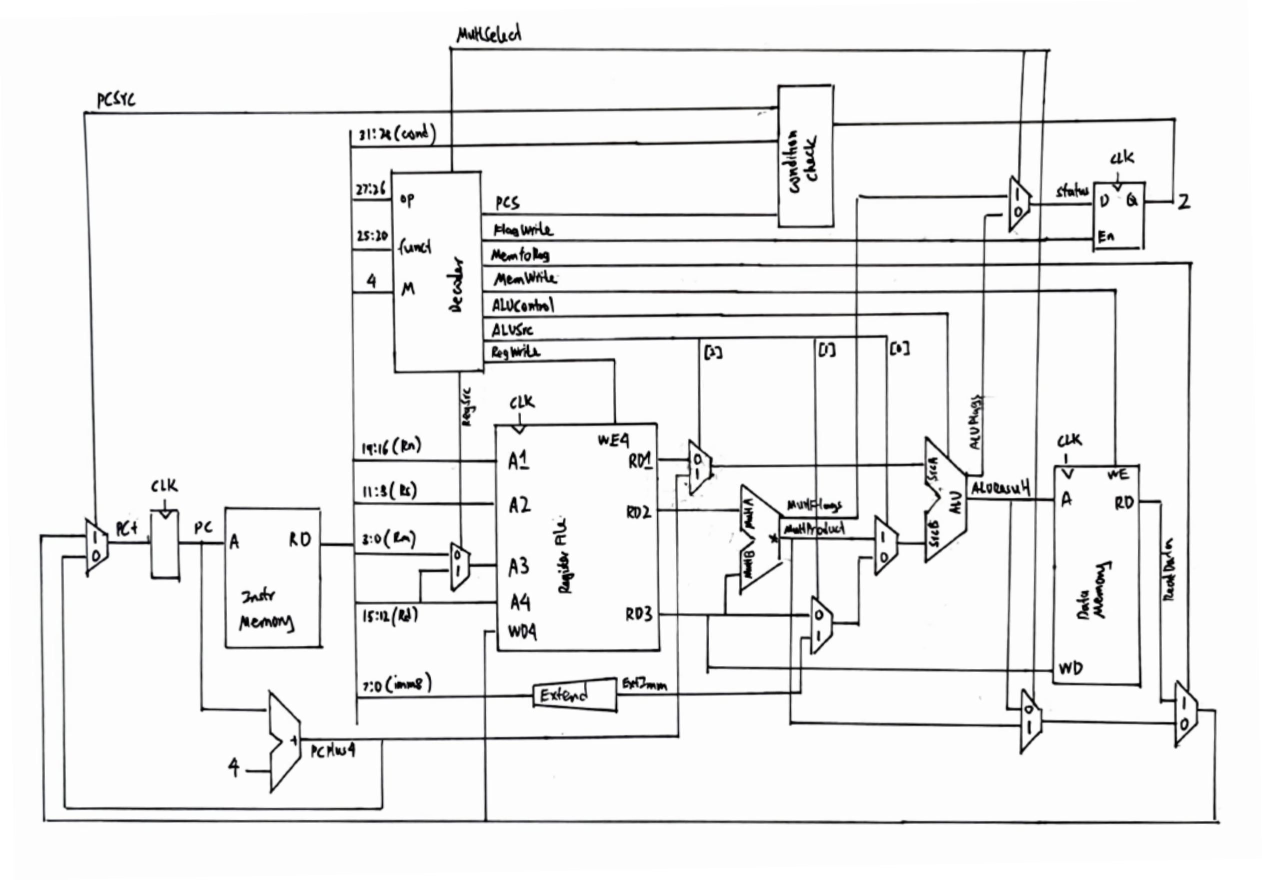
31:28				27:26		25	24:21				20	19:16				15:12				11:5							4	3:0			
0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0

MOV R12, R5: 0b0000,0001,1010,0000,1100,0000,0000,0101 = 0x01A0C005 MOV Rd, Rm



31:28				27:26		25	24:21				20	19:16				15:12				11:5							4	3:0			
0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Section 3: Microarchitecture Design



RegSrc = (op == 01) && (L == 0)

RegWrite = (op == 00) || ((op == 01) && (L == 1))

ALUSrc[2] = PCS = (op == 10)

ALUSrc[1] = !((op == 00) && (I == 0))

ALUSrc[0] = (op == 00) && (I == 0) && (M == 1)

ALUControl = (op == 00) ? [((I == 0) && (M == 1)) ? 0100 : cmd] : [U ? 0100 : 0010]

MemWrite = (op == 01) && (L == 0)

MemtoReg = (op == 01) && (L == 1)

FlagWrite = (op == 00) && (S == 1)

PCS = (op == 10)

PCSrc = (op == 10) && ((cond == 0000) ? (Z == 1) : 1)

MultSelect = (op == 00) && (I == 0) && (M == 1) && (cmd = 0000)

DP Operations : cmd

cmd	**Instruction**	**Operation**
0000	AND	Logical AND
0001	EOR	Logical Exclusive OR
0010	SUB	Subtract
0011	RSB	Reverse Subtract
0100	ADD	Add
0101	ADC	Add with Carry
0110	SBC	Subtract with Carry
0111	RSC	Reverse Subtract with Carry
1000	TST	Test Update flags after AND
1001	TEQ	Test Equivalence Update flags after EOR
1010	CMP	Compare Update flags after SUB
1011	CMN	Compare Negated Update flags after ADD
1100	ORR	Logical OR
1101	MOV	Move
1110	BIC	Bit Clear
1111	MVN	Move Not

Note : Multiplication is not one of the 16 ALU operations, though it is considered a DP operation. Multiplication is done in a separate multiplication unit and is a bit different from other DP operations.

- Flags are set by instructions with suffix S
  - Example : ADDS affects flags, ADD doesn’t
  - Exceptions : CMP, CMN, TST, TEQ which are used only to set flags (result is discarded)

cond	Mnemonic	Name	Condition Checked
0000	EQ	Equal	Z
0001	NE	Not equal	$\bar{Z}$
0010	CS / HS	Carry set / Unsigned higher or same	C
0011	CC / LO	Carry clear / Unsigned lower	$\bar{C}$
0100	MI	Minus / Negative	N
0101	PL	Plus / Positive of zero	$\bar{N}$
0110	VS	Overflow / Overflow set	V
0111	VC	No overflow / Overflow clear	$\bar{V}$
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z\ OR\ \bar{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\bar{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	$Z\ OR\ (N \oplus V)$
1110	AL (or none)	Always / unconditional	ignored

Interpretation based on SUBS/CMP

- ALUControl = (op==00)? cmd : (U? 0100:0010)
  - U = funct[3] = Instr[23]
  - 0100 – ALUControl for addition, 0010 – ALUControl for subtraction
  - For DP, ALUControl is cmd. For memory and branch, U bit decides whether imm8 is added or subtracted (i.e., whether the offset is positive or negative)
- ALUSrc[0] = !((op==00) && (I==0))
  - I = funct[5] = Instr[25]
  - For all except DP with register as Operand2, ALU_SrcB is immediate
- ALUSrc[1] = PCS
  - ALU_SrcA is PCPlus4 only for branch (doesn’t matter whether branch is taken or not. ALUResult is discarded when the branch is not taken anyway)