

# Informe de la Pràctica de Fundaments de Computadors

---

**Albert Martí i Arnau Mosquera Assignatura:**Fundaments de Computadors

**Pràctica:**Conversió de codi C a ARM - Anàlisi i processament de temperatures

**Data de lliurament:**22 d'abril de 2025

---

## Índex

### 1. 1\_celsfahr\_E9M22

#### 1.1 Especificacions

- 1.1.1 E9M22\_add
- 1.1.2 E9M22\_sub
- 1.1.3 E9M22\_neg
- 1.1.4 E9M22\_abs
- 1.1.5 E9M22\_mul\_s
- 1.1.6 E9M22\_are\_eq
- 1.1.7 E9M22\_are\_unordered\_s

#### 1.2 Disseny

- 1.2.1 E9M22\_add
- 1.2.2 E9M22\_sub
- 1.2.3 E9M22\_neg
- 1.2.4 E9M22\_abs
- 1.2.5 E9M22\_mul\_s
- 1.2.6 E9M22\_are\_eq
- 1.2.7 E9M22\_are\_unordered\_s

#### 1.3 Implementació

- 1.3.1 E9M22\_add
  - 1.3.2 E9M22\_sub
  - 1.3.3 E9M22\_neg
  - 1.3.4 E9M22\_abs
  - 1.3.5 E9M22\_mul\_s
  - 1.3.6 E9M22\_are\_eq
  - 1.3.7 E9M22\_are\_unordered\_s
- 

### 2. 2\_GeoTemp

#### 2.1 Especificacions

- 2.1.1 avgmaxmin\_city
- 2.1.2 avgmaxmin\_month

## 2.2 Disseny

- 2.2.1 avgmaxmin\_city
- 2.2.2 avgmaxmin\_month

## 2.3 Implementació

- 2.3.1 avgmaxmin\_city
- 2.3.2 avgmaxmin\_month

---

# 1\_celsfahr\_E9M22

## 1.1 Especificació de les funcions bàsiques E9M22

### 1.1.1 E9M22\_add

Aquesta funció rep dos nombres en format E9M22 (valors de 32 bits amb 1 bit de signe, 9 bits d'exponent i 22 bits de mantissa), i retorna la seva suma també en format E9M22. Es fa una normalització prèvia per alinear els exponents, es sumen les mantisses i posteriorment es renormalitza el resultat.

- **Paràmetres\*\*:**
  - **r0**: operand A (E9M22)
  - **r1**: operand B (E9M22)
- **Retorn\*\*:**
  - **r0**: resultat de la suma (E9M22)

---

### 1.1.2 E9M22\_sub

Realitza la resta entre dos valors E9M22. Internament, inverteix el signe del segon operand i reutilitza l'algorisme de la suma (**E9M22\_add**).

- **Paràmetres\*\*:**
  - **r0**: operand A (E9M22)
  - **r1**: operand B (E9M22)
- **Retorn\*\*:**
  - **r0**: resultat A - B en format E9M22

---

### 1.1.3 E9M22\_mul

Multiplica dos valors E9M22. La funció calcula el signe resultant, suma els exponents, i multiplica les mantisses amb el desplaçament adequat per mantenir la precisió. El resultat es normalitza.

- **Paràmetres\*\*:**
  - **r0**: operand A (E9M22)

- **r1**: operand B (E9M22)
  - **Retorn\*\***:
    - **r0**: resultat de la multiplicació (E9M22)
- 

#### 1.1.4 E9M22\_normalize\_and\_round\_s

Aquesta funció ajusta un valor E9M22 perquè estigui normalitzat (el bit més significatiu de la mantissa és 1) i realitza l'arrodoniment segons els bits descartats.

- **Paràmetres\*\***:
    - **r0**: mantissa no normalitzada
    - **r1**: exponent associat
  - **Retorn\*\***:
    - **r0**: valor normalitzat i arrodonit (E9M22)
- 

#### 1.1.5 count\_trailing\_zeros\_s

Compta quants bits a zero hi ha consecutius des del **bit menys significatiu cap amunt\*\*** dins d'un registre de 32 bits.

- **Paràmetres\*\***:
    - **r0**: valor d'entrada
  - **Retorn\*\***:
    - **r0**: nombre de zeros al final
- 

#### 1.1.6 count\_leading\_zeros\_s

Compta quants bits a zero hi ha consecutius des del **bit més significatiu cap avall\*\*** dins d'un registre de 32 bits.

- **Paràmetres\*\***:
    - **r0**: valor d'entrada
  - **Retorn\*\***:
    - **r0**: nombre de zeros al començament
- 

#### 1.1.7 E9M22\_neg

Canvia el signe d'un valor en format E9M22. Només cal invertir el bit de signe.

- **Paràmetres\*\***:
    - **r0**: valor original (E9M22)
  - **Retorn\*\***:
    - **r0**: valor amb signe invertit
- 

#### 1.1.8 E9M22\_abs

Retorna el valor absolut d'un nombre E9M22. Elimina el signe (posa el bit de signe a 0).

- **Paràmetres\*\*:**
    - **r0**: valor original (E9M22)
  - **Retorn\*\*:**
    - **r0**: valor absolut
- 

### 1.1.9 E9M22\_are\_eq\_s

Comprova si dos valors E9M22 són iguals. Té en compte el cas especial de NaN, que mai és igual a res (ni tan sols a ell mateix).

- **Paràmetres\*\*:**
    - **r0**: operand A
    - **r1**: operand B
  - **Retorn\*\*:**
    - **r0**: 1 si són iguals, 0 altrament
- 

### 1.1.10 E9M22\_are\_unordered\_s

Retorna 1 si algun dels dos operands és NaN. Aquest cas indica que no es poden comparar (unordered en terminologia IEEE-754).

- **Paràmetres\*\*:**
    - **r0**: operand A
    - **r1**: operand B
  - **Retorn\*\*:**
    - **r0**: 1 si algun operand és NaN, 0 altrament
- 

## 1.2 Disseny de les funcions bàsiques E9M22

### 1.2.1 E9M22\_add

El disseny segueix l'algorisme clàssic de suma de nombres en coma flotant:

1. **Extracció de camps\*\*:** Es separen el signe, exponent i mantissa dels dos operands.
  2. **Alineació d'exponents\*\*:** Si els exponents són diferents, es desplaça cap a la dreta la mantissa del nombre amb exponent més petit, per fer coincidir els exponents.
  3. **Operació de mantisses\*\*:** Si els signes són iguals, es sumen les mantisses. Si són diferents, es resten, i es conserva el signe del nombre més gran.
  4. **Normalització\*\*:** Es desplaça la mantissa resultant cap a l'esquerra (si hi ha zeros a l'inici) i s'ajusta l'exponent en conseqüència.
  5. **Arrodoniment\*\*:** Es tenen en compte els bits descartats per aplicar arrodoniment correcte.
  6. **Recomposició\*\*:** Finalment es torna a empaquetar el signe, l'exponent i la mantissa en format E9M22.
-

### 1.2.2 E9M22\_sub

Es basa en la suma, però amb el signe del segon operando invertit:

- Es canvia el bit de signe del segon operando (**B**), convertint la resta en una suma:  $A - B = A + (-B)$ .
  - A continuació, es reutilitza el mateix algoritme de **E9M22\_add**.
- 

### 1.2.3 E9M22\_mul

Per multiplicar dos nombres E9M22:

1. **Extracció de camps\*\***: Signe, exponent i mantissa dels dos operandes.
  2. **Càlcul del signe final\*\***: S'usa XOR entre els signes.
  3. **Exponent resultant\*\***: Es sumen els exponents i es resta el bias (511).
  4. **Multiplicació de mantisses\*\***: S'inclou el bit implícit (1) abans de multiplicar les mantisses.
  5. **Normalització\*\***: Es fa un desplaçament si cal, i es corregeix l'exponent.
  6. **Arrodoniment i saturació\*\***: Si es sobrepassa el rang, es genera  $+\infty$  o  $-\infty$ .
  7. **Recomposició\*\***: Es generen els bits finals del format E9M22.
- 

### 1.2.4 E9M22\_normalize\_and\_round\_s

Aquesta funció ajusta una mantissa i exponent després d'una operació:

1. **Normalització\*\***: Es detecta el primer bit 1 per saber quants llocs s'ha de desplaçar la mantissa.
  2. **Ajust de l'exponent\*\***: Cada desplaçament a l'esquerra incrementa l'exponent.
  3. **Arrodoniment\*\***: Es fa servir l'últim bit descartat i el següent per decidir si s'arrodoneix cap amunt.
  4. **Tractament de casos límit\*\***: Es gestiona el desbordament d'exponent i la generació de zero.
- 

### 1.2.5 count\_trailing\_zeros\_s

Implementa un algorisme per comptar zeros consecutius des del bit menys significatiu (LSB) cap a l'esquerra:

- Es fa un bucle que desplaça cap a la dreta fins que troba un 1.
  - Alternativament, es pot usar **rbit** (invertir bits) i **clz** (comptar zeros des del MSB).
- 

### 1.2.6 count\_leading\_zeros\_s

Utilitza la instrucció **clz** per comptar quants zeros hi ha des del MSB fins al primer bit 1:

- És útil per normalitzar mantisses.
  - Si no es pot usar **clz**, es pot fer un bucle que desplaça cap a l'esquerra i compta fins trobar un 1.
- 

### 1.2.7 E9M22\_neg

Inverteix el signe d'un nombre E9M22:

- Es fa un XOR amb `0x80000000`, que només modifica el bit més significatiu (bit 31).
- La resta del valor roman intacta.

### 1.2.8 E9M22\_abs

Retorna el valor absolut:

- Es posa el bit de signe a 0 fent un AND amb `0x7FFFFFFF`.
- Això manté l'exponent i la mantissa intactes.

### 1.2.9 E9M22\_are\_eq\_s

Compara si dos nombres E9M22 són iguals:

1. **Gestió de NaN\*\***: Si qualsevol operand és NaN, retorna 0.
2. **Comparació binària\*\***: Si cap és NaN, es comparen directament tots els bits.

### 1.2.10 E9M22\_are\_unordered\_s

Determina si qualsevol dels dos operands és NaN:

- Comprova si l'exponent és tot uns (`0x1FF`) i la mantissa no és zero.
- Si algun operand és NaN, retorna 1 (estan "desordenats" i no comparables).

## 1.3 Implementació

Fragments de codi ARM rellevants amb comentaris. Explica:

- Com s'ha traduït el codi C a codi ARM
- Com s'han fet les operacions aritmètiques i lògiques
- Com s'ha controlat el flux i gestionat els registres

#### 1.1.1 E9M22\_add

```
.global E9M22_add_s
E9M22_add_s:
    push {r4-r11, lr}           @ Guardem els registres que utilitzarem

    mov r4, r0                  @ r4 = num1
    mov r5, r1                  @ r5 = num2

    @ Obtenim els signes de num1 i num2
    ldr r6, =E9M22_MASK_SIGN
    and r7, r4, r6              @ signe1 → r7
    and r8, r5, r6              @ signe2 → r8

    @ Comprovem si num1 és NaN
    ldr r9, =E9M22_MASK_EXP
```

```

and r10, r4, r9
cmp r10, r9
bne check_nan2          @ Si no és NaN, mirem num2
ldr r9, =E9M22_MASK_FRAC
and r10, r4, r9
cmp r10, #0
bne return_nan1         @ num1 és NaN → retorna num1

```

#### check\_nan2:

```

@ Comprovem si num2 és NaN
ldr r9, =E9M22_MASK_EXP
and r10, r5, r9
cmp r10, r9
bne check_inf
ldr r9, =E9M22_MASK_FRAC
and r10, r5, r9
cmp r10, #0
bne return_nan2         @ num2 és NaN → retorna num2
pop {r4-r11, pc}

```

#### check\_inf:

```

@ Comprovem si num1 és infinit
ldr r9, =E9M22_MASK_EXP
and r10, r4, r9
cmp r10, r9
bne check_inf2

@ num1 és infinit
and r10, r5, r9
cmp r10, r9
bne return_inf1         @ num2 no és infinit → retorna num1

@ Tots dos infinits → comparar signes
cmp r7, r8
beq return_inf1         @ mateix signe → retorna num1
ldr r0, =E9M22_qNaN
ldr r0, [r0]            @ signes oposats → retorna NaN
b end_add

```

#### check\_inf2:

```

@ Comprovem si num2 és infinit
and r10, r5, r9
cmp r10, r9
bne check_zero

@ num2 és infinit → retorna num2
mov r0, r5
b end_add

```

#### return\_inf1:

```

mov r0, r4
b end_add

```

@ ----- Tractament de zeros -----

```

check_zero:
    @ Comprovem si num1 == 0
    ldr r9, =E9M22_MASK_EXP
    ldr r0, =E9M22_MASK_FRAC
    orr r9, r9, r0
    and r10, r4, r9
    cmp r10, #0
    bne check_zero2          @ num1 ≠ 0

    @ num1 és 0 → retorna num2
    mov r0, r5
    b end_add

check_zero2:
    and r10, r5, r9
    cmp r10, #0
    bne normal_add          @ num2 ≠ 0

    @ num2 és 0 → retorna num1
    mov r0, r4
    b end_add

@ ----- Tractament de nombres normals/denormals -----

normal_add:
    @ num1 → normal o denormal?
    ldr r9, =E9M22_MASK_EXP
    and r10, r4, r9
    cmp r10, #0
    beq num1_denorm

    @ num1 normal → obtenir mantissa i exponent
    ldr r9, =E9M22_MASK_FRAC
    and r6, r4, r9
    orr r6, r6, #E9M22_1_IMPLICIT_NORMAL
    mov r9, r4
    lsr r9, r9, #E9M22_m
    sub r9, r9, #E9M22_bias
    mov r10, r9              @ exp1 → r10
    b get_num2

num1_denorm:
    @ num1 denormal
    ldr r9, =E9M22_MASK_FRAC
    and r6, r4, r9
    ldr r10, =E9M22_Emin     @ exp1 = Emin

get_num2:
    @ num2 → normal o denormal?
    ldr r9, =E9M22_MASK_EXP
    and r11, r5, r9
    cmp r11, #0
    beq num2_denorm

```



```

    @ num2 normal
    ldr r9, =E9M22_MASK_FRAC
    and r11, r5, r9
    orr r11, r11, #E9M22_1_IMPLICIT_NORMAL
    mov r9, r5
    lsr r9, r9, #E9M22_m
    sub r9, r9, #E9M22_bias
    mov r12, r9                @ exp2 → r12
    b align_exponents

num2_denorm:
    @ num2 denormal
    ldr r9, =E9M22_MASK_FRAC
    and r11, r5, r9
    ldr r12, =E9M22_Emin      @ exp2 = Emin

@ ----- Alineació de mantisses -----

align_exponents:
    cmp r10, r12
    beq do_addition          @ exponents iguals → passar a suma

    @ Si exp1 < exp2 → desplaçar mant2
    movlt r0, r12
    sublt r0, r0, r10        @ dif_exp = exp2 - exp1
    cmp r0, #E9M22_e
    blt shift_mant2_left
    mov r1, #E9M22_e - 1
    lsl r11, r11, r1
    sub r12, r12, r1
    sub r0, r12, r10
    lsr r6, r6, r0
    add r10, r10, r0
    b do_addition

shift_mant2_left:
    lsl r11, r11, r0
    sub r12, r12, r0
    b do_addition

@ Cas contrari: exp1 > exp2 → (tractament simètric, intercanviant mantisses)

@ ----- Suma de mantisses -----

do_addition:
    cmp r7, r8
    beq same_sign

    @ Si signes diferents → negar una mantissa
    tst r7, r7
    rsbne r6, r6, #0
    rsbeq r11, r11, #0

```

```

same_sign:
    add r0, r6, r11                @ mantissa suma

    @ Si resultat negatiu → convertir a valor absolut
    cmp r0, #0
    bge mantissa_ok
    rsb r0, r0, #0

mantissa_ok:
    @ Calcular signe del resultat
    cmp r7, r8
    beq signe1_ok

    @ comparar magnituds
    mov r1, r4
    mov r2, r5
    bic r1, r1, #E9M22_MASK_SIGN
    bic r2, r2, #E9M22_MASK_SIGN
    cmp r1, r2
    movge r9, r7
    movlt r9, r8
    b call_normalize

signe1_ok:
    mov r9, r7

@ ----- Normalització i arrodoniment -----

call_normalize:
    mov r1, r10
    mov r2, r9
    bl E9M22_normalize_and_round_s

return_nan1:
    mov r0, r4
    b end_add

return_nan2:
    mov r0, r5
    b end_add

end_add:
    pop {r4-r11, pc}              @ Retornem de la funció

```

### 1.1.2 E9M22\_sub

```

.global E9M22_sub_s
E9M22_sub_s:
    push {lr}                      @ Guardem l'enllaç de retorn

```

```

    @ Neguem num2: canviem el bit de signe
    ldr r2, =E9M22_MASK_SIGN    @ Carreguem la màscara del bit de
signe
    eor r1, r1, r2                @ num2 = num2 XOR SIGN → canviem
signe

    @ Cridem a la funció de suma amb num1 i -num2
    bl E9M22_add_s

    pop {pc}                      @ Retornem

```

### 1.1.3 E9M22\_mul

```

.global E9M22_mul_s
E9M22_mul_s:
    push {r4-r11, lr}            @ Guardem els registres de treball i el
return

    mov r4, r0                    @ r4 = num1
    mov r5, r1                    @ r5 = num2

    @ Calcular el signe del producte: signe1 XOR signe2
    ldr r6, =E9M22_MASK_SIGN
    and r7, r4, r6                @ signe1 → r7
    and r8, r5, r6                @ signe2 → r8
    eor r9, r7, r8                @ signe_prod → r9

    @ -----
    @ Tractament de NaNs

    ldr r6, =E9M22_MASK_EXP
    and r0, r4, r6
    cmp r0, r6
    bne .check_nan2
    ldr r6, =E9M22_MASK_FRAC
    and r0, r4, r6
    cmp r0, #0
    bne .return_nan1              @ num1 és NaN

.check_nan2:
    ldr r6, =E9M22_MASK_EXP
    and r0, r5, r6
    cmp r0, r6
    bne .check_inf_zero
    ldr r6, =E9M22_MASK_FRAC
    and r0, r5, r6
    cmp r0, #0
    bne .return_nan2              @ num2 és NaN

.return_nan1:

```

```

    mov r0, r4
    b .end_mul

.return_nan2:
    mov r0, r5
    b .end_mul

@ -----
@ Tractament  $\infty \times 0 \rightarrow \text{NaN}$ ,  $\infty \times x \rightarrow \infty$ 

.check_inf_zero:
    ldr r6, =E9M22_MASK_EXP
    and r0, r4, r6
    cmp r0, r6
    bne .check_inf_num2          @ num1 no és  $\infty$ 

    @ num1 és  $\infty$ 
    ldr r6, =E9M22_MASK_EXP | E9M22_MASK_FRAC
    and r1, r5, r6
    cmp r1, #0
    moveq r0, r9
    ldreq r1, =E9M22_qNaN
    orreq r0, r0, r1              @  $\infty \times 0 \rightarrow \text{NaN}$ 
    movne r0, r9
    ldrne r1, =E9M22_INF_POS
    orrne r0, r0, r1             @  $\infty \times x \rightarrow \infty$ 
    b .end_mul

.check_inf_num2:
    ldr r6, =E9M22_MASK_EXP
    and r0, r5, r6
    cmp r0, r6
    bne .check_zeros            @ num2 no és  $\infty$ 

    @ num2 és  $\infty$ 
    ldr r6, =E9M22_MASK_EXP | E9M22_MASK_FRAC
    and r1, r4, r6
    cmp r1, #0
    moveq r0, r9
    ldreq r1, =E9M22_qNaN
    orreq r0, r0, r1             @  $0 \times \infty \rightarrow \text{NaN}$ 
    movne r0, r9
    ldrne r1, =E9M22_INF_POS
    orrne r0, r0, r1             @  $x \times \infty \rightarrow \infty$ 
    b .end_mul

@ -----
@ Tractament de zeros normals  $\rightarrow$  retorna  $\pm 0$ 

.check_zeros:
    ldr r6, =E9M22_MASK_EXP | E9M22_MASK_FRAC
    and r0, r4, r6
    cmp r0, #0
    moveq r0, r9                  @ signe del resultat

```

```

    beq .end_mul

    and r0, r5, r6
    cmp r0, #0
    moveq r0, r9
    beq .end_mul

    @ -----
    @ Extracció d'exponents i mantisses

.extract_mant_exp:
    @ num1
    ldr r6, =E9M22_MASK_EXP
    and r0, r4, r6
    cmp r0, #0
    beq .denorm1

    @ num1 normalitzat
    lsr r6, r0, #E9M22_m
    sub r6, r6, #E9M22_bias           @ exp1
    ldr r0, =E9M22_MASK_FRAC
    and r1, r4, r0
    orr r1, r1, #E9M22_1_IMPLICIT_NORMAL @ mant1
    b .done_num1

.denorm1:
    ldr r6, =E9M22_Emin
    ldr r0, =E9M22_MASK_FRAC
    and r1, r4, r0                   @ mant1 per denormalitzat

.done_num1:
    mov r10, r6                      @ exp1 → r10
    mov r6, r1                       @ mant1 → r6

    @ num2
    ldr r0, =E9M22_MASK_EXP
    and r1, r5, r0
    cmp r1, #0
    beq .denorm2

    lsr r1, r1, #E9M22_m
    sub r1, r1, #E9M22_bias           @ exp2
    ldr r0, =E9M22_MASK_FRAC
    and r2, r5, r0
    orr r2, r2, #E9M22_1_IMPLICIT_NORMAL @ mant2
    b .done_num2

.denorm2:
    ldr r1, =E9M22_Emin
    ldr r0, =E9M22_MASK_FRAC
    and r2, r5, r0                   @ mant2 per denormalitzat

.done_num2:
    mov r11, r1                      @ exp2 → r11

```

```
    mov r7, r2                                @ mant2 → r7

    @ -----
    @ Eliminar zeros finals per optimitzar precisió

    mov r0, r6
    bl count_trailing_zeros_s
    cmp r0, #0
    beq .mant2_trail
    lsr r6, r6, r0
    add r10, r10, r0

.mant2_trail:
    mov r0, r7
    bl count_trailing_zeros_s
    cmp r0, #0
    beq .do_product
    lsr r7, r7, r0
    add r11, r11, r0

.do_product:
    mov r0, r6
    mov r1, r7
    bl umul32x32_2x32                        @ r0=prod64lo, r1=prod64hi

    mov r2, r0
    mov r3, r1

    @ Calcular exponent del producte
    add r4, r10, r11
    sub r4, r4, #E9M22_m                    @ exp_prod

    cmp r3, #0
    beq .no_shift

    mov r0, r3
    bl count_leading_zeros_s
    rsb r5, r0, #32                        @ despl = 32 - clz

    @ Calcular sticky bit
    mov r1, #1
    lsl r1, r1, r5
    sub r1, r1, #1
    and r1, r2, r1
    cmp r1, #0
    moveq r1, #0
    movne r1, #1

    @ Construir mant_prod
    lsl r0, r3, r0
    lsr r5, r2, r5
    orr r0, r0, r5
    orr r0, r0, r1
```

```

    add r4, r4, r5                @ ajustem exponent

    b .normalize

.no_shift:
    mov r0, r2                    @ mant_prod

.normalize:
    mov r1, r4                    @ exponent del producte
    mov r2, r9                    @ signe del producte
    bl E9M22_normalize_and_round_s

.end_mul:
    pop {r4-r11, pc}             @ Restaurar i retornar

.data
.align 2

```

#### 1.1.4 E9M22\_normalize\_and\_round\_s

```

.global E9M22_normalize_and_round_s
E9M22_normalize_and_round_s:
    @ Aquesta és una versió dummy que només retorna el mantissa amb signe
    i exponent com a resultat.
    @ Hauràs d'implementar la versió real més endavant.

    push {lr}                     @ Guardem el registre de retorn (lr)
    per tornar després

    @ Suposant que:
    @ - r0 conté la mantissa
    @ - r1 conté l'exponent
    @ - r2 conté el signe del nombre (1 per negatiu, 0 per positiu)
    @ Aquesta funció hauria de normalitzar i arrodonir la mantissa,
    ajustant l'exponent i el signe.

    @ Però, de moment, només retornarem el registre r0 tal com està,
    @ sense realitzar cap operació de normalització o arrodoniment real.

    pop {pc}                      @ Recuperem el registre de retorn
    (pc) i tornem a la funció cridant

```

#### 1.1.5 count\_trailing\_zeros\_s

```

.global count_trailing_zeros_s
count_trailing_zeros_s:

```

```

    push {lr}                @ Guardem l'adreça de retorn a la pila
    mov r1, #0               @ Inicialitzem el comptador de zeros a r1

.loop_ctz:
    tst r0, #1               @ Comprovem si el bit menys significatiu és 1
    bne .end_ctz            @ Si ho és, sortim del bucle
    lsr r0, r0, #1           @ Desplacem r0 un bit cap a la dreta (r0 = r0
>> 1)
    add r1, r1, #1           @ Incrementem el comptador de zeros finals
    b .loop_ctz              @ Tornem a comprovar el següent bit

.end_ctz:
    mov r0, r1               @ El resultat (comptador) el posem a r0
    pop {pc}                 @ Recuperem l'adreça de retorn i retornem

```

### 1.1.6 count\_leading\_zeros\_s

```

.global count_leading_zeros_s
count_leading_zeros_s:
    push {lr}                @ Guarda l'adreça de retorn a la pila
    mov r1, #0               @ r1 comptador de zeros inicialitzat a 0
    mov r2, #1 << 31         @ r2 comença amb un 1 a la posició més
significativa (bit 31)

.loop_clz:
    tst r0, r2               @ Comprova si el bit corresponent de r0 és 1
    bne .end_clz            @ Si és 1, sortim del bucle
    lsr r2, r2, #1           @ Desplacem el bit de r2 cap a la dreta (bit
següent)
    add r1, r1, #1           @ Incrementem el comptador de zeros
    cmp r1, #32              @ Si hem comptat 32 bits, sortim (tots són
zeros)
    bge .end_clz
    b .loop_clz              @ Tornem a comprovar el següent bit

.end_clz:
    mov r0, r1               @ Retornem el nombre de zeros al principi
    (leading zeros)
    pop {pc}                 @ Recuperem l'adreça de retorn

```

### 1.1.7 E9M22\_neg

```

.global E9M22_neg_s
E9M22_neg_s:
    push {lr}                @ Guardem el registre de retorn (lr) per
poder tornar més endavant

    ldr r1, =E9M22_MASK_SIGN @ Carreguem la màscara per al bit de signe

```



```

    eor r0, r0, r1          @ Realitzem una operació XOR per invertir
                             el bit de signe (canviar el signe de num)

    pop {pc}                @ Recuperem el registre de retorn (pc) i
                             tornem a la funció cridant

@; E9M22_abs_s(): valor absolut de num

```

### 1.1.8 E9M22\_abs

```

.global E9M22_abs_s
E9M22_abs_s:
    push {lr}               @ Guardem el registre de retorn (lr) per
                             poder tornar més endavant

    ldr r1, =E9M22_MASK_SIGN @ Carreguem la màscara per al bit de signe
    bic r0, r0, r1          @ Esborrem el bit de signe de r0,
                             convertint el nombre en el seu valor absolut

    pop {pc}                @ Recuperem el registre de retorn (pc) i
                             tornem a la funció cridant

```

### 1.1.9 E9M22\_are\_eq\_s

```

.global E9M22_are_eq_s
E9M22_are_eq_s:
    push {r2, r3, lr}      @ Guardem els registres r2, r3 i lr (link
                             register) per poder tornar després

    mov r2, r0              @ r2 = num1 (copiem num1 a r2)
    mov r3, r1              @ r3 = num2 (copiem num2 a r3)

    @; Comprovar si algun operand és NaN
    ldr r1, =E9M22_MASK_EXP @ Carreguem la màscara per al camp
                             d'exponent
    and r0, r2, r1          @ Apliquem la màscara per obtenir l'exponent
                             de num1
    cmp r0, r1              @ Comprovem si l'exponent és el màxim (NaN)
    bne .check_nan2_mul     @ Si no és NaN, anem a comprovar num2
    ldr r1, =E9M22_MASK_FRAC @ Carreguem la màscara per al camp de la
                             fracció
    and r0, r2, r1          @ Apliquem la màscara per obtenir la fracció
                             de num1
    cmp r0, #0              @ Comprovem si la fracció és zero (NaN)
    bne .return_false       @ Si la fracció no és zero, num1 és NaN, i
                             retornem false

    .check_nan2_mul:

```

```

    ldr r1, =E9M22_MASK_EXP    @ Carreguem la màscara per al camp
d'exponent
    and r0, r3, r1             @ Apliquem la màscara per obtenir l'exponent
de num2
    cmp r0, r1                 @ Comprovem si l'exponent de num2 és el
màxim (NaN)
    bne .check_equal           @ Si no és NaN, anem a comprovar si els dos
nombres són iguals
    ldr r1, =E9M22_MASK_FRAC    @ Carreguem la màscara per al camp de la
fracció
    and r0, r3, r1             @ Apliquem la màscara per obtenir la fracció
de num2
    cmp r0, #0                 @ Comprovem si la fracció és zero (NaN)
    bne .return_false           @ Si la fracció no és zero, num2 és NaN, i
retornem false

.check_equal:
    cmp r2, r3                 @ Comprovem si num1 és igual a num2
    moveq r0, #1               @ Si són iguals, retornem 1
    bne .check_zero_eq         @ Si no són iguals, comprovem si són zero

.check_zero_eq:
    orr r0, r2, r3             @ Apliquem OR entre num1 i num2
    ldr r1, =E9M22_MASK_EXP | E9M22_MASK_FRAC @ Carreguem la màscara per
al camp d'exponent i fracció
    and r0, r0, r1             @ Apliquem la màscara per obtenir el valor
combinat
    cmp r0, #0                 @ Comprovem si el resultat és zero
    moveq r0, #1               @ Si és zero, retornem 1 (són iguals)
    movne r0, #0               @ Si no és zero, retornem 0 (no són iguals)
    b .end_eq

.return_false:
    mov r0, #0                 @ Si es troba un NaN o alguna altra
condició, retornem 0

.end_eq:
    pop {r2, r3, pc}          @ Recuperem els registres i tornem a la
funció cridant

```

### 1.1.10 E9M22\_are\_unordered\_s

```

.global E9M22_are_unordered_s
E9M22_are_unordered_s:
    push {r2, r3, lr}          @ Guardem els registres r2, r3 i lr (link
register) per poder tornar després

    mov r2, r0                 @ r2 = num1 (copiem num1 a r2)
    mov r3, r1                 @ r3 = num2 (copiem num2 a r3)

    ldr r1, =E9M22_MASK_EXP    @ Carreguem la màscara per al camp

```

```

d'exponent
    and r0, r2, r1                @ Apliquem la màscara per obtenir l'exponent
de num1
    cmp r0, r1                    @ Comprovem si l'exponent és el màxim (NaN)
    bne .check_nan2_addu          @ Si no és NaN, anem a comprovar num2
    ldr r1, =E9M22_MASK_FRAC     @ Carreguem la màscara per al camp de la
fracció
    and r0, r2, r1                @ Apliquem la màscara per obtenir la fracció
de num1
    cmp r0, #0                    @ Comprovem si la fracció és zero (NaN)
    bne .return_true             @ Si la fracció no és zero, num1 és NaN, i
retornem true

.check_nan2_addu:
    ldr r1, =E9M22_MASK_EXP      @ Carreguem la màscara per al camp
d'exponent
    and r0, r3, r1                @ Apliquem la màscara per obtenir l'exponent
de num2
    cmp r0, r1                    @ Comprovem si l'exponent de num2 és el
màxim (NaN)
    bne .check_nan2uf           @ Si no és NaN, anem a comprobar si num2 és
NaN
    ldr r1, =E9M22_MASK_FRAC     @ Carreguem la màscara per al camp de la
fracció
    and r0, r3, r1                @ Apliquem la màscara per obtenir la fracció
de num2
    cmp r0, #0                    @ Comprovem si la fracció és zero (NaN)
    bne .return_true             @ Si la fracció no és zero, num2 és NaN, i
retornem true

.check_nan2uf:
    mov r0, #0                    @ Si cap dels dos números és NaN, retornem
0 (no estan unordered)
    b .end_unordered

.return_true:
    mov r0, #1                    @ Si algun número és NaN, retornem 1 (estan
unordered)

.end_unordered:
    pop {r2, r3, pc}              @ Recuperem els registres i tornem a la
funció cridant

```

## 2\_GeoTemp

### 2.1 Especificacions

#### 2.1.1 avgmaxmin\_city

- **Nom** 😬 \*avgmaxmin\_city

- **Objectiu** 🤖 \*Calcular la temperatura mitjana, màxima i mínima d'una ciutat concreta al llarg dels 12 mesos de l'any.
- **Paràmetres d'entrada** \*\*:
  - `E9M22 temperatures[] []`: Matriu de temperatures (E9M22) de ciutats (files) i mesos (columnes).
  - `int num_cities`: Nombre total de ciutats.
  - `int city_index`: Índex de la ciutat sobre la qual es vol calcular les dades.
- **Sortida**:
  - `maxmin_t *result`: Estructura amb els valors calculats:
    - `avg`: Temperatura mitjana anual.
    - `max`: Temperatura màxima.
    - `max_pos`: Mes de la màxima.
    - `min`: Temperatura mínima.
    - `min_pos`: Mes de la mínima.

## 2.1.2 avgmaxmin\_month

- **Nom** 🤖 \*`avgmaxmin_month`
  - **Objectiu** 🤖 \*Calcular la temperatura mitjana, màxima i mínima d'un mes concret entre totes les ciutats disponibles.
  - **Paràmetres d'entrada** \*\*:
    - `E9M22 temperatures[] []`: Matriu de temperatures (E9M22).
    - `int num_cities`: Nombre total de ciutats.
    - `int month_index`: Índex del mes a analitzar.
  - **Sortida**:
    - `maxmin_t *result`: Estructura amb els valors calculats:
      - `avg`: Temperatura mitjana del mes.
      - `max`: Temperatura màxima registrada.
      - `max_pos`: Índex de la ciutat amb la màxima.
      - `min`: Temperatura mínima registrada.
      - `min_pos`: Índex de la ciutat amb la mínima.
- 

## 2.2 Disseny

### 2.2.1 avgmaxmin\_city

1. **Inicialització**:
  - Assignar la temperatura del primer mes de la ciutat a `min`, `max`, i `sum`.
  - Guardar posició inicial per `min_pos` i `max_pos` (habitualment 0).
2. **Recorregut dels mesos (0–11)**:
  - Per cada mes:
    - Comparar la temperatura amb els valors `min` i `max`.
    - Actualitzar `min`, `min_pos`, `max`, `max_pos` si escau.
    - Afegir la temperatura a la suma total.

### 3. Càlcul de mitjana:

- Dividir la suma acumulada entre 12 (nombre de mesos) amb la funció de divisió E9M22.

### 4. Retorn:

- Escriure `avg`, `min`, `min_pos`, `max`, `max_pos` a la struct `maxmin_t`.

## 2.2.2 avgmaxmin\_month

### 1. Inicialització:

- Assignar la temperatura de la primera ciutat al mes donat com a valor inicial per `min`, `max`, i `sum`.
- Inicialitzar `min_pos` i `max_pos` a 0.

### 2. Recorregut de ciutats (0–num\_cities-1):

- Per cada ciutat:
  - Accedir a la temperatura del mes.
  - Comparar amb `min` i `max` i actualitzar valors i posicions si cal.
  - Sumar la temperatura a la suma total.

### 3. Càlcul de mitjana:

- Dividir la suma total entre `num_cities` amb la rutina de divisió E9M22.

### 4. Retorn:

- Emmagatzemar `avg`, `min`, `min_pos`, `max`, `max_pos` a la struct `maxmin_t`.

## 2.3 Implementació

Fragments de codi ARM rellevants amb comentaris. Explica:

- Com s'ha traduït el codi C a codi ARM
- Com s'han fet les operacions aritmètiques i lògiques
- Com s'ha controlat el flux i gestionat els registres

### 2.3.1 avgmaxmin\_city

```
.global avgmaxmin_city
avgmaxmin_city:
    push {r4-r10, lr}

    mov r4, #1           @ i = 1
    mov r9, #0           @ idmax = 0
    mov r10, #0          @ idmin = 0

    mov r5, #48          @ 12 columnes x 4 bytes = 48
    mul r11, r2, r5       @ r11 = offset fila ciutat
    add r11, r0, r11      @ r11 = &ttemp[id_city][0]
```

```

    ldr r5, [r11]           @ avg = ttemp[id_city][0]
    mov r6, r5              @ max = avg
    mov r7, r5              @ min = avg

loop_city:
    cmp r4, #12
    beq end_loop_city

    mov r8, r4
    lsl r8, r8, #2           @ offset columna i * 4
    add r8, r11, r8          @ &ttemp[id_city][i]
    ldr r8, [r8]             @ tvar = ttemp[id_city][i]

    mov r0, r5               @ avg
    mov r1, r8               @ tvar
    bl E9M22_add
    mov r5, r0               @ avg += tvar

    mov r0, r8
    mov r1, r6
    bl E9M22_is_gt
    cmp r0, #0
    beq check_min_city

    mov r6, r8               @ max = tvar
    mov r9, r4               @ idmax = i

check_min_city:
    mov r0, r8
    mov r1, r7
    bl E9M22_is_lt
    cmp r0, #0
    beq next_city
    mov r7, r8               @ min = tvar
    mov r10, r4              @ idmin = i

next_city:
    add r4, r4, #1
    b loop_city

end_loop_city:
    ldr r1, =0x40A00000      @ 12.0 en E9M22
    mov r0, r5
    bl E9M22_div
    mov r5, r0               @ avg /= 12

    str r7, [r3, #MM_TMINC]
    str r6, [r3, #MM_TMAXC]

    mov r0, r7
    bl Celsius2Fahrenheit
    str r0, [r3, #MM_TMINF]

    mov r0, r6

```

```

bl Celsius2Fahrenheit
str r0, [r3, #MM_TMAXF]

strh r10, [r3, #MM_IDMIN]
strh r9, [r3, #MM_IDMAX]

mov r0, r5
pop {r4-r10, pc}

```

### 2.3.2 avgmaxmin\_month

```

.global avgmaxmin_month
avgmaxmin_month:
    push {r4-r10, lr}

    mov r4, #1                @ i = 1
    mov r9, #0                @ idmax = 0
    mov r10, #0               @ idmin = 0

    lsl r11, r2, #2           @ offset columna = id_month * 4
    add r6, r0, r11           @ &ttemp[0][id_month]
    ldr r5, [r6]              @ avg = ttemp[0][id_month]
    mov r6, r5                @ max = avg
    mov r7, r5                @ min = avg

loop_month:
    cmp r4, r1                @ i < nrow
    bge end_loop_month

    mov r8, r4
    mov r12, #48
    mul r8, r8, r12           @ offset fila i * 48
    add r8, r0, r8
    add r8, r8, r11           @ &ttemp[i][id_month]
    ldr r8, [r8]              @ tvar = ttemp[i][id_month]

    mov r0, r5
    mov r1, r8
    bl E9M22_add
    mov r5, r0

    mov r0, r8
    mov r1, r6
    bl E9M22_is_gt
    cmp r0, #0
    beq check_min_month
    mov r6, r8
    mov r9, r4

check_min_month:
    mov r0, r8

```

```
    mov r1, r7
    bl E9M22_is_lt
    cmp r0, #0
    beq next_month
    mov r7, r8
    mov r10, r4

next_month:
    add r4, r4, #1
    b loop_month

end_loop_month:
    mov r0, r1                @ n rows
    bl int_to_E9M22
    mov r1, r0                @ divisor
    mov r0, r5                @ avg
    bl E9M22_div
    mov r5, r0                @ avg /= n rows

    str r7, [r3, #MM_TMINC]
    str r6, [r3, #MM_TMAXC]

    mov r0, r7
    bl Celsius2Fahrenheit
    str r0, [r3, #MM_TMINF]

    mov r0, r6
    bl Celsius2Fahrenheit
    str r0, [r3, #MM_TMAXF]

    strh r10, [r3, #MM_IDMIN]
    strh r9, [r3, #MM_IDMAX]

    mov r0, r5
    pop {r4-r10, pc}
```