



UNIVERSITAT  
ROVIRA i VIRGILI

# FONAMENTS DE COMPUTADORS

## PRÀCTICA ARM

**Estudiants:**

Raúl Martín Morales

Amb material realitzat amb

Jan Torres Rodríguez

**Lab L3**

Curs 2022-23

# INDEX

<b>1. Fase1</b>	
1.1. Especificacions.....	3
1.2. Disseny.....	4
1.3. Implementació.....	5
1.4. Joc de proves.....	7
<b>2. Fase 2</b>	
2.1. Especificacions.....	8
2.2. Disseny.....	9
2.3. Implementació.....	10
2.4. Joc de proves.....	16
<b>3. Fase 3</b>	
3.1. Especificacions.....	17
3.2. Disseny.....	18
3.3. Implementació.....	19
3.4. Joc de proves.....	22

# 1. Fase 1

## 1.1 Especificacions

La primera fase es basa en traduir el codi source/CelsiusFahrenheit.c a llenguatge ensamblador. S'intenta seguir les instruccions que el codi en C proporciona, però allà on no hem sabut traduir "literalment", ho hem fet amb el propòsit d'obtenir el resultat més pròxim possible. El codi en C utilitza MAKE\_Q15 per passar els valors a coma fixa 1:16:15.

En aquest apartat dissenyarem dues funcions, una per passar de graus Celsius a Fahrenheit i l'altre per fer la operació contrària.

A la funció Celsius2Fahrenheit, apliquem una simple operació que s'encarregarà de fer la conversió d'unitats. El valor el qual hem de convertir en Celsius l'introduïm pel registre r0, i retorna el valor convertit a través del mateix registre.

Farem servir l'operació R0 ->  $\text{output} = (\text{input} * 9/5) + 32.0$ .

A la funció Fahrenheit2Celsius entra el valor pel registre r0 en Fahrenheit, i es retorna el valor passat a Celsius pel mateix registre. En aquesta rutina farà la conversió inversa, utilitzant aquesta operació R0 ->  $\text{output} = (\text{input} - 32.0) * 5/9$ .

## 1.2 Disseny

Primerament, al veure que els nombres estan codificats en Q15, coma fixa 1:16:15, hem decidit implementar una multiplicació de  $2^{15}$  a tot valor que utilitzem en l'operació.

Per altra banda, com les temperatures que entren a la rutina poden ser negatives, no podem utilitzar l'instrucció mul (doncs no té en compte els signes), llavors hem decidit fer ús de l'instrucció smull, que conserva el signe negatiu a la multiplicació. A l'utilitzar smull però, hem de tenir en ment que es dividirà el resultat en dues parts, RdLo, que seran el 32 bits baixos, i RdHi que seran els 32 bits alts, que haurem d'ordenar-los per obtenir el resultat de la multiplicació correctament.

# 1.3 Implementació

CelsiusFahrenheit.s

.include "Q15.i"

.text

.align 2

.arm

.global Celsius2Fahrenheit

Celsius2Fahrenheit:

push {r1-r3, lr}

mov r1, #0

ldr r3, =58982

@;  $9/5 * 2^{15} = 58.982$

smull r1, r2, r0, r3

@;  $r1 = r0 * 9/5$  (32 bits resultants amb menys pes)

@;  $r2 = r0 * 9/5$  (32 bits resultants amb mes pes)

mov r1, r1, lsr #15

@; Desplaçament de 15 bits de la part baixa a la dreta

orr r1, r2, lsl #17

@; S'omplen 17 bits de la part alta (preferentment) a

@; l'esquerra

add r0, r1, #1048576

@;  $r0 = r1 + 1.048.576$  |  $1.048.576 = 32 * 2^{15}$

pop {r1-r3, pc}

.global Fahrenheit2Celsius

Fahrenheit2Celsius:

push {r1-r3, lr}

mov r1, #0

ldr r3, =18204

@;  $5/9 * 2^{15} = 18.204$

sub r0, r0, #1048576

@;  $r0 = (\text{input} - 32) | 1.048.576 = 32 * 2^{15}$

smull r1, r2, r0, r3

@;  $r1 = r0 * 5/9$  (32 bits resultants amb menys pes)

@;  $r2 = r0 * 5/9$  (32 bits resultants amb més pes)

mov r1, r1, lsr #15

@; Desplaçament de 15 bits de la part baixa a la dreta

orr r1, r2, lsl #17

@; S'omplen 17 bits de la part alta (preferentment) a

@; l'esquerra

mov r0, r1

@;  $r0 = \text{temperatura en Celsius resultant (r1)}$

pop {r1-r3, pc}

## demo.s

.data

```
        .align 2
temp1C: .word 0x00119AE1      @; temp1C = 35.21 C
temp2F: .word 0xFFFF42000    @; temp2F = -23.75 F
```

.bss

```
        .align 2
temp1F: .space 4             @; expected conversion: 95.377532958984375 F
temp2C: .space 4             @; expected conversion: -30.971466064453125 C
```

.text

```
        .align 2
        .arm
        .global main
main:
    push {r1-r3, lr}
    ldr r1, =temp1C
    ldr r2, =temp1F
    ldr r0, [r1]
    bl Celsius2Fahrenheit      @; temp1F = Celsius2Fahrenheit(temp1C);
    str r0, [r2]

    ldr r1, =temp2F
    ldr r3, =temp2C
    ldr r0, [r1]
    bl Fahrenheit2Celsius     @; temp2C = Fahrenheit2Celsius(temp2F);
    str r0, [r3]

    @; TESTING POINT: check the results
    @; (gdb) p /x temp1F      -> 0x002FB053
    @; (gdb) p /x temp2C      -> 0xFFFF083A7
    @; BREAKPOINT
    mov r0, #0                @; return(0)

    pop {r1-r3, pc}

.end
```

## 1.4. Joc de proves

S'han executat sense errors els jocs de proves proporcionats.

*demo.s*

temp1C:	.word 0x00119AE1	temp1C = 35.21 °C
temp2F:	.word 0xFFFF42000	temp2F = -23.75 °F

### Resultats

temp1F	=	0x2fb053	=	95,37753°F
temp2C	=	0xff083a7	=	-30.9714°C

### Resultats de *debug* (test\_CelsiusFahrenheit.c)

```
7% Console Window

Breakpoint 3, main () at tests/test_CelsiusFahrenheit.c:118

(gdb) p num_ok
$3 = 24

(gdb) p num_ko
$4 = 0

(gdb) p num_tests
$5 = 24

(gdb)
```

A més a més, modificarem el test\_CelsiusFahrenheit.c per comprovar nous valors:

### Resultats del test:

```
/* the list of test case values */
test_ops_struct test_case[] =
/* Tests Celsius -> Fahrenheit */
[
    {'C', MAKE_Q15(-150.0),
     MAKE_Q15(-238.0)},
    {'C', MAKE_Q15(-6.6),
     MAKE_Q15(20.12)},
    {'C', MAKE_Q15(3.0),
     MAKE_Q15(37.4)},
    /* Tests Fahrenheit -> Celsius */
    {'F', MAKE_Q15(-367.42),
     MAKE_Q15(-221.9)},
    {'F', MAKE_Q15(2.0),
     MAKE_Q15(-16.6667)},
    {'F', MAKE_Q15(32.0),
     MAKE_Q15(1.1111)},
    {'F', MAKE_Q15(31.111),
     MAKE_Q15(-0.493888889)},
    {'F', MAKE_Q15(42345.5),
     MAKE_Q15(23507.5)}
];
```

```
7% Console Window

(gdb) p num_ok
$1 = 7

(gdb) p num_ko
$2 = 1

(gdb) p num_tests
$3 = 8
```

## 2. Fase 2

### 2. 1 Especificacions

A la segona fase, l'objectiu principal serà crear dues rutines per calcular els valor mitjans, màxims i mínims d'una taula de temperatures expressades en graus Celsius o Fahrenheit, en format Q15, és a dir, coma fixa 1:16:15.

Creem l'arxiu avgmaxmintemp.s a la carpeta /source, on escriurem el codi en llenguatge ensamblador.

La primera rutina avgmaxmin\_city recorrerà tots els mesos d'una ciutat on calcularà la seva temperatura mitjana, màxima i mínima de la taula de temperatures ttemp[][12], expresades en graus Celsius, on les files son les ciutats i les columnes són els mesos. Es passarà el valor id\_city pel registre r2, que serà l'índex de la fila a calcular. I com últim paràmetre, tenim la variable \*mmres, on es guardaran els valors de les temperatures màximes i mínimes en Celsius i Fahrenheit i la posició en la qual es troben.

Aquesta rutina retornarà com a valor la variable avg, que serà la temperatura mitjana.

La segona rutina avgmaxmin\_month recorrerà totes les files (ciutats) d'un mes en específic, per calcular la temperatura mitjana, màxima i mínima de la taula ttempt[][12] de temperatures expressades en graus Celsius. Passarem la variable nrows com a r1 que contindrà el número de files de la taula i la variable id\_month en r2 que indicarà el mes del qual volem estudiar les temperatures.

En aquesta rutina també tindrem la variable \*mmres, on guardarem els valors de les temperatures màximes i mínimes en Celsius i Fahrenheit i la posició en la que es troben aquestes.

I per últim, la rutina retornarà el valor de avg on es troba la temperatura mitjana del mes escollit.



## 2. 2 Disseny

Pel disseny del Makefile hem agafat de base l'arxiu de la primera fase, i a partir d'aquest hem anat canviant tot allò que fos necessari. Pel geotemp.elf vam introduir tots els build/ necessaris per executar-lo excepte el build/, ja que no influeix.

Pel build/geotemp.o hem vist que dins source/geotemp.c ens indica quins arxius .h hem d'incloure.

Per saber quins arxius hem d'incloure dins el build/avgmaxmintemp.o ens fixem dins source/avgmaxmintemp.s on indica que hem d'incloure l'arxiu avgmaxmintemp.i.

En els fitxers que tenim, observem que tenim source/data.c, el qual també haurem d'implementar. Aleshores, dins el source/data.o ens indicarà que hem d'incloure Q15.h geotemp.h i data.h, aleshores el fiquem dins del build/data.o.

Per acabar el Makefile, necessitem les comandes que generen el debugger i serviran per comprovar els tests. En test\_geotemp.elf introduïrem els arxius del build/ excepte build/geotemp.o, ja que ja es troba implementat al build/test\_geotemp.o. També afegim tots els arxius de p\_lib/.

Per últim, al build/test\_geotemp.o haurem d'incloure els arxius .h que s'indiquen dins tests/test\_geotemp.c.

A l'hora de dissenyar les rutines de avgmaxmin\_city i avgmaxmin\_month, haurem de saber com llegir les diferents temperatures de cada mes (que formen les columnes). Cal afegir que el procediment per llegir les temperatures serà diferent en les dos rutines.

Per saber la posició d'inici en la rutina avgmaxmin\_city, multiplicarem id\_city (índex de la ciutat), pels 4 bytes que ocupa cada posició i pels 12 mesos que indiquen les 12 columnes. A partir d'aquí ja podríem anar passant per cada posició sumant de 4 en 4.

A la rutina avgmaxmin\_month, per saber la posició inicial multiplicarem id\_month (que indica el mes a estudiar), per 4 (que són els bytes que ocupa cada casella), i guardarem a un registre el valor 48 obtingut de la multiplicació  $4 \times 12$  (4 bytes pels 12 mesos) que s'anirà sumant per arribar a la següent posició, és a dir, anirà passant de fila en fila.

I per acabar, quan volem guardar el valor dels mínims i màxims en graus Celsius i Fahrenheit hauríem de tenir en ment que són nombres enters(int), que ocupen 32 bits. Els guardarem amb un str, però per guardar els id\_min i id\_max, com són variables short que ocupen 16 bits, farem servir strh.

## 2. 3 Implementació

avgmaxmintemp.s

```
.include "avgmaxmintemp.i"
```

```
.bss
```

```
    .align 2
```

```
q: .space 4
```

```
r: .space 4
```

```
.text
```

```
    .align 2
```

```
    .arm
```

```
    .global avgmaxmin_city
```

```
avgmaxmin_city:
```

```
    push {r4-r12, lr}
```

```
    mov r7, #4
```

@; 4 bytes (tamany de cada element, de  
cada columna)

```
    mov r5, #12
```

```
    mul r4, r2, r5
```

@; 12 pels mesos (columnes) i 4 pels bytes que  
ocupa cada element

```
    mul r4, r7
```

```
    add r0, r4
```

@; Es sumen a la posicio 0 els elements que hi ha abans de la  
@; ciutat determinada (ttemp(id\_city,month) = (0,0) + (filla  
@; ciutat\*NC)\*Tamany)

```
    mov r5, #1
```

@; r5 = i = 1

```
    ldr r8, [r0]
```

@; avg = temperatura del primer mes  
(ttemp[id\_city][0])

```
    mov r9, r8
```

@; min = temperatura del primer mes

```
    mov r10, r8
```

@; max = temperatura del primer mes

```
    mov r11, #0
```

@; idmin = 0

```
    mov r12, #0
```

@; idmax = 0

```
    .Lfor:
```

```
        cmp r5, #12
```

```
        bhi .Lendfor
```

@; Si i > 12 (mesos de l'any) surt del  
bucle

```
        ldr r6, [r0, r7]
```

@; Llegeix els primers 4 bytes a r6

```
        add r8, r6
```

```
    .Lif1:
```

```
        cmp r6, r10
```

@; Si tvar <= max surt de l'if

```
        ble .Lendif1
```

```
        mov r10, r6
```

@; max = tvar

```
        mov r12, r5
```

@; idmax = i

.Lendif1:

.Lif2:

cmp r6, r9	@; Si tvar >= min surt de l'if
bhs .Lendif2	
mov r9, r6	@; min = tvar
mov r11, r5	@; idmin = i

.Lendif2:

add r5, #1	@; i++
add r7, #4	@; següent columna
b .Lfor	

.Lendfor:

```
mov r0, r8
mov r1, #12
mov r5, #32768
smull r6, r7, r1, r5
mov r6, r6, lsl #15
orr r6, r7, lsr #17
mov r1, r6
ldr r2,=q
bl div_mod
ldr r6,=q
```

str r9, [r3]	@; Es guarda en la primera posició de r3 la temp mínima en Celsius
--------------	--

str r10, [r3, #4]	@; Es guarda en la segona posició de r3 la temp maxima en Celsius
-------------------	---

mov r0, r9	@; Es posa a r0 la temp min porque es el registre que la rutina fa servir
------------	---

bl Celsius2Fahrenheit	
str r0, [r3, #8]	@; Es guarda en la tercera posició de r3 la temp mínima en Fahrenheit

mov r0, r10	
bl Celsius2Fahrenheit	
str r0, [r3, #12]	@; Es guarda en la quarta posició de r3 la temp max en Fahrenheit

strh r11, [r3, #16]	@; Es guarda en la cinquena posició de r3 la posició de la temp min
---------------------	---

strh r12, [r3, #18]	@; Es guarda en la sisena posició de r3 la posició de la temp max
---------------------	---

mov r0, r6	@; Es guarda avg a R0 -> return(avg)
pop {r4-r12, pc}	

```

.global avgmaxmin_month
avgmaxmin_month:
    push {r4-r12, lr}
    mov r4, #0           @; idmin = 0
    mov r5, #0           @; idmax = 0
    mov r6, #4           @; Tamany de cada element
    mul r10, r6, r2       @; R10 = 4 * id_month (elements abans de l'escollit)
    add r0, r10
    ldr r7, [r0]         @; R7 = avg
    mov r8, r7           @; R8 = max
    mov r9, r7           @; R9 = min

    mov r10, #1          @; R10 = i = 1
    mov r11, #48         @; R11 = 12 * 4 (Elements fins la proxima temperatura
                        del mes multiplicat pel tamany)

    ldr r12, [r0, r11]   @; Temperatura actual
    add r7, r12

.Lwhile:
    cmp r10, r1
    bhs .Lendwhile

    .Lif3:
        cmp r12, r8
        ble .Lendif3
        mov r8, r12
        mov r5, r10       @; R5(idmax) = R10
    .Lendif3:

    .Lif4:
        cmp r12, r9
        bhs .Lendif4
        mov r9, r12
        mov r4, r10
    .Lendif4:

    add r11, #48          @; Es sumen els elements per arribar a la
                        proxima temp a la següent fila
    add r10, #1          @; i++
    ldr r12, [r0, r11]   @; Temperatura actual
    add r7, r12
    b .Lwhile
.Lendwhile:

```

.Lif5:	
cmp r7, #0	@; Es comprova si avg es negatiu o positiu
bge .Lelse5	
mov r11, #-1	
smull r6, r10, r7, r11	@; Es canvia de signe i s'assignen els bits de mes bits a R6
mov r6, r6, lsr #15	
orr r6, r10, lsl #17	
mov r0, r6	
b .Lendif5	
.Lelse5:	
mov r0, r7	@; R0 = avg sense dividir entre n rows
.Lendif5:	
ldr r2, =q	
ldr r3, =r	
bl div_mod	@; Es guarda el quocient (avg) a R2
ldr r10, =q	
str r9, [r3]	@; Es guarda en la primera posició de r3 la temp mínima en Celsius
str r8, [r3, #4]	@; Es guarda en la segona posició de r3 la temp máxima en Celsius
mov r0, r9	@; Es posa a r0 la temp min porque es el registre que la rutina fa servir
bl Celsius2Fahrenheit	
str r0, [r3, #8]	@; Es guarda en la tercera posició de r3 la temp mínima en Fahrenheit
mov r0, r8	
bl Celsius2Fahrenheit	
str r0, [r3, #12]	@; Es guarda en la quarta posició de r3 la temp max en Fahrenheit
strh r4, [r3, #16]	@; Es guarda en la cinquena posició de r3 la posició de la temp min
strh r5, [r3, #18]	@; Es guarda en la sisena posició de r3 la posició de la temp max
ldr r0, [r10]	@; Es guarda avg a R0 -> return(avg)
pop {r4-r12, pc}	

## Makefile (Canvis explicats en el disseny)

```
ARCH := -march=armv5te -mlittle-endian
INCL  := -I./include
ASFLAGS := $(ARCH) $(INCL) -g
CCFLAGS := -Wall -gdwarf-3 -O0 $(ARCH) $(INCL)
LDFLAGS := -z max-page-size=0x8000 -Lp_lib

#-----
# make commands
#-----
geotemp.elf : build/avgmaxmintemp.o build/data.o build/geotemp.o
             arm-none-eabi-ld $(LDFLAGS) build/avgmaxmintemp.o build/data.o build/geotemp.o \
             \
             p_lib/startup.o p_lib/CelsiusFahrenheit.o
p_lib/libfoncompus.a -o geotemp.elf

build/geotemp.o: source/geotemp.c include/avgmaxmintemp.h include/CelsiusFahrenheit.h
include/data.h \
             include/divmod.h include/geotemp.h include/Q15.h
             arm-none-eabi-gcc $(CCFLAGS) -c source/geotemp.c -o build/geotemp.o

build/avgmaxmintemp.o: source/avgmaxmintemp.s include/avgmaxmintemp.h
             arm-none-eabi-as $(ASFLAGS) -c source/avgmaxmintemp.s -o
build/avgmaxmintemp.o

build/data.o: source/data.c include/Q15.h include/geotemp.h include/data.h
             arm-none-eabi-gcc $(CCFLAGS) -c source/data.c -o build/data.o

test_geotemp.elf : build/test_geotemp.o build/avgmaxmintemp.o
                 arm-none-eabi-ld $(LDFLAGS) build/test_geotemp.o build/avgmaxmintemp.o \
                 build/data.o p_lib/CelsiusFahrenheit.o p_lib/startup.o p_lib/libfoncompus.a -o
test_geotemp.elf

build/test_geotemp.o: tests/test_geotemp.c include/avgmaxmintemp.h include/Q15.h
                 arm-none-eabi-gcc $(CCFLAGS) -c tests/test_geotemp.c -o build/test_geotemp.o

#-----
# clean commands
#-----
clean :
        @rm -fv build/*
        @rm -fv *.elf
```

```
#-----  
# run commands  
#-----  
run : geotemp.elf  
      arm-eabi-insight geotemp.elf &  
  
#-----  
# debug commands  
#-----  
debug : test_geotemp.elf  
        arm-eabi-insight test_geotemp.elf &
```

## 2.4 Joc de proves

Pel joc de proves de la fase dos farem servir el testgeotemp.c i agafarà diferents valors amb els que comprovarà si les rutines creades en aquesta fase funcionen correctament.

A partir del test tenim un resultat erroni, per tant els resultats que surten de les rutines utilitzades no son correctes.

```
7% Console Window
(gdb) p num_ok
$1 = 1

(gdb) p num_ko
$2 = 5

(gdb) p num_tests
$3 = 6

(gdb) |
```



# 3. Fase 3

## 3. 1 Especificacions

L'objectiu principal és implementar en llenguatge ensamblador les quatre operacions aritmètiques (la suma, la resta, la multiplicació i divisió) amb números codificats en coma fixa 1:16:15 (Q15). També voldrem controlar el desbordament en cada rutina, i amb els tests predeterminats (test\_Q15.c) i veure si els resultats son correctes.

Aquestes operacions seran introduïdes dins d'una llibreria (libQ15.a).

A la primer rutina add\_Q15, es farà el càlcul de la suma de dos nombres, per r0 el num1 i per r1 el num2. La rutina retornarà el valor de la suma per r0. Entrarà de 3r paràmetre overflow per referència, on serà r2.

També es calcularà el desbordament. Quan no n'hi hagi serà 0, i quan sí n'hi hagi serà 1, és a dir, quan la suma de dos nombres positius arrosseguen un resultat negatiu o la suma de dos nombres negatius arrosseguen un resultat positiu el overflow serà 1.

A la segona rutina sub\_Q15, es voldrà fer la resta de nombres (r0-r1) on r1 serà num1 i r2 serà num2, codificats en coma fixa Q15. La rutina retornarà el valor de la resta per r0.

De la mateixa manera que a l'anterior rutina, entrarà de 3r paràmetre per referència l'overflow per r2.

Per la tercera rutina mul\_Q15 es basarà en multiplicar els dos nombres, num1(r0) i num2(r1), en coma fixa Q15.

El resultat d'aquesta operació es retornarà per r0.

Com totes les altres rutines també valorarà l'overflow que es produeix a la multiplicació d'aquests dos nombres. En el cas de la multiplicació serà més difícil, doncs haurem de valorar si el resultat del càlcul és negatiu.

Com última rutina tenim div\_Q15, amb la qual dividirem dos nombres, num1/num2, on num1 és r0 i num2 és r1. Amb aquesta rutina retornarem el valor de la divisó.

Per assolir la divisió en ensamblador haurem de fer servir l'anterior rutina (mul\_Q15), ja que no podem dividir dos nombres com tal, sinó que farem la inversa del denominador i multiplicar-lo pel numerador.

## 3. 2 Disseny

En una primera instància, les rutines que hem de programar poden semblar més senzilles del que són en realitat, doncs ja existeixen comandes que faran algunes de les operacions aritmètiques que es sol·liciten. Però al necessitar saber si hem tingut overflow, haurem d'utilitzar màscares.

En les dues primeres operacions, add i sub, s'ha decidit no utilitzar-les, doncs és més senzill fent servir altres mètodes. Dits mètodes són les *flags* que ens indiquen, si es van actualitzant, aquella informació que sigui necessària.

En la multiplicació i la divisió però, sí s'ha fet ús de les màscares. Compararem el producte amb el resultat d'utilitzar la operació booleana and amb el bit de signe del producte. En funció del resultat, sabrem si s'ha produït overflow. Si tots dos números tenen el mateix signe, però el del resultat és diferent, s'haurà produït desbordament.

### 3. 3 Implementació

```
.bss
    .align 2
    q: .space 4
    r: .space 4

.text
    .align 2
    .arm
    .global add_Q15

add_Q15:
    push {r3, r4, lr}

    mov r4, #0          @; S'estableix que no hi ha overflow en el principi
    adds r3, r0, r1      @; Es sumen els dos números i s'actualitzen els flags (el
                        flag de overflow

    bvs .LifOv1
    b .LelseOv1

.LifOv1:
    mov r4, #1          @; S'estableix que sí hi ha overflow a variable local
.LelseOv1:

    strb r4, [r2]        @; S'estableix si hi ha overflow a la variable passada per
                        referència
    mov r0, r3           @; Es retorna la suma
    pop {r3, r4, pc}

    .global sub_Q15

sub_Q15:
    push {r3, r4, lr}

    mov r4, #0          @; S'estableix que no hi ha overflow en el principi
    subs r3, r0, r1      @; Es resten els dos números i s'actualitzen els flags (el flag de
                        overflow s'actualitza aquí)

    bvs .LifOv2
    b .LelseOv2

.LifOv2:
    mov r4, #1          @; S'estableix que sí hi ha overflow a variable local
.LelseOv2:

    strb r4, [r2]        @; S'estableix si hi ha overflow a la variable passada per
                        referència
    mov r0, r3           @; Es retorna la resta
    pop {r3, r4, pc}
```

```

.global mul_Q15
mul_Q15:
    push {r3-r7, lr}

    mov r5, #0

    smull r3, r4, r0, r1
    mov r3, r3, lsr #15    @; Es prepara el resultat de la multiplicació que s'ha de
                           retornar

    orr r3, r4, lsl #17
    mov r4, r4, asr #15    @; Es desplacen tants bits com part fraccionària
                           @; hi ha cap a l'esquerra per a poder veure el bit de
                           signe

    ldr r6, =0x80000000    @; S'estableix la màscara
    and r7, r4, r6         @; R7 = R4 && MASK_SIGN

.Lif3:
    cmp r3, #0             @; Es mira si el resultat de la multiplicació és negatiu
    bge .Lelse3
    .Lif4:
        cmp r7, r6         @; Es compara el registre amb bit de signe(passat per
                           la màscara), amb la màscara
        beq .Lendif3
        mov r5, #1         @; Si R7 != R6, llavors hi ha hagut overflow
    .Lendif4:
.Lelse3:
    cmp r4, #0             @; Si el registre amb el bit de signe(passat per la
                           màscara)
    beq .Lendif3           @; és diferent de 0, significarà que hi ha hagut
                           overflow
    mov r5, #1
.Lendif3:

    mov r0, r3             @; Es retorna el resultat del producte
    strb r5, [r2]          @; S'estableix la variable passada per referència
                           overflow

    pop {r3-r7, pc}

```

```

.global div_Q15
div_Q15:
    push {r3-r10, lr}
    mov r8, r0
    mov r7, #1
    .Lif5:
        cmp r1, #0
        bne .Lelse5
        mov r11, r2
        mov r0, #0                @; Resultat = 0
        mov r4, #1                @; Overflow = 1
        b .Lendif5

    .Lelse5:
        cmp r1, #0
        bhi .Lcontinue1
        mov r7, #-1                @; Es passa el segon operand a valor absolut
        smull r5, r6, r1, r7
        mov r5, r5, lsr #15
        orr r5, r6, lsl #17
        mov r1, r5

    .Lcontinue1:
        ldr r2, =q
        mov r0, #32768            @; 32768 = 1 * 2^15
        mov r0, r0, lsl #15
        bl div_mod
        ldr r1, [r2]              @; S'obté el valor del quocient

        mov r2, r11
        mov r0, r8                @; Es recupera R0 original (numerador de la
                                divisió)

        bl mul_Q15
        mov r4, r2

        cmp r7, #0                @; Si op2 era negatiu, se li canvia el signe al
                                resultat de la divisió

        bhi .Lendif5
        smull r9, r10, r0, r7
        mov r9, r9, lsr #15
        orr r9, r10, lsl #17
        mov r0, r9

    .Lendif5:
        strb r4, [r2]
        pop {r3-r10, pc}


.end

```

## 3. 4 Joc de proves

En aquest apartat es farà un joc de prova amb el test\_Q15 on posarem a prova les nostres rutines de les operacions aritmètiques. Aquest test utilitzarà diferents valors i els intrudirà en cada una d'aquestes rutines i comprovarà si el valor resultant es correcte

### Resultats de *debug* (test\_Q15)

 Console Window

```
(gdb) disp num_ops_ok  
3: num_ops_ok = 38
```

```
(gdb) disp num_ovf_ok  
4: num_ovf_ok = 38
```


```
(gdb) disp num_tests  
5: num_tests = 38
```

```
(gdb) disp num_errors  
6: num_errors = 0
```

Ara utilitzem el joc de proves que hem creat nosaltres:

```
/* Tests operació SUMA (add) */  
{add, 0, 0, 0, 0},  
{add, MAKE_Q15(2.0),  
  MAKE_Q15(2.5),  
  MAKE_Q15(4.5), 0},  
{add, MAKE_Q15(-2.125),  
  MAKE_Q15(2.5),  
  MAKE_Q15(0.375), 0},  
{add, MAKE_Q15(2.125),  
  MAKE_Q15(-1.5),  
  MAKE_Q15(0.625), 0},  
{add, MAKE_Q15(-5.125),  
  MAKE_Q15(-2.5),  
  MAKE_Q15(-7.625), 0},  
  
/* Tests operació RESTA (sub) */  
{sub, 0, 0, 0, 0},  
{sub, MAKE_Q15(2.125),  
  MAKE_Q15(1.5),  
  MAKE_Q15(0.625), 0},  
{sub, MAKE_Q15(-2.0),  
  MAKE_Q15(7.5),  
  MAKE_Q15(-9.5), 0},  
{sub, MAKE_Q15(4),  
  MAKE_Q15(-6.25),  
  MAKE_Q15(10.25), 0},  
{sub, MAKE_Q15(-2.25),  
  MAKE_Q15(-3.75),  
  MAKE_Q15(1.5), 0},  
  
/* Tests operació MULTIPLICACIÓ (mul) */  
{mul, 0, 0, 0, 0},  
{mul, MAKE_Q15(2.5),  
  MAKE_Q15(3.0),  
  MAKE_Q15(7.5), 0},  
{mul, MAKE_Q15(-5.25),  
  MAKE_Q15(2.5),  
  MAKE_Q15(-13.125), 0},  
  
/* Tests operació DIVISIÓ (div) */  
{div, 0, 0, 0, 1},  
{div, MAKE_Q15(50.0),  
  MAKE_Q15(2.0),  
  MAKE_Q15(25.0), 0},  
{div, MAKE_Q15(12.25),  
  MAKE_Q15(-4.0),  
  MAKE_Q15(-3.0625), 0},  
{div, MAKE_Q15(-9.125),  
  MAKE_Q15(6.0),  
  MAKE_Q15(-1.52), 0},  
{div, MAKE_Q15(-100.0),  
  MAKE_Q15(-4.0),  
  MAKE_Q15(25.0), 0},  
};
```

I aquests són els resultats:

 Console Window

```
(gdb) disp num_ops_ok  
3: num_ops_ok = 17
```

```
(gdb) disp num_ovf_ok  
4: num_ovf_ok = 18
```

```
(gdb) disp num_tests  
5: num_tests = 18
```

```
(gdb) disp num_errors  
6: num_errors = 1
```