Fonaments de Computadors
Pràctica ARM 1
curs 2021-22

Alumnes: Alan Rocha - Florian Serb

Professors: Carlos Soriano

Data lliurament: 22/04/22

Grup Labs: L07

Índex

1. Fase 1

1.1 Especificacions

La tasca de les dues rutines d'aquesta primera fase és fer una conversió de temperatures.

La rutina Celsius2Fahrenheit s'encarrega de convertir una temperatura en graus Celsius a la temperatura equivalent en graus Fahrenheit, utilitzant valors codificats en coma fixa 1:19:12.

```
@; Celsius2Fahrenheit(): converteix una temperatura en graus Celsius a la
@;                       temperatura equivalent en graus Fahrenheit, utilitzant
@;                       valors codificats en Coma Fixa 1:19:12.
@;  Entrada:
@;       input   -> R0
@;  Sortida:
@;       R0      -> output = (input * 9/5) + 32.0;
    .global Celsius2Fahrenheit
Celsius2Fahrenheit:
        push {r1-r3, lr}
            @; prod64 = r0 * Q12(9/5)
            mov r1, #0x1C00         @; guardem el valor del 9/5 en Q12 = 0x1CCD
            add r1, #0x00CD         @; Aquest valor el separem en dos perque el tamany del operador te un tamany de 8 bits
            smull r2, r3, r0, r1    @; Multipliquem i la guardem en els registres r2 i r3

            @; r2 (part baixa), r3 (part alta)

            mov r2, r2, lsr #12     @; Mou el registre cap a la dreta 12bits
            orr r2, r3, lsl #20     @; Passem els bits de menys pes de r3 cap a r2
            mov r3, r3, lsr #12     @; Mou el r3 cap a la dreta

            add r2, #0x20000        @; Al registre r2 li sumem 32

            mov r0, r2              @; El resultat de r2, el moguem a r0

        pop {r1-r3, pc}             @; Recuperem els valor dels registres i actualitzem el pc pero tornar al main.
```

La rutina Fahrenheit2Celsius s'encarrega de convertir una temperatura en graus a Fahrenheit a la temperatura equivalent en graus Celsius, utilitzant valors codificats en coma fixa 1:19:12.

```
@; Fahrenheit2Celsius(): converteix una temperatura en graus Fahrenheit a la
@;                       temperatura equivalent en graus Celsius, utilitzant
@;                       valors codificats en Coma Fixa 1:19:12.
@;  Entrada:
@;       input   -> R0
@;  Sortida:
@;       R0      -> output = (input - 32.0) * 5/9;
    .global Fahrenheit2Celsius
Fahrenheit2Celsius:
        push {r1-r3, lr}            @; Salvem les dades dels registres

        sub r0, #0x20000            @; Li restem 32 al r0
        mov r1, #0x1C00             @; Guardem la primera part de 5/9 en el registre
        add r1, #0x00CD             @; Guardem la segona part de 5/9 en el registre
        smull r2, r3, r0, r1        @; Multipliquem amb signe i guardem el resultat en els registres r2 i r3

        @; r2 (part baixa), r3 (part alta)
        mov r2, r2, lsr #12         @; Mou a la dreta el registre 12 bits
        orr r2, r3, lsl #20         @; Passem els bits de menys pes de r3 a r2
        mov r3, r3, asr #12         @; Mogem a la dreta el registre

        mov r0, r2                  @; El resultat de r2, el passem a r0

        pop {r1-r3, pc}             @; Recuperem els valors del registres i actualitzem el pc pero tornar al main.
```

Les dues rutines comparteixen la mateixa entrada i sortida que seria r0, i la temperatura que s'ha de convertir també arriba per r0 i la conversió també.

A continuació tenim el fitxer demo.s en el qual definim i inicialitzem els paràmetres amb les temperatures indicades.

```
.data
    .align 2
    temp1C: .word 0x0002335C        @; temp1C = 35.21 ºC
    temp2F: .word 0xFFFE8400        @; temp2F = -23.75 ºF

.bss
    .align 2
    temp1F: .space 4                @; expected conversion:  95.379638671875 ºF
    temp2C: .space 4                @; expected conversion: -30.978271484375 ºC
```

Seguidament, afegim la funció "main" i la fem global per a poder cridar-la des de qualsevol fitxer, després de inicialitzar el main, passem les temperatures definides a la part superior".data" als paràmetres a utilitzar, passem les temperatures a la funció "Celsius2Fahrenheit".

```
.text
    .align 2
    .arm
    .global main
main:
    push {lr}
    @; temp1F = Celsius2Fahrenheit(temp1C);
    ldr r1, =temp1C             @;Guardem en r1 la direccio de temp1C
    ldr r0, [r1]                @;Tenim el contingut de la direccio r1
    bl Celsius2Fahrenheit       @;Fem una crida a la rutina que modificara r0
    ldr r2, =temp1F             @;Guardem en r2 la direccio de temp1F
    str r0, [r2]                @;Ara passem el contingut que tenim a r0 per a modificar Celsius2Fahrenheit
```

A continuació farem el mateix, però amb "Fahrenheit2Celsius", passem el contingut dels paràmetres inicialitzats al principi "r1, =temp2F/ r2, =temp2C", d'aquesta manera passem la temperatura a les funcions de "Fahrenheit2Celsius.s".

```
@; temp2C = Fahrenheit2Celsius(temp2F);
ldr r1, =temp2F             @;Guardem en r1 la direccio de temp2F
ldr r0, [r1]                @;Tenim el contingut de la direccio r1
bl Fahrenheit2Celsius       @;Fem una crida a la rutina que modificara r0
ldr r2, =temp2C             @;Guardem en r2 la direccio de temp2C
str r0, [r2]                @;Ara passem el contingut que tenim a r0 per a modificar Fahrenheit2Celsius
```

Finalment, copiem el contingut de la memòria sobre la llista ordenada de registre i alliberem espai en memòria. També hem d'indicar el final del fitxer.

```
    mov r0, #0

    pop {pc}

.end
```

1.2 Disseny

Calcul de valors a Q12

El primer que vam necessitar fer per a aquesta primera fase, va ser passar els valors 9/5 i 5/9 a hexadecimal en Q12.

- Per a passar un nombre a hexadecimal Q12 as que multiplicar el nombre per 2^12, al nostre cas seràn 1.8 i 0.55.

  Com el resultat de les nostres operacions donaven 7372,8 i 2252,8 i per a passar a hexa no podem tenir decimals, vam arrodonir els valors a 7373 i 2253 que passats a hexadecimal eran 1CCD i 8E4 respectivament.

Explicació de CelsiusFahrenheit.s

El primer problema va arribar quan vam voler implementar els nostres valors en hexadecimal i ens vam adonar que havíem d'implementar els nostres valors en dues instruccions.

```
@; prod64 = r0 * Q12(9/5)
mov r1, #0x1C00          @; guardem el valor del 9/5 en Q12 = 0x1CCD
add r1, #0x00CD          @; Aquest valor el separem en dos perque el tamany del operador te un tamany de 8 bits
```

Això era degut a la diferència de 8 bits que hi havia entre l'1 de major pes i l'1 de menor pes, a més que en ser un operant immediat no el podíem ficar a una sola instrucció.

Per al segon valor va ser suficient amb carregar al registre amb la instrucció ldr.

```
ldr r1, =0x8E4              @; Igualem a r1 el valor hexadecimal "0x8E4"
```

Per a la multiplicació vam realitzar-la amb una instrucció **s**mull **(extensió de signe**), ja que els registres són de 32 bits i el producte de 64. D'aquesta manera separem el producte en dos registres; un que guarda la part alta i altre que guarda la part baixa. Després del producte desplacem el registre 12 bits a la dreta per a mantenir el registre en Q12.

```
smull r2, r3, r0, r1    @; Multipliquem i la guardem en els registres r2 i r3
@;s = signe
@; r2 (part baixa), r3 (part alta)
mov r2, r2, lsr #12     @; Mou el registre a la dreta 12bits
orr r2, r3, lsl #20     @; Passem els bits de menys pes de r3 cap a r2
mov r3, r3, lsr #12     @; Mou el r3 cap a la dreta
```

Explicació representació inicial demo.s

Les d'edicions que hem pres per al fitxer "demo.s", són utilitzar els paràmetres predefinits i inicialitzats amb les temperatures".data", i igualar-ho a direcció més fàcil d'usar "r1/r2..", seguidament el que faríem seria carregar a una direcció el contingut que tenim a r1 d'aquesta manera després podrem fer una crida a la rutina "Celsius2Fahrenheit" i podrem utilitzar el contingut de la condició "temp1C". Seguidament, faríem el mateix, però en aquest cas afegiríem una altra direcció en la qual la igualaríem a "temp1F".

```
.data
    .align 2
temp1C: .word 0x0002335C
temp2F: .word 0xFFFE8400
```

A continuació hem decidit seguir la mateixa estructura per a passar de "Fahrenheit" a "Celsius", utilitzarem una condició en la qual la igualarem al paràmetre predefinit i inicialitzat amb la temperatura en hexadecimal "=temp2F", seguidament passarem aquest paràmetre a la funció "Fahrenheit2Celsius" d'aquesta manera li passarem les dades definides, les dades de "temps1C" i "temp2F".

Per acabar, faríem un pop per a alliberar espai de memòria i el punt més important afegiríem un .end per a indicar el final del "main:".

```
    pop {pc}

.end
```

1.3 Implementació

Algorismes CelsiusFahrenheit.s - Demo.s

El codi de CelsiusFahrenheit.s:

```
@; Celsius2Fahrenheit(): converteix una temperatura en graus Celsius a la
@;                       temperatura equivalent en graus Fahrenheit, utilitzant
@;                       valors codificats en Coma Fixa 1:19:12.
@;  Entrada:
@;      input   -> R0
@;  Sortida:
@;      R0      -> output = (input * 9/5) + 32.0;
    .global Celsius2Fahrenheit
Celsius2Fahrenheit:
        push {r1-r3, lr}
            @; prod64 = r0 * Q12(9/5)
            mov r1, #0x1C00         @; guardem el valor del 9/5 en Q12 = 0x1CCD
            add r1, #0x00CD         @; Aquest valor el separem en dos perque el tamany del operador te un tamany de 8 bits
            smull r2, r3, r0, r1    @; Multipliquem i la guardem en els registres r2 i r3
            @; r2 (part baixa), r3 (part alta)

            mov r2, r2, lsr #12     @; Mou el registre cap a la dreta 12bits
            orr r2, r3, lsl #20     @; Passem els bits de menys pes de r3 cap a r2
            mov r3, r3, lsr #12     @; Mou el r3 cap a la dreta

            add r2, #0x20000        @; Al registre r2 li sumem 32

            mov r0, r2              @; El resultat de r2, el moguem a r0

        pop {r1-r3, pc}             @; Recuperem els valor dels registres i actualitzem el pc pero tornar al main.

@; Fahrenheit2Celsius(): converteix una temperatura en graus Fahrenheit a la
@;                       temperatura equivalent en graus Celsius, utilitzant
@;                       valors codificats en Coma Fixa 1:19:12.
@;  Entrada:
@;      input   -> R0
@;  Sortida:
@;      R0      -> output = (input - 32.0) * 5/9;
    .global Fahrenheit2Celsius
Fahrenheit2Celsius:
        push {r1-r3, lr}            @; Salvem les dades dels registres

        sub r0, #0x20000            @; Li restem 32 al r0
        mov r1, #0x1C00             @; Guardem la primera part de 5/9 en el registre
        add r1, #0x00CD             @; Guardem la segona part de 5/9 en el registre
        smull r2, r3, r0, r1        @; Multipliquem amb signe i guardem el resultat en els registres r2 i r3

        @; r2 (part baixa), r3 (part alta)
        mov r2, r2, lsr #12         @; Mou a la dreta el registre 12 bits
        orr r2, r3, lsl #20         @; Passem els bits de menys pes de r3 a r2
        mov r3, r3, asr #12         @; Mogem a la dreta el registre

        mov r0, r2                  @; El resultat de r2, el passem a r0

        pop {r1-r3, pc}             @; Recuperem els valors del registres i actualitzem el pc pero tornar al main.
```

El codi de Demo.s:

```
.data
    .align 2
    temp1C: .word 0x0002335C        @; temp1C = 35.21 ºC
    temp2F: .word 0xFFFE8400        @; temp2F = -23.75 ºF

.bss
    .align 2
    temp1F: .space 4                @; expected conversion:  95.379638671875 ºF
    temp2C: .space 4                @; expected conversion: -30.978271484375 ºC

.text
    .align 2
    .arm
    .global main
main:
    push {lr}
    @; temp1F = Celsius2Fahrenheit(temp1C);
    ldr r1, =temp1C             @;Guardem en r1 la direccio de temp1C
    ldr r0, [r1]                @;Carreguem a r0 el contingut de la direccio r1
    bl Celsius2Fahrenheit       @;Fem una crida a la rutina que modificara r0
    ldr r2, =temp1F             @;Guardem en r2 la direccio de temp1F
    str r0, [r2]                @;Ara passem el contingut que tenim a r0 per a modificar Celsius2Fahrenheit
    @; temp2C = Fahrenheit2Celsius(temp2F);
    ldr r1, =temp2F             @;Guardem en r1 la direccio de temp2F
    ldr r0, [r1]                @;Carreguem a r0 el contingut de la direccio r1
    bl Fahrenheit2Celsius       @;Fem una crida a la rutina que modificara r0
    ldr r2, =temp2C             @;Guardem en r2 la direccio de temp2C
    str r0, [r2]                @;Ara passem el contingut que tenim a r0 per a modificar Fahrenheit2Celsius

@; TESTING POINT: check the results
@;  (gdb) p /x temp1F       -> 0x0005F613
@;  (gdb) p /x temp2C       -> 0xFFFE1059
@; BREAKPOINT
    mov r0, #0                  @; return(0)

    pop {pc}

.end
```

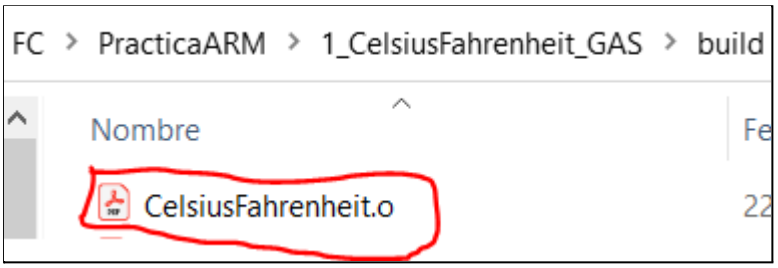
1.4 Joc de proves "ampliat"

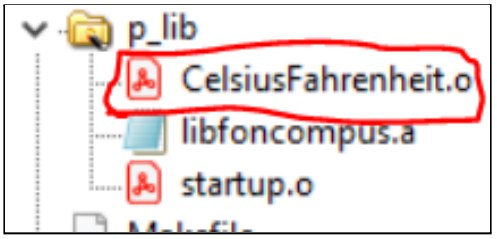
PROVA	DESCRIPCIÓ	RESULTAT ESPERAT	RESULTAT OBTINGUT	OK?
1	p /x temp1F	0x0005F613	0x5f613	CORRECTE
2	p /x temp2C	0xFFFE1059	0xfffe1059	CORRECTE

2. Fase 2

2.1 Especificacions

El primer que hem de fer a treballar correctament en la Fase 2, és afegir el fitxer "**CelsiusFahrenheit**.o", aquest fitxer l'hem de copiar de la fase 1, hem d'anar a la fase 1 a la carpeta: (1_CelsiusFahrenheit_GAS/build) i copiar el fitxer "CelsiusFahrenheit.o" i apegar-lo a dins de (2_GeoTemp/p_lib).





El segon que haurem de fer en la Fase2, és importar i modificar el fitxer "**MAKEFILE**", ja que sense aquest fitxer no podrem executar i fer corre el nostre programà.

A continuació hem modificat totes les instruccions a partir del comentari "make commands", perquè el que hem de fer s'importa tots els fitxers .c"codi d'alt nivell/C" i convertir-los en .o "codi comprimit" d'aquesta manera després podrem cridar tots els .o i crear els executables .elf i podrem fer corre el programa.

1. Creem el "data.o", per a crear-lo hem d'anar al fitxer data.c "source/data.c" i fer include de tots els fitxers que tenim com a "include".

Seguidament fem "CCFLAGS -c" (-c: indicant que és codi d'alt nivell) per a crear el fitxer "-o"(-o:indicant que volem que sigui fitxer de CODI COMPRES) "data.o" a partir del "data.c".

```
# fem els includes de tots els fitxer de dins de data.c per a poder generar el .o
build/data.o : include/Q12.h include/geotemp.h include/data.h
    arm-none-eabi-gcc $(CCFLAGS) -c source/data.c -o build/data.o
```

2. Creem el "avgmaxmintemp.o", per a crear-lo hem d'afegir la font de "avgmaxmintemp.s" i l'include que tenim dins d'aquest fitxer.

A continuació fem ASFLAGS (perquè AS, perquè volem passar un fitxer .s ENSAMBLADOR GAS a .o CODI COMPRES), fem ASFLAGS el fitxer avgmaxmintemp.s -o (indicant que volem un fitxer .o) build/avgmaxmintemp.o (indicant en quin directori volem que es creï).

```
# en aquest cas tenim .s "ENSAMBLADOR GAS" i volem passar aquest fitxer a .o per a que estigui mes comprimit
# per a passar de .s fem = arm-none-eabi-as ASFLAGS "fitxer .s" -o "fitxer .o" per a crar el fitxer .o
build/avgmaxmintemp.o : source/avgmaxmintemp.s include/avgmaxmintemp.i
    arm-none-eabi-as $(ASFLAGS) source/avgmaxmintemp.s -o build/avgmaxmintemp.o
```

3. Creem el "geotem.o", per a poder crear-lo hem d'afegir el fitxer geotemp.c "source/geotemp.c" i tots els includes que tenim dins d'aquest fitxer.

Seguidament fem CCFLAGS (perquè volem passar d'un fitxer .c a .o), -c (codi d'alt nivell) source/geotemp.c(ruta i nom del fitxer a convertir) -o build/geotemp.o(directori al qual volem crear el fitxer + nom del fitxer acabat amb .o).

```
# fem els includes de tots els fitxer de dins de data.c per a poder generar el .o
build/geotemp.o : source/geotemp.c include/Q12.h include/divmod.h include/avgmaxmintemp.h include/geotemp.h include/data.h include/CelsiusFahrenheit.h
    arm-none-eabi-gcc $(CCFLAGS) -c source/geotemp.c -o build/geotemp.o
```

4. Creem el "test_geotemp.o"(perquè test_geotemp, perquè hem de tindre dos fitxers.elf= geotemp.elf + test_geotemp.elf), per a poder crear-lo hem d'anar al fitxer test_geotemp.c "source/test_geotemp.c" i afegir tots els includes que tenim dins d'aquest fitxer.

Ara fem CCFLAGS (perquè volem passar de .c a .o) -c (codi d'alt nivell) tests/test_geotemp.c (ruta del fitxer) -o (codi comprimit) build/test_geotemp.o (ruta a desar el fitxer).

```
# fem els includes que tenim dins de "test_geotemp.c"
# seguidament convertim .c a .o "codi comprimit"
build/test_geotemp.o : include/avgmaxmintemp.h include/Q12.h
    arm-none-eabi-gcc $(CCFLAGS) -c tests/test_geotemp.c -o build/test_geotemp.o
```

5. Per a generar el fitxer "geotemp.elf", hem de cridar els "build" que hem afegit anteriorment per a poder generar-se, en aquest cas cridem els fitxers (build/avgmaxmintemp.o build/data.o build/geotemp.o).

A continuació fem LDFLAGS per a poder cridar tots els "build/avgmaxmintemp.o build/data.o build/geotemp.o" i també poder cridar tots els .o referits a aquests fitxers, això ho fem gràcies a "\".

Per a generar el fitxer "geotem.elf", hem d'afegir tots els .o de dins del fitxer "p_lib" i el fitxer libfoncompus.a el convertit a -o (fitxer comprimit).

```
# cridem tots els fitxer .o per a poder crear l'executable "geotem.elf"
# LDFLAGS es la regla principal para poder crear el .elf
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
```

6. Per a generar el fitxer "test_geotemp.elf", hem de cridar els "build" que hem generat anteriorment, en aquest cas cridem (build/test_geotemp.o build/avgmaxmintemp.o).

A continuació fem LDFLAGS per a poder cridar els "build/test_geotemp.o build/avgmaxmintemp.o".

També afegim "\" per a cridar tots els .o vinculats a aquests.

Per a generar el fitxer "test_geotem.elf", hem d'afegir tots els .o de dins del fitxer "p_lib" i el fitxer libfoncompus.a el convertit a -o (fitxer comprimit).

```
# aqui cridem tots els .o per a poder crear l'executable test_geotemp.elf, utilitzem LDFLAGS
test_geotemp.elf : build/test_geotemp.o build/avgmaxmintemp.o
    arm-none-eabi-ld $(LDFLAGS) build/test_geotemp.o build/avgmaxmintemp.o \
                    p_lib/startup.o p_lib/CelsiusFahrenheit.o p_lib/libfoncompus.a -o test_geotemp.elf
```

Explicació de avgmaxmintemp.s

Quant al fitxer avgmaxmintemp.s tenim dues rutines: avgmaxmin_city i avgmaxmin_month. Aquestes dues rutines s'encarreguen de calcular la temperatura màxima, mitjana i mínima a una ciutat durant els dotze mesos de l'any i la temperatura d'un mes de diverses ciutats.

El paràmetre d'entrada és r0, que fa la funció de taula que conté les temperatures de les ciutats durant els dotze mesos de l'any:

- A r1 tenim el nombre de files de la taula que serien les ciutats.
- A r2 per a la rutina avgmaxmin_city: passem la fila o la ciutat que volem calcular.
- A r2 per a la rutina avgmaxmin_month: passem la columna del mes que volem calcular.
- A r3 passem per referència la direcció del struct per on es tornaran els màxims i mínims.

L'output o sortida serà per r0 que retornarà la mitjana de les temperatures.

```
@; Funcio Calcular mitjana/maxima/minima ciutats
@; r0 = ttemp[][12]
@; r1 = nrows
@; r2 = id_city
@; r3 = *mmres
@; Output = r0 el cocient de la divisio "variable avg" i *mmres "temp min y max"
@; r4 = avg
@; r5 = max
@; r6 = min
@; r7 = idmin = 0
@; r8 = idmax = 0
@; r9 = i
@; r10 = tvar

    .global avgmaxmin_city      @; Definim la funcio com a global per a poder ser cridada desde altres funcions
avgmaxmin_city:                 @; Inicialitzem la funcio "avgmaxmin_city"
```

```
@; Funcio calcular mijana, maxim i minim de un mes de varies ciutats
@; r0 -> ttemp[][12]
@; r1 -> nrows
@; r2 -> id_month
@; r3 -> *mmres
@; Output -> r0 el cocient de la divisio "variable avg" i *mmres "temp min y max")
@; r4->avg
@; r5->max
@; r6->min
@; r7->idmin = 0
@; r8->idmax = 0
@; r9->i
@; r10 -> tvar

    .global avgmaxmin_month     @;Definim la funcio com a global per a poder ser cridada desde altres funcions
avgmaxmin_month:                @; Inicialitzem la funcio "avgmaxmin_month"
```

2.2 Disseny

CelsiusFahrenheit.o

El primer que hem decidit d'implementar és el fitxer CelsiusFahrenheit.o, ja que aquest sols l'hem de copiar de la fase1 (1_CelsiusFahrenheit_GAS/build/CelsiusFahrenheit.o) i pegar-lo dins de (2_GeoTemp/p_lib).

Makefile

Seguidament, hem decidit copiat el fitxer Makefile de la Fase1 i li hem pegat dins de la Fase2, i a partir d'aquí començar a afegir tots els fitxers necessaris per a poder generar tots els fitxers .o necessaris per a poder generar els fitxers .elf.

1. El primer fitxar a implementar sirà:

En el qual afegirem tots els includes respectius als que tenim dins de "data.c" i convertirem data.c a data.o, gràcies a CCFLAGS.

```
build/data.o
```

2. Seguidament afegirem:

Tots els includes respectius als que tenim dins de “avgmaxmintemp.s” i convertirem avgmaxmintemp.s a avgmaxmintemp.o, gràcies a ASFLAGS.

```
build/avgmaxmintemp.o
```

3. També hem d'afegir:

Tots els includes respectius als que tenim dins de "geotemp.c" i convertirem geotemp.c a geotemp.o, gràcies a CCFLAGS.

```
build/geotemp.o
```

4. A continuació afegim:

Tots els includes respectius als que tenim dins de “test_geotemp.c” i convertirem test_geotemp.c a test_geotemp.o, gràcies a CCFLAGS.

```
build/test_geotemp.o
```

5. Seguidament fem:

Cridarem tots els build necessaris per a generar el fitxer .elf, i convertirem libfoncompus.a a geotemp.elf, gràcies a LDFLAGS.

```
geotemp.elf
```

6. Per últim fem:

Cridarem tots els build necessaris per a generar el fitxer .elf, i convertirem libfoncompus.a a test_geotemp.elf, gràcies a LDFLAGS.

test_geotemp.elf

Avgmaxmintemp.s

Per a la solució d'aquest codi, vam decidir controlar el bucle per mitjà de salts a les etiquetes en codi assemblador.
A la rutina de avgmaxmin_city vam afegir una comparació al final del bucle, ja que en aquesta rutina el bucle s'executa almenys una vegada.

```
@; BUCLE
    mov r9, #1                  @; Inicialtzem la i a 1 "r9 = 1"
.Lfor:                          @; Inicialitzem el ".Lfor"
    add r12, r9, r11            @; A r12 guardem el recorregut de la "i" fins a 12"maxim de messos"
    ldr r10, [r0, r12, lsl #2]  @; Carreguem el registre a tvar = taula tempereatura"ttemp[id_city][i]"
    add r4, r10                 @; Afegim a avg += tvar
    @; IF per trobar el MAX
    cmp r10, r5                 @; Comparem si r10 es mes gran que r5 "tvar > max" si es compleix entra a la condicio
    ble .Lifmin                 @; Si tvar es mes petit < que max saltara al IF MIN
    mov r5, r10                 @; Igualem el contingut de max = tvar "ja que em trobat el valor maxim"
    mov r8, r9                  @; Igualem el contingut de idmax = i
    @; IF per trobar el MIN
.Lifmin:                        @; Inicialitzem el ".Lifmin"
    cmp r10, r6                 @; Comparem si tvar < min si es compleix entra a la condicio
    bge .Lfif                   @; Si no es compleix salta al .Lfif
    mov r6, r10                 @; Igualem el min = tvar "ja que em trobat el valor minim"
    mov r7, r9                  @; Igualem el idmin = i
.Lfif:                          @; Inicialitzem el ".Lfif"
    add r9, #1                  @; Per acabar amb el for, em d'afegir el i++
    cmp r9, #12                 @; Tambe em d'afegir la comparacio, si la i < 12 entra a for
    blo .Lfor                   @; Si no es compleix la condicio salta al if i < 12 fins trobar el valor mes petit que 12
@; FINAL del BUCLE
```

Per altra banda, després del bucle, ens de què el resultat estigui en positiu, però això fem un valor absolut i això poder utilitzar el div_mod.

En aquesta part en lloc de fer servir salts per controlar el condicional, hem utilitzat instruccions predicades per optimitzar una miqueta mes.

```
@; r0->ttemp[][12], r1->den=12, r2->&cociente, r3-> &resto, r4->avg, r11-> avgNeg
mov r0, r4                  @; Movem avg a r0
mov r11, #0                 @; Inicialitzem avg a Negatiu
cmp r0, #0                  @; Comparem el contingut de avg si es mes petit que 0 "if (avg < 0)"
movlt r11, #1               @; Si es compleix la condicio avgNeg verifiquem que es negatiu
@; lt = menos que movlt = movemos si es menos que 1
rsblt r0, #0                @; Una vegada verifiquem que es NEGATIU ho passem a POSITIU
@; rsblt fa una resta si el valor de r0 es MENOR que 0
```

```
mov r1, #12                 @; Afegim a r1 el denominador 12"messos"
mov r4, r3                  @; Passem avg a r3 "residu"

ldr r2, =data_c             @; Carreguem a r2 el registre de data_c
ldr r3, =data_r             @; Carreguem a r3 el registre de data_r

bl div_mod                  @; Cridem la funcio per a fer la divisio entre 12
@; Sols ens importa el COCIENT
ldr r1, [r2]                @; Carreguem a r1 el registre de r2"data_c"

cmp r11, #1                 @; Comparem si avgNeg si es NEGATIU
rsbeq r1, #0                @; Si es NEGATIU ho passem a POSITIU
```

Finalment, hem decidit transferir les dades al struct per memoria.

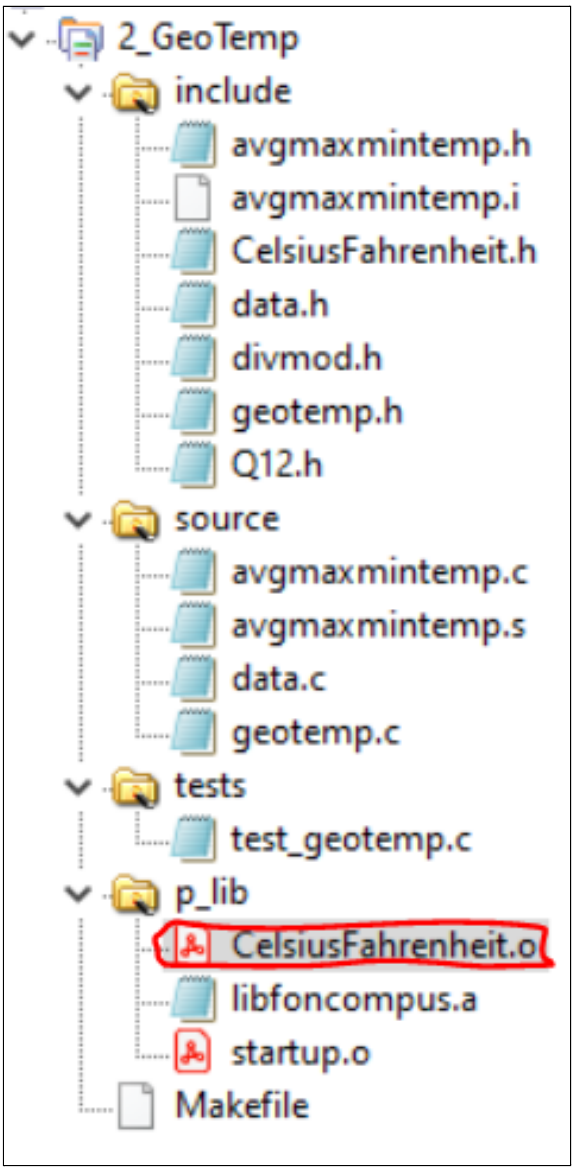
```
@; transferir datos al struct
str r6, [r4, #MM_TMINC]     @; Guardem les dades en memoria, en aquest cas temperatura minima en MM_TMINC "mmres->tmin_C = min;"
str r5, [r4, #MM_TMAXC]     @; Guardem les dades en memoria, en aquest cas temperatura maxima en MM_TMAXC "mmres->tmax_C = max;"
mov r0, r6                  @; Igualem el contingut de "ttemp a min"
bl Celsius2Fahrenheit       @; Invoquem/Cridem la funcio "Celsius2Fahrenheit(min);"
str r0, [r4, #MM_TMINF]     @; Guardem les dades en memoria, en aquest cas temperatura minima Fahrenheit en MM_TMINF
                            @; "mmres->tmin_F = Celsius2Fahrenheit(min)"
mov r0, r5                  @; Igualem el contingut de "ttemp a max"
bl Celsius2Fahrenheit       @; Invoquem/Cridem la funcio "Celsius2Fahrenheit(max);"
str r0, [r4, #MM_TMAXF]     @; Guardem les dades en memoria, en aquest cas temperatura maxima Fahrenheit en MM_TMAXF
                            @; "mmres->tmax_F = Celsius2Fahrenheit(max);"
strh r7, [r4, #MM_IDMIN]    @; Guardem les dades en una memoria de 16 bits"short", la ID de temperatura minima MM_IDMIN
                            @; "mmres->id_min = idmin (2bytes);"
strh r8, [r4, #MM_IDMAX]    @; Guardem les dades en una memoria de 16 bits"short", la ID de temperatura maxima MM_IDMAX
                            @; "mmres->id_max = idmax (2bytes);"

mov r0, r1                  @; Igualem el contingut de ttemp a files "return(cociente=avg);"
pop {r1-r12, pc}            @; Recuperem les dades i tornem a la funcio que crida aquesta rutina
```

Vam fer un intent de fer-lo per pila, però no ens va sortir bé tot i que de totes maneres en aquesta fase no hem aconseguit tots els resultats que esperàvem.

2.3 Implementació

Importar CelsiusFahrenheit.o de la Fase1



Makefile

```
#-------------------------------------------------------------------------------
#   Programador/a 1: florian.serb@estudiants.urv.cat
#   Programador/a 2: alan.rocha@estudiants.urv.cat
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
# Variables para la generacion de codigo
#-------------------------------------------------------------------------------
ARCH    := -march=armv5te -mlittle-endian
# FLAGS = comprovar si tenim overflow o no
ASFLAGS := $(ARCH) -g
# ASFLAGS = s'utilitza per a passar els fitxers .s "ENSAMBLADOR GAS" a .o "CODI COMPRES"
CCFLAGS := -Wall -gdwarf-3 -O0 $(ARCH) -I./include
# CCFLAGS = s'utilitza per a passar els fitxers .c "CODI C/ALT NIVELL" a .o "CODI COMPRES"
LDFLAGS := -z max-page-size=0x8000 -Lp_lib
# LDFLAGS = s'utilitza per a cridar tots els fitxers amb l'ajuda de \ i crear el fitxer .elf
```

```
# fem els includes de tots els fitxer de dins de data.c per a poder generar el .o
build/data.o : include/Q12.h include/geotemp.h include/data.h
    arm-none-eabi-gcc $(CCFLAGS) -c source/data.c -o build/data.o

# en aquest cas tenim .s "ENSAMBLADOR GAS" i volem passar aquest fitxer a .o per a que estigui mes comprimit
# per a passar de .s fem = arm-none-eabi-as ASFLAGS "fitxer .s" -o "fitxer .o" per a crar el fitxer .o
build/avgmaxmintemp.o : source/avgmaxmintemp.s include/avgmaxmintemp.i
    arm-none-eabi-as $(ASFLAGS) source/avgmaxmintemp.s -o build/avgmaxmintemp.o

# fem els includes de tots els fitxer de dins de data.c per a poder generar el .o
build/geotemp.o : source/geotemp.c include/Q12.h include/divmod.h include/avgmaxmintemp.h include/geotemp.h include/data.h include/CelsiusFahrenheit.h
    arm-none-eabi-gcc $(CCFLAGS) -c source/geotemp.c -o build/geotemp.o

# fem els includes que tenim dins de "test_geotemp.c"
# seguidament convertim .c a .o "codi comprimit"
build/test_geotemp.o : include/avgmaxmintemp.h include/Q12.h
    arm-none-eabi-gcc $(CCFLAGS) -c tests/test_geotemp.c -o build/test_geotemp.o

# cridem tots els fitxer .o per a poder crear l'executable "geotem.elf"
# LDFLAGS es la regla principal para poder crear el .elf
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

# aqui cridem tots els .o per a poder crear l'executable test_geotemp.elf, utilitzem LDFLAGS
test_geotemp.elf : build/test_geotemp.o build/avgmaxmintemp.o
    arm-none-eabi-ld $(LDFLAGS) build/test_geotemp.o build/avgmaxmintemp.o \
        p_lib/startup.o p_lib/CelsiusFahrenheit.o p_lib/libfoncompus.a -o test_geotemp.elf
```

```
#-------------------------------------------------------------------------------
# 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 &
```

Algorisme CelsiusFahrenheit.s

```
@; Funcio Calcular mitjana/maxima/minima ciutats
@; r0 = ttemp[][12]
@; r1 = nrows
@; r2 = id_city
@; r3 = *mmres
@; Output = r0 el cocient de la divisio "variable avg" i *mmres "temp min y max"
@; r4 = avg
@; r5 = max
@; r6 = min
@; r7 = idmin = 0
@; r8 = idmax = 0
@; r9 = i
@; r10 = tvar

    .global avgmaxmin_city      @; Definim la funcio com a global per a poder ser cridada desde altres funcions
avgmaxmin_city:                 @; Inicialitzem la funcio "avgmaxmin_city"
    push {r1-r12, lr}           @; Guardem les dades

    mov r7, #0                  @; Inicialitzem les variables locals a 0
    mov r8, #0                  @; Inicialitzem les variables locals a 0

    mov r11, #12                @; Afegim a r11 -> 12 messos = "12 columnes"
    mul r11, r2, r11            @; Ara fem una multiplicacio a r11 = multipliquem  id_city * 12 messos
    ldr r4, [r0, r11, lsl #2]   @; Carreguem a r4 el calcul de: r0 + r11 * 4 (dades tipus word = 32bits)
    mov r5, r4                  @; Igualem el contingut de r4 a r5 "avg = max"
    mov r6, r4                  @; Igualem el contingut de r4 a r6 "min = avg"

@; BUCLE
    mov r9, #1                  @; Inicialtzem la i a 1 "r9 = 1"
.Lfor:                          @; Inicialitzem el ".Lfor"
    add r12, r9, r11            @; A r12 guardem el recorregut de la "i" fins a 12"maxim de messos"
    ldr r10, [r0, r12, lsl #2]  @; Carreguem el registre a tvar = taula tempereatura"ttemp[id_city][i]"
    add r4, r10                 @; Afegim a avg += tvar
    @; IF per trobar el MAX
    cmp r10, r5                 @; Comparem si r10 es mes gran que r5 "tvar > max" si es compleix entra a la condicio
    ble .Lifmin                 @; Si tvar es mes petit < que max saltara al IF MIN
    mov r5, r10                 @; Igualem el contingut de max = tvar "ja que em trobat el valor maxim"
    mov r8, r9                  @; Igualem el contingut de idmax = i
    @; IF per trobar el MIN
.Lifmin:                        @; Inicialitzem el ".Lifmin"
    cmp r10, r6                 @; Comparem si tvar < min si es compleix entra a la condicio
    bge .Lfif                   @; Si no es compleix salta al .Lfif
    mov r6, r10                 @; Igualem el min = tvar "ja que em trobat el valor minim"
    mov r7, r9                  @; Igualem el idmin = i
.Lfif:                          @; Inicialitzem el ".Lfif"
    add r9, #1                  @; Per acabar amb el for, em d'afegir el i++
    cmp r9, #12                 @; Tambe em d'afegir la comparacio, si la i < 12 entra a for
    blo .Lfor                   @; Si no es compleix la condicio salta al if i < 12 fins trobar el valor mes petit que 12
@; FINAL del BUCLE
```

```
@; /12
    @; r0->ttemp[][12], r1->den=12, r2->&cociente, r3-> &resto, r4->avg, r11-> avgNeg
    mov r0, r4                  @; Movem avg a r0
    mov r11, #0                 @; Inicialitzem avg a Negatiu
    cmp r0, #0                  @; Comparem el contingut de avg si es mes petit que 0 "if (avg < 0)"
    movlt r11, #1               @; Si es compleix la condicio avgNeg verifiquem que es negatiu
    @; lt = menos que movlt = movemos si es menos que 1
    rsblt r0, #0                @; Una vegada verifiquem que es NEGATIU ho passem a POSITIU
    @; rsblt fa una resta si el valor de r0 es MENOR que 0
    mov r1, #12                 @; Afegim a r1 el denominador 12"messos"
    mov r4, r3                  @; Passem avg a r3 "residu"

    ldr r2, =data_c             @; Carreguem a r2 el registre de data_c
    ldr r3, =data_r             @; Carreguem a r3 el registre de data_r

    bl div_mod                  @; Cridem la funcio per a fer la divisio entre 12
    @; Sols ens importa el COCIENT
    ldr r1, [r2]                @; Carreguem a r1 el registre de r2"data_c"

    cmp r11, #1                 @; Comparem si avgNeg si es NEGATIU
    rsbeq r1, #0                @; Si es NEGATIU ho passem a POSITIU
@; FINAL /12

    @; transferir datos al struct
    str r6, [r4, #MM_TMINC]     @; Guardem les dades en memoria, en aquest cas temperatura minima en MM_TMINC "mmres->tmin_C = min;"
    str r5, [r4, #MM_TMAXC]     @; Guardem les dades en memoria, en aquest cas temperatura maxima en MM_TMAXC "mmres->tmax_C = max;"
    mov r0, r6                  @; Igualem el contingut de "ttemp a min"
    bl Celsius2Fahrenheit       @; Invoquem/Cridem la funcio "Celsius2Fahrenheit(min);"
    str r0, [r4, #MM_TMINF]     @; Guardem les dades en memoria, en aquest cas temperatura minima Fahrenheit en MM_TMINF
                                @; "mmres->tmin_F = Celsius2Fahrenheit(min)"
    mov r0, r5                  @; Igualem el contingut de "ttemp a max"
    bl Celsius2Fahrenheit       @; Invoquem/Cridem la funcio "Celsius2Fahrenheit(max);"
    str r0, [r4, #MM_TMAXF]     @; Guardem les dades en memoria, en aquest cas temperatura maxima Fahrenheit en MM_TMAXF
                                @; "mmres->tmax_F = Celsius2Fahrenheit(max);"
    strh r7, [r4, #MM_IDMIN]    @; Guardem les dades en una memoria de 16 bits"short", la ID de temperatura minima MM_IDMIN
                                @; "mmres->id_min = idmin (2bytes);"
    strh r8, [r4, #MM_IDMAX]    @; Guardem les dades en una memoria de 16 bits"short", la ID de temperatura maxima MM_IDMAX
                                @; "mmres->id_max = idmax (2bytes);"

    mov r0, r1                  @; Igualem el contingut de ttemp a files "return(cociente=avg);"
    pop {r1-r12, pc}            @; Recuperem les dades i tornem a la funcio que crida aquesta rutina
```

```
@; Funcio calcular mijana, maxim i minim de un mes de varies ciutats
@; r0 -> ttemp[][12]
@; r1 -> nrows
@; r2 -> id_month
@; r3 -> *mmres
@; Output -> r0 el cocient de la divisio "variable avg" i *mmres "temp min y max")
@; r4->avg
@; r5->max
@; r6->min
@; r7->idmin = 0
@; r8->idmax = 0
@; r9->i
@; r10 -> tvar

    .global avgmaxmin_month     @;Definim la funcio com a global per a poder ser cridada desde altres funcions
avgmaxmin_month:                @; Inicialitzem la funcio "avgmaxmin_month"
    push {r1-r12, lr}           @; Guardem les dades

    mov r7, #0                  @; Inicialitzem les variables locals a 0
    mov r8, #0                  @; Inicialitzem les variables locals a 0

    ldr r4, [r0, r2, lsl #2]    @; Carreguem a r4 el calcul de: r0 + r11 * 4 (dades tipus word = 32bits)
    mov r5, r4                  @; Igualem el contingut de r4 a r5 "max = avg"
    mov r6, r4                  @; Igualem el contingut de r4 a r6 "min = avg"

@; BUCLE
    mov r9, #1                  @; Inicialitzem la i a 1 "r9 = i = 1"
    mov r12, #12                @; Inicialitzem r12 amb 12 volors "12 messos"
.Lfor2:                         @; Inicialitzem el ".Lfor2"
    cmp r9, r1                  @; Comparem si r1 es major a r9 "i < n_rows"
    bhs .Lfifor2                @; Si la condicio no es compleix salta al final del for
    mul r11, r9, r12            @; Ara fem una multiplicacio de "i*12 messos" i ho guardem en r11
    add r11, r2                 @; Al resultat de la multiplicacio sumem r2 "r11 = (i*12 + r2)"
    ldr r10, [r0, r11, lsl #2]  @; Carreguem a r10 la ttempertura calculada en r11 i desplacem dos bits
                                @; "tvar = ttemp[id_city][i] = r0 + (r9*12 + r2) * 4"
    add r4, r10                 @; Afegim a avg el contingut de tvar
    @; IF per trobar el MAX
    cmp r10, r5                 @; Comparem si tvar es MAJOR que MAX "tvar > max"
    ble .Lifmin2                @; Si tvar es mes petit < que max saltara al IF MIN
    mov r5, r10                 @; Igualem el contingut de max = tvar "perque es el valor maxim"
    mov r8, r9                  @; Igualem el la ID MAXIMA a i
    @;IF per trobar el MIN
.Lifmin2:                       @; Inicialitzem el ".Lifmin2"
    cmp r10, r6                 @; Comparem si tvar < min si es compleix entra a la condicio
    bge .Lfif2                  @; Si no es compleix salta a .lfif2
    mov r6, r10                 @; Igualem el min = tvar "ja que em trobat el valor minim"
    mov r7, r9                  @; idmin = i
.Lfif2:                         @; Inicialitzem el ".Lfif2"
    add r9, #1                  @; Per acabar amb el for, em d'afegir el i++
    b .Lfor2                    @; Al acabar d'incrementar la "i++" fem un salt a .Lfor2
.Lfifor2:                       @; Inicialitzem el ".Lfifor2"
@; FINAL DEL BUCLE
```

```
@; /12
    @; r0->ttemp[][12], r1->den=12, r2->&cociente, r3-> &resto, r4->avg, r11-> avgNeg
    mov r0, r4                  @; Movem avg a r0
    mov r11, #0                 @; Inicialitzem avg a Negatiu
    cmp r0, #0                  @; Comparem el contingut de avg si es mes petit que 0 "if (avg < 0)"
    movlt r11, #1               @; Si avgNeg es menos que 1 se movera el contenido
    @; lt = menos que movlt = movemos si es menos que 1
    rsblt r0, #0                @; Una vegada verifiquem que es NEGATIU ho passem a POSITIU
    @; rsblt fa una resta si el valor de r0 es MENOR que 0
    mov r1, #12                 @; Afegim a r1 el denominador 12"messos"
    mov r4, r3                  @; Passem avg a r3 "residu"

    ldr r2, =data_c             @; Carreguem a r2 el registre de data_c
    ldr r3, =data_r             @; Carreguem a r3 el registre de data_r

    bl div_mod                  @; Cridem la funcio per a fer la divisio entre 12
    @; Sols ens importa el COCIENT
    ldr r1, [r2]                @; Carreguem a r1 el registre de r2"data_c"

    cmp r11, #1                 @; Comparem si avgNeg si es NEGATIU
    rsbeq r1, #0                @; Si es NEGATIU ho passem a POSITIU
@; FINAL /12

    @; transferir datos al struct
    str r6, [r4, #MM_TMINC]     @; Guardem les dades en memoria, en aquest cas temperatura minima en MM_TMINC "mmres->tmin_C = min;"
    str r5, [r4, #MM_TMAXC]     @; Guardem les dades en memoria, en aquest cas temperatura maxima en MM_TMAXC "mmres->tmax_C = max;"
    mov r0, r6                  @; Igualem el contingut de "ttemp a min"
    bl Celsius2Fahrenheit       @; Invoquem/Cridem la funcio "Celsius2Fahrenheit(min);"
    str r0, [r4, #MM_TMINF]     @; Guardem les dades en memoria, en aquest cas temperatura minima Fahrenheit en MM_TMINF
                                @; "mmres->tmin_F = Celsius2Fahrenheit(min)"
    mov r0, r5                  @; Igualem el contingut de "ttemp a max"
    bl Celsius2Fahrenheit       @; Invoquem/Cridem la funcio "Celsius2Fahrenheit(max);"
    str r0, [r4, #MM_TMAXF]     @; Guardem les dades en memoria, en aquest cas temperatura maxima Fahrenheit en MM_TMAXF
                                @; "mmres->tmax_F = Celsius2Fahrenheit(max);"
    strh r7, [r4, #MM_IDMIN]    @; Guardem les dades en una memoria de 16 bits"short", la ID de temperatura minima MM_IDMIN
                                @; "mmres->id_min = idmin (2bits);"
    strh r8, [r4, #MM_IDMAX]    @; Guardem les dades en una memoria de 16 bits"short", la ID de temperatura maxima MM_IDMAX
                                @; "mmres->id_max = idmax (2bits);"

    mov r0, r1                  @; Igualem el contingut de ttemp a files "return(cociente=avg);"
    pop {r1-r12, pc}            @; Recuperem les dades i tornem a la funcio que crida aquesta rutina
```

2.4 Joc de proves "ampliat"

PROVA	DESCRIPCIÓ	RESULTAT ESPERAT	RESULTAT OBTINGUT	OK?
1	p avgres	{114254, 103387, 52906, 78255}	{-337683, -33099, 52906, 123904}	{-337683, -33099, **52906**, 123904}
2	p maxminres[0]	{105606, 121766, 321167, 350256, 0, 7}	{ -365702, -313353, -133994, -39763, 0, 7}	{ -365702, -313353, -133994, -39763, **0**, **7**}
3	p maxminres[1]	{22528, 157286, 171623, 414194, 4, 13}	{-376023, -157286, -152572, 807410, 7, 13}	{-376023, **157286**, -152572, 807410, 7, **13**}
4	p info_HNord[4].name	"Dikson"	"Dikson"	CORRECTE
5	p info_HNord[413].name	"Reggane"	"Reggane	CORRECTE
6	p maxminres[2]	{36454, 70451, 196690, 257887, 6, 1}	{36454, 70451, 589906, 651103, 6, 1}	{**36454**, **70451**, 589906, 651103, **6**, **1**}
7	p maxminres[3]	{33178, 113869, 190794, 336041, 15, 12}	{33178, 113869, 584010, 729257, 15, 12}	{**33178**, **113869**, 584010, 729257, **15**, **12**}
8	p info_HSud[15].name	"Stanley"	"Stanley"	CORRECTE
9	p info_HSud[12].name	"Port Moresby"	"Port Moresby"	CORRECTE

3. Fase 3

3.1 Especificacions

El que hem de fer per a implementar la Fase 3, es anar al directori "sources/" i afegir el fitxer Q12.s, en la primera línia del codi farem un include Q12.i per a importar els paràmetres a sumar/restar.

Suma

Primerament el que hem fet a sigut definir la funció "add_Q12" i l'hem inicialitzat.

Tenim els mateixos paràmetres d'entrada i de sortida. R0 per a introduir el primer número i r1 per al segon numero de forma que la rutina "r0 = r0+r1".

Seguidament mirem amb r3 si tenim overflow o no, segons si detecta que tenim overflow o no, anirà per movvc o movvs.

```
    .global add_Q12
add_Q12:
    push {r3, lr}                   @; Guardem les dades
        adds r0, r1                 @; Realitzem la suma a r0
        @;Afegim una "s" a la instruccio "add" per tal de poder actualitzar amb la opercaio els flags...
        @; ... i comprovar si ha hagut overflow o no
        movvc r3, #0                @; Si v=0 llavors direm a r3 que no hi ha overflow
        @; Amb la instrucci□ "vc" que afegim a "mov" volem dir que no hi ha overflow
        movvs r3, #1                @; si v=1 llavors direm a r3 que si hi ha overflow
        @; Amb la instrucci□ "sc" que afegim a "mov" volem dir que hi ha overflow
        strb r3, [r2]               @; Escriure resultat a "overflow"
        @; "b" = escriu bits alts al signe
    pop {r3, pc}                    @; recuperem les dades y tornem a la funcio que crida aquesta rutina.
```

Resta

La resta funciona de la mateixa manera que la suma.
Hem definit la funció "sub_Q12" i l'hem inicialitzat.
Tenim els mateixos paràmetres d'entrada i de sortida. R0 per a introduir el primer número i r1 per al segon numero de forma que la rutina "r0 = r0-r1".

Seguidament mirem amb r3 si tenim overflow o no, segons si detecta que tenim overflow o no, anirà per movvc o movvs.

```
.global sub_Q12
sub_Q12:
    push {r3, lr}                   @; Guardem les dades
        subs r0, r1                 @; realitzem la resta a r0
        @;Afegim una "s" a la instruccio "sub" per tal de poder actualitzar amb la opercaio els flags...
        @; ... i comprovar si ha hagut overflow o no.
        movvc r3, #0                @; Si v=0 llavors direm a r3 que no hi ha overflow
        @; Amb la instrucci□ "vc" que afegim a "mov" volem dir que no hi ha overflow
        movvs r3, #1                @; si v=1 llavors direm a r3 que si hi ha overflow
        @; Amb la instrucci□ "sc" que afegim a "mov" volem dir que hi ha overflow
        strb r3, [r2]               @; Escriure resultat a "overflow"
        @; "b" = escriu bits alts al signe
    pop {r3, pc}                    @; recuperem les dades y tornem a la funcio que truca aquesta rutina.
```

Divisió i Multiplicació, aquestes dues funcions les tenim que inicialitzar, ja que si no ho fem el programa entra en conflicte amb el MAKEFILE, ja que aquest espera la crida de les funcions i al no rebre res aquest es queixa i no deixa executar el programa.

```
@; MULTI
    .global mul_Q12                 @; Si no definim ni incialitzem la funcio mul_Q12 peta el programa
mul_Q12:                            @; Perque les llibreries del "makefile" les crida i al no estar "peta"
    push {lr}
    pop {pc}

@; DIVI
    .global div_Q12                 @; Si no definim ni incialitzem la funcio div_Q12 peta el programa
div_Q12:                            s@; Perque les llibreries del "makefile" les crida i al no estar "peta"
    push {lr}
    pop {pc}
```

3.2 Disseny

SUMA i RESTA

En la rutina suma i resta al fer les instruccions (add i sub), s'afegeix una "s", d'aquesta manera seran(adds i subs), aixo ho fem per a poder actualitzar amb aquestes operacions els flags i per poder comprobar directament si tenim overflow o no, gràcies a les predicacions de les instruccions vc(no tenim overflow) i vs(tenim overflow).

```
    .global add_Q12
add_Q12:
    push {r3, lr}                   @; Guardem les dades
        adds r0, r1                 @; Realitzem la suma a r0
        @;Afegim una "s" a la instruccio "add" per tal de poder actualitzar amb la opercaio els flags...
        @; ... i comprovar si ha hagut overflow o no
        movvc r3, #0                @; Si v=0 llavors direm a r3 que no hi ha overflow
        @; Amb la instrucci "vc" que afegim a "mov" volem dir que no hi ha overflow
        movvs r3, #1                @; si v=1 llavors direm a r3 que si hi ha overflow
        @; Amb la instrucci "sc" que afegim a "mov" volem dir que hi ha overflow
        strb r3, [r2]               @; Escriure resultat de si tenim overflow o no "overflow"
        @; "b" = escriu bits alts al signe
    pop {r3, pc}                    @; recuperem les dades y tornem a la funcio que crida aquesta rutina.
```

```
.global sub_Q12
sub_Q12:
    push {r3, lr}                   @; Guardem les dades
        subs r0, r1                 @; realitzem la resta a r0
        @;Afegim una "s" a la instruccio "sub" per tal de poder actualitzar amb la opercaio els flags...
        @; ... i comprovar si ha hagut overflow o no.
        movvc r3, #0                @; Si v=0 llavors direm a r3 que no hi ha overflow
        @; Amb la instrucci "vc" que afegim a "mov" volem dir que no hi ha overflow
        movvs r3, #1                @; si v=1 llavors direm a r3 que si hi ha overflow
        @; Amb la instrucci "sc" que afegim a "mov" volem dir que hi ha overflow
        strb r3, [r2]               @; Escriure resultat a "overflow"
        @; "b" = escriu bits alts al signe
    pop {r3, pc}                    @; recuperem les dades y tornem a la funcio que truca aquesta rutina.
```

3.3 Implementació

```
@;-----------------------------------------------------
@;  Programador/a 1: florian.serb@estudiants.urv.cat |
@;  Programador/a 2: alan.rocha@estudiants.urv.cat   |
@;-----------------------------------------------------

.include "includes/Q12.i"

.text
    .align 2
    .arm

@;SUMA
@; r0 -> num1
@; r1 -> num2
@; r2 -> unsigned char *overflow    @; r2 -> &overflow
                                    @; r3 -> ov
@; Output -> sortida per r0 de num1 + num2, es modifica el valor de r2 per parametre (si r2=0 -> !overflow, si r2=1 -> overflow)

    .global add_Q12
add_Q12:
    push {r3, lr}                   @; Guardem les dades
        adds r0, r1                 @; Realitzem la suma a r0
        @;Afegim una "s" a la instruccio "add" per tal de poder actualitzar amb la opercaio els flags...
        @; ... i comprovar si ha hagut overflow o no
        movvc r3, #0                @; Si v=0 llavors direm a r3 que no hi ha overflow
        @; Amb la instrucci□ "vc" que afegim a "mov" volem dir que no hi ha overflow
        movvs r3, #1                @; si v=1 llavors direm a r3 que si hi ha overflow
        @; Amb la instrucci□ "sc" que afegim a "mov" volem dir que hi ha overflow
        strb r3, [r2]               @; Escriure resultat de si tenim overflow o no "overflow"
        @; "b" = escriu bits alts al signe
    pop {r3, pc}                    @; recuperem les dades y tornem a la funcio que crida aquesta rutina.
```

```
@; RESTA
@; r0 -> num1
@; r1 -> num2
@; r2 -> unsigned char *overflow
@; output -> sortida per r0 de num1 - num2, es modifica el valor de r2 per parametre (si r2=0 -> !overflow, si r2=1 -> overflow)
.global sub_Q12
sub_Q12:
    push {r3, lr}                   @; Guardem les dades
        subs r0, r1                 @; realitzem la resta a r0
        @;Afegim una "s" a la instruccio "sub" per tal de poder actualitzar amb la opercaio els flags...
        @; ... i comprovar si ha hagut overflow o no.
        movvc r3, #0                @; Si v=0 llavors direm a r3 que no hi ha overflow
        @; Amb la instrucci□ "vc" que afegim a "mov" volem dir que no hi ha overflow
        movvs r3, #1                @; si v=1 llavors direm a r3 que si hi ha overflow
        @; Amb la instrucci□ "sc" que afegim a "mov" volem dir que hi ha overflow
        strb r3, [r2]               @; Escriure resultat a "overflow"
        @; "b" = escriu bits alts al signe
    pop {r3, pc}                    @; recuperem les dades y tornem a la funcio que truca aquesta rutina.

@; MULTI
    .global mul_Q12                 @; Si no definim ni incialitzem la funcio mul_Q12 peta el programa
mul_Q12:                            @; Perque les llibreries del "makefile" les crida i al no estar "peta"
    push {lr}
    pop {pc}

@; DIVI
    .global div_Q12                 @; Si no definim ni incialitzem la funcio div_Q12 peta el programa
div_Q12:                            s@; Perque les llibreries del "makefile" les crida i al no estar "peta"
    push {lr}
    pop {pc}
```

3.4 Joc de proves "ampliat"

PROVA	DESCRIPCIÓ	RESULTAT ESPERAT	RESULTAT OBTINGUT	OK?
1	disp num_ops_ok	38	16	CASI
2	disp num_ovf_ok	38	14	CASI
3	disp num_tests	38	38	CORRECTE
4	disp num_errors	0	46	CASI

Conclusions

Aquesta pràctica ens ha semblat bastant complicada, ja que des de l'inici vam tindre complicacions en saber com començar. Considerem que les classes de laboratori no han sigut suficient per poder realitzar la pràctica satisfactòriament.

La primera fase va ser la més assequible, perquè sí que s'entenia el que es demanava i va ser més fàcil traduir el codi de C a assemblador.

A la segona fase, van arribar els problemes, perquè no hem aconseguit tots els valors correctes que es demanaven. De totes maneres, ja hem comentat aquest resultat amb el professor i esperem que es tingui en compte l'esforç.

A la tercera fase vam tornar a tindre complicacions amb les funcions de multiplicació i divisió, i sumant el baix nivell d'ARM que tenim i la mancança de temps vam decidir deixar aquestes funcions de banda.

En general, ens ha quedat una sensació d'insatisfacció perquè no hem aconseguit assolir la pràctica satisfactòriament.