

**2. GAIA- AGINDU MULTZOA.****2.1. Ondoko aldagaien definizioak emanda:**

- A: .word 7  
 B: .word -12  
 C: .word 28

Adierazi ondoren agertzen den agindu multzo bakoitzaren exekuzioan memoriain eta erregistroetan sortzen diren aldatak. Kasu bakoitzean, aldagaien hasierako balioa kontuan hartu.

a)	ldr r5,A mov r1, #6 add r5,r5,r1 str r5,B	b)	ldr r2,B lsl r3,r2,#2 lsl r4,r2,#1 asr r5,r2,#1 str r3,A str r5,C	c)	ldr r4,C ldr r6,A and r5,r4,#1 and r7,r6,#1 or r8,r4,#1 or r9,r6,#1 str r9,A
----	--	----	--	----	--

**2.2. Ondoko aldagaien definizioak emanda:**  
hreibidea

- 40 Y: .word 12,15  
 48 Z: .word 2

Adierazi ondoren agertzen den agindu multzo bakoitzaren exekuzioan memoriain eta erregistroetan sortzen diren aldatak. Kasu bakoitzean X eta Y aldagaien hasieraren hartzan duten balioa kontuan hartu. R1 erregistroaren hasierako balioa 40 da.

a)	ldr r5,Y str r5,Z	b)	adr r2,Z mov r5,#0 ldr r6,[r1,#4]	c)	ldr r5,Y lsl r6,[r1,#4] add r4,r5,r6 str r5,#2 str r6,[r2,r5]
					r3,[r1,#12]

**2.4. Idatz ezazu mihiztadura-lengoiaiaz ZENB izeneko aldagaien gordetako zenbakia oso baten balio absolutua kalkulatu eta ABS aldagaien gordetako programa.****2.5. Idatz ezazu programa bat mihiztadura-lengoiaiaz ZENB aldagaien dagoen balioa bikoitia badda BI aldagaien 1eko bat itzultzen duena eta Oko bat bestelakoan.****2.6. Idatz ezazu mihiztadura-lengoiaiaz, ZENB aldagaien dagoen zenbakia positiboa edo negatiboa bada, hurrenez hurren, POSI aldagaien 1eko edo Okoa gordetzen duen programa.****2.7. ARM mihiztadura lengoiaian idatzitako ondoko programa ematen da. Suposatuz aldagaiak memoriako 0 helbidekit aurerra gordetzen direla, adierazi agindu bakoitzaren exekuzioaren ondorioz erregistroetan eta memoriain gertatzeren diren aldataketaik.**

```
.code 32
.global main, __cpsz_mask
main:
    adr r2, P
    adr r3,T1
    adr r4,T2
    mov r1,#3
    ldr r5,[r3,r1, 1s1 #2]
    ldr r6,[r4,r1, 1s1 #2]
    add r7,r5,r6
    etik2:
    add r1,r1,#1
    b etik1:
    str r1,[r2]
    mov pc,lr
```

**2.8. Mihiztadura lengoiaian idatzitako ondoko programa emanda:**

```
T1:
.word 10,-7,27,-4,0,564,611,
12
T2:
.word 18,12,2,4,80,-564
P:
.word 1

.code 32
.global main, __cpsz_mask
main:
    adr r2, P
    adr r3,T1
    adr r4,T2
    mov r1,#3
    ldr r5,[r3,r1, 1s1 #2]
    ldr r6,[r4,r1, 1s1 #2]
    add r7,r5,r6
    etik2:
    add r1,r1,#1
    b etik1:
    str r1,[r2]
    mov pc,lr
```

```
1s1 z2,r0,#2
1dr r3,[r1,r2]
cmp r3,#0
bts BUK
add r0, r0, #1
b FOR
str r3, N
BUK:
    mov pc, lr
```

Erantzun galadera hauek:  
 a ) Esan zein den N aldagaiaren balioa programa bukatzean  
 b ) Zenbat aldiriz exekutatzen da FOR: (cmp r0,#10) etiketa duen agindua adibide zehatz honetan?  
 c ) Zergatik behar da 1s1 r2,r0,#2 agindua?  
 d ) Azaldu programa honek egiten duena. Ez da azken emaitza eskatzen, ezta agindu aginduko azalpen bat ere, ebazten duen problemaren definizioa  
 eskatzen da.

Erantzun galadera hauak:  
 a ) Adierazi NEG aldagaiak programaren bukaeran hartzan duen balioa.  
 b ) Zergatik da beharrezko b BUK agindua?  
 c ) Azaldu programa honek egiten duena. Ez da eskatzen programa agindu agindua azaltzea, ebazten duen problemaren definizioa nahi da.



2.24

	B	D	EM(IR)	VAL	MEM	EM(ID)
move r3, #0x E6	IR = M[PC] PC = PC + 4	desk	-	magi tv		
ldr r2, [r3, #6]	"	"	r3 = 000000E6	r3 + 6 = M[r3+6] = 000000EC	2nd line	r2 = FF'FF FF F8
ldr r5, [r3, R4]	"	"	r3 = 000000E6 r4 = 000000A	R4 + r4 = >00000F0 M[r3+0F0]	3rd line	r3 = 000000F0 r5 = 00000100
str r5, [r3, #4]	"	"	r3 = 000000F0	r3 + (-4) = M[r3-4] = 000000E8	4th line	
ldr r6, [r5], #7	"	"	r5 = 00000100	M[r5+7] = 000000E7		
str r4, [r5, r2]	"	"	r5 = 00000107	r5 + 7 = M[r5+100] = 00000100	5th line	r5 = 00000107 r6 = 00 00010C
			r2 = FF'FF FF F8	r5 + r2 = M[r5+100] = 00000100	6th line	r4 = 0000000A

2.26

	B	D	EM(IR)	VAL	MEM	EM(ID)
ldr r3, x	Ir = M[PC] PC = PC + 4	desk	-	-	M[00000000]	
adr r4, y	"	"				
ldr r5, [r4]	"	"				
ldr r6, [r4, #4]	"	"				
ldr r7, [r4, #n]	"	"				



2.26

00000080	00 00 00 08 : x 00 00 00 04 : y 00 00 00 05 00 00 00 06
00000084	
00000088	
0000008C	
00000090	

34 21 23 45 : z

ldr r3, x

$$\begin{aligned} r3 &= M[x] & r3 &= M[00000080] \\ r3 &= 00 00 00 08 \end{aligned}$$

adr r4, y

$$\begin{aligned} r4 &= @y \\ r4 &= 00000084 \end{aligned}$$

ldr r5, [r4]

$$\begin{aligned} r5 &= M[00000084] & r5 &= 00 00 00 04 \\ r4 &= 00 00 00 84 \end{aligned}$$

ldr r6, [r4, #4]

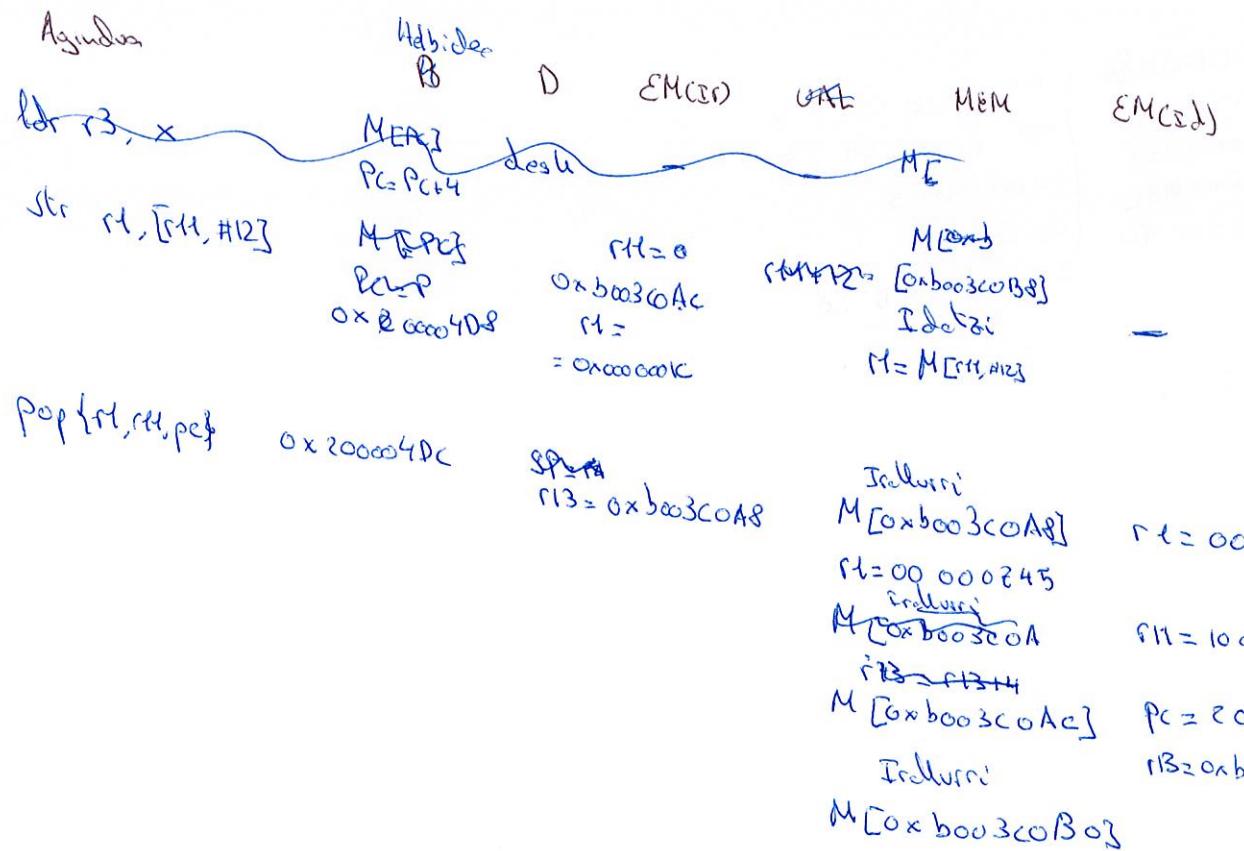
$$\begin{aligned} r4 &= 00000084 \\ r4+4 &= 00000088 \end{aligned}$$

$$\begin{aligned} r6 &= M[00000088] & r4 &= 00000084 \\ r6 &= 00 00 00 05 \end{aligned}$$

ldr r7, [r4] #4

$$\begin{aligned} r4 &= 00000088 \\ r8 &= M[00000088] & r8 &= 00 00 00 05 \\ r4+r4 &= 0000008C \end{aligned}$$

27



add SP, SP, #4 OR 20 00 03 BC      R13 = 0x3003C0B8  
 LDR R2 [SP], #4      0x200003BC0      R13 = 0x3003C0B8  
 M[0x20003C0B8]      R13 = 0x3003C0B8  
 R2 = 00 00 00 1C  
 1000      push {lr}      M(M)      EM(D)  
 1004      adr R1, A      Rr = 100      M(M)  
 1008      adr R0, B      Sp = 2000      M[3000]\_R1  
 100C                          Sp = 2004

## 2.1 ARIKETA

Hasterako aldagaien balioa

A.: word 7

B.: word -12

C.: word 28

Memoiako Edukia		
A	B	C
00	FF	00
00	FF	FF
00	00	00
07	F4	1C

Zenbaki osoak birako osagarran adierazten dira.

a)

```
ldr r5, A      r5 = M[A], r5= 7
mov r1, #5     r1= 6
add r5,r5,#1   r5 = r5 + r1= 13
str r5, B      M[B]=13
```

b)

```
ldr r2, B      r2 = M[B], r2=-12
lsr r3, r2, #2  r3 = r2 * 4, r2= -48

lsr r4, r2, #1
r2= 1111 1111 1111 1111 1111 1111 1111 0100 (-12)
eskunera desplazatzu posizio bat baina Okoa sartuta ezkerretik
r4= 0111 1111 1111 1111 1111 1111 1111 1010
r4= 7F FF FF FA (horrela utzi dezakegu balio hamartarra eman gabe)

asr r5, r2, #1
r2= 1111 1111 1111 1111 1111 1111 1111 0100 (-12)
Hau desplazamendu aritmetikoa da, ezkerretik pisu handieneko bita
sartzen da
r5 = 1111 1111 1111 1111 1111 1111 1111 1010
r5 = -6 (hau da r2/2)
```

```
str r3, A      M[A]=-48
str r3, C      M[C]=-6
```

c)

```
ldr r4, C      r4 = M[C], r4=28
ldr r6, A      r6 = M[A], r6=7
and r5, r4, #1   r5 = r4 and 1 =0
r4      0000 0000 0000 0000 0000 0000 0001 1100
#1      0000 0000 0000 0000 0000 0000 0001 0001
```

and 0000 0000 0000 0000 0000 0000 0000 0000

and r7, r6, #1	r7 = r6 and 1 = 1
r6	0000 0000 0000 0000 0000 0000 0000 0111
#1	0000 0000 0000 0000 0000 0000 0000 0001
and	0000 0000 0000 0000 0000 0000 0000 0001
or r8, r4, #1	r8 = r4 or 1 = 29
r4	0000 0000 0000 0000 0000 0000 0000 0001 1100
#1	0000 0000 0000 0000 0000 0000 0000 0000 0001
or	0000 0000 0000 0000 0000 0000 0000 0001 1101
or r9, r6, #1	r9 = r6 or 1 = 7
r6	0000 0000 0000 0000 0000 0000 0000 0000 0111
#1	0000 0000 0000 0000 0000 0000 0000 0000 0001
or	0000 0000 0000 0000 0000 0000 0000 0000 0111
str r9, A      M[A] = 7	



2.15

3

Code 32

global memory - cpsr, - mask

word 0, 1, 2, 3, 4, 5;

main:

addr 505

more often

for: cmp r1, A3

bag bike

push front

be Adm.

add sp sp #8

add fl, st, #1

↳ for

Autograph

push free, with }

Nov 1st, 1981

push { r2, r6, r5, r4, r3, r2 }  
mov r2, H2

ldr r2 [r0, r3 lsl #2]

Class Sub File No. 71/22

LDR +5 [R0,R4] LSP #27

83

Ldr 16.5.23

ldr \$2, [ST]

Str 16. [85]

str 58, Er 23

Pop 18, 16, 15

... , 113, 129, 111, pg }

62

void main()

Count by 1's, 2's, 3's, 4's, 5's, 6's, 7's, 8's, 9's

Beto Bettonec (\* B[EG], Bator)

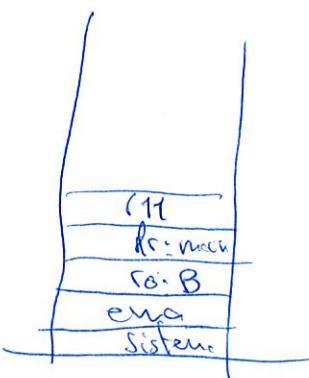
Batu

B :=  
Befores 0;  
main:

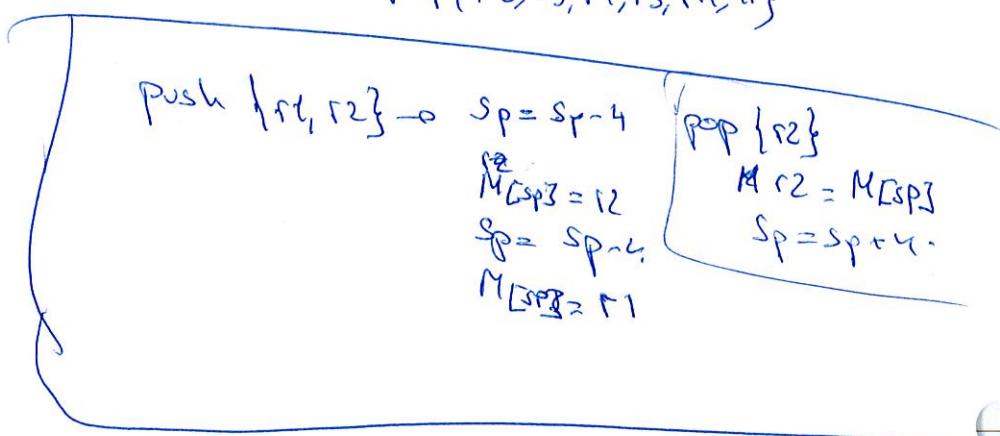
push {r3}  
adr r0, B  
mov r1, #0  
~~for : cmp r1, #10~~  
bge b4  
push f  
sub sp, sp, #4  
push {r0f}  
bl Befor.  
~~pop add sp, sp #4~~  
pop {r2f}  
Str r2, Befor.  
pop {pcf}

Befor.:

push {r1, r3}  
mov r1, sp  
push {r6, r5, r4, r3}  
lsl r5 [r1, #8]  
ldr r4, [r3, r5 lsl,  
ldr r3 [r1, #8]  
mov r4, #0  
~~for : mov r6, #0~~  
~~for : cmp r4, #10~~  
bge b4  
ldr r5 [r3, r4, lsl #2]  
add r6, r6, r5  
add r4, r4, #1  
b for  
b4: Str r4 [r1, #12]  
pop {r6, r5, r4, r3, r1, lr}



? 28



Habbiere

0x200003B8

0x200003DC

0x200003B0

mov r11, sp

Avgindue

bl A8P

push {r1, r2}

Erlagtschook

r13 = 0x200003B8

Individ:

$M_1 = 0x200A001$

ldori

$r1 = 0x200003BC$

$SP = SP + 4, SP = SP - 4$

$SP = 0x200300B4$

ldori

$r11 = 0x200300B4$

Memomoria

—

—

—

1.- Codifica el programa principal que está en C en ensamblador ARM.

2.- Dibuja la pila correspondiente a toda la ejecución.

3.- Intenta responder a las preguntas que se plantean a continuación.

a ). ¿Para qué se utilizan los registros r1, r2 y r3?

b ). La subrutina SUB1 es una función o un procedimiento? Justifica tu respuesta.

d ). ¿Qué cambiaría de no utilizar el registro r11? Justifica tu respuesta.

2.21.

1.- Codifica en ensamblador ARM la subrutina que está implementada en C.

2.- Dibuja la pila de toda la ejecución indicando con qué instrucción se corresponde cada entrada y cada salida de la pila.

3.- Explica para qué sirve el programa. No se pide lo que hace el programa instrucción por instrucción, sino una explicación de que problema solucionaría el programa.

4.- Si al pasar por la instrucción 12 el registro r1 valiera 4, ¿qué valor se cargaría en el registro r4?

```

1: .code 32
2: .global main, __cprsr_mask
3: main:
4:     push {lr}
5:     mov r1, #0
6:     ldr r2, n
7:     adr r7, A
8:     adr r8, B
9:     adr r9, C
10:    for: cmp r1, r2
11:    beq bfor
12:    ldr r4, [r7, r1, lsl#2]
13:    ldr r5, [r8, r1, lsl#2]
14:    mul r6, r4, r5
15:    str r6, [r9, r1, lsl#2]
16:    add r1, r1, #1
17:    b for
18:    bfor: sub sp, sp, #4
19:    ldr r10, m
20:    push {r10, r2, r9}
21:    bl SUB1
22:    add sp, sp, #12
23:    pop r3, res
24:    str r3, res
25:    pop {pc}

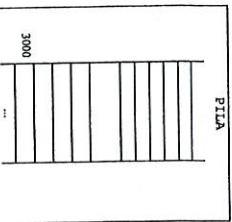
```

2.22. Se da el siguiente programa escrito en lenguaje ensamblador del ARM:

```

.code 32
.global main, __cprsr_mask
a: .word 5
b: .word 6
main:
    push {lr}
    adr r1, a
    adr r0, b
    push {r0, r1}
    bl SWAP
    add sp, sp, #8
    pop {pc}
SWAP:
    push {lr}
    ldr r3, [sp, #4]
    ldr r5, [r3]
    ldr r4, [sp, #8]
    ldr r6, [r4]
    str r5, [r4]
    str r6, [r3]
    pop {pc}

```



El programa completo se ha almacenado a partir de la dirección 1000 de memoria. Completa la traza de ejecución para las primeras 7 instrucciones del programa sabiendo que el valor inicial del registro lr (r14) es 100. Dibuja como queda la pila después de la ejecución de estas instrucciones, indicando las direcciones y el contenido de las posiciones de la pila. El valor inicial del registro sp (r13) es 3000. Para cada acceso a memoria o a un registro indica si el acceso es una lectura o una escritura.

1000 push {lr} R\_\_\_\_\_ ... R\_\_\_\_\_ @M:\_\_\_\_\_ EM:\_\_\_\_\_

```

int SUB1(int *d, int e, int f)
{
    int cont=0;
    for(i=0;i<e;i++){
        if(d[i] < f)
            cont++;
    }
    return cont;
}

```

**EJERCICIOS TEMA 2.- SUBRUTINAS**

- 2.14.** Escribe un programa que invierta los elementos de una tabla de 5 elementos utilizando una pila, para las tres siguientes posibilidades:
- sin utilizar instrucciones push y pop, y suponiendo que la pila crece hacia direcciones inferiores,
  - sin utilizar instrucciones push y pop, y suponiendo que la pila crece hacia direcciones superiores,
  - utilizando las instrucciones push y pop.

- 2.15.** En la siguiente tabla se muestra el contenido de algunas palabras de memoria y de algunos registros generales de una máquina ARM.

**MEMORIA**

Dirección	Contenido
0x000000DC	FF FF FF R11 = 000001D4
0x000000E0	00 00 00 R13 (sp) = 002000D4
0x000000E4	00 00 00 OF
0x000000E8	00 00 00 F0 push {r11}
.....	mov r11, sp
0x020000D0	00 00 01 00 ldr r5, [r11, #8]
0x020000D4	00 00 0A 41 ldr r4, [r5]
0x020000D8	00 00 00 DC add r4, r4, #7
0x020000DC	FF FF B4 str r4, [r11, #12]
0x020000E0	00 00 01 DC pop {r11}

- a) Refleja el efecto que produce, tanto en la memoria como en los registros generales, la ejecución de las siguientes instrucciones.

- b) Para cada una de las instrucciones anteriores, indica las acciones que se llevan a cabo durante la ejecución de cada instrucción.

- 2.16.** Escribe en C y en ensamblador del ARM una subrutina que recibe como parámetro un vector y devuelve la suma de todos los elementos del vector. Escribe también el programa principal que llama a esa subrutina. Supón que el vector es de 10 elementos.

- 2.17.** Pasa a ensamblador la siguiente subrutina escrita en C. Escribe un programa en ensamblador que llame a la subrutina fact.

```
int fact (int i)
{
    if (i==1) return(1);
    else return (fact(i-1)*i);
}
```

- 2.18.** Pasa a ensamblador el siguiente programa escrito en C.

<pre>int calcular (int x, int *y)</pre>	<pre>void main ()</pre>
<pre>{ int primero, segundo, resultado;</pre>	<pre>    primero=10;</pre>
<pre>    (*y) = z / 2;</pre>	<pre>    segundo=20;</pre>
<pre>    resultado = calcular(primerero, &amp;segundo);</pre>	<pre>}</pre>

- 2.19.** Dados un programa principal escrito en C y una subrutina SUB escrita en ensamblador ARM:

- 1.- Codifica el programa principal que está en C en ensamblador ARM.

- 2.- Dibuja la pila correspondiente a toda la ejecución.

```
void main ()
{
    int A[8] = {15,12,7,-8,42,21,-3,4};
    int n=0;
    for (i=0; i<8;i++)
        if (A[0]>A[i])
        {
            SUB (&A[0], &A[i]);
            n++;
        }
}
```

3.- Responde a las preguntas que se plantean a continuación.

- a ) ¿Para qué se utilizan los registros r3 y r5 en la subrutina?
- b ) La subrutina SUB ¿es una función o un procedimiento? Justifica tu respuesta.

- c ) ¿Cuántas veces se ejecuta la subrutina SUB en este caso concreto?

- d ) ¿Cuál es el valor de r4 al ejecutar la subrutina SUB?
- e ) ¿Qué cambiaría de no utilizar el registro r11? Justifica tu respuesta.

- f ) Explica que hace la instrucción: std r5,[r3].

- 2.20.** Dados un programa principal escrito en C y una subrutina SUB1 escrita en ensamblador ARM:

```
void main()
{
    int b[6]={7,-8,42,21,-71,0};
    int n=6, i, res[6];
    SUB1(b, n, res);
}
```

FOR:

```
    cmp r1,r3
    beg FIN
    ldr r5, [r2,r1,ls1#2]
    cmp r5, #0
    bgt IF
    mov r6, #0
    b FINIF
    sub sp, sp, #4
    push {r5}
    bl SUB2
    add sp, sp, #4
    pop r6
    FINIF: ldr r7, [r11,#8]
    str r6, [r7,r1,ls1#2]
    add r1, r1, #1
    b FOR
FIN:  pop {r3-r7,r11,pc}
```

2. 20

1)

code 32

· globe main, -cpr\$,-mask

A: .word 15, 12, 7, -8, 42, 21, -3, 4

N: .word 0

main:

adr r0, A

mov r1, #1

ldr r10, N

For: cmp r1, #8

bge BUK

ldr r7, [r0, r1, lsl #2]

ldr r8, [r0]

if: cmp r8, r7

ble for bsl r8, r1, lsl #2

push {r0, r7} add r8, r0, r8

bl AZP r9 sagaratu helbideak

add r1, r1, #1 add sp, sp, #8

add r10, r10, #1

b for

BUK: mov pc lr str r10, r pop {pc}

3)

a)

r3 eta r5 A bektoreko A[0] eta A[i] -ren helbideak gordetzeako erabiltzen dira ✓

b) AZP prozedura da, A bektorea eraldatzen baita. ✓

c) 3 aldiz exekutatuko da AZP azpirutine. ✓

d) r4 enregistroan A[0] bektorearen balioa gordetxo du.

e) r11 enregistroari esker pilako paragonotroak emezago atxikitzen dira.

f) r6 enregistroaren balioa r3 helbidearteko balioaren ordez aldatzen du. idazten du memoriako

2)

r3

r4

r5

r6

r11

lr:main

r0: A[0]

r7: A[i]

lr: sistema

Iulen Fernández

Aitzol Elu

2.17

```
void main ()  
{ int B[6] = {3,4,-7,8,9,10};  
    int Batura = 0;  
    Batura = Batu (B),
```

```
int Batu (int * Bck)  
{ int Bat=0  
    int i=0  
    for (i=0; i<6; i++)  
        bat = bat + Bck [i]  
    return bat }
```

· code 32  
· global main, - cpsr - mask

B: .word 3, 4, -7, 8, 9, 10  
Batura: .word 0

main:

```
push {lr}  
adr r0, B  
sub sp, sp, #4  
push {r0} ✓  
bl BATU  
add sp, sp, #4  
pop {r6}  
str r6, Batura  
pop {pc}
```

BATU: push {r11, lr}

```
mov r11, sp  
push {r1, r2, r3, r4, r5}  
ldr r3, [r11, #8]
```

mov r1, #0

mov r2, #0

FOR: cmp r1, #6

beq BUK

lsl r4, r1, #2

ldr r5, [r3, r4]

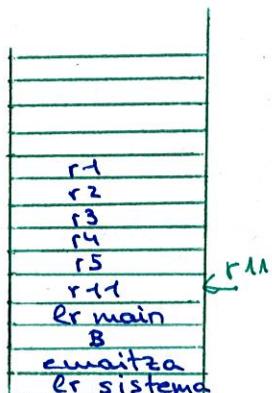
add r2, r2, r5

add r1, r1, #1

b FOR

BUK: str r5 [r11, #12]

pop {r1, r2, r3, r4, r5, r11, pc}



2.22

push { r11, lr }  
mov r11, sp ✓  
push { r3, r4, r5 }  
mov r3, #0  
mov r4, #0  
for: cmp r4, r2  
bge buk  
ldr r5 [r9, r4, lsl #2]  
add r4, r4, #1  
cmp r5, r10  
bge for  
add r3, r3, #1  
b for  
buk: str r3 [r11, #20] ✓  
pop { r3, r4, r5, r11, pc }

parametroak pilan dantza eta  
pulalek urakurri behar dira

r3	← r11
r4	
r5	
r11	
lr main	
r10: m	
r2: n	
r9: c	
cueaitza	
er sistema	

ldr r9, [r11, #16]  
ldr r2, [r11, #12]  
ldr r10, [r11, #8]

↓  
erregistro zentzaki  
biarrek edozein  
izan da.

Jolen Fernández  
UNOZ

Aitzol EPV



# Ariadna Elu

Code 32

glade\_main, -cpos, -maddr

n : word -5

emai\_t3o : , word 0

main:

if : ldr r0, n  
adr r1, n  
wzr r0,r1  
cmp r0, #0  
blt else

push {r0}

bl positivo

~~else:~~ b bult

push {r1}

bl negativo

add sp, sp #8

pop {r0}

str r0, emai

b bult

else: push {r1}

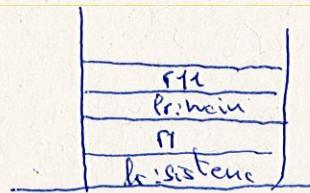
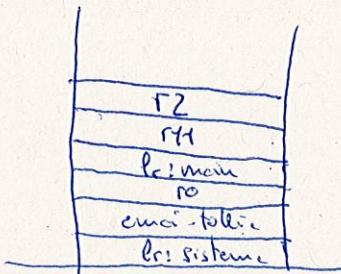
bl negativo

add sp, sp #8

pop {r1}

str [r1], n

bult:



Positivo:

~~push {r1, lr}~~

~~mov r1, sp~~

~~push {r2}~~

~~push {lr}~~

push {r1, lr}

mov r1, sp

push {r2}

ldr r2, [r1, #8]

add r2, r2, #1

str r2, [r1, #12]

pop {r2, r1, blr}

Negativo

push {r1, lr}

mov r1, sp

push {r3}

ldr r3, [r1, #8]

sub r3, r3, #1

pop {r3, r1, lr}



### **3. Laborategia : Azpirrutinak**

1. ARTIKELA

Ondoren agertzen diren programa nagusia eta azpirrutinak aztertzea nahi ditugu. Lehenik kodetu mihiztadura lengoainan programa nagusia eta azpirrutinak. Ondoren, aztertu bere exekuzioa.

```
void main ()
{ int n=-5, emaitza;

    if (n>=0)
        emaitza=positibo(n);
    else
        negatibo(&n);
}
```

```
int positibo (int zenb)
{
    return(zenb+1);
}

void negatibo (int *zenb)
{
    (*zenb) --;
}
```

Simulatu eta araztatu programa hau eta erantzun ondorengo galderak.

- a) Pila memoriako zein helbidetatik aurrera gordetzen da?

b) Osatu pila positibo azpirrutinaren exekuziorako, bere posizio bakoitzerako edukiaren balioa eta helbidea adieraziz.

Zein da positibo azpirrutinatik programa nagusirako itzulera helbidea?

positibo azpirrutina funtzioa edo prozedura da?

## 2. gaiko laborategiak

## *Konputagailuen Egitura*

c) Osatu pila negatibo azpirrutinaren exekuziorako, bere posizio bakoitzerako edukiaren balioa eta helbidea adieraziz.

Zein helbidetan dago gordeta n aldagaia memorian?

Zein da negatibo azpirrutinatik programa nagusirako itzulera helbidea?

negatibo azpirrutina funtzioa edo prozedura da?

## Laborategiak

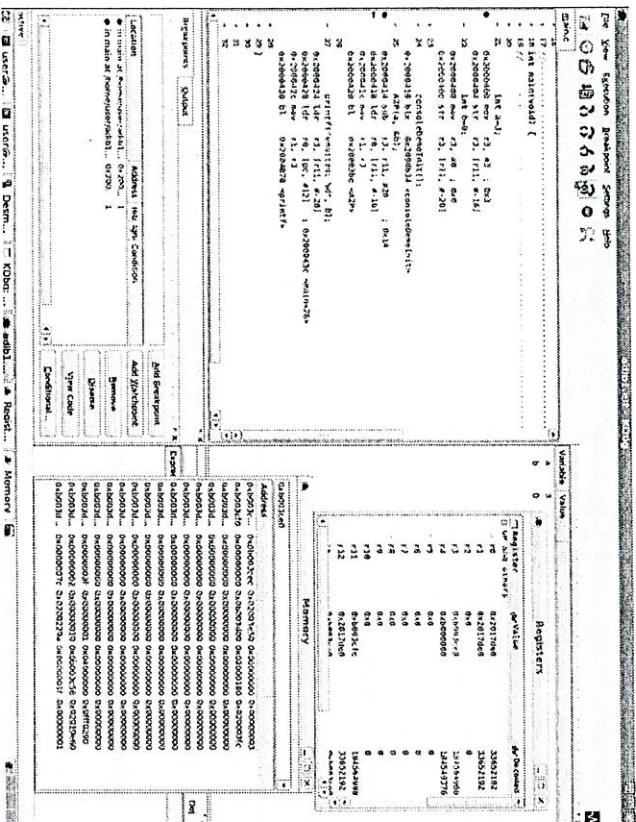
## Konputagailuen Egitura

## Laborategiak

## Konputagailuen Egitura

\$ kdbg prog.elf

Araztu nahi dugun exekutamaria (prog.elf goiko programan) kdbg-a martxan jarri ondoren ere adieraz daitelk, File | Executable menuko aukeraren bidez. Aplikazio honek aldagien, erregistroen eta memoriako helbideen edukiak ikusteko aukera ematen digu. Ondokoak da aplikazio honen pantaila nagusia.



ere ikus dezakegu View | Memory aukeraren bidez. Memoriako helbide zehatz baten edukia ikusi nahi badugu helbide hori sartu beharko dugu lehoan bertan agertzen den goiko gelaxkan.

## Arazketa

ARM9 prozesadore batean gure programen exekuzioa probatu, arazu eta aztertzeko desmume eta kdbg (gdb) trenak erabiliko ditugu. Kontuan hartu behar da gure programak exekutatzeko, ez badugu zuzenean NDS makina batean egiten simuladore bat beharko dugula, komplilazioren ondoren sortzen den kodea ez baita PC-rako, beste makina desberdin batentzat baizik. Horrexegatik erabilitzen dugun kompliladore kompliladore gurutzatu bat da: PC batean exekutatzen den kompliladore bat da baina NDS makina batentzat kodea sortzen du. Hau horrela izanda, kdbg-a erabiliz programa aratzeko, desmume programa erabiliz gure programa simulatu behar da eta simulazioaren irteera aratzialdearekin loru. Hori egiteko portu bat erabilitzan da, edozein izan daitetik ondoko balio tartetik: 0000 eta 9999.

adibide karpetan daudagun txantiloitik abiatuko gara programatzeko bat idazteko. Lehen programa honek egingo duen gauza bakarra "Hello, world" mezua pantailatzea da. Hau da bere kodea:

```
#include <nds.h>
#include <stdio.h>

void main()
{
    consoleDemoInit();
    printf("Hello, world\n");
}
```

Programa hau probatu aurretek kopiatu egin behar dugu. Horretarako lehen esan dugun bezala malke erabili dugu, azian ikus daitetik bezala. Komplilazio erroreko sortzen badira zuzendu eta berriro kopilatu beharko dugu. Komplilazio prozesua zuren bukatzen denean, adibide karpetaren barnuan adibide .elf eta adibide.nds fitxategiak aurkituko ditugu.

```
/adibide/sources cd ..
```

```
/adibide/sources make
```

Orain bi kontsola irekiko ditugu. Lehengongo simuladorearen exekuzioa abiaraziko dugu hemen ikusten dugun bezala:

```
/adibide$ desmume-glade --arm9gdb=4444 adibide.nds
```

Bigarrenean aratzialdearen exekuzioa abiaraziko da azian ikusten den bezala. Adibide honetan bi aplikazioak komunikatzeko 4444 portua erabili da.

```
/adibide$ kdbg -r :4444 adibide.elf
```

## 1. Irudia. Kdbg-aren pantaila nagusia

1 irudiko kode lehoak gure programaren (main.c) iturru kodesa erakusten du. Guik idatzitako programaz gain, programaren lerro bakoitzeko (+ ikura sakarزان) kompliladoreak sortutako mihiztadura lengoaiko kodesa ikus dezakigu. Programaren gainean breakpoint-ak jar daitelk kik biloitza eginez breakpoint-a jartzea nahi dugun kode-lerroaren gainean (une horretan puntu gorri bat agertuko da kode-lerroaren ezkerrean, bertan breakpoint bat jari dugula adieraziz). Programaren exekuzioa zein puntuaren dagoen gezi berde batetik adierazten du, aginduen ezkerrean agertzen da hau ere. Programa exekutatzen has ditudin, breakpoint bat jari behar dugu aginduen batetan eta ondoren Execution | Run aukerau. Honen ondorioz, programa jarri dugun lehen breakpoint-erarte exekutatuko da.

Aldagai lokalen haloak ikusteko View | Locals lehoia daukagu. Stack lehoia ikusteko beriz View | Stack aukeratu behar da. Exekuzioa gelditzen denean kdbg-k prozesadoreen erregistroen edukia erakusten du erregistroen lehoan (View | Registers). Memoriaren edukia

# LABORATEGIAK

## Laborategiak

## Konputagailuen Egitura

### Sarrera

Dakizun bezala, NDS makinak 2 prozesadore ditu, ARM9 bat eta ARM7 bat. ARM9a da prozesadore nagusia eta programaren logika berak kontrolatzen du. ARM7a berritz, bigarren mailako prozesadorea da, eta batez ere soinua, harirk gabeko sarea (WiFi) eta teknika batzuen kudeaketaz arduratzan da. Bi prozesadore dituenet, bi exekutagarri sortu behar dira NDS-an, prozesadore bakoitzerako bat. ARM7-aren exekutagarriak ARM9-aren exekutagarriaren morroi bezala jokatzen du. Laborategian idatziko ditugun programa guztiek ARM9 prozesadorako dira.

NDS-tako kode exekutagarria sortzeko, lehenik eta behin objektu kodea sortu behar da. ARM7 eta ARM9-rako, horretarako konpiladore gunitzatu bat erabiliz (arm-eabi-gcc). Konpialdoreak sortzen duen objektu kodea estekatzaleak erabiliko du (arm-eabi-ld) prozesadore bakoitzerako fitxategi exekutagarri bat sortzeko. Fitxategi exekutagarriek, arazketarako informazioa gordetzen duen formatu estandar bat jarraitzen dute (.eif). Arm-eabi-obj copy tresnarekin bi exekutagarri muniztu sortutu dira azkenik nōtoo1 tresnaren bidez exekutagarri bakar batean elkartzen direnak (.nds fitxategi bat).

Laborategietan aritekta egin eta aztertzeko LINUX-erako ondoko tresnak erabiliko ditugut:

- DevKitPro: NDS-rako aplikazioak garatzeko liburutegi, konpiladore eta tresna multzo bat da. Beste gauza batzuen artean liburutegia dautu, gure NDS-aren hardwarea egokitzten laguntzen diena.
- Testu editoreak: gure programak idatzeko erabiliko dugu.
- DeSmuMe: NDS-aren simuladore edo emuladorea.

- Kdbg: (gdb): gdb programa araztaietako bere kdbg ingurune grafikoarekin.
- Kdbg: gure programak araztu eta aztertzeko

### make: programak konpilatzeko

Idazten dugun kodearen konpilazioa automatizatzeko (bai C-z idatzitakoarena eta bai mihizadura leongoian idatzitakoarena) make tresna erabiliko dugu. Tresna hau erabili ahal izateko Makefile izeneko fitxategi batean koden sortzeko exaztu behar diren fitxategien arteko elkarloturak eta komplazio arauak adierazi behar dira. Ikerkai honen helburuetatik kanpo geratzen da Makefile fitxategia sortzeko sintaxia zein den aztertzea, eta beraz, make erabiliko dugu dauzkagun Makefile fitxategien txantiloietatik abiatuz.

adibide karpetan gure praktikak egiteko erabiliko dugun txantilo bat sortuko dugu. devkitpro-arm-eabi (devkitARM) paketearekin batera programa txikien adibide multzo bat instalatuko da /opt/devkitPro/examples/nds direktorioan. Gogoratu NDS-ak bi prozesadore dituela eta guk, momentutz, bakanik ARM9-tako kodea sortuko dugula. Horregatik, erabiliko dugun txantilioa, /opt/devkitPro/examples/nds/templates/arm9 direktorioan dagoeneko abiatuta sortu da. Azpian ikusi dezakeugu nola kopiatu txantilio hori adibide izena duen karteta batetara.

```
$ cp -r /opt/devkitPro/examples/nds/templates/arm9 adibide
$ cd adibide
```

## Konputagailuen Egitura

Hasteke, adibide hau konpilatzen probatu dezakegu:

```
/adibide$ make
```

Makefile fitxategiari begirada bat emanda, ikusiko duzue ez dela batere erraza ulertzan hor dagoen kodesa. Makefile fitxategiaren helburua .nds lutzapena duen fitxategi exekutagarri bat sortzea da, aplikazioa kokatzen den karpetaren izen bera hartuko duena (adibide .nds). Ithurburu kodea source apzirektioan aurkitzen da. Ithurburu kodea mihizadura leongoian idatzita badago, kodea duen fitxategiaren lutzapena .s izan behar da, eta C-z idatzita badago berrixt, fitxategiaren lutzapena .c izango da. Objektu kodea duten fitxategiak, .o, build direktorioan sortzen dira eta direktorio nagusian sortuko den exekutagarri bakar batean konbinatzentzira .elf (adibide.elf). Exekutagarri hori (ARM9-aren exekutagarria da) lehenetsitako ARM7-aren exekutagarriarekin konbinatzen da fitxategi bakarra sortzeko, adibide .nds.

Programa berri bat idatzeari nahiz dugunean, adibide karpetako dugu, izena aldatuko digu eta gure programa idatziko dugu source karpetan dagoen main.c/main.s fitxategian.

### Testu editoreak

Bakoitzak nahi duen testu editorea erabili dezakeen arren, kontuan hartu behar da gedit editorea simplea dela erabilizteko. Sistemako kontsolatik abiarrazi dezakegu edo "Aplicaciones-Accesos" barruan aurkitu dezakegu. adibide karpetako ithurburu kodea duen fitxategia editatzako ondoko egingo dugu:

```
/adibide$ cd source
/adibide/sources/gedit main.c
```

### DeSmuMe: programak simulatzeko

Librea den NDS-aren simuladore bakarretako bat. Gainera bakarra da arazketarako (gdb-arekin) bide gainean bat integratzen duena.

DeSmuME funtzionaltasun desberdinak eskaintzen dituzten interfaze desberdinak exekutatu daiteke (glade edo cli adibidez). Adibidez, glade interfazeak S/Iko erregistroak eta memoriako erregistroa atitzeko aukera ematen du.

```
$ desmume prog.nds
$ desmume-cll prog.nds
$ desmume-glade prog.nds
```

### Kdbg: gure programak araztu eta aztertzeko

Kdbg front-end (ingurune grafiko) bat da gdb araztailerako. Beraz, kdbg ez da berez araztaile bat, eta bere lana ingurune grafikoko aginduak gdb-rr bidartzea da. gdb-a beriz GNU-ren programa araztaile aurerratu bat da, arkitektura, programazio lengoia eta exekutagarri formatu desberdinak lantzen dezkeena. Kontsolatik kdbg abiarrizko horrela egingo dugu:

Programazio Modulu  
2014/2015 ikasturtean

۲۰

baino lehen aurkituz gero

```

procedure Bukaeraako_Irakurri (F : in out Ada.Text_IO.File_Type;
                                 IM : out Matrize);

```

-- Sarriera: Fm, testu-fitxategia  
-- Irteera: LM, matxirea, non soluzioko balioko kokatu diren  
-- Ergina: fitxategirik gero zenhaki-matxize bat irakurzen du, eta  
balioko laukietan kokatzen ditu, hasierako baliogisa  
-- False dutela  
-- Salbuespenak: Errorea\_Irakurketan, lerro gutxiegi baldin baduude, edo  
lerron bat Jabetzegia bada; Zenhaki\_Desegokia, '1 .. '9 ..  
tartekoa ez den karaktereren bat aurkitzen bada

Honako eragiketa hauek izango ditu:

orrean dagoen laukia itzulzen du.  
function Laukia (M : in Matrize;  
E : in Bedarratziko\_Indizeak\_Berreankada\_Indize;

Z : in Bederatziko\_Indizeak.Zutabe\_Indize)  
return Laukiak.Lauki;

Laukia\_Kokatu: LM matrizea, E errenkada-zenbakia, Z zutabe-zenbakia eta L laukia

— ۲۷ —

procedur *Lauka\_Kontrata*  
(LM : in out Matrice;  
R : in Bederatziko Indizeak.Errenkada\_Indize;

**L** : in Laukiak.Lauki);

**Inkonsistentziarik\_Dago:** LM matrizea eta MA matrize-atal baten erreferentziak adierazten du emaitza boolean batetik

```
function Inkontsistenziarik_Dago  
  (IM : in Marize;_  
   MA : in Marize_Atalak.Atal)  
  return Boolean;
```

-- Sarreia: LM, lauki-matrirea;  
-- MA, matrize-atala  
-- Irteera: True, LM matrizezero MA atalean inkonsistenziarik  
baldin badago;  
-- False, bestela  
-- Ohaarra: Matrize-atal batean (errenkada, zutabe edo karratu) inkonsistentzia dago, zenbaki bat behin baino geniagotan agertzen baldin bada atal horretan

**Idatzia proba-programa, paketea ongi implementatuta dagoela egiazatzeko.** Eginda ematen zaiztu paketea probatzeko prozedura bat<sup>6</sup> eta bai prozedura horrek irakurriko diuen marizeek dazukan testu-fitxategia ere.

**Saiobespenak** Iau iuriagungun, uan...  
**Errorea\_Irakurketan:** Hasierakoa\_Irakurri eta Bukaerakoa\_Irakurri eragiketean altzarriz beharko da, Iero-amaierako marka bat edo fibategiaren amaitera behar.

<sup>5</sup> *inkonsistenziarik\_dago\_funtzioa.xt*  
<sup>6</sup> *probatu\_lauki\_matrizeak.adb*  
<sup>7</sup> *probarako\_matrizeak.txt*

<sup>8</sup> Hitzak paketea eginda ematen zaizu: *hitzak.ads* eta *hitzak.adt*.

**Entrigau beharreko oinarrizko laborategi-saio honetan ez da ezer igo behar eGelara, baina litekeena da egindako zerbait aurerago eskatzea.**

Matrizeak dauzkan datu-fitxategia.

## 1. laborategi-saioa

**Ada laborategien helburu orokorra**

- **Ikasturte honek Ado Laborategi-saioen atzen helburua sudokuak egiteko lagunza emango duen aplikazio bat egitea da. Gai horren inguruan landuko dira irakasgaiko hainbat aspeku: diseinua, modulartasuna, unitate generikoen erabilera eta errorearen tratamendua.**
- batik bat.

**Laborategi-saio honen helburua**

- **Modularizasunaren oinarritzko konzeptuak apilatzeara, Ado lengoia erabiliz. Paketeen erabilera lanzea, datu-mota abstraktuak implementatzeko. Salbuespenei dagokien aratuk, aurerrago landuko dira, beste saio hanean.**

**Laborategi-saio honetan egin beharrekoak**

Sudokuak egiteko lagunza emango duen aplikazioaren diseinuak hainbat modulu ditu. Horietako batzuk implementatuta emango zaizkizu; adibidez: Zenbakia, Hitzak, Bederatziko – Indizeak, Matrize\_Atalak... Beste asko,jakina, zeuk egin beharko dituzu.

Enunziatu honetan Laukiak, Lauki\_Matrizeak eta Jokalariak moduluak daude implemenatutako duena, Lauki motako elementuz osatutako matrize gisa. Matrize honen errenkadak eta zutabeak implementatuta ematen zaizun Bederatziko\_Indizeak paketeko Errenkada\_Indize eta Zutabe\_Indize motetakoak izango dira, hurrenez hurren. Pakete honak implementatuta ematen zaizun Matrize\_Atalak paketea<sup>3</sup> erabilizten du, zeinen helburua matrizeko 9 errenkadak, 9 zutabeak eta 9 karatatak (3\*3 laukioak) erreferentziatzea baita.

Honako eragiketa hauek izango diru paketeak:  
 1. **Idazi Lauki datu-nota abstraktuak implementatuko duen modulu (Laukiak paketea).**  
 Sudoko batzuk kokitzeko balontza adieraziko du DMA honek, eta horako hauek gordeko dira berian: laukia libre dagoen ala ez (egoera: libre edo beterik); balioa hasieratik emana izan den ala ez (hasierako balioa denetx: booleana); laukian dagoen balioa (Zenbakia. Zenbakia motako bat).

Honako eragiketa hauek izango diru paketeak:

**Hutsa\_Sortu:** Lauki motako objektu bat sortzen du, egoeratzat libre duela, eta hasierako balioa ez dela adierazten duela (hasierako balioa denetx: False).

**Procedure Hutsa\_Sortu (L : out Lauki);**

**Libre\_Dago:** Lauki motako objektu bat emanda, True itzultzen du haren egoera libre baldin bada, eta False bestela.  
 function Libre\_Dago (L : in Lauki) return Boolean;

**Hasierakoa\_Da:** Lauki motako objektu bat emanda, True itzultzen du haren balioa baldin bada, eta False bestela.

hasierako balioa baldin bada, eta False bestela.

**Zenbakia:** Lauki motako objektu bat emanda, bertako balioa (zenbakia) itzultzen du.

```
function Zenbakia (L : in Lauki) return Zenbakia; 
```

**Zenbakia\_Kokatu:** Lauki bat eta zenbakia bat emanda, zenbakia laukian kokatzen du, eta hasierako balioa hainbat kokatzen du. Laukiaaren egerra “beterik” izango da eragiketa exekutatu ondoren.

```
procedure Zenbakia_Kokatu (L : in out Lauki; N : in Zenbakia; Zenbakia); 
```

**Hasierako\_Zenbakia\_Kokatu:** Lauki bat eta zenbakia bat emanda, zenbakia laukian kokatzen du, eta hasierako balioa denetx adierazten den lekuaren balio egoikia (True) ezartzen du. Laukiaaren egoera “beterik” izango da eragiketa exekutatu ondoren.

**Zenbakia\_Kendu:** Lauki bat emanda, “libre” egoeran utzen du.

**Idati proba-programa, paketea ongi implementatuta dagoela egiazatzeko.**  
**Salbuespenak [aurerrago egindo da, utzi orangozo]:**

Zenbakirik\_Ez: Zenbakia eta Zenbakia\_Kendu eragiketetan altxarazi beharko da, laukian zenbakirik ez baldin badago.

2. **Idazi Lauki\_Matrizeak izeneko pakete bat, Matrize datu-mota abstraktua implementatuko duena, Lauki motako elementuz osatutako matrize gisa.** Matrize honen errenkadak eta zutabeak implementatuta ematen zaizun Bederatziko\_Indizeak paketeko Errenkada\_Indize eta Zutabe\_Indize motetakoak izango dira, hurrenez hurren. Pakete honak implementatuta ematen zaizun Matrize\_Atalak paketea<sup>3</sup> erabilizten du, zeinen helburua matrizeko 9 errenkadak, 9 zutabeak eta 9 karatatak (3\*3 laukioak) erreferentziatza baita.

Honako eragiketa hauek izango diru paketeak:  
**Hasierakoa\_Irakurri:** F testu-fixategia emanda, LM matriza eratzen du, non hasierako balioak kokatu diren, fixatxitgitik irakurria. Eragiketa honek fixatxitgitik 9\*9ko zenbakimatrize bat irakurten du, eta balioak laukietan kokatzen dira gisa kokatzen diru; hasieran matrizeko lauki libre gisa geratu behar dutenak ‘0’ batez errepresentatuko dira fibategian. Eragiketa honek implementatzeko, erreparratu Bukaerakoa\_Irakurri nola dagoen implementatuta, oso antzekoak bainitza (hala ere, ez hartu kontuan salbuespenei dagokiena, tratamendu hori aurrerago egingo baita).

```
procedure Hasierakoa_Irakurri (F : in out Ada.Text_IO.File_Type;
                                LM : out Matrize); 
```

**Bukaerakoa\_Irakurri:** F testu-fixategia emanda, LM matriza eraitzten du, non soluzioko balioak kokatu diren. Eragiketa honek fixatxitgitik 9\*9ko zenbakimatrize bat irakurten du, eta balioak laukietan kokatzen diru, hasierako balio gisa False dutela. Eragiketa hau implementatuta ematen zaiztu<sup>4</sup>, baina zeuk irakatu behar duzu dagokion paketean.

<sup>2</sup> Espezifikazioa egindako ematen zaiztu: bederatziko\_imlizeak.ads  
<sup>3</sup> matrize\_atalak.ads eta matrize\_atalak.adb  
<sup>4</sup> bukaerakoa\_irakurri\_prozedura.txt

<sup>1</sup> Zenbakia ikontzearren espezifikazioa egindako ematen zaiztu: zenbakia.ads

```

Void main()
{
    int min;
    int a=8, b=10
    min = minima(a, b)
}

```

```

int minima(int a, int b)
{
    if (a < b)
        return a
    else return b
}

```

Code 32  
global main - CPSR, - mask

```

word a : . word 8;
        b : . word 10;
        min : . word 0;

```

```

main()
{
    push{lr}
    ldr r1, a
    ldr r2, b
    push
    sub sp, sp, #4
    push{r3}
    push{r2}
    bl minima
    add SP, SP, #8
    str
    pop [r6]
    str r6 !min
    pop {lr pc}
}

```

minima:

push {r3, r4, r5, lr} Espera garde

ldr r3, [SP, #4] @r3, [SP, #16]

ldr r4, [SP, #8] ldr r4, [SP, #20]

mov r5, r4

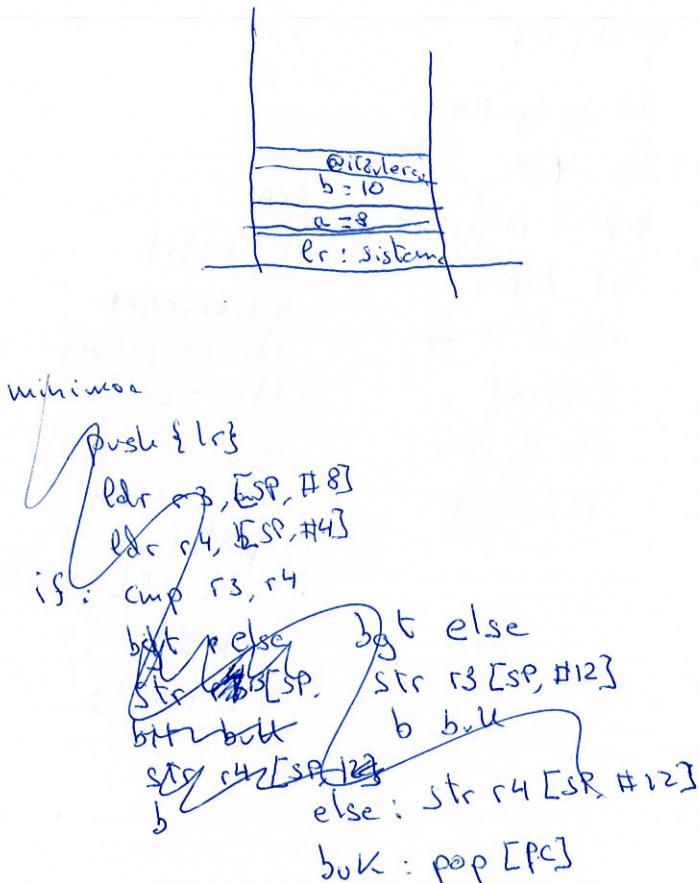
cmp r4, r3

blt trilia

move r5, r3

trilia: str r5, [SP, #12] str r5, [SP, #24]

pop {r3, r4, r5, pc} Espera bressuillatu

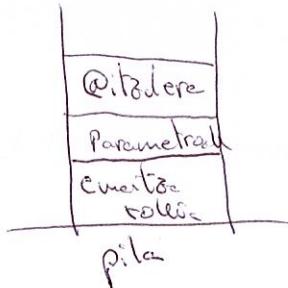


```

int A3P (int i)
{
    return (i*i)
}

Void main()
{
    int a=3
    int ema
    ema = A3P(a)
}

```



Code: 32

global main, -cpsr, -mask

a: .word 3

ema: .word 0

main:

push {lr}

Sub SP, SP, #4

ldr r3, a

push {r1}

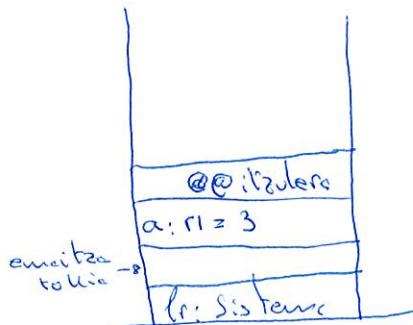
bl A3P

add SP, SP, #8

pop {r6}

str r6, ema

pop {pc}



A3P:

push {lr}

sub sp, sp, #8

ldr r3, [SP, #4]

ldr r4, [SP, #4]

mul r5, r4, r3

str r5, [SP, #8]

push f

str r5, [SP, #8]

bl A3P

pop {pc}





00/00/11/000

## 2. gaiko laborategiak

### Konputagailuen Egitura

2.gaiko laborategiak

### Konputagailuen Egitura

- e) Aldatu goiko programan B aldagaiaren balioa 47 balioa har dezan. Adierazi zein den kasu honetan exekutatzentzen den agindu sekuentzia eta agindu horretako bakoitzaren helbidea memorian. Horretarako bete ondorengo taula:

	<i>Agindua</i>	<i>Helbidea</i>
1	ldr r3, A	0x200038C
2	ldr r2, B	0x2000380
3	cmp r2, r3	0x2000394
4	bit etik1	0x2000398
5	mov r4, r3	0x200039C
..	beq r2	0x20003A0
	etik1: str r4	0x20003A8
	mov pc lr	0x20003AC

- a) Memoriako zein helbidetatik aurerra gorde dira programako aginduak? Zein da programako bigarren aginduaren helbidea? **0x200038C** *Bisaren aginduaren helbidea: 0x2000390*

b) Zein memoriako helbidetatik aurerra gorde dira programako aldagaiak? Jari helbide hori "Memory"

- pantailan programako aldagaien edukia ikusteko. Osatu ondoko taula:
- |          |            |            |            |
|----------|------------|------------|------------|
| aldataa  | A          | B          | E          |
| helbidea | 0x2000380  | 0x2000384  | 0x2000388  |
| edukia   | 0x00000012 | 0x00000005 | 0x00000000 |

- c) Ze balio hartzen dituzte r2, r3 eta r4 erregistroek?

**x2: 5      x3: 18      x4: 5**

- d) Programako agindu guziak exekutatzentzen dira? Azaldu zure erantzuna.

Eg. "mov r4, r3" eta "b etik2" Salto esinago ikar  
"cmp r2, r3" betzen dabillo.

Gure datuak definitzeraoan datu tamaina desberdinak erabilizten baditugu, hauek memoria ondo lerrokatuta daudela ziurtatu behar dugu. Horretanako dauskagu **.align** sasi-agindua.  
.align 1 jarzen badugu, hitz erdiko tamainaren mugan lerrokatuko da.

Datu tamaina desberdinaren atzipena eta memoria lerrokatzea aztertzeko ondoko programaren arazketa aztertuko dugu, desmume eta kdbg erabiliz.

```
.code 32
.global main, __cpsr_mask
main:
    ldrb r1, x1
    ldrb r4, x2
    ldrb r2, Y
    add r3, r1, r2
    str r3, z
    mov pc, lr
```

Erantzun ondoko galderak:

- Goiko programan zenbat byte okupatzentzu datu bakoitzak (x1, x2, Y, z) memorian? Irudikatu memoria datuak nola dauden gordeta azalduz.
- Egin ondoko aldagaiaren definizioa eta baita ldrb r4, x2 agindua. Probatu programa orain. Nola daude datuak orain gordeta? Zer geratzen da aginduekin?
- Orain b) ataleko datu definizioei gehitu .align sasi-agindua ondoan ikusten den bezala. Zer geratzen da orain programa arazean? Ze balio hartzen du r2 erregistroak lrdh r2, y agindua exekutatzean? Zergatik?

- ```
x1: .byte 5
y: .hword 6
.y: .align 1
z: .word 7
```
- Aldatu c) atalean jarritako datu definizioa azpian ikusten den bezala utziz. Ze balio hartzen du orain r2 erregistroak programaren exekuzioan? Irudikatu nola geratzen diren datu hauek memorian gordeta.
  - x1: .byte 5
.y: .align 1
y: .hword 6
z: .word 7

```

Void main()
{
    int a=5, b=6,
        truletu (&a, &b),
}

```

```

Void truletu (int *t1, int *t2)
{
    int aux,
        aux = *t1;
    *t1 = *t2;
    *t2 = aux,
}

```

Code 32

global main, -CPSR, mask

main:

push {lr}

~~str~~

ldr r0, &a

ldr r1, b

push {r0, r1}

bl truletu

add sp sp, #8

pop {pc}

truletu: .word 5  
.word 6

ldr push {lr}

ldr r6 [sp, #8]

~~str~~

ldr r7 [sp, #4]

~~ldr r8, [r7]~~

str r6, [r6]

str r8, [r8]

pop {pc}

|            |    |
|------------|----|
| lr: main   | #4 |
| @a : t1    | #8 |
| @b : t2    |    |
| lr: system |    |

r6 = @b

r7 = @a

r2 = b

r3 = 5

r6 → @b = 5

r7 → @a = 6

push {lr}

ldr r6 [sp, #8]

ldr r7 [sp, #4]

ldr r2, [r6]

ldr r3, [r7]

str r3, [r6]

str r2, [r7]

pop {pc}



28

| PC   | B             | P                         | EN(IR) | VAL    | MEM        | EM(ID)                    |    |
|------|---------------|---------------------------|--------|--------|------------|---------------------------|----|
| 1000 | mov r0 #0     | IR = M[1000]<br>PC = 1004 | Desk   | -      | mugtu(r0)  | -                         | r0 |
| 1004 | adr r1 B      | IR = M[1004]<br>PC = 1008 | Desk   | -      | mugtu(B)   | -                         | r1 |
| 1008 | Cmp r0 #10    | "                         | Desk   | r0     | r0 - 10    | -                         | -  |
| 1012 | beq B,B       | "                         | Desk   | -      | Komp?      | -                         | -  |
| 1016 | lsl r2,r0,H2  | "                         | Desk   | r0     | (r0, H2)   | -                         | r2 |
| 1020 | ldr r3[r1,r2] | "                         | Desk   | r1, r2 | r1 + r2    | irelli(r1+r2)<br>hebbidec | r3 |
| 1024 | comp r3,#0    | "                         | Desk   | r3     | r3 - #0    | -                         | -  |
| 1028 | blt B,B       | "                         | Desk   | -      | Komp?      | -                         | -  |
| 1032 | add r0,r0,#1  | "                         | Desk   | r0     | r0 + #1    | -                         | r0 |
| 1036 | b far         | "                         | Desk   | -      | PC + despl | -                         | -  |
| 1040 | str r3,N      | "                         | Desk   | r3     | regtu(r3)  | idate(r3)                 | PC |
| 1044 | mov,pc,lr     |                           |        | lr     | mugtu(lr)  | mugtu(lr)                 | PC |

## Konzeptgaben eastura

2.8

- a)  $N := -1$  zwingt da.
- b) 3 add's exklusive der 4. adden Billens Salto erginge kein
- c) belltorelo alleigen hebbaren sequenzelle.
- d) programme konell erkennen da sein den belltorelo leben symboli ueberlappen.

2.9

code 32

global main, -cpsr, -mask

B1: .word .... f5

X: .word 15

main:

adr  
mov r0, #0

adr r1, B1

mov r2, X

For: cmp r0, #15

beq b1ke ob if

Comp

lsl r3, r0 #2

ldr r4 [r1, r3]

Comp r4, r2

beg buk

add r0, r0, #1

b for

Bellinstr

st r5

if : str p<sub>ss</sub>, # -1

b finish lsr p<sub>ss</sub>, r0

Bulk: str p<sub>ss</sub>, r0

finish: ldp pc

mov pc, lr



**Konputagailuen Egitura / Estructura de Computadores**  
Curso 2013/2014 ikasturtea

**Azterketa eredua / Examen Modelo A**

**Informatika Fakultatea**  
Facultad de Informática

**Konputagailuen Arquitectura eta Teknologia Saila**  
Departamento de Arquitectura y Tecnología de Computadores

2014ko martxaren 18a / 18 de marzo de 2014

**1. (puntu 1)**

Ondorengo helbideak byteria helbideratzan den eta 4 byte-tako hitzak dituen memoria batia dagozkio.

0x230  
0x82E

Erantzun ondorengo galderei:

- Zein da memoriaren tamaina?
- Memoriako zein hitzi eta hitzaren barruko zein byte-ri dagokio helbide bakoitza?

**2. (2,5 puntu)**

ARM makinen mihiztadura lenhoaian idatz ezazu ondorengo C-z idatzitako programa gauzatzetan duen kodea.

```
main ()
{
    int x = 40;
    int y = -4
    while (y < x)
    {
        if (y < 0) y = y * y;
    }
}
```

**3. (3,5 puntu)**

- AZP1 azpierrutinaren zehaztapenak akats bat du. Adieraz ezazu zein den.
- ARM makinen mihiztadura lenhoaian idatz ezazu AZP1 azpierrutinari dagokion eta C lenhoaiaz idatzita dagoen kodea.
- AZP1 azpierrutinari dagokion aktibazio blokea erakusten duen pila marraz ezazu.
- AZP1 azpierrutinaren pop {pc} aginduak ze balio kargatuko du PC erregistroan?
- Aldar ezazu AZP1 azpierrutina AZP2 azpierrutinari parametroak pasa diazakion push agindua erabilii gabe.
- Zein da programa nagusien add sp, sp, #12 aginduaren eginkizuna?

```
.code 32
.global main, __cprsr_mask
A: .word 0, 42, 21, 71, 0;
B: .word 3;
C: .word 0, 0, 0, 0, 0, 0;

main:
@000000DC:    push {lr}
                a2 r7, A
                ldr r8, B
                a2 r9, C
                push {r7, r8, r9}
                bl AZP1
                add sp, sp, #12
                pop {pc}
```

## Akkubazio Blokearen Uztedaketa

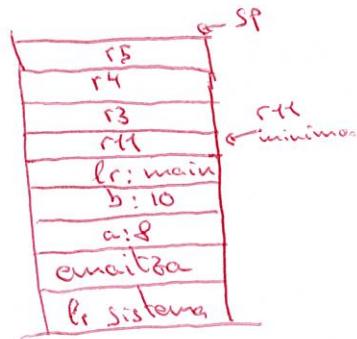
beste eeregistrok bat:  $r11 \equiv (\text{FP} \equiv \text{frame pointer})$

| eeregistroak |            |
|--------------|------------|
| r11          | onelli Azp |
| @.tzelera    |            |
| Param        |            |
| emaitza      |            |

minimoa:

```

push {r11, lr}
mov r11, sp
push {r3, r4, r5}
ldr r3, [r11, #8]
ldr r4, [r11, #12]
mov r5, r4
cmp r4, r3
blt txikia
mov r5, r3
txikia: str r5, [r11, #16]
    
```



## 2.17 arrilera

Void main()

```

{ int Batu(int *Bew)
    int Batuera;
    Batuera = Batu(B);
    
```

int Batu (int \*Bew)

```

{ int Bat = 0
    int c = 0,
        for (i=0, i<10, i++)
            Bat = bat + Bew[i];
    return bat;
    } 
```

Code 32:

global main; -cper, -mabi

B: .word 3,4,-2,8,9,10

Batuera: .word 0;

main:

```

    push lr
    adr r1, Bat
    ldr r3, Batura
    push {r1, r3}
    bl Batu
    
```

Batu:

```

    push {r1, r2}
    mov r11, sp
    push {r4-r10}
    ldr r12, Bat
    mov r0, #6
    mov r1, #0
    for: cmp r5, #0 r0, #
        beq Batuera
        add r5, r5, r6
        add r3, r3, #1
        ldr r3, [r1, #12]
        ldr r6, [r3, r0, lsl #2]
        add r5, r5, r6
        add r3, r3, #1
        b for
        beq Batuera, str {r5, r6}, [r4-r10, lr]
    
```



**2.17.** Idatzi C-z eta ARM mihiztadura lengoaien azpirrutina bat parametro bezala bektore bat jasotzen duena eta bektoreko osagai guztien batura itzultzen duena. Idatzi ere programa nagusi bat azpirrutina horri deia egiten diona. Bektoreak 10 osagai izango ditu.

**2.18.** C-z idatxitako azpirrutina ARM mihiztadura-lengoiaza idatz ezazu. Fakt azpirutinari dei egiten dion programa idatz ezazu mihiztadura-lengoiaza.

```
int fakt (int i)
{
    if (i==1) return(1);
    else return (fakt (i-1)*i);
}
```

**2.19. Idatzi ezazu ARM mihiztadura-lengoiaza, C-z idatxitako ondorengoko programa.**

```
int kalkulatu (int x, int *y)
{
    int z;
    z = x + (*y);
    (*y) = z / 2;
    return (2*z);
}
```

**2.20. C-z idatxitako programa nagusi bat eta ARM mihiztadura lengoaien idatxitako AZP azpirrutina emanik:**

- C-z dagoen programa nagusia kode ezazu ARM mihiztadura lengoaien.
- Exekuzio osoari dagokion pila marraz ezazu.

```
void main()
{
    int A[8] = {15,12,7,-6,-42,21,-3,-4};
    int n=0;
    for (i=1; i<8;i++)
        if (A[i]>A[i+1])
            AZP (A[i], &A[i+1]);
    n++;
}
printf ("%d",n);
```

## 2.22.

- o C lengoaiaz idatzita dagoen azpirrutina, ARM mihiztadura lengoaiaz kodetu ezazu.
- o Exekuzio osoa erakusten duen pila marraz ezazu, pilako sarrera eta irteera bakoitzaren sein agindurean ondorioz geritzen den adieraziz.
- o Programa osaren eginkizuna sein den adieraz ezazu. Azaldu zein eragiketa ebatzeko erabili ahal izango den, eta ez leroz lerro zer egiten duen.
- o 12. agindutik igarotzean r1 erregistroan r1 erregistroan?
- o 12. agindutik kargatu da r4 erregistroan?
- o Azaldu zer egiten duen ondorengoko aginduak: str r6,[r3].

**2.21. C-z idatxitako programa nagusi bat eta ARM mihiztadura lengoaien idatxitako AZP azpirrutina emanik:**

1.- C-z dagoen programa nagusia kode ezazu ARM mihiztadura lengoaien.

2.- Exekuzio osoari dagokion pila marraz ezazu.

3.- Saitez ondoren agertzen diren galderet erantzuna ematen.

a.). Zertarako erabilten dira r1, r2 eta r3 erregistroak?

b.). AZPI1 azpirrutina, funtzioa ala prozedura da? Justifikatu zure erantzuna.

c.). Zer aldatuko litzateke r11 erregistroa erabiliko ez balitz?

|                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                    |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>void main() {     int b[6]={7,-8,-42,21,-71,0};     int n=6, i, res[6];     AZP1(b, n, res);     for (i=0; i&lt;n; i++)         printf("%d", res[i]); }</pre> | <pre>AZP1: push {r11, lr} mov r11, sp push {r2, r3,r4,r5,r6,r7} ldr r2, [r11,#16] mov r1, #0 ldr r3, [r11,#12] cmp r1,r3 beq BUK ldr r5, [r2,r4,r11,#2] cmp r5, #0 BKF IF mov r6, #0 B BUKIF sub sp, sp, #4 push {r5} bl AZP2 add sp, sp, #4 pop r6 BUKF:ldr r7, [r11,#8] str r6, [r7,r1,1s1#2] add r1, r1, #1 B FOR BUK: pop {r3-r7,r11,pc}</pre> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

1: .code 32
2: .global main, __cprt_mask
3: main:
4:     push {lr}
5:     mov r1, #0
6:     ldr r2, n
7:     add r7, A
8:     add r8, B
9:     add r9, C
10:    for: cmp r1, r2
11:        bge bfor
12:        ldr r4, [r7, r1, lsl#2]
13:        ldr r5, [r8, r1, lsl#2]
14:        mul r6, r4, r5
15:        str r6, [r9, r1, lsl#2]
16:        add r1, r1, #1
17:    b for
18: bfor: sub sp, sp, #4
19:     ldr r10, m
20:     push {r10, r2, r9}
21:     bl AZPL
22:     add sp, sp, #12
23:     pop r3
24:     str r3, ema
25:     pop {pc}

```

2. 23. ARM mihiztadura engoainan idatzitako ondoko programa daukagu;

```

.a: .word 5
b: .word 6

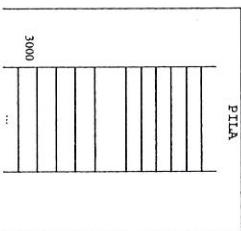
main:
    push {lr}
    adr r1, a
    adr r0, b
    push {r0, r1}
    bl TRUKATU
    add sp, sp, #8
    pop {pc}

TRUKATU:
    push {lr}
    ldr r3, [sp, #4]
    ldr r5, [r3]
    ldr r4, [sp, #8]
    ldr r6, [r4]
    str r5, [r4]
    str r6, [r3]
    pop {pc}

```

Programa hau memoriako 1000 helbidek aurerrera gorde da dena jarraian. Bete eza zu exekutatzuen diren lehenengo 7 aginduen exekuzio traza, kontuan izanik 1r (r14) erregistroaren hasierako balioa 100 dela. Irudikatu plia nola denetzen den agindu hauren exekuzioaren bukaera, pilako helbideak eta edukia adierazit, Jakinda sp (r13) erregistroaren hasierako balioa 3000 dela. Atzizten den erregistro eta memoriako helbide bakotzeeko adierazi irakurketa edo idatzeta bat egiteko den.

1000 push {R} R [ ] ... R [ ] @M: EM:



a) Adierazi hola aldatzen den memoraren eta erregistroen edukia goiko agindu

bakoitzaren exekuzioarekin.  
b) Agindu bakoitzoan exekuzioan ematen diren pausoak eta pauso bakoitzean burutzenean diren ekintzak adierazi.

```
        .l: .long -  
b: .word 6  
main:  
    push {lr}
```

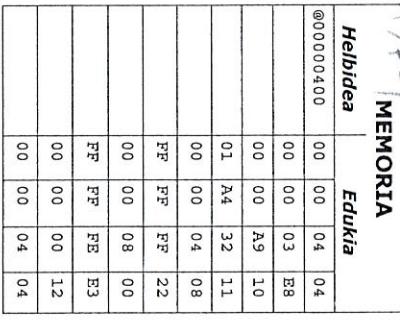
```

add sp, sp, #8
pop {pc}
TRUKATU:
push {lr}
ldr r3, [sp, #8]
ldr r5, [r3]
ldr r4, [sp, #8]
ldr r6, [r4]
str r5, [r4]
str r6, [r5]
pop {pc}

```

Programa hau memoriako 1000 helbidek aurerrera gorde da dena jarraian. Bete eza zu exekutatzuen diren lehenengo 7 aginduen exekuzio traza, kontuan izanik 1r (r14) erregistroaren hasierako balioa 100 dela. Irudikatu plia nola denetzen den agindu hauren exekuzioaren bukaera, pilako helbideak eta edukia adierazit, Jakinda sp (r13) erregistroaren hasierako balioa 3000 dela. Atzizten den erregistro eta memoriako helbide bakotzeeko adierazi irakurketa edo idatzeta bat egiteko den.

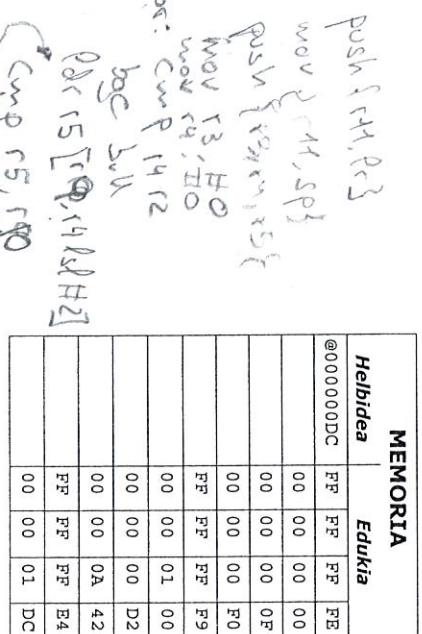
1000 push {R} R [ ] ... R [ ] @M: EM:



**25.** Ondoko taulian, oxu0000400 heibideetik hastea, AKM makinaren memoriako 10 hitzen edukia da. Horretaz gain, makinaren hainbat erregistroren edukia ere ikus daiteke.

Programa hau memoriako 1000 helbidek aurerrera gorde da dena jarraian. Bete ezazu exekutatzenean lehengero 7 agindunen exekuzio traza, kontuan izanik 1x14 (r14) erregistroaren hasierako balioa 100 dela. Irudikatu plia nola denetan den agindu hauren exekuzioaren bukaera, pilako helbideak eta edukiatik adierazit, Jakinda sp (r13) erregistroaren hasierako balioa 3000 dela. Atzizten den erregistro eta memoriako helbide bakotzeeko adierazi irakurketa edo idatzeta bat egiteko den.

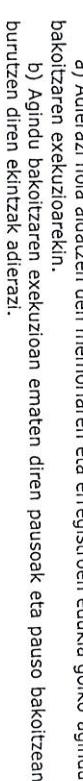
1000 push {R} R [ ] ... R [ ] @M: EM:



a) Adieraz, nola aidatzet den memoriaaren eta erregistroen edukia goik aginatu bakoitzaren exekuzoarekin.  
b) Agindutako bakoitzaren exekuzioan ematen diren pausoak eta pauso bakoitzean urutzen diren ekintzak adierazi.

Programa hau memoriako 1000 helbidek aurerrera gorde da dena jarraian. Bete ezazu exekutatzenean lehengero 7 aginduen exekuzio traza, kontuan izanik 1x14 (r14) erregistroaren hasierako balioa 100 dela. Irudikatu plia nola denetan den agindu hauren exekuzioaren bukaera, pilako helbideak eta edukia adierazit, Jakinda sp (r13) erregistroaren hasierako balioa 3000 dela. Atzizten den erregistro eta memoriako helbide bakotzeeko adierazi irakurketa edo idatzeta bat egiteko den.

1000 push {R} R [ ] ... R [ ] @M: EM:



|  |    |    |    |    |                       |
|--|----|----|----|----|-----------------------|
|  |    |    |    |    | ... . . . . ,         |
|  |    |    |    |    | .. . . . ,            |
|  | 00 | 00 | 00 | 12 | 5. str r2, [r5, #-4]! |
|  | 00 | 00 | 04 | 04 | 6. 1dr r6, [r2, r4]!  |

**2.24.** Ondoko taulian, 0x000000DC helbidetik hasita, ARM makinaren memoriako 10 hitzen edukaketa erakusten da. Horretaz gain, makinaren hainbat erregistratu edukia ere ikus daiteke.

## 2. Gaia: ariketak

**2. 26.** ARM makina batean 0x00000080 helbidetik hasita memorian ondoko datuak gordeko dira:

x: .word 8

v: word 4, 5, 6

卷之三

z : .WORU U\*34212345

Adierazi nola geratuko den memoriaren edukia datu horiek gordetzean. Adierazi nola aldazten den memoriaren eta erregistroen edukia ondoko aginduak exekutatu direnean.

**2.27.** Ondoren ARM makinenaren mihiztadura lengoain idatzitza dauden programa batzen eta azpiroturra baten zati bat erakusten dira. Gainera, memoriako hainbat nositzietako edukia ere anteriorean dagoenak.

|      |            | DIRECCIÓN        | MEMORIA    | CONTENIDO   |
|------|------------|------------------|------------|-------------|
| main | 0x200003A0 | push {lr}        |            |             |
|      |            | ldr r3, X        |            |             |
|      |            | cmp r3, #10      |            |             |
|      |            | beq b1,k         |            |             |
|      |            | sub sp, sp, #4   | 0xb003C0A4 |             |
|      |            | push {r3}        |            |             |
|      |            | b1 A2P           | 0xb003C0A8 |             |
|      |            | add sp, sp, #4   | 0xb003C0AC | 00 00 07 45 |
|      |            | ldr r2, [sp], #4 | 0xb003C0B0 | 10 00 01 88 |
|      |            | str r2, ema      | 0xb003C0B4 | 20 00 03 BC |
|      |            | pop {pc}         |            | 00 00 00 17 |
|      |            |                  | 0xb003C0B8 | FF FF FF    |
|      |            |                  |            |             |
| A2P: | 0x20000424 | push {r11,lr}    |            |             |
|      |            | mov r11,sp       |            |             |
|      |            | push {r1}        |            |             |
|      |            | ldr r1, [r11,    |            |             |
|      |            | add r1, r1, #5   |            |             |
|      |            | str r1, [r11,    |            |             |
|      | #8]        | pop {r1,r11,pc}  |            |             |
|      | #12]       |                  |            |             |

**2. 26.** ARM makina batean 0x00000080 helbidetik hasita memorian ondoko datuak gordeko dira:

卷之二

xii

Y: -word 4, 5, 6

**z:** .word 0x34212345

Adierazi nola geratuko den memoriaren edukia datu horiek gordetzean. Adierazitako memoria eta erregistroen edukia ondoko aginduak exekutatzeko direnean.

Programa nagusienaren aginduak 0x200003A0 helbidekit aurerrera gordetzen dira eta azpierrutina 0x200003C4 helbidekit aurerrera. Unean, erregistro batzuen balioa ondoren adierazitakoa da:

**R1** = 0x00000001C      R11 = 0xb003C0AC  
**R13 (SP)** = 0xb003C0A8      R15 (PC) = 0x200004D8

Duzun informazioak abiatuta, exekutatuko diren hurrengo lau agindu egitarazte traza (zer geratzen den) bere ezazti ondokoan adieraziz.

**2. 28.** Ondoren ARM makinen mihiztadura lengoaian idatzia dauden programa baten eta azpierutina baten zati bat erakusten dira. Gainera, memoriako hainbat posiziotako edukia ere ageri da.

|            | MEMORIA   | EDU <sup>K</sup> IA |
|------------|-----------|---------------------|
| ...        |           |                     |
| main:      | push {lr} | HELBIDEA            |
| 0x200003A0 | push {r1} |                     |

```
        cmp    r3, #10
        breq   buk
        sub    sp, #4
        ldmia  sp!, {r0-r3, r14}
        bx     r14
```

|      |          |                |            |    |    |    |    |
|------|----------|----------------|------------|----|----|----|----|
| buk: | b1 A2P   | add sp, sp, #4 | 0xb003C0A8 | 00 | 00 | 07 | 45 |
|      | pop {r2} | str r2, ema    | 0xb003C0AC | 00 | 00 | 01 | 88 |
|      | pop {pc} |                | 0xb003C0B0 | 00 | 00 | A1 | F4 |
|      |          |                | 0xb003C0B4 | 00 | 00 | 00 | 1F |

|             |                     |            |    |    |    |
|-------------|---------------------|------------|----|----|----|
| A2P:        | push {r11,lr}       | 0xb003C0B8 | E5 | FF | FF |
| 0x2000003DC | mov r11,sp          |            |    |    |    |
|             | push r11            |            |    |    |    |
|             | ldr r11, [r11, #8]  |            |    |    |    |
|             | add r11, r11, #5    |            |    |    |    |
|             | str r11, [r11, #12] |            |    |    |    |

Programa nagusiaren aginduak **0x200003A0** helbibetik aurerrera gordetzen dira eta azpierrutina **0x200003DC** helbibetik aurerrera. Erekuzioaren une zehatz batzen

*R11 = 0x20000A001*

Duzun informazioetik abiatuta, exekutatuko diren **hurrengo Hiru aginduen** egikarizte traza (zer geratzen den) bete ezzazu, ondokoak adieraziz:

| <b>Memoria</b>                                                  | <b>Erregistroak</b>                                                                                             | <b>Agindua</b>          | <b>Heibidea</b> |
|-----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|-------------------------|-----------------|
| (adierazitzean eta<br>edukia, memoria atitzten<br>den kasuetan) | (atitzentz direnaren,<br>adierazitzean irakurkun<br>dituen edo bertan idazten duen, eta<br>dgoozkiien baliozak) | (ezkutatzeko<br>denean) | (aginduarena)   |



(1)

0x230

bytere helbideretv eto 4 byte lito hitec

a) Sein da Memoria tecina?

12 by 2<sup>12</sup>

b)

0010 0011 0000  
2<sup>3</sup> 2<sup>2</sup> 2<sup>0</sup>

40 byte hitz berrea  
140 hitzari.

b) @1000 0010 1110  
512 + 11 = 523 hitz.

26aic

(2.1)

a)

ldr r5 A → r5 = 7

mov r1, #6 → r1 = 6

add r5, r5 r1 → r5 = 13

str r5, B → B = 13

(2.2)

code 32

global main -- CPSF, -mesk

n: .word 5

fall: .word 1

main:

|                 |                    |
|-----------------|--------------------|
| ldr r0, n       | mul r2, r2, r1     |
| ldr r2, fall    | b for              |
| if: cmp r0, 0   | fall: str r2, fall |
| btl             | b fall             |
| for: mov r1, #2 | then: str #1, fall |
| cmp r0, 1       | bul, mov pc, lr    |
| btl             | loop               |

main:

global

Code 32

global main, -cpsr, -msh

X: .word 40;

Y: .word ~4;

main:

ldr d, X

ldr r2, Y

while: cmp r2, #1

bge bkh

if: r2 cmp, r2, #0

bge else

mul r2, r2, r2

b while

else: add r2, r2, #1

b while

bkh: mov pc

str r2, Y

mov pc, lr

④

R1 = 0x 0000001C R11 = 0x b003C0AC

R13 (SP) = 0x b003C0A8 R15 (PC) = 200003D8

Register

0x200003D8

Agindur

0x200003DC

Str r1, [R11, #12]

0x200003Bc

pop [R1, R11, PC]

add sp sp #4

Erregister

Indurrí:

r1 = 0x 0000001C

r4 = 0xb003C0AC

Indurrí:

sp = 0xb003C0A8

Indurrí:

sp = 0xb003C0AC

Memoria

Indurrí:

M[0xb003C0B8] = 0x 0000001C

ldr r2

0x 200003E0 ldr r2, [sp], #4

Indurrí:

sp = 0x b003C0AC

Indurrí:

sp = 0x b003C0B8

00 00 00 1C

r2 = 0x 400001ff

①

KK byteles memoria etc 32 bit Adress-Bus, so Hauptges. b. Hitze 4 byte  
 Hebbideordne unitate -> byte  
 $\rightarrow \text{Hitze}$

a) Hebbide attive?

a) Mem-tacarea  $\rightarrow 16 \cdot 2^{10} = 2^4 \cdot 2^{10} = 2^{14}$   
 Hebbide attive  $\rightarrow 2^{14} - 1$

2)  $2^{14} - 1$

b)

$2^{14} \rightarrow$  hebbideordne unitate byte  $\Rightarrow n=14$  bit

$\rightarrow \text{Hitze} \rightarrow \frac{4 \text{ byte}}{2^2} \cdot 2^{14} = 2^{12} \Rightarrow n=12$  bit

c) atoimen bit.

4 atoimen

②

4 byteles hitze, edukiere 1Mbyte =  $2^{20}$  bit etc 4 byteles hebbide-Bus  
 etc 32 bit Adress-Bus, atoimen denbare loops etc undichtasen 200 MHz

a)  $\frac{250 \text{ byte}}{\frac{4 \text{ byte}}{5 \text{ ns}}} = 281 \text{ atoimen}$  63 atoimen, atoimen denbare, loops 63 ms

$f = 200 \text{ MHz} \Rightarrow f = \frac{1}{T} \Rightarrow T = \frac{1}{f} = 5 \text{ ns}$

$\frac{63 \cdot 100 \text{ ns}}{5 \text{ ns}} = 1260 \text{ z. Mb}$

(1)

a) 4Mbytes memoria  $\rightarrow$  4bytes geladen.

$4 \cdot 2^{20}$  byte  $\rightarrow$  4bytes geladen

Berech,  $2^{20}$  geladen da.

b) Leibide handien  $2^{20}$  bytes da

c) Leibide bus 20 bitellen da.

(2)

2bytes geladen etc 8bit-eles Leibide bus.

a)  $2 \cdot 2^8$  byte =  $2^9$  bytes

b)  $2^9$  bytes eddliere  $\Rightarrow \frac{2^9}{2} = 2^8$

(3)

4096 bits  $\Rightarrow$   $2^{4096}$  bytes =  $2^{13}$  bytes

a) Berech, memoria tamain bytes =  $2^{13}$  bytes da.

b)  $2^{13}-1$  geladen

c) 13 bit-eles

d)  $2^{13}$  da Leibide handien.

(4)

Adress 2 byte, Leibideretze-unitdee = Hitzee, Leibide-bus 15 bit

data-bus 16 bit,

a) Mem tracie  $\Rightarrow 2 \cdot 2^{15} = 2^{16}$

Dateneh =  $\frac{2^{16}}{2^2} = 2^{14}$  daten

b) Zatzipan

Code 32

global -e psr, -m ask  
A: .word 2,-8,3,-5,1;  
EMA: .word 0;

main:

ldr r0, A  
adr r1, EMA  
mov r2, #0  
for:  
    cmp r2, #5  
    bge bulk  
    ldr r3 [r0, r2 lsl #2]  
    if:  
        cmp r3 #0  
        blt ~~bulk~~ endif  
        sub sp sp #4  
        push {r3, r1}  
        bl A2p1  
        add sp, sp, #8  
        ~~move~~ pop {r2}  
        str r2, [r1]  
    endif:  
        add r2, r2, #1  
    bulk:  
        pop mask  
        pop {pc}



word -1

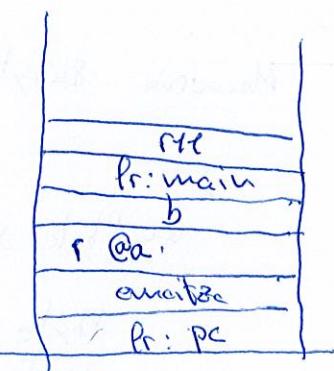
S: push lr  
ldr r3, C  
ldr r0, b

if: qdr r1, a  
ldr r2, [r2]  
cmp r2, r0  
bge brk  
mvn r3, r0  
brk: str r3, C

est. 45

⇒

push {r11, lr}  
mov sp, r11, sp  
push {r3, r2, r1, r0}  
ldr r0 [r11, #8]  
ldr r1 [r11, #12]  
ldr r4 [r11]  
if: cmp r4, r0  
ble brk  
mvn r3, r0  
brk: str r3 [r11, #16]



①

Wörde 4 Byte  
Habilderatze-unitate Byte  
Habide-buse 16 bit  
datu-bus 16 bit

a)

Sein da memorieren edukiere!

$$2^{16} \text{ byte} \approx 2^4 \cdot K \text{ byte}$$

b)

$$\frac{2^{16} \text{ byte}}{2^2} = 2^{14} \text{ byte}$$

$2^{16}$  - 1 position

c)

0101 0011 0101 0000      0 bytes

(1)

Memory 8Kbytes, halfword 12 bit data-bus 2 bytes

a)

one byte data  $\rightarrow$  2 byteello data-bus

$$\frac{8 \text{ bytes}}{2 \text{ bytes}} = 4$$

b)

$$8Kbytes \rightarrow 2^3 \cdot 2^{10} = 2^{13}$$

$$2^{12} \rightarrow \frac{2^{13}}{2^{12}} = 2 \rightarrow \text{halfword unit}$$

c)

$$2^{12} \text{ position}$$

d)

$$2 \text{ bytes}$$

(2)

~~global~~

code 32

global var, -cpsr, -main;

A: .word 3, 4, 2, -1, 5, 6, R0, 12, 3, 2;

B: .word 12, 14, 2, 43, 12, 67, 1, 1, 1;

C: .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;

main:

```

    ldr r0, EA
    ldr r1, B
    add r2, c
    mov r3, R0
    for: cmp r3, #10
          bge b1
    ldr r4 [r3, lsl #2]
    ldr r5 [r1, r3, lsl #2]
    cmp r4 r5
    bge else

```

```

    sub r6, r5, r4
    str r6, [r2, r3 lsl #0]
    a b for, r3, #1
    else: sub r7, r2, r3 lsl #0
    add r8, r3, #1
    for
    b1: mov pc, lr

```

d) 000000F4 -> change into pc-all.

e)

str r<sub>1</sub> [sp, #-4]

str r<sub>2</sub> [sp, #4]

f)

parametric calculation

①

0x230 → 12 bit

0x82E

a) Sein da memorieren termino?

$$\text{fam. mem} = 2^{12} \text{ bytes} = 4 \text{ Kbytes}$$

b)

0000 0011 0000

$$2^2 + 2^3 + 2^7 = 140 \quad \text{bytes}$$

140 bytes etc 0 bytes bytes bewahren

②

main()

{

main:

ldr r1, x

ldr r2, y

while: cmp r<sub>2</sub>, r<sub>1</sub>  
bge b1L

cmp r<sub>2</sub>, #0

bge else

else, a fewl b while r<sub>2</sub>, r<sub>2</sub>, r<sub>2</sub>  
else, add r<sub>2</sub> r<sub>2</sub> #1

b while

b1L: mov pc lr

③

a)

Azp1 aspiratziaren espedifizioen, hor da, int Azp1 (int 'a', 'b', 'c') egiten da, baina zitzen, esan nahi duen a, b eta c-ren helbideak ditzakete, baina b-ren klasen ez ditzakoa helbidea baitikoa dela, programea negoziazioa baliola hartzan dagozale. Eta prozedura bat da eta ez funtzioa.

b)

push {r11, lr}

mov r11, sp

push {r4, r2, r6, r4, r3, r2, r1}

adr r1, [r11, #8]

ldr r2, [r11, #12]

mov r3, #0 ldr r3, [r11, #16]

bnez r2, l1

For: cmp r0, #6

bge l0

ldr r4, [r11, r0 lsl #2]

ldr r5, [r5, r0 lsl #2]

~~push {r4, r2, r5}~~ sub sp, sp #4

push {r2, r4, r5}

bl Azp2

~~pop {r2}~~

add sp, sp #8

pop {r6}

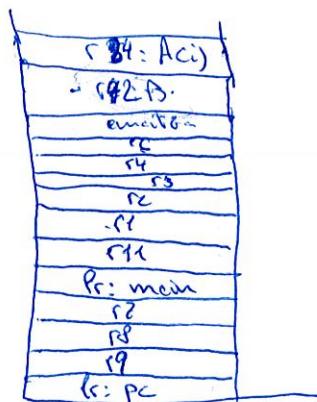
~~push {r2, r4, r5}~~

str r6, [r3, r0 lsl #2]

r0 add r0, r0, #1

bl For

l0: pop {r6, r4, r3, r2, r1, r11, pc}



# Agiindu Multzoa

Multzoa lengoaia eta multzoa lengoaia

mikroinstrukzioak  
 add r3, r1, r2 → 111000001000000100110000000000010  
 mikroinstrukzioa lengoaia      ↑ Kompilezorak  
 A := b + c

$$a = b + c$$

multzoa lengoaia

a)

add, a, b, c

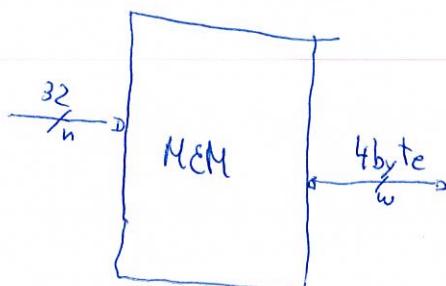
↓

Cisc

b) ldr r1, b  
 ldr r2, c  
 add r3, r1, r2  
 str r3, a  
 ↓  
 Risc

ARM

32 biteko Risc multzoa



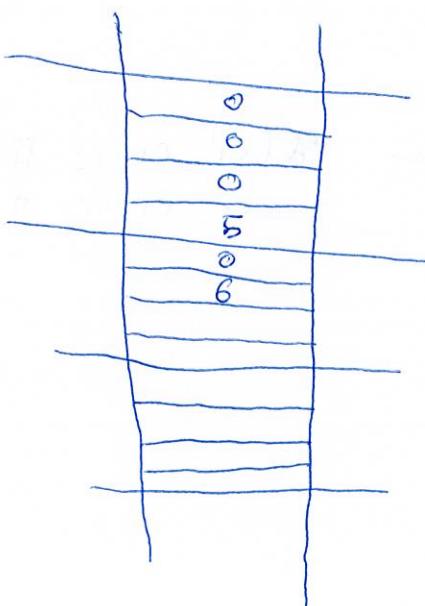
Heldibideetako-unitateen byteak  
 Hitza: 4 byte  
 Datu-tamainuak  
 ↗ 2 byte (half-word)  
 ↗ 4 byte (word)

A: word 5

B: half word 6

C: word 4

D: byte 7



@Adibidce

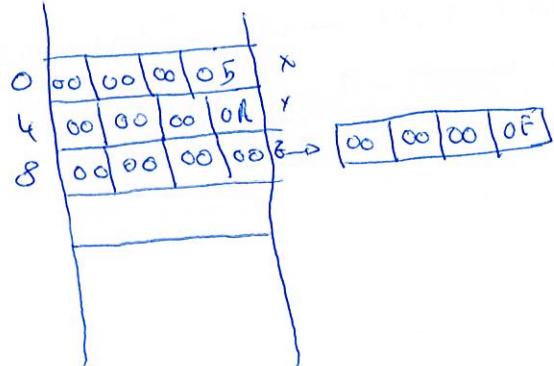
Code: 32

global main, — Cpsr, mask

X: . word 5

Y: . word 10

Z: . word 0



main

ldr r2, x → r2 ← M[2] r2=5

ldr r3, y → r3 ← M[3] r3=10

add r5, r2, r3 → r5 = r2 + r3 = 15

str r5, z → M[2] ← 15

mov pc, lr

Arithmetic

ldr rh, @mem

str rh, @mem

add, sub, mul, rh, ri, er2

adds, subs, muls rh, ri, er2

and, eor, or rh, ri, er2

ands, eors, ors rh, ri, er2

mov rh, er2

movs rh, er2

lsl rh ri er2 →

asr rh ri er2 →

lsr rh ri er2

cmp rn, er2

lsl r1, r2 #1 r1 ← r2 < 1

asr r1 r2 #1 r1 ← r2 > 1

# Ajindo mult3cc Arithm

(2.1)

A: .word ?

B: .word -12

C: .word 28

|    |    |    |    |
|----|----|----|----|
| 00 | 00 | 00 | 07 |
| FF | FF | FF | FF |
| 00 | 00 | 00 | FF |

IC

$$\begin{array}{r}
 101100 \\
 10011 \\
 +1 \\
 \hline
 111001 \\
 +1 \\
 \hline
 1110100
 \end{array}$$

a)

lds r5, A  $\rightarrow r5 = M[A]$   $\Rightarrow r5 = 7$

mov r1, #6  $\rightarrow r1 = 6$

add r5, r5, r1  $\rightarrow r5 := r5 + r1 \Rightarrow r5 = 13$

str r5, B

## Konparaketa

Konparaketa egitello agindu balaria dago:

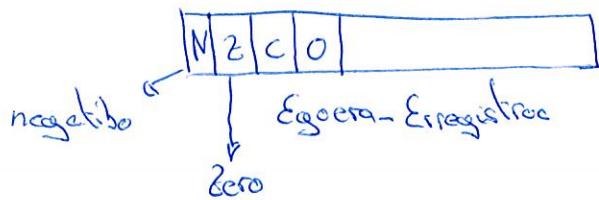
CMP r1, r2

CMP r1, #5

Erabilliko ditugun erregistreak r1 → r12 joango dira.

Konparaketaren emaitzak taldeko erregistro bat existitzen da. hori

CPSR (Current Program Status Register) esaten da. Zaiola.



Laukitxo balioitzean bit bat gordetzen du.

## Jauzi-aginduak

Agindu desberdinak dode:

breq (=)

bne(≠)

blt(<)

ble(<=)

blt(>=)

bge(>=)

Adib:

CMP r1, r2

breq etill1

$$\left\{ \begin{array}{l} \\ \downarrow \\ r1 \neq r2 \end{array} \right.$$

b etill2 → (balidintza gabeko saltua)

etill1: \_\_\_\_\_ r1 = r2

etill2: \_\_\_\_\_

C-ko programazio bat pastille dugo miliatadura lenguaia

```
main()
{
    int a=5, b=16, handi;
    if (a>b) handi=a,
    else handi=b; // a ≤ b (bile)
```

Miliatadura lenguaian horrela gelditako da:

- Code 32

- global main - CPSR - mask

a = .word 5

b = .word 16

handi: .word 0

Main:

ldr r4, a

ldr r5, b

CMP r4, r5 @ r4-r5

bgt if

str r5, handi

b: bult

if: str r4, handi

bult: mov pc, lr

# Aritmetik

2.3

- NEG aldagiekt + bilde hertillo dv.
- Bestela emaitza beti izango da!
- NKO ordun NEG-ot bestela NEG->0

2.6

Code 32  
 global main -> CPS, masl

:
 :
 :
 :

main:

r3 

|      |       |      |      |
|------|-------|------|------|
| xxxx | ..... | .... | xxxx |
|------|-------|------|------|

1000 

|      |       |      |
|------|-------|------|
| 1000 | ..... | 0000 |
|------|-------|------|

xxxx 

|      |       |     |
|------|-------|-----|
| xxxx | ..... | xxx |
|------|-------|-----|

and r3, r2, # 80000000

verarbeiten

zemb: . word -3

Posi: . word 0

main:

ldr r2, zemb

and r3, r2, # 80000000 → ands

cmp r3, #0

beq if

mov r4, #0

b bulk

if: move r4, #1

bulk: str r4, Posi

crabilita comp... regindua soberan ega  
litzatelle.

## C. baia, arildtall

2.4

- Code 32
- global main, - cpsr, - mask

Zerb: . word ~3

Abs: . word 0

main:

ldr r1, Zerb

not

cmp Ror r1, #0

bpt if →

mul r1, Abs

}

if : mul r1, #-1

