

# My Project

Generado por Doxygen 1.8.10

Martes, 20 de Octubre de 2015 07:54:40



# Índice general

<b>1</b>	<b>Índice de estructura de datos</b>	<b>1</b>
1.1	Estructura de datos	1
<b>2</b>	<b>Índice de archivos</b>	<b>3</b>
2.1	Lista de archivos	3
<b>3</b>	<b>Documentación de las estructuras de datos</b>	<b>5</b>
3.1	Referencia de la Estructura <code>ins_t</code>	5
3.1.1	Documentación de los campos	5
3.1.1.1	array	5
3.2	Referencia de la Estructura <code>instruction_t</code>	5
3.2.1	Documentación de los campos	5
3.2.1.1	mnemonic	5
3.2.1.2	op1_type	6
3.2.1.3	op1_value	6
3.2.1.4	op2_type	6
3.2.1.5	op2_value	6
3.2.1.6	op3_type	6
3.2.1.7	op3_value	6
3.2.1.8	registers_list	6
3.3	Referencia de la Estructura <code>port_t</code>	6
3.3.1	Documentación de los campos	6
3.3.1.1	DDR	6
3.3.1.2	Interrupts	6
3.3.1.3	PIN	6
3.3.1.4	Pins	6
3.3.1.5	PORT	6
<b>4</b>	<b>Documentación de archivos</b>	<b>7</b>
4.1	Referencia del Archivo <code>ALU.c</code>	7
4.1.1	Documentación de las funciones	8
4.1.1.1	<code>ACTNZ(uint32_t *Rd, uint32_t *banderas)</code>	8

4.1.1.2	ADCS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	8
4.1.1.3	ADD(uint32_t *Rd, uint32_t Rm, uint32_t Rn)	8
4.1.1.4	ADDS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	9
4.1.1.5	ANDS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	9
4.1.1.6	BICS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	9
4.1.1.7	CMN(uint32_t Rm, uint32_t Rn, uint32_t *banderas)	9
4.1.1.8	CMP(uint32_t Rm, uint32_t Rn, uint32_t *banderas)	10
4.1.1.9	EORS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	10
4.1.1.10	MOV(uint32_t *Rm, uint32_t Rn)	10
4.1.1.11	MOVS(uint32_t *Rm, uint32_t Rn, uint32_t *banderas)	10
4.1.1.12	MULS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	11
4.1.1.13	MVNS(uint32_t *Rd, uint32_t Rm, uint32_t *banderas)	11
4.1.1.14	NOP()	11
4.1.1.15	ORRS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	11
4.1.1.16	RSBS(uint32_t *Rd, uint32_t Rm, uint32_t *banderas)	12
4.1.1.17	SBCS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	12
4.1.1.18	SUB(uint32_t *Rd, uint32_t Rm, uint32_t Rn)	12
4.1.1.19	SUBS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	12
4.1.1.20	TST(uint32_t Rm, uint32_t Rn, uint32_t *banderas)	13
4.2	Referencia del Archivo ALU.h	13
4.2.1	Documentación de las funciones	14
4.2.1.1	ACTNZ(uint32_t *Rd, uint32_t *banderas)	14
4.2.1.2	ADCS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	14
4.2.1.3	ADD(uint32_t *Rd, uint32_t Rm, uint32_t Rn)	14
4.2.1.4	ADDS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	15
4.2.1.5	ANDS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	15
4.2.1.6	BICS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	15
4.2.1.7	CMN(uint32_t Rm, uint32_t Rn, uint32_t *banderas)	15
4.2.1.8	CMP(uint32_t Rm, uint32_t Rn, uint32_t *banderas)	16
4.2.1.9	EORS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	16
4.2.1.10	MOV(uint32_t *Rm, uint32_t Rn)	16
4.2.1.11	MOVS(uint32_t *Rm, uint32_t Rn, uint32_t *banderas)	16
4.2.1.12	MULS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	17
4.2.1.13	MVNS(uint32_t *Rd, uint32_t Rm, uint32_t *banderas)	17
4.2.1.14	NOP()	17
4.2.1.15	ORRS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	17
4.2.1.16	RSBS(uint32_t *Rd, uint32_t Rm, uint32_t *banderas)	18
4.2.1.17	SBCS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	18
4.2.1.18	SUB(uint32_t *Rd, uint32_t Rm, uint32_t Rn)	18
4.2.1.19	SUBS(uint32_t *Rd, uint32_t Rm, uint32_t Rn, uint32_t *banderas)	18

4.2.1.20	TST(uint32_t Rm, uint32_t Rn, uint32_t *banderas)	19
4.3	Referencia del Archivo branch.c	19
4.3.1	Documentación de las funciones	20
4.3.1.1	B(uint32_t *R, uint32_t b)	20
4.3.1.2	BAL(uint32_t *R, uint32_t b, uint32_t *banderas)	20
4.3.1.3	BCC(uint32_t *R, uint32_t b, uint32_t *banderas)	20
4.3.1.4	BCS(uint32_t *R, uint32_t b, uint32_t *banderas)	21
4.3.1.5	BEQ(uint32_t *R, uint32_t b, uint32_t *banderas)	21
4.3.1.6	BGE(uint32_t *R, uint32_t b, uint32_t *banderas)	21
4.3.1.7	BGT(uint32_t *R, uint32_t b, uint32_t *banderas)	21
4.3.1.8	BHI(uint32_t *R, uint32_t b, uint32_t *banderas)	22
4.3.1.9	BL(uint32_t *LR, uint32_t *PC, uint32_t salto)	22
4.3.1.10	BLE(uint32_t *R, uint32_t b, uint32_t *banderas)	22
4.3.1.11	BLS(uint32_t *R, uint32_t b, uint32_t *banderas)	22
4.3.1.12	BLT(uint32_t *R, uint32_t b, uint32_t *banderas)	23
4.3.1.13	BLX(uint32_t *LR, uint32_t *PC, uint32_t direccion)	23
4.3.1.14	BMI(uint32_t *R, uint32_t b, uint32_t *banderas)	23
4.3.1.15	BNE(uint32_t *R, uint32_t b, uint32_t *banderas)	23
4.3.1.16	BPL(uint32_t *R, uint32_t b, uint32_t *banderas)	24
4.3.1.17	BVC(uint32_t *R, uint32_t b, uint32_t *banderas)	24
4.3.1.18	BVS(uint32_t *R, uint32_t b, uint32_t *banderas)	24
4.3.1.19	BX(uint32_t *PC, uint32_t *direccion)	24
4.4	Referencia del Archivo branch.h	25
4.4.1	Documentación de las funciones	26
4.4.1.1	B(uint32_t *R, uint32_t b)	26
4.4.1.2	BAL(uint32_t *R, uint32_t b, uint32_t *banderas)	27
4.4.1.3	BCC(uint32_t *R, uint32_t b, uint32_t *banderas)	27
4.4.1.4	BCS(uint32_t *R, uint32_t b, uint32_t *banderas)	27
4.4.1.5	BEQ(uint32_t *R, uint32_t b, uint32_t *banderas)	27
4.4.1.6	BGE(uint32_t *R, uint32_t b, uint32_t *banderas)	28
4.4.1.7	BGT(uint32_t *R, uint32_t b, uint32_t *banderas)	28
4.4.1.8	BHI(uint32_t *R, uint32_t b, uint32_t *banderas)	28
4.4.1.9	BL(uint32_t *RL, uint32_t *PC, uint32_t salto)	28
4.4.1.10	BLE(uint32_t *R, uint32_t b, uint32_t *banderas)	29
4.4.1.11	BLS(uint32_t *R, uint32_t b, uint32_t *banderas)	29
4.4.1.12	BLT(uint32_t *R, uint32_t b, uint32_t *banderas)	29
4.4.1.13	BLX(uint32_t *RL, uint32_t *PC, uint32_t direccion)	29
4.4.1.14	BMI(uint32_t *R, uint32_t b, uint32_t *banderas)	30
4.4.1.15	BNE(uint32_t *R, uint32_t b, uint32_t *banderas)	30
4.4.1.16	BPL(uint32_t *R, uint32_t b, uint32_t *banderas)	30

4.4.1.17	BVC(uint32_t *R, uint32_t b, uint32_t *banderas)	30
4.4.1.18	BVS(uint32_t *R, uint32_t b, uint32_t *banderas)	31
4.4.1.19	BX(uint32_t *PC, uint32_t *direccion)	31
4.5	Referencia del Archivo decoder.c	31
4.5.1	Documentación de las funciones	32
4.5.1.1	countLines(FILE *fp)	32
4.5.1.2	decodeInstruction(instruction_t instruction, uint32_t *Reg, uint32_t *banderas, uint8_t *SR, uint16_t *operacion)	32
4.5.1.3	getInstruction(char *instStr)	32
4.5.1.4	readFile(char *filename, ins_t *instructions)	32
4.6	Referencia del Archivo decoder.h	32
4.6.1	Documentación de las funciones	33
4.6.1.1	countLines(FILE *fp)	33
4.6.1.2	decodeInstruction(instruction_t instruction, uint32_t *Reg, uint32_t *banderas, uint8_t *pila, uint16_t *operacion)	33
4.6.1.3	getInstruction(char *instStr)	33
4.6.1.4	readFile(char *filename, ins_t *instructions)	33
4.7	Referencia del Archivo desplazamiento.c	33
4.7.1	Documentación de las funciones	33
4.7.1.1	ASRS(uint32_t *Rd, uint32_t Rm, uint32_t num, uint32_t *banderas)	33
4.7.1.2	LSLS(uint32_t *Rd, uint32_t Rm, uint32_t num, uint32_t *banderas)	34
4.7.1.3	LSRS(uint32_t *Rd, uint32_t Rm, uint32_t num, uint32_t *banderas)	34
4.7.1.4	REV(uint32_t *Rd, uint32_t Rm)	34
4.7.1.5	REV16(uint32_t *Rd, uint32_t Rm)	34
4.7.1.6	REVSH(uint32_t *Rd, uint32_t Rm)	35
4.7.1.7	RORS(uint32_t *Rd, uint32_t Rm, uint32_t num, uint32_t *banderas)	35
4.8	Referencia del Archivo desplazamiento.h	35
4.8.1	Documentación de las funciones	36
4.8.1.1	ASRS(uint32_t *Rd, uint32_t Rm, uint32_t num, uint32_t *banderas)	36
4.8.1.2	LSLS(uint32_t *Rd, uint32_t Rm, uint32_t num, uint32_t *banderas)	36
4.8.1.3	LSRS(uint32_t *Rd, uint32_t Rm, uint32_t num, uint32_t *banderas)	36
4.8.1.4	REV(uint32_t *Rd, uint32_t Rm)	36
4.8.1.5	REV16(uint32_t *Rd, uint32_t Rm)	37
4.8.1.6	REVSH(uint32_t *Rd, uint32_t Rm)	37
4.8.1.7	RORS(uint32_t *Rd, uint32_t Rm, uint32_t num, uint32_t *banderas)	37
4.9	Referencia del Archivo interrupciones.c	37
4.9.1	Documentación de las funciones	38
4.9.1.1	NVIC(uint32_t *Reg, uint32_t *banderas, uint8_t *SR, uint8_t *Rin)	38
4.10	Referencia del Archivo interrupciones.h	38
4.10.1	Documentación de las funciones	38

4.10.1.1	NVIC(uint32_t *Reg, uint32_t *banderas, uint8_t *SR, uint8_t *Rin)	38
4.11	Referencia del Archivo io.c	38
4.11.1	Documentación de las funciones	39
4.11.1.1	changePinPortA(uint8_t pin, uint8_t value)	39
4.11.1.2	changePinPortB(uint8_t pin, uint8_t value)	39
4.11.1.3	initIO(void)	39
4.11.1.4	IOAccess(uint8_t address, uint8_t *data, uint8_t r_w)	39
4.11.1.5	showFrame(int x, int y, int w, int h)	39
4.11.1.6	showPorts(void)	39
4.11.2	Documentación de las variables	39
4.11.2.1	irq	39
4.11.2.2	PORTA	39
4.11.2.3	PORTB	39
4.12	Referencia del Archivo io.h	39
4.12.1	Documentación de los 'defines'	40
4.12.1.1	BLUEBLACK	40
4.12.1.2	HIGH	40
4.12.1.3	LOW	40
4.12.1.4	Read	40
4.12.1.5	REDBLACK	40
4.12.1.6	WHITEBLACK	40
4.12.1.7	Write	40
4.12.1.8	XINIT	40
4.12.1.9	YINIT	40
4.12.2	Documentación de las funciones	40
4.12.2.1	changePinPortA(uint8_t pin, uint8_t value)	40
4.12.2.2	changePinPortB(uint8_t pin, uint8_t value)	40
4.12.2.3	initIO(void)	40
4.12.2.4	IOAccess(uint8_t address, uint8_t *data, uint8_t r_w)	40
4.12.2.5	showFrame(int x, int y, int w, int h)	40
4.12.2.6	showPorts(void)	40
4.13	Referencia del Archivo LoadStore.c	41
4.13.1	Documentación de las funciones	41
4.13.1.1	LDR(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM, uint32_t imm)	41
4.13.1.2	LDRB(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM)	42
4.13.1.3	LDRH(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM, uint32_t imm)	43
4.13.1.4	LDRSB(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM)	43
4.13.1.5	LDRSH(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM)	43
4.13.1.6	STR(uint32_t Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM, uint32_t imm)	44
4.13.1.7	STRB(uint32_t Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM)	45

4.13.1.8	STRH(uint32_t Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM, uint32_t imm)	45
4.13.2	Documentación de las variables	45
4.13.2.1	adress	45
4.13.2.2	mem	45
4.14	Referencia del Archivo LoadStore.h	45
4.14.1	Documentación de las funciones	46
4.14.1.1	LDR(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM, uint32_t imm)	46
4.14.1.2	LDRB(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM)	46
4.14.1.3	LDRH(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM, uint32_t imm)	46
4.14.1.4	LDRSB(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM)	47
4.14.1.5	LDRSH(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM)	47
4.14.1.6	STR(uint32_t Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM, uint32_t imm)	47
4.14.1.7	STRB(uint32_t Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM)	48
4.14.1.8	STRH(uint32_t Rt, uint32_t Rn, uint32_t Rm, uint8_t *SRAM, uint32_t imm)	48
4.15	Referencia del Archivo main.c	48
4.15.1	Documentación de las funciones	49
4.15.1.1	main(void)	49
4.15.2	Documentación de las variables	49
4.15.2.1	irq	49
4.15.2.2	PORTA	49
4.15.2.3	PORTB	49
4.16	Referencia del Archivo PILA.c	49
4.16.1	Documentación de las funciones	49
4.16.1.1	POP(uint8_t registros[16], uint32_t *Reg, uint8_t *SR)	49
4.16.1.2	PUSH(uint8_t registros[16], uint32_t *Reg, uint8_t *SR)	50
4.17	Referencia del Archivo PILA.h	51
4.17.1	Documentación de las funciones	51
4.17.1.1	POP(uint8_t registros[16], uint32_t *Reg, uint8_t *SR)	51
4.17.1.2	PUSH(uint8_t registros[16], uint32_t *Reg, uint8_t *SR)	51
4.18	Referencia del Archivo ports.c	51
4.19	Referencia del Archivo registros.c	52
4.19.1	Documentación de las funciones	52
4.19.1.1	mostrar_banderas(uint32_t banderas[4])	52
4.19.1.2	mostrar_operacion(char *op)	52
4.19.1.3	mostrar_registros(uint32_t *registro)	52
4.19.1.4	mostrar_SRam(uint8_t *SRam)	53
4.19.2	Documentación de las variables	53
4.19.2.1	i	53
4.19.2.2	j	53
4.20	Referencia del Archivo registros.h	53



4.20.1 Documentación de las funciones . . . . .	53
4.20.1.1 mostrar_banderas(uint32_t banderas[4]) . . . . .	53
4.20.1.2 mostrar_operacion(char *op) . . . . .	53
4.20.1.3 mostrar_registros(uint32_t *registro) . . . . .	54
4.20.1.4 mostrar_SRam(uint8_t *SRam) . . . . .	54
4.21 Referencia del Archivo test.c . . . . .	54



# Capítulo 1

## Índice de estructura de datos

### 1.1. Estructura de datos

Lista de estructuras con una breve descripción:

<a href="#">ins_t</a> . . . . .	5
<a href="#">instruction_t</a> . . . . .	5
<a href="#">port_t</a> . . . . .	6



## Capítulo 2

# Indice de archivos

### 2.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

ALU.c	7
ALU.h	13
branch.c	19
branch.h	25
decoder.c	31
decoder.h	32
desplazamiento.c	33
desplazamiento.h	35
interrupciones.c	37
interrupciones.h	38
io.c	38
io.h	39
LoadStore.c	41
LoadStore.h	45
main.c	48
PILA.c	49
PILA.h	51
ports.c	51
registros.c	52
registros.h	53
test.c	54



## Capítulo 3

# Documentación de las estructuras de datos

### 3.1. Referencia de la Estructura ins\_t

```
#include <decoder.h>
```

#### Campos de datos

- char \*\* [array](#)

#### 3.1.1. Documentación de los campos

##### 3.1.1.1. char\*\* array

La documentación para esta estructura fue generada a partir del siguiente fichero:

- [decoder.h](#)

### 3.2. Referencia de la Estructura instruction\_t

```
#include <decoder.h>
```

#### Campos de datos

- char [mnemonic](#) [10]
- char [op1\\_type](#)
- char [op2\\_type](#)
- char [op3\\_type](#)
- uint32\_t [op1\\_value](#)
- uint32\_t [op2\\_value](#)
- uint32\_t [op3\\_value](#)
- uint8\_t [registers\\_list](#) [16]

#### 3.2.1. Documentación de los campos

##### 3.2.1.1. char mnemonic[10]

3.2.1.2. char op1\_type

3.2.1.3. uint32\_t op1\_value

3.2.1.4. char op2\_type

3.2.1.5. uint32\_t op2\_value

3.2.1.6. char op3\_type

3.2.1.7. uint32\_t op3\_value

3.2.1.8. uint8\_t registers\_list[16]

La documentación para esta estructura fue generada a partir del siguiente fichero:

- [decoder.h](#)

### 3.3. Referencia de la Estructura port\_t

```
#include <io.h>
```

#### Campos de datos

- uint8\_t [DDR](#)
- uint8\_t [PORT](#)
- uint8\_t [PIN](#)
- uint8\_t [Pins](#)
- uint8\_t [Interrupts](#)

#### 3.3.1. Documentación de los campos

3.3.1.1. uint8\_t DDR

3.3.1.2. uint8\_t Interrupts

3.3.1.3. uint8\_t PIN

3.3.1.4. uint8\_t Pins

3.3.1.5. uint8\_t PORT

La documentación para esta estructura fue generada a partir del siguiente fichero:

- [io.h](#)



## Capítulo 4

# Documentación de archivos

### 4.1. Referencia del Archivo ALU.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "ALU.h"
#include <string.h>
```

#### Funciones

- void **ADDS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion que suma*
- void **ADD** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn)  
*funcion que suma pero no actualiza banderas*
- void **SUBS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion que resta y actualiza las banderas*
- void **SUB** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn)  
*funcion que resta*
- void **ANDS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion que producto logico a nivel de bit*
- void **ORRS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion OR*
- void **MOVS** (uint32\_t \*Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion que mueve datos de un registro a otro*
- void **MOV** (uint32\_t \*Rm, uint32\_t Rn)  
*funcion que mueve datos de un registro a otro pero no actualiza banderas*
- void **BICS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion para ejecutar una AND entre un registro y el negado del otro*
- void **EORS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion para ejecutar una exclusive OR bit a bit con los dos registros de entrada*
- void **MVNS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t \*banderas)  
*funcion para ejecutar la operacion logica NOT a Rm*
- void **NOP** ()  
*funcion para no hacer nada*
- void **CMN** (uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion sumar pero no guarda el resultado*

- void **CMP** (uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion restar pero no guarda el valor solo actualiza banderas*
- void **TST** (uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion realiza la operacion AND pero no guarda el valor solo actualiza banderas*
- void **ACTNZ** (uint32\_t \*Rd, uint32\_t \*banderas)  
*funcion que actualiza las banderas N y Z*
- void **ADCS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion que efectua la suma normal y suma el carry*
- void **SBCS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion que efectua la resta normal y resta el carry*
- void **RSBS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t \*banderas)  
*funcion que efectua el complemento a dos de un registro*
- void **MULS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion que efectue el producto de dos registros sin signo*

#### 4.1.1. Documentación de las funciones

##### 4.1.1.1. void ACTNZ ( uint32\_t \* Rd, uint32\_t \* banderas )

funcion que actualiza las banderas N y Z

Parámetros

*Rd	resultado
*banderas	direccion de memoria de las banderas

Devuelve

no hay retorno

##### 4.1.1.2. void ADCS ( uint32\_t \* Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \* banderas )

funcion que efectua la suma normal y suma el carry

Parámetros

*Rd	registro donde se guarda el resultado
Rm	registro a sumar
Rn	registro a sumar
*banderas	direccion de memoria de la bandera cero "N"

Devuelve

no hay retorno

##### 4.1.1.3. void ADD ( uint32\_t \* Rd, uint32\_t Rm, uint32\_t Rn )

funcion que suma pero no actualiza banderas

Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado

Devuelve

no hay retorno

4.1.1.4. void ADDS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que suma

Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado
<i>*banderas</i>	direccion de memoria de las banderas

Devuelve

no hay retorno

4.1.1.5. void ANDS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que producto logico a nivel de bit

Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado
<i>*banderas</i>	direccion de memoria de las banderas

Devuelve

no hay retorno

4.1.1.6. void BICS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion para ejecutar una AND entre un registro y el negado del otro

Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	Registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado.
<i>*banderas</i>	direccion de memoria de las banderas

Devuelve

no hay retorno

4.1.1.7. void CMN ( uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion sumar pero no guarda el resultado

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

**4.1.1.8. void CMP ( uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )**

funcion restar pero no guarda el valor solo actualiza banderas

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*banderas</i>	direccion en memoria de las banderas

**Devuelve**

no hay retorno

**4.1.1.9. void EORS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )**

funcion para ejecutar una exclusive OR bit a bit con los dos registros de entrada

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	Registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado.
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

**4.1.1.10. void MOV ( uint32\_t \* *Rm*, uint32\_t *Rn* )**

funcion que mueve datos de un registro a otro pero no actualiza banderas

**Parámetros**

<i>Rn</i>	Registro que se va a mover
<i>*Rm</i>	lugar donde se almacenaria el resultado.

**Devuelve**

no hay retorno

**4.1.1.11. void MOVS ( uint32\_t \* *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )**

funcion que mueve datos de un registro a otro

## Parámetros

<i>Rn</i>	Registro que se mueve
* <i>Rm</i>	lugar donde se almacenara el resultado.
* <i>banderas</i>	direccion de memoria de las banderas

## Devuelve

no hay retorno

4.1.1.12. void MULS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que efectue el producto de dos registros sin signo

## Parámetros

* <i>Rd</i>	registro donde se guarda el resultado
<i>Rm</i>	registro a multiplicar
<i>Rn</i>	registro a multiplicar
* <i>banderas</i>	direccion de memoria de las bandera

## Devuelve

no hay retorno

4.1.1.13. void MVNS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t \* *banderas* )

funcion para ejecutar la operacion logica NOT a *Rm*

## Parámetros

<i>Rm</i>	Registro de entrada
* <i>Rd</i>	lugar donde se almacenara el resultado
* <i>banderas</i>	direccion de memoria de las bandera

## Devuelve

no hay retorno

4.1.1.14. void NOP ( )

funcion para no hacer nada

## Parámetros

<i>no</i>	tiene parametros de entrada
-----------	-----------------------------

## Devuelve

no hay retorno

4.1.1.15. void ORRS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion OR

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	Registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado
<i>*banderas</i>	direccion de memoria de la bandera cero "N"

**Devuelve**

no hay retorno

4.1.1.16. void RSBS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t \* *banderas* )

funcion que efectua el complemento a dos de un registro

**Parámetros**

<i>*Rd</i>	registro donde se guarda el resultado
<i>Rm</i>	registro al que se le aplicara el complemento
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

4.1.1.17. void SBCS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que efectua la resta normal y resta el carry

**Parámetros**

<i>*Rd</i>	registro donde se guarda el resultado
<i>Rm</i>	registro a de entrada
<i>Rn</i>	registro a restar con el de entrada
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

4.1.1.18. void SUB ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn* )

funcion que resta

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado

**Devuelve**

no hay retorno

4.1.1.19. void SUBS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que resta y actualiza las banderas

## Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado
<i>*banderas</i>	direccion de memoria de las banderas

## Devuelve

no hay retorno

4.1.1.20. void TST ( uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion realiza la operacion AND pero no guarda el valor solo actualiza banderas

## Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*banderas</i>	direccion de memoria de las banderas

## Devuelve

no hay retorno

## 4.2. Referencia del Archivo ALU.h

## Funciones

- void **ADCS** (uint32\_t \**Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \**banderas*)  
*funcion que efectua la suma normal y suma el carry*
- void **ADDS** (uint32\_t \**Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \**banderas*)  
*funcion que suma*
- void **ADD** (uint32\_t \**Rd*, uint32\_t *Rm*, uint32\_t *Rn*)  
*funcion que suma pero no actualiza banderas*
- void **ANDS** (uint32\_t \**Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \**banderas*)  
*funcion que producto logico a nivel de bit*
- void **BICS** (uint32\_t \**Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \**banderas*)  
*funcion para ejecutar una AND entre un registro y el negado del otro*
- void **CMN** (uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \**banderas*)  
*funcion sumar pero no guarda el resultado*
- void **CMP** (uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \**banderas*)  
*funcion restar pero no guarda el valor solo actualiza banderas*
- void **EORS** (uint32\_t \**Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \**banderas*)  
*funcion para ejecutar una exclusive OR bit a bit con los dos registros de entrada*
- void **MOV** (uint32\_t \**Rm*, uint32\_t *Rn*)  
*funcion que mueve datos de un registro a otro pero no actualiza banderas*
- void **MOVS** (uint32\_t \**Rm*, uint32\_t *Rn*, uint32\_t \**banderas*)  
*funcion que mueve datos de un registro a otro*
- void **MULS** (uint32\_t \**Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \**banderas*)  
*funcion que efectue el producto de dos registros sin signo*
- void **MVNS** (uint32\_t \**Rd*, uint32\_t *Rm*, uint32\_t \**banderas*)  
*funcion para ejecutar la operacion logica NOT a *Rm**

- void **NOP** ()  
*funcion para no hacer nada*
- void **ORRS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion OR*
- void **RSBS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t \*banderas)  
*funcion que efectua el complemento a dos de un registro*
- void **SBCS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion que efectua la resta normal y resta el carry*
- void **SUBS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion que resta y actualiza las banderas*
- void **SUB** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t Rn)  
*funcion que resta*
- void **TST** (uint32\_t Rm, uint32\_t Rn, uint32\_t \*banderas)  
*funcion realiza la operacion AND pero no guarda el valor solo actualiza banderas*
- void **ACTNZ** (uint32\_t \*Rd, uint32\_t \*banderas)  
*funcion que actualiza las banderas N y Z*

#### 4.2.1. Documentación de las funciones

##### 4.2.1.1. void ACTNZ ( uint32\_t \* Rd, uint32\_t \* banderas )

funcion que actualiza las banderas N y Z

###### Parámetros

*Rd	resultado
*banderas	direccion de memoria de las banderas

###### Devuelve

no hay retorno

##### 4.2.1.2. void ADCS ( uint32\_t \* Rd, uint32\_t Rm, uint32\_t Rn, uint32\_t \* banderas )

funcion que efectua la suma normal y suma el carry

###### Parámetros

*Rd	registro donde se guarda el resultado
Rm	registro a sumar
Rn	registro a sumar
*banderas	direccion de memoria de la bandera cero "N"

###### Devuelve

no hay retorno

##### 4.2.1.3. void ADD ( uint32\_t \* Rd, uint32\_t Rm, uint32\_t Rn )

funcion que suma pero no actualiza banderas



## Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado

## Devuelve

no hay retorno

4.2.1.4. void ADDS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que suma

## Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado
<i>*banderas</i>	direccion de memoria de las banderas

## Devuelve

no hay retorno

4.2.1.5. void ANDS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que producto logico a nivel de bit

## Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado
<i>*banderas</i>	direccion de memoria de las banderas

## Devuelve

no hay retorno

4.2.1.6. void BICS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion para ejecutar una AND entre un registro y el negado del otro

## Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	Registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado.
<i>*banderas</i>	direccion de memoria de las banderas

## Devuelve

no hay retorno

4.2.1.7. void CMN ( uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion sumar pero no guarda el resultado

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

**4.2.1.8. void CMP ( uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )**

funcion restar pero no guarda el valor solo actualiza banderas

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*banderas</i>	direccion en memoria de las banderas

**Devuelve**

no hay retorno

**4.2.1.9. void EORS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )**

funcion para ejecutar una exclusive OR bit a bit con los dos registros de entrada

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	Registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado.
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

**4.2.1.10. void MOV ( uint32\_t \* *Rm*, uint32\_t *Rn* )**

funcion que mueve datos de un registro a otro pero no actualiza banderas

**Parámetros**

<i>Rn</i>	Registro que se va a mover
<i>*Rm</i>	lugar donde se almacenaria el resultado.

**Devuelve**

no hay retorno

**4.2.1.11. void MOVS ( uint32\_t \* *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )**

funcion que mueve datos de un registro a otro

## Parámetros

<i>Rn</i>	Registro que se mueve
* <i>Rm</i>	lugar donde se almacenara el resultado.
* <i>banderas</i>	direccion de memoria de las banderas

## Devuelve

no hay retorno

4.2.1.12. void MULS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que efectue el producto de dos registros sin signo

## Parámetros

* <i>Rd</i>	registro donde se guarda el resultado
<i>Rm</i>	registro a multiplicar
<i>Rn</i>	registro a multiplicar
* <i>banderas</i>	direccion de memoria de las bandera

## Devuelve

no hay retorno

4.2.1.13. void MVNS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t \* *banderas* )

funcion para ejecutar la operacion logica NOT a *Rm*

## Parámetros

<i>Rm</i>	Registro de entrada
* <i>Rd</i>	lugar donde se almacenara el resultado
* <i>banderas</i>	direccion de memoria de las bandera

## Devuelve

no hay retorno

4.2.1.14. void NOP ( )

funcion para no hacer nada

## Parámetros

<i>no</i>	tiene parametros de entrada
-----------	-----------------------------

## Devuelve

no hay retorno

4.2.1.15. void ORRS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion OR

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	Registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado
<i>*banderas</i>	direccion de memoria de la bandera cero "N"

**Devuelve**

no hay retorno

4.2.1.16. void RSBS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t \* *banderas* )

funcion que efectua el complemento a dos de un registro

**Parámetros**

<i>*Rd</i>	registro donde se guarda el resultado
<i>Rm</i>	registro al que se le aplicara el complemento
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

4.2.1.17. void SBCS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que efectua la resta normal y resta el carry

**Parámetros**

<i>*Rd</i>	registro donde se guarda el resultado
<i>Rm</i>	registro a de entrada
<i>Rn</i>	registro a restar con el de entrada
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

4.2.1.18. void SUB ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn* )

funcion que resta

**Parámetros**

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado

**Devuelve**

no hay retorno

4.2.1.19. void SUBS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion que resta y actualiza las banderas

## Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*Rd</i>	lugar donde se almacenara el resultado
<i>*banderas</i>	direccion de memora de las banderas

## Devuelve

no hay retorno

4.2.1.20. void TST ( uint32\_t *Rm*, uint32\_t *Rn*, uint32\_t \* *banderas* )

funcion realiza la operacion AND pero no guarda el valor solo actualiza banderas

## Parámetros

<i>Rn</i>	Registro de entrada
<i>Rm</i>	registro de entrada
<i>*banderas</i>	direccion de memoria de las banderas

## Devuelve

no hay retorno

## 4.3. Referencia del Archivo branch.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "branch.h"
```

## Funciones

- void **B** (uint32\_t \**R*, uint32\_t *b*)  
*funcion que aumenta o disminuye b posiciones el pc*
- void **BEQ** (uint32\_t \**R*, uint32\_t *b*, uint32\_t \**banderas*)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera Z esta en uno*
- void **BNE** (uint32\_t \**R*, uint32\_t *b*, uint32\_t \**banderas*)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera Z esta en cero*
- void **BCS** (uint32\_t \**R*, uint32\_t *b*, uint32\_t \**banderas*)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en uno*
- void **BCC** (uint32\_t \**R*, uint32\_t *b*, uint32\_t \**banderas*)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en cero*
- void **BMI** (uint32\_t \**R*, uint32\_t *b*, uint32\_t \**banderas*)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera N esta en uno*
- void **BPL** (uint32\_t \**R*, uint32\_t *b*, uint32\_t \**banderas*)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera N esta en cero*
- void **BVS** (uint32\_t \**R*, uint32\_t *b*, uint32\_t \**banderas*)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera V esta en uno*
- void **BVC** (uint32\_t \**R*, uint32\_t *b*, uint32\_t \**banderas*)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera V esta en cero*

- void **BHI** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en uno y la Z en cero*
- void **BLS** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en cero o la Z en uno*
- void **BGE** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera N y V son iguales*
- void **BLT** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera N y V son diferentes*
- void **BGT** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera Z es cero y N igual a V*
- void **BLE** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera Z es cero 0 N diferente de V*
- void **BAL** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc*
- void **BL** (uint32\_t \*LR, uint32\_t \*PC, uint32\_t salto)  
*funcion que llama una subrutina que esta en una direccion relativa al pc*
- void **BLX** (uint32\_t \*LR, uint32\_t \*PC, uint32\_t direccion)  
*funcion que llama una subrutina que esta en la posicion direccion y RL avanza a la siguiente direccion*
- void **BX** (uint32\_t \*PC, uint32\_t \*direccion)  
*funcion que salta a una direccion espesifica por un registro*

#### 4.3.1. Documentación de las funciones

##### 4.3.1.1. void B ( uint32\_t \* R, uint32\_t b )

funcion que aumenta o disminuye b posiciones el pc

Parámetros

*R	direccion del PC
b	valor que se va a cambiar el PC

Devuelve

no hay retorno

##### 4.3.1.2. void BAL ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc

Parámetros

*R	direccion del PC
b	valor que va a cambiar el PC
*banderas	Direccion de las banderas utilizada para evaluar la condicion

Devuelve

no hay retorno

##### 4.3.1.3. void BCC ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en cero

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.3.1.4. void BCS ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en uno

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.3.1.5. void BEQ ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera Z esta en uno

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.3.1.6. void BGE ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera N y V son iguales

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.3.1.7. void BGT ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera Z es cero y N igual a V

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

4.3.1.8. void BHI ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )

funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en uno y la Z en cero

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

4.3.1.9. void BL ( uint32\_t \* *RL*, uint32\_t \* *PC*, uint32\_t *salto* )

funcion que llama una subrutina que esta en una direccion relativa al pc

**Parámetros**

<i>salto</i>	el numero de direcciones que avanzara desde su posicion actual
<i>*PC</i>	tomara la direccion de la subrutina *PC=Reg[15]
<i>*LR</i>	tomara el valor de la siguiente direccion apartir de la posicion actual *LR=Reg[14]

**Devuelve**

no hay retorno

4.3.1.10. void BLE ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )

funcion que aumenta o disminuye b posiciones el pc si la bandera Z es cero 0 N diferente de V

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

4.3.1.11. void BLS ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )

funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en cero o la Z en uno



## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.3.1.12. void BLT ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera N y V son diferentes

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.3.1.13. void BLX ( uint32\_t \* RL, uint32\_t \* PC, uint32\_t direccion )

funcion que llama una subrutina que esta en la posicion direccion y RL avanza a la siguiente direccion

## Parámetros

<i>direccion</i>	es un registro cuyo valor es la direccion de la subrutina
<i>*PC</i>	tomara la direccion de la subrutina *PC=Reg[15]
<i>*LR</i>	tomara el valor de la siguiente direccion apartir de la posicion actual *LR=Reg[14]

## Devuelve

no hay retorno

## 4.3.1.14. void BMI ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera N esta en uno

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.3.1.15. void BNE ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera Z esta en cero

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.3.1.16. void BPL ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )**

funcion que aumenta o disminuye b posiciones el pc si la bandera N esta en cero

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.3.1.17. void BVC ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )**

funcion que aumenta o disminuye b posiciones el pc si la bandera V esta en cero

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.3.1.18. void BVS ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )**

funcion que aumenta o disminuye b posiciones el pc si la bandera V esta en uno

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.3.1.19. void BX ( uint32\_t \* *PC*, uint32\_t \* *direccion* )**

funcion que salta a una direccion espesifica por un registro

## Parámetros

<i>direccion</i>	es un registro cuyo valor es la direccion
<i>*PC</i>	tomara la direccion de la subrutina *PC=Reg[15]

## Devuelve

no hay retorno

## 4.4. Referencia del Archivo branch.h

## Funciones

- void **B** (uint32\_t \*R, uint32\_t b)  
*funcion que aumenta o disminuye b posiciones el pc*
- void **BEQ** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera Z esta en uno*
- void **BNE** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera Z esta en cero*
- void **BCS** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en uno*
- void **BCC** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en cero*
- void **BMI** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera N esta en uno*
- void **BPL** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera N esta en cero*
- void **BVS** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera V esta en uno*
- void **BVC** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera V esta en cero*
- void **BHI** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en uno y la Z en cero*
- void **BLS** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en cero o la Z en uno*
- void **BGE** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera N y V son iguales*
- void **BLT** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera N y V son diferentes*
- void **BGT** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera Z es cero y N igual a V*
- void **BLE** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc si la bandera Z es cero 0 N diferente de V*
- void **BAL** (uint32\_t \*R, uint32\_t b, uint32\_t \*banderas)  
*funcion que aumenta o disminuye b posiciones el pc*
- void **BL** (uint32\_t \*RL, uint32\_t \*PC, uint32\_t salto)  
*funcion que llama una subrutina que esta en una direccion relativa al pc*
- void **BLX** (uint32\_t \*RL, uint32\_t \*PC, uint32\_t direccion)  
*funcion que llama una subrutina que esta en la posicion direccion y RL avanza a la siguiente direccion*
- void **BX** (uint32\_t \*PC, uint32\_t \*direccion)  
*funcion que salta a una direccion espesifica por un registro*

#### 4.4.1. Documentación de las funciones

##### 4.4.1.1. void B ( uint32\_t \* *R*, uint32\_t *b* )

funcion que aumenta o disminuye b posiciones el pc

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que se va a cambiar el PC

## Devuelve

no hay retorno

4.4.1.2. void BAL ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )

funcion que aumenta o disminuye b posiciones el pc

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

4.4.1.3. void BCC ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )

funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en cero

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

4.4.1.4. void BCS ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )

funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en uno

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

4.4.1.5. void BEQ ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )

funcion que aumenta o disminuye b posiciones el pc si la bandera Z esta en uno

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.4.1.6. void BGE ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )**

funcion que aumenta o disminuye b posiciones el pc si la bandera N y V son iguales

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.4.1.7. void BGT ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )**

funcion que aumenta o disminuye b posiciones el pc si la bandera Z es cero y N igual a V

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.4.1.8. void BHI ( uint32\_t \* *R*, uint32\_t *b*, uint32\_t \* *banderas* )**

funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en uno y la Z en cero

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.4.1.9. void BL ( uint32\_t \* *RL*, uint32\_t \* *PC*, uint32\_t *salto* )**

funcion que llama una subrutina que esta en una direccion relativa al pc

## Parámetros

<i>salto</i>	el numero de direcciones que avanzara desde su posicion actual
<i>*PC</i>	tomara la direccion de la subrutina *PC=Reg[15]
<i>*LR</i>	tomara el valor de la siguiente direccion apartir de la posicion actual *LR=Reg[14]

## Devuelve

no hay retorno

## 4.4.1.10. void BLE ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera Z es cero 0 N diferente de V

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.4.1.11. void BLS ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera C esta en cero o la Z en uno

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.4.1.12. void BLT ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )

funcion que aumenta o disminuye b posiciones el pc si la bandera N y V son diferentes

## Parámetros

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

## Devuelve

no hay retorno

## 4.4.1.13. void BLX ( uint32\_t \* RL, uint32\_t \* PC, uint32\_t direccion )

funcion que llama una subrutina que esta en la posicion direccion y RL avanza a la siguiente direccion

**Parámetros**

<i>direccion</i>	es un registro cuyo valor es la direccion de la subrutina
<i>*PC</i>	tomara la direccion de la subrutina *PC=Reg[15]
<i>*LR</i>	tomara el valor de la siguiente direccion apartir de la posicion actual *LR=Reg[14]

**Devuelve**

no hay retorno

**4.4.1.14. void BMI ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )**

funcion que aumenta o disminuye b posiciones el pc si la bandera N esta en uno

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.4.1.15. void BNE ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )**

funcion que aumenta o disminuye b posiciones el pc si la bandera Z esta en cero

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.4.1.16. void BPL ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )**

funcion que aumenta o disminuye b posiciones el pc si la bandera N esta en cero

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.4.1.17. void BVC ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )**

funcion que aumenta o disminuye b posiciones el pc si la bandera V esta en cero



**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.4.1.18. void BVS ( uint32\_t \* R, uint32\_t b, uint32\_t \* banderas )**

funcion que aumenta o disminuye b posiciones el pc si la bandera V esta en uno

**Parámetros**

<i>*R</i>	direccion del PC
<i>b</i>	valor que va a cambiar el PC
<i>*banderas</i>	Direccion de las banderas utilizada para evaluar la condicion

**Devuelve**

no hay retorno

**4.4.1.19. void BX ( uint32\_t \* PC, uint32\_t \* direccion )**

funcion que salta a una direccion espesifica por un registro

**Parámetros**

<i>direccion</i>	es un registro cuyo valor es la direccion
<i>*PC</i>	tomara la direccion de la subrutina *PC=Reg[15]

**Devuelve**

no hay retorno

**4.5. Referencia del Archivo decoder.c**

```
#include "decoder.h"
#include "ALU.h"
#include "registros.h"
#include "desplazamiento.h"
#include "curses.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include "branch.h"
#include "LoadStore.h"
#include "PILA.h"
```

## Funciones

- void `decodeInstruction` (`instruction_t` instruction, `uint32_t` \*Reg, `uint32_t` \*banderas, `uint8_t` \*SR, `uint16_t` \*operacion)
- `instruction_t` `getInstruction` (`char` \*instStr)  
*Obtiene la instrucción separada por partes.*
- int `readFile` (`char` \*filename, `ins_t` \*instructions)
- int `countLines` (`FILE` \*fp)

### 4.5.1. Documentación de las funciones

4.5.1.1. `int` `countLines` ( `FILE` \* *fp* )

4.5.1.2. `void` `decodeInstruction` ( `instruction_t` *instruction*, `uint32_t` \* *Reg*, `uint32_t` \* *banderas*, `uint8_t` \* *SR*, `uint16_t` \* *operacion* )

4.5.1.3. `instruction_t` `getInstruction` ( `char` \* *instStr* )

Obtiene la instrucción separada por partes.

Parámetros

<i>instStr</i>	cadena que contiene la instrucción.
* <i>operacion</i>	almacenara la direccion asignada en memoria para una instruccion especifica

Devuelve

`instruction_t` la instrucción separada por partes.

4.5.1.4. `int` `readFile` ( `char` \* *filename*, `ins_t` \* *instructions* )

## 4.6. Referencia del Archivo decoder.h

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>
```

### Estructuras de datos

- struct `ins_t`
- struct `instruction_t`

## Funciones

- void `decodeInstruction` (`instruction_t` instruction, `uint32_t` \*Reg, `uint32_t` \*banderas, `uint8_t` \*pila, `uint16_t` \*operacion)
- `instruction_t` `getInstruction` (`char` \*instStr)  
*Obtiene la instrucción separada por partes.*
- int `readFile` (`char` \*filename, `ins_t` \*instructions)
- int `countLines` (`FILE` \*fp)

#### 4.6.1. Documentación de las funciones

4.6.1.1. `int countLines ( FILE * fp )`

4.6.1.2. `void decodeInstruction ( instruction_t instruction, uint32_t * Reg, uint32_t * banderas, uint8_t * pila, uint16_t * operacion )`

4.6.1.3. `instruction_t getInstruction ( char * instStr )`

Obtiene la instrucción separada por partes.

Parámetros

<i>instStr</i>	cadena que contiene la instrucción.
<i>*operacion</i>	almacenara la direccion asignada en memoria para una instruccion especifica

Devuelve

`instruction_t` la instrucción separada por partes.

4.6.1.4. `int readFile ( char * filename, ins_t * instructions )`

### 4.7. Referencia del Archivo desplazamiento.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "desplazamiento.h"
#include "ALU.h"
```

#### Funciones

- void **LSLS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t num, uint32\_t \*banderas)  
*funcion para desplazar el dato Rm cierta cantidad de veces hacia la izquierda*
- void **LSRS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t num, uint32\_t \*banderas)  
*funcion para desplazar el dato Rm cierta cantidad de veces hacia la derecha*
- void **RORS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t num, uint32\_t \*banderas)  
*funcion para rotar hacia la derecha un dato*
- void **ASRS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t num, uint32\_t \*banderas)  
*funcion para hacer un desplazamiento aritmetico a la derecha*
- void **REV** (uint32\_t \*Rd, uint32\_t Rm)  
*funcion para desplazar paquetes de 8 bits*
- void **REV16** (uint32\_t \*Rd, uint32\_t Rm)  
*funcion para desplazar paquetes de 16 bits*
- void **REVSH** (uint32\_t \*Rd, uint32\_t Rm)  
*funcion para intercambiar los dos byts menos significativos.*

#### 4.7.1. Documentación de las funciones

4.7.1.1. `void ASRS ( uint32_t * Rd, uint32_t Rm, uint32_t num, uint32_t * banderas )`

funcion para hacer un desplazamiento aritmetico a la derecha

**Parámetros**

<i>*Rd</i>	variable donde se almacena el desplazamiento
<i>Rm</i>	registro que se desplazara
<i>num</i>	numero de bits que se desplazara
<i>*banderas</i>	direccion de memoria de la las banderas

**Devuelve**

no hay retorno

4.7.1.2. void LSLS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *num*, uint32\_t \* *banderas* )

funcion para desplazar el dato Rm cierta cantidad de veces hacia la izquierda

**Parámetros**

<i>*Rd</i>	registro donde se guardara el dato desplazado
<i>Rm</i>	registro que se desplazara
<i>num</i>	numero de bits que se desplazara el dato Rm
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

4.7.1.3. void LSRS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *num*, uint32\_t \* *banderas* )

funcion para desplazar el dato Rm cierta cantidad de veces hacia la derecha

**Parámetros**

<i>*Rd</i>	registro donde se guardara el dato desplazado
<i>Rm</i>	registro que se desplazara
<i>num</i>	numero de bits que se desplazara el dato Rm
<i>*banderas</i>	direccion de memoria de las banderas

**Devuelve**

no hay retorno

4.7.1.4. void REV ( uint32\_t \* *Rd*, uint32\_t *Rm* )

funcion para desplazar paquetes de 8 bits

**Parámetros**

<i>*Rd</i>	variable donde se almacena el desplazamiento
<i>Rm</i>	registro que se desplazara

**Devuelve**

no hay retorno

4.7.1.5. void REV16 ( uint32\_t \* *Rd*, uint32\_t *Rm* )

funcion para desplazar paquetes de 16 bits

## Parámetros

<i>*Rd</i>	variable donde se almacena el desplazamiento
<i>Rm</i>	registro que se desplazara

## Devuelve

no hay retorno

## 4.7.1.6. void REVSH ( uint32\_t \* Rd, uint32\_t Rm )

funcion para intercambiar los dos byts menos significativos.

## Parámetros

<i>*Rd</i>	variable donde se almacena el desplazamiento
<i>Rm</i>	registro que se desplazara

## Devuelve

no hay retorno

## 4.7.1.7. void RORS ( uint32\_t \* Rd, uint32\_t Rm, uint32\_t num, uint32\_t \* banderas )

funcion para rotar hacia la derecha un dato

## Parámetros

<i>*Rd</i>	variable donde se almacena la rotacion
<i>Rm</i>	registro que se rotara
<i>num</i>	numero de bits que se rotara el dato Rm
<i>*banderas</i>	direccion de memoria de las banderas

## Devuelve

no hay retorno

## 4.8. Referencia del Archivo desplazamiento.h

## Funciones

- void **ASRS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t num, uint32\_t \*banderas)  
*funcion para hacer un desplazamiento aritmetico a la derecha*
- void **LSLS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t num, uint32\_t \*banderas)  
*funcion para desplazar el dato Rm cierta cantidad de veces hacia la izquierda*
- void **LSRS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t num, uint32\_t \*banderas)  
*funcion para desplazar el dato Rm cierta cantidad de veces hacia la derecha*
- void **REV** (uint32\_t \*Rd, uint32\_t Rm)  
*funcion para desplazar paquetes de 8 bits*
- void **REV16** (uint32\_t \*Rd, uint32\_t Rm)  
*funcion para desplazar paquetes de 16 bits*
- void **REVSH** (uint32\_t \*Rd, uint32\_t Rm)  
*funcion para intercambiar los dos byts menos significativos.*
- void **RORS** (uint32\_t \*Rd, uint32\_t Rm, uint32\_t num, uint32\_t \*banderas)  
*funcion para rotar hacia la derecha un dato*

#### 4.8.1. Documentación de las funciones

##### 4.8.1.1. void ASRS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *num*, uint32\_t \* *banderas* )

funcion para hacer un desplazamiento aritmetico a la derecha

###### Parámetros

<i>*Rd</i>	variable donde se almacena el desplazamiento
<i>Rm</i>	registro que se desplazara
<i>num</i>	numero de bits que se desplazara
<i>*banderas</i>	direccion de memoria de la las banderas

###### Devuelve

no hay retorno

##### 4.8.1.2. void LSLS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *num*, uint32\_t \* *banderas* )

funcion para desplazar el dato Rm cierta cantidad de veces hacia la izquierda

###### Parámetros

<i>*Rd</i>	registro donde se guardara el dato desplazado
<i>Rm</i>	registro que se desplazara
<i>num</i>	numero de bits que se desplazara el dato Rm
<i>*banderas</i>	direccion de memoria de las banderas

###### Devuelve

no hay retorno

##### 4.8.1.3. void LSRS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *num*, uint32\_t \* *banderas* )

funcion para desplazar el dato Rm cierta cantidad de veces hacia la derecha

###### Parámetros

<i>*Rd</i>	registro donde se guardara el dato desplazado
<i>Rm</i>	registro que se desplazara
<i>num</i>	numero de bits que se desplazara el dato Rm
<i>*banderas</i>	direccion de memoria de las banderas

###### Devuelve

no hay retorno

##### 4.8.1.4. void REV ( uint32\_t \* *Rd*, uint32\_t *Rm* )

funcion para desplazar paquetes de 8 bits

###### Parámetros

<i>*Rd</i>	variable donde se almacena el desplazamiento
<i>Rm</i>	registro que se desplazara

Devuelve

no hay retorno

4.8.1.5. void REV16 ( uint32\_t \* *Rd*, uint32\_t *Rm* )

funcion para desplazar paquetes de 16 bits

Parámetros

<i>*Rd</i>	variable donde se almacena el desplazamiento
<i>Rm</i>	registro que se desplazara

Devuelve

no hay retorno

4.8.1.6. void REVSH ( uint32\_t \* *Rd*, uint32\_t *Rm* )

funcion para intercambiar los dos byts menos significativos.

Parámetros

<i>*Rd</i>	variable donde se almacena el desplazamiento
<i>Rm</i>	registro que se desplazara

Devuelve

no hay retorno

4.8.1.7. void RORS ( uint32\_t \* *Rd*, uint32\_t *Rm*, uint32\_t *num*, uint32\_t \* *banderas* )

funcion para rotar hacia la derecha un dato

Parámetros

<i>*Rd</i>	variable donde se almacena la rotacion
<i>Rm</i>	registro que se rotara
<i>num</i>	numero de bits que se rotara el dato Rm
<i>*banderas</i>	direccion de memoria de las banderas

Devuelve

no hay retorno

## 4.9. Referencia del Archivo interrupciones.c

```
#include <stdint.h>
#include <stdio.h>
#include "interrupciones.h"
```

## Funciones

- void **NVIC** (uint32\_t \*Reg, uint32\_t \*banderas, uint8\_t \*SR, uint8\_t \*Rin)

*funcion que me controla la interrupcion que se va a ejecutar*

### 4.9.1. Documentación de las funciones

#### 4.9.1.1. void NVIC ( uint32\_t \* Reg, uint32\_t \* banderas, uint8\_t \* SR, uint8\_t \* Rin )

funcion que me controla la interrupcion que se va a ejecutar

##### Parámetros

*Reg	direccion en memoria de los registros
*banderas	direccion en memoria de las banderas
*SR	direccion en memoria de la SRAM
*Rin	direccion en memoria del arreglo donde se encuentra la interrupccion que se va a ejecutar

##### Devuelve

no hay retorno

## 4.10. Referencia del Archivo interrupciones.h

### Funciones

- void **NVIC** (uint32\_t \*Reg, uint32\_t \*banderas, uint8\_t \*SR, uint8\_t \*Rin)

*funcion que me controla la interrupcion que se va a ejecutar*

### 4.10.1. Documentación de las funciones

#### 4.10.1.1. void NVIC ( uint32\_t \* Reg, uint32\_t \* banderas, uint8\_t \* SR, uint8\_t \* Rin )

funcion que me controla la interrupcion que se va a ejecutar

##### Parámetros

*Reg	direccion en memoria de los registros
*banderas	direccion en memoria de las banderas
*SR	direccion en memoria de la SRAM
*Rin	direccion en memoria del arreglo donde se encuentra la interrupccion que se va a ejecutar

##### Devuelve

no hay retorno

## 4.11. Referencia del Archivo io.c

```
#include "io.h"
```



## Funciones

- void `initIO` (void)
- void `changePinPortA` (uint8\_t pin, uint8\_t value)
- void `changePinPortB` (uint8\_t pin, uint8\_t value)
- void `IOAccess` (uint8\_t address, uint8\_t \*data, uint8\_t r\_w)
- void `showPorts` (void)
- void `showFrame` (int x, int y, int w, int h)

## Variables

- `port_t` `PORTA`
- `port_t` `PORTB`
- `uint8_t` `irq` [16]

### 4.11.1. Documentación de las funciones

4.11.1.1. void `changePinPortA` ( uint8\_t *pin*, uint8\_t *value* )

4.11.1.2. void `changePinPortB` ( uint8\_t *pin*, uint8\_t *value* )

4.11.1.3. void `initIO` ( void )

4.11.1.4. void `IOAccess` ( uint8\_t *address*, uint8\_t \* *data*, uint8\_t *r\_w* )

4.11.1.5. void `showFrame` ( int *x*, int *y*, int *w*, int *h* )

4.11.1.6. void `showPorts` ( void )

### 4.11.2. Documentación de las variables

4.11.2.1. `uint8_t` `irq`[16]

4.11.2.2. `port_t` `PORTA`

4.11.2.3. `port_t` `PORTB`

## 4.12. Referencia del Archivo io.h

```
#include <stdint.h>
#include <curses.h>
```

## Estructuras de datos

- struct `port_t`

## 'defines'

- #define `XINIT` 10
- #define `YINIT` 40
- #define `HIGH` 1
- #define `LOW` 0

- `#define Read 1`
- `#define Write 0`
- `#define BLUEBLACK 10 /*Text Blue Background Black*/`
- `#define REDBLACK 20 /*Text Red Background Black*/`
- `#define WHITEBLACK 30 /*Text White Background White*/`

## Funciones

- `void IOAccess (uint8_t address, uint8_t *data, uint8_t r_w)`
- `void changePinPortA (uint8_t pin, uint8_t value)`
- `void changePinPortB (uint8_t pin, uint8_t value)`
- `void initIO (void)`
- `void showPorts (void)`
- `void showFrame (int x, int y, int w, int h)`

### 4.12.1. Documentación de los 'defines'

4.12.1.1. `#define BLUEBLACK 10 /*Text Blue Background Black*/`

4.12.1.2. `#define HIGH 1`

4.12.1.3. `#define LOW 0`

4.12.1.4. `#define Read 1`

4.12.1.5. `#define REDBLACK 20 /*Text Red Background Black*/`

4.12.1.6. `#define WHITEBLACK 30 /*Text White Background White*/`

4.12.1.7. `#define Write 0`

4.12.1.8. `#define XINIT 10`

4.12.1.9. `#define YINIT 40`

### 4.12.2. Documentación de las funciones

4.12.2.1. `void changePinPortA ( uint8_t pin, uint8_t value )`

4.12.2.2. `void changePinPortB ( uint8_t pin, uint8_t value )`

4.12.2.3. `void initIO ( void )`

4.12.2.4. `void IOAccess ( uint8_t address, uint8_t * data, uint8_t r_w )`

4.12.2.5. `void showFrame ( int x, int y, int w, int h )`

4.12.2.6. `void showPorts ( void )`

## 4.13. Referencia del Archivo LoadStore.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "LoadStore.h"
#include <string.h>
#include "io.h"
```

### Funciones

- void **LDR** (uint32\_t \*Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM, uint32\_t imm)  
*funcion para cargar un dato de la RAM al registro Rt*
- void **LDRB** (uint32\_t \*Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM)  
*funcion para cargar 8 bits de la RAM al registro Rt*
- void **LDRH** (uint32\_t \*Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM, uint32\_t imm)  
*funcion para cargar 16 bits de la RAM al registro Rt*
- void **LDRSB** (uint32\_t \*Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM)  
*funcion para cargar 8 bits de la RAM al registro Rt con extension de signo*
- void **LDRSH** (uint32\_t \*Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM)  
*funcion para cargar 16 bits de la RAM al registro Rt con extension de signo*
- void **STR** (uint32\_t Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM, uint32\_t imm)  
*funcion que toma un registro y lo escribe en cierta direccion de memoria*
- void **STRB** (uint32\_t Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM)  
*funcion almacena los primeros 8 bits de un registro en cierta direccion de la RAM*
- void **STRH** (uint32\_t Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM, uint32\_t imm)  
*funcion almacena los primeros 16 bits de un registro en cierta direccion de la RAM*

### Variables

- uint32\_t **adress**
- uint8\_t **mem**

#### 4.13.1. Documentación de las funciones

4.13.1.1. void **LDR** ( uint32\_t \* *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM*, uint32\_t *imm* )

funcion para cargar un dato de la RAM al registro Rt

#### Parámetros

<i>*Rt</i>	resultado
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram puede ser un dato o un inmediato
<i>imm</i>	es 1 si Rm es un inmediato cero en caso de ser un registro
<i>*SRAM</i>	posicion cero de la memoria RAM

#### Devuelve

no hay retorno

4.13.1.2. void LDRB ( uint32\_t \* *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM* )

funcion para cargar 8 bits de la RAM al registro *Rt*

## Parámetros

<i>*Rt</i>	resultado
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram puede ser un dato o un inmediato
<i>*SRAM</i>	posicion cero de la memoria RAM

## Devuelve

no hay retorno

4.13.1.3. void LDRH ( uint32\_t \* *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM*, uint32\_t *imm* )

funcion para cargar 16 bits de la RAM al registro Rt

## Parámetros

<i>*Rt</i>	resultado
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram puede ser un dato o un inmediato
<i>*SRAM</i>	posicion cero de la memoria RAM
<i>imm</i>	es 1 si Rm es un inmediato cero en caso de ser un registro

## Devuelve

no hay retorno

4.13.1.4. void LDRSB ( uint32\_t \* *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM* )

funcion para cargar 8 bits de la RAM al registro Rt con extension de signo

## Parámetros

<i>*Rt</i>	resultado
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
<i>*SRAM</i>	posicion cero de la memoria RAM

## Devuelve

no hay retorno

4.13.1.5. void LDRSH ( uint32\_t \* *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM* )

funcion para cargar 16 bits de la RAM al registro Rt con extension de signo

## Parámetros

<i>*Rt</i>	resultado
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
<i>*SRAM</i>	posicion cero de la memoria RAM

## Devuelve

no hay retorno

4.13.1.6. void STR ( uint32\_t *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM*, uint32\_t *imm* )

funcion que toma un registro y lo escribe en cierta direccion de memoria

## Parámetros

<i>Rt</i>	dato a escribir
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
* <i>SRAM</i>	posicion cero de la memoria RAM
<i>imm</i>	indica si Rm es inmediato si es 1 cero de lo contrario

## Devuelve

no hay retorno

4.13.1.7. void STRB ( uint32\_t *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM* )

funcion almacena los primeros 8 bits de un registro en cierta direccion de la RAM

## Parámetros

<i>Rt</i>	dato a escribir
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
* <i>SRAM</i>	posicion cero de la memoria RAM

## Devuelve

no hay retorno

4.13.1.8. void STRH ( uint32\_t *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM*, uint32\_t *imm* )

funcion almacena los primeros 16 bits de un registro en cierta direccion de la RAM

## Parámetros

<i>Rt</i>	dato a escribir
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
* <i>SRAM</i>	posicion cero de la memoria RAM
<i>imm</i>	indica si Rm es inmediato si es 1 cero de lo contrario

## Devuelve

no hay retorno

## 4.13.2. Documentación de las variables

## 4.13.2.1. uint32\_t adress

## 4.13.2.2. uint8\_t mem

## 4.14. Referencia del Archivo LoadStore.h

## Funciones

- void **LDR** (uint32\_t \**Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \**SRAM*, uint32\_t *imm*)  
funcion para cargar un dato de la RAM al registro *Rt*

- void **LDRB** (uint32\_t \*Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM)  
*funcion para cargar 8 bits de la RAM al registro Rt*
- void **LDRH** (uint32\_t \*Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM, uint32\_t imm)  
*funcion para cargar 16 bits de la RAM al registro Rt*
- void **LDRSB** (uint32\_t \*Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM)  
*funcion para cargar 8 bits de la RAM al registro Rt con extension de signo*
- void **LDRSH** (uint32\_t \*Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM)  
*funcion para cargar 16 bits de la RAM al registro Rt con extension de signo*
- void **STR** (uint32\_t Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM, uint32\_t imm)  
*funcion que toma un registro y lo escribe en cierta direccion de memoria*
- void **STRB** (uint32\_t Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM)  
*funcion almacena los primeros 8 bits de un registro en cierta direccion de la RAM*
- void **STRH** (uint32\_t Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \*SRAM, uint32\_t imm)  
*funcion almacena los primeros 16 bits de un registro en cierta direccion de la RAM*

#### 4.14.1. Documentación de las funciones

##### 4.14.1.1. void LDR ( uint32\_t \* Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \* SRAM, uint32\_t imm )

funcion para cargar un dato de la RAM al registro Rt

Parámetros

*Rt	resultado
Rn	posicion de memoria en ram
Rm	posicion de memoria en ram puede ser un dato o un inmediato
imm	es 1 si Rm es un inmediato cero en caso de ser un registro
*SRAM	posicion cero de la memoria RAM

Devuelve

no hay retorno

##### 4.14.1.2. void LDRB ( uint32\_t \* Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \* SRAM )

funcion para cargar 8 bits de la RAM al registro Rt

Parámetros

*Rt	resultado
Rn	posicion de memoria en ram
Rm	posicion de memoria en ram puede ser un dato o un inmediato
*SRAM	posicion cero de la memoria RAM

Devuelve

no hay retorno

##### 4.14.1.3. void LDRH ( uint32\_t \* Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \* SRAM, uint32\_t imm )

funcion para cargar 16 bits de la RAM al registro Rt



## Parámetros

<i>*Rt</i>	resultado
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram puede ser un dato o un inmediato
<i>*SRAM</i>	posicion cero de la memoria RAM
<i>imm</i>	es 1 si Rm es un inmediato cero en caso de ser un registro

## Devuelve

no hay retorno

4.14.1.4. void LDRSB ( uint32\_t \* *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM* )

funcion para cargar 8 bits de la RAM al registro Rt con extension de signo

## Parámetros

<i>*Rt</i>	resultado
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
<i>*SRAM</i>	posicion cero de la memoria RAM

## Devuelve

no hay retorno

4.14.1.5. void LDRSH ( uint32\_t \* *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM* )

funcion para cargar 16 bits de la RAM al registro Rt con extension de signo

## Parámetros

<i>*Rt</i>	resultado
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
<i>*SRAM</i>	posicion cero de la memoria RAM

## Devuelve

no hay retorno

4.14.1.6. void STR ( uint32\_t *Rt*, uint32\_t *Rn*, uint32\_t *Rm*, uint8\_t \* *SRAM*, uint32\_t *imm* )

funcion que toma un registro y lo escribe en cierta direccion de memoria

## Parámetros

<i>Rt</i>	dato a escribir
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
<i>*SRAM</i>	posicion cero de la memoria RAM

<i>imm</i>	indica si Rm es inmediato si es 1 cero de lo contrario
------------	--

**Devuelve**

no hay retorno

**4.14.1.7. void STRB ( uint32\_t Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \* SRAM )**

funcion almacena los primeros 8 bits de un registro en cierta direccion de la RAM

**Parámetros**

<i>Rt</i>	dato a escribir
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
<i>*SRAM</i>	posicion cero de la memoria RAM

**Devuelve**

no hay retorno

**4.14.1.8. void STRH ( uint32\_t Rt, uint32\_t Rn, uint32\_t Rm, uint8\_t \* SRAM, uint32\_t imm )**

funcion almacena los primeros 16 bits de un registro en cierta direccion de la RAM

**Parámetros**

<i>Rt</i>	dato a escribir
<i>Rn</i>	posicion de memoria en ram
<i>Rm</i>	posicion de memoria en ram
<i>*SRAM</i>	posicion cero de la memoria RAM
<i>imm</i>	indica si Rm es inmediato si es 1 cero de lo contrario

**Devuelve**

no hay retorno

**4.15. Referencia del Archivo main.c**

```
#include "decoder.h"
#include "ALU.h"
#include "registros.h"
#include "desplazamiento.h"
#include "curses.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include "PILA.h"
#include "LoadStore.h"
#include "interrupciones.h"
#include "io.h"
```

## Funciones

- int `main` (void)

## Variables

- `port_t` `PORTA`
- `port_t` `PORTB`
- `uint8_t` `irq` [16]

### 4.15.1. Documentación de las funciones

4.15.1.1. int main ( void )

### 4.15.2. Documentación de las variables

4.15.2.1. uint8\_t irq[16]

4.15.2.2. port\_t PORTA

4.15.2.3. port\_t PORTB

## 4.16. Referencia del Archivo PILA.c

```
#include <stdint.h>
#include <stdio.h>
#include "PILA.h"
```

## Funciones

- void `PUSH` (uint8\_t registros[16], uint32\_t \*Reg, uint8\_t \*SR)  
*funcion para escribir datos en la pila*
- void `POP` (uint8\_t registros[16], uint32\_t \*Reg, uint8\_t \*SR)  
*funcion para escribir datos en la pila*

### 4.16.1. Documentación de las funciones

4.16.1.1. void POP ( uint8\_t *registros*[16], uint32\_t \* *Reg*, uint8\_t \* *SR* )

funcion para escribir datos en la pila

#### Parámetros

<i>registros</i> [16]	
* <i>Reg</i>	dato del registro cero
* <i>PILA</i>	valor en la primer direccion de memoria en la pila

#### Devuelve

no hay retorno

4.16.1.2. void PUSH ( uint8\_t *registros*[16], uint32\_t \* *Reg*, uint8\_t \* *SR* )

funcion para escribir datos en la pila

## Parámetros

<i>registros[16]</i>	
<i>*Reg</i>	dato del registro cero
<i>*PILA</i>	valor en la primer direccion de memoria en la pila

## Devuelve

no hay retorno

## 4.17. Referencia del Archivo PILA.h

## Funciones

- void **PUSH** (uint8\_t registros[16], uint32\_t \*Reg, uint8\_t \*SR)  
*funcion para escribir datos en la pila*
- void **POP** (uint8\_t registros[16], uint32\_t \*Reg, uint8\_t \*SR)  
*funcion para escribir datos en la pila*

### 4.17.1. Documentación de las funciones

#### 4.17.1.1. void POP ( uint8\_t registros[16], uint32\_t \* Reg, uint8\_t \* SR )

funcion para escribir datos en la pila

## Parámetros

<i>registros[16]</i>	
<i>*Reg</i>	dato del registro cero
<i>*PILA</i>	valor en la primer direccion de memoria en la pila

## Devuelve

no hay retorno

#### 4.17.1.2. void PUSH ( uint8\_t registros[16], uint32\_t \* Reg, uint8\_t \* SR )

funcion para escribir datos en la pila

## Parámetros

<i>registros[16]</i>	
<i>*Reg</i>	dato del registro cero
<i>*PILA</i>	valor en la primer direccion de memoria en la pila

## Devuelve

no hay retorno

## 4.18. Referencia del Archivo ports.c

## 4.19. Referencia del Archivo registros.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "registros.h"
#include <curses.h>
```

### Funciones

- void `mostrar_registros` (uint32\_t \*registro)  
*Funcion que toma los valores de los registros y los imprime en pantalla.*
- void `mostrar_SRam` (uint8\_t \*SRam)  
*Funcion que muestra los valores almacenados en la SRam.*
- void `mostrar_banderas` (uint32\_t banderas[4])  
*Funcion que toma los valores de las banderas y los imprime en pantalla.*
- void `mostrar_operacion` (char \*op)  
*Funcion que muestra la operacion que se realizo.*

### Variables

- unsigned int `i`
- unsigned int `j`

### 4.19.1. Documentación de las funciones

#### 4.19.1.1. void `mostrar_banderas` ( uint32\_t *banderas*[4] )

Funcion que toma los valores de las banderas y los imprime en pantalla.

##### Parámetros

<code>banderas[4]</code>	Arreglo que contiene cada 1 de las banderas
--------------------------	---

##### Devuelve

No tiene retorno

#### 4.19.1.2. void `mostrar_operacion` ( char \* *op* )

Funcion que muestra la operacion que se realizo.

##### Parámetros

<code>op[5]</code>	variable tipo caracter donde se gusrda el nombre de la operacion realizada
--------------------	--

##### Devuelve

No tiene retorno

#### 4.19.1.3. void `mostrar_registros` ( uint32\_t \* *registro* )

Funcion que toma los valores de los registros y los imprime en pantalla.

## Parámetros

<code>registro[13]</code>	Arreglo que contiene cada 1 de los registros
---------------------------	--

## Devuelve

No tiene retorno

## 4.19.1.4. void mostrar\_SRam ( uint8\_t \* SRam )

Funcion que muestra los valores almacenados en la SRam.

## Parámetros

<code>SRam[65]</code>	Memoria disponible
-----------------------	--------------------

## Devuelve

No tiene retorno

## 4.19.2. Documentación de las variables

## 4.19.2.1. unsigned int i

## 4.19.2.2. unsigned int j

## 4.20. Referencia del Archivo registros.h

## Funciones

- void `mostrar_registros` (uint32\_t \*registro)  
*Funcion que toma los valores de los registros y los imprime en pantalla.*
- void `mostrar_banderas` (uint32\_t banderas[4])  
*Funcion que toma los valores de las banderas y los imprime en pantalla.*
- void `mostrar_operacion` (char \*op)  
*Funcion que muestra la operacion que se realizo.*
- void `mostrar_SRam` (uint8\_t \*SRam)  
*Funcion que muestra los valores almacenados en la SRam.*

## 4.20.1. Documentación de las funciones

## 4.20.1.1. void mostrar\_banderas ( uint32\_t banderas[4] )

Funcion que toma los valores de las banderas y los imprime en pantalla.

## Parámetros

<code>banderas[4]</code>	Arreglo que contiene cada 1 de las banderas
--------------------------	---

## Devuelve

No tiene retorno

## 4.20.1.2. void mostrar\_operacion ( char \* op )

Funcion que muestra la operacion que se realizo.

**Parámetros**

<i>op[5]</i>	variable tipo caracter donde se gusrda el nombre de la operacion realizada
--------------	--

**Devuelve**

No tiene retorno

**4.20.1.3. void mostrar\_registros ( uint32\_t \* *registro* )**

Funcion que toma los valores de los registros y los imprime en pantalla.

**Parámetros**

<i>registro[13]</i>	Arreglo que contiene cada 1 de los registros
---------------------	--

**Devuelve**

No tiene retorno

**4.20.1.4. void mostrar\_SRam ( uint8\_t \* *SRam* )**

Funcion que muestra los valores almacenados en la SRam.

**Parámetros**

<i>SRam[65]</i>	Memoria disponible
-----------------	--------------------

**Devuelve**

No tiene retorno

**4.21. Referencia del Archivo test.c**