

Proyecto De Micros

Generado por Doxygen 1.8.10

Domingo, 18 de Octubre de 2015 05:47:05

Índice general

1	Índice de estructura de datos	1
1.1	Estructura de datos	1
2	Índice de archivos	3
2.1	Lista de archivos	3
3	Documentación de las estructuras de datos	5
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
4	Documentación de archivos	7
4.1	Referencia del Archivo <code>alu.c</code>	7
4.1.1	Documentación de las funciones	8
4.1.1.1	<code>ADCS(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)</code>	8
4.1.1.2	<code>ADD(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)</code>	9
4.1.1.3	<code>ADDS(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)</code>	10
4.1.1.4	<code>AND(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)</code>	10
4.1.1.5	<code>ASRS(uint32_t *Rm, uint32_t *Rn)</code>	10
4.1.1.6	<code>BICS(uint32_t *Rm, uint32_t *Rn)</code>	10
4.1.1.7	<code>CMN(uint32_t *Rm, uint32_t *Rn)</code>	11
4.1.1.8	<code>CMP(uint32_t *Rm, uint32_t *Rn)</code>	11

4.1.1.9	EOR(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	11
4.1.1.10	LSL(uint32_t *Rm, uint32_t *Rn)	11
4.1.1.11	LSLS(uint32_t *Rd, uint32_t *Rm, uint32_t Rn)	11
4.1.1.12	LSR(uint32_t *Rm, uint32_t *Rn)	12
4.1.1.13	LSRS(uint32_t *Rd, uint32_t *Rm, uint32_t Rn)	13
4.1.1.14	MOV(uint32_t *Rm, uint32_t *Rn)	13
4.1.1.15	MOVS(uint32_t *Rm, uint32_t Rn)	13
4.1.1.16	MULS(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	13
4.1.1.17	MVN(uint32_t *Rm, uint32_t *Rn)	13
4.1.1.18	NOP(void)	14
4.1.1.19	obtenerBandera(bool *bands)	14
4.1.1.20	ORR(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	14
4.1.1.21	REV(uint32_t *Rm, uint32_t *Rn)	14
4.1.1.22	REV16(uint32_t *Rm, uint32_t *Rn)	14
4.1.1.23	RORS(uint32_t *Rm, uint32_t Rn)	14
4.1.1.24	RSBS(uint32_t *Rm, uint32_t *Rn)	15
4.1.1.25	SalvarBanderas(bool *bands)	15
4.1.1.26	SBC(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	15
4.1.1.27	SUB(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	15
4.1.1.28	SUBS(uint32_t *Rd, uint32_t *Rm, uint32_t Rn)	15
4.1.1.29	TST(uint32_t *Rm, uint32_t *Rn)	16
4.1.2	Documentación de las variables	16
4.1.2.1	banderas	16
4.1.2.2	comp	16
4.2	Referencia del Archivo alu.h	16
4.2.1	Documentación de las funciones	17
4.2.1.1	ADCS(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	17
4.2.1.2	ADD(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	18
4.2.1.3	ADDS(uint32_t *Rd, uint32_t *Rm, uint32_t Rn)	18
4.2.1.4	AND(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	18
4.2.1.5	ASRS(uint32_t *Rm, uint32_t Rn)	18
4.2.1.6	BICS(uint32_t *Rm, uint32_t Rn)	19
4.2.1.7	CMN(uint32_t *Rm, uint32_t *Rn)	19
4.2.1.8	CMP(uint32_t *Rm, uint32_t *Rn)	19
4.2.1.9	EOR(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	19
4.2.1.10	LSL(uint32_t *Rm, uint32_t *Rn)	19
4.2.1.11	LSLS(uint32_t *Rd, uint32_t *Rm, uint32_t Rn)	20
4.2.1.12	LSR(uint32_t *Rm, uint32_t *Rn)	21
4.2.1.13	LSRS(uint32_t *Rd, uint32_t *Rm, uint32_t Rn)	21
4.2.1.14	MOV(uint32_t *Rm, uint32_t *Rn)	21

4.2.1.15	MOVS(uint32_t *Rm, uint32_t Rn)	21
4.2.1.16	MULS(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	21
4.2.1.17	MVN(uint32_t *Rm, uint32_t *Rn)	22
4.2.1.18	NOP(void)	22
4.2.1.19	obtenerBandera(bool *bands)	22
4.2.1.20	ORR(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	22
4.2.1.21	REV(uint32_t *Rm, uint32_t *Rn)	22
4.2.1.22	REV16(uint32_t *Rm, uint32_t *Rn)	22
4.2.1.23	RORS(uint32_t *Rm, uint32_t Rn)	23
4.2.1.24	RSBS(uint32_t *Rm, uint32_t *Rn)	23
4.2.1.25	SalvarBanderas(bool *bands)	23
4.2.1.26	SBC(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	23
4.2.1.27	SUB(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)	23
4.2.1.28	SUBS(uint32_t *Rd, uint32_t *Rm, uint32_t Rn)	24
4.2.1.29	TST(uint32_t *Rm, uint32_t *Rn)	24
4.3	Referencia del Archivo decoder.c	24
4.3.1	Documentación de las funciones	25
4.3.1.1	countLines(FILE *fp)	25
4.3.1.2	decodeInstruction(instruction_t instruction)	25
4.3.1.3	getInstruction(char *instStr)	25
4.3.1.4	iniciaram(void)	25
4.3.1.5	obtener_memoria(uint32_t *pcount)	25
4.3.1.6	obtener_registros(uint32_t *pcount)	25
4.3.1.7	obtenerPC(uint32_t *pcount)	25
4.3.1.8	readFile(char *filename, ins_t *instructions)	25
4.3.2	Documentación de las variables	25
4.3.2.1	bin	26
4.3.2.2	exnum	26
4.3.2.3	guardar	26
4.3.2.4	i	26
4.3.2.5	indicador	26
4.3.2.6	memoria	26
4.3.2.7	registers	26
4.4	Referencia del Archivo decoder.h	26
4.4.1	Documentación de las funciones	26
4.4.1.1	countLines(FILE *fp)	26
4.4.1.2	decodeInstruction(instruction_t instruction)	26
4.4.1.3	getInstruction(char *instStr)	27
4.4.1.4	iniciaram(void)	28
4.4.1.5	obtener_memoria(uint32_t *pcount)	28

4.4.1.6	obtener_registros(uint32_t *pcount)	28
4.4.1.7	obtenerPC(uint32_t *pcount)	28
4.4.1.8	POPHINTERRUPT(registers, memoria, guardar)	28
4.4.1.9	PUSHINTERRUPT(registers, memoria, guardar)	28
4.4.1.10	readFile(char *filename, ins_t *instructions)	28
4.5	Referencia del Archivo flags.c	29
4.5.1	Documentación de las funciones	29
4.5.1.1	flag(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn, bool *banderas, uint8_t *compar)	29
4.5.2	Documentación de las variables	29
4.5.2.1	H	29
4.6	Referencia del Archivo flags.h	29
4.6.1	Documentación de los 'defines'	30
4.6.1.1	C	30
4.6.1.2	N	30
4.6.1.3	V	30
4.6.1.4	Z	30
4.6.2	Documentación de las funciones	30
4.6.2.1	flag(uint32_t *Rd, uint32_t *Rm, uint32_t *Rn, bool *banderas, uint8_t *compar)	30
4.7	Referencia del Archivo main.c	30
4.7.1	Documentación de los 'defines'	31
4.7.1.1	AZUL	31
4.7.1.2	BLANCO	31
4.7.1.3	BRILLO	31
4.7.1.4	CELESTE	31
4.7.1.5	INTERMITENTE	31
4.7.1.6	INVERSO	31
4.7.1.7	MARRON	31
4.7.1.8	MAXCAD	31
4.7.1.9	MAXLINEAS	31
4.7.1.10	NEGRO	31
4.7.1.11	NORMAL	31
4.7.1.12	ROJO	31
4.7.1.13	ROSA	31
4.7.1.14	SEMIBRILLO	31
4.7.1.15	VERDE	31
4.7.2	Documentación de las funciones	32
4.7.2.1	Disp(void)	32
4.7.2.2	Exit(void)	32
4.7.2.3	Inivideo(void)	32
4.7.2.4	IniVideo(void)	32

4.7.2.5	main(void)	32
4.7.3	Documentación de las variables	32
4.7.3.1	data	32
4.7.3.2	memoria	32
4.7.3.3	win	32
4.7.3.4	wine	32
4.8	Referencia del Archivo NVIC.c	32
4.8.1	Documentación de las funciones	32
4.8.1.1	NVIC_DisableIRQ(uint8_t *Exnumb, uint8_t numb)	32
4.8.1.2	NVIC_EnableIRQ(uint8_t *Exnumb, uint8_t numb)	32
4.9	Referencia del Archivo NVIC.h	33
4.9.1	Documentación de las funciones	33
4.9.1.1	NVIC_DisableIRQ(uint8_t *Exnumb, uint8_t numb)	33
4.9.1.2	NVIC_EnableIRQ(uint8_t *Exnumb, uint8_t numb)	33
4.10	Referencia del Archivo ram.c	33
4.10.1	Documentación de las funciones	34
4.10.1.1	bitcount(uint32_t *R)	34
4.10.1.2	inimemoria(uint32_t *memoria, int tama)	34
4.10.1.3	LDR(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	34
4.10.1.4	LDRB(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	34
4.10.1.5	LDRH(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	35
4.10.1.6	mostrar_memoria(uint32_t *memoria, int tama)	35
4.10.1.7	POP(uint32_t *registros, uint32_t *memory, uint32_t *res)	35
4.10.1.8	POPINTERRUPT(uint32_t *registros, uint32_t *memory, uint32_t *res)	35
4.10.1.9	PUSH(uint32_t *registros, uint32_t *memory, uint32_t *res)	35
4.10.1.10	PUSHINTERRUPT(uint32_t *registros, uint32_t *memory, uint32_t *res)	36
4.10.1.11	STR(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	36
4.10.1.12	STRB(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	36
4.10.1.13	STRH(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	36
4.11	Referencia del Archivo ram.h	36
4.11.1	Documentación de los 'defines'	37
4.11.1.1	DIRMAXMEM	37
4.11.1.2	MEMORIA	37
4.11.2	Documentación de las funciones	37
4.11.2.1	bitcount(uint32_t *R)	37
4.11.2.2	inimemoria(uint32_t *memoria, int tama)	37
4.11.2.3	LDR(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	38
4.11.2.4	LDRB(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	38
4.11.2.5	LDRH(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	38
4.11.2.6	mostrar_memoria(uint32_t *memoria, int tama)	38

4.11.2.7	POP(uint32_t *registros, uint32_t *memory, uint32_t *res)	38
4.11.2.8	POPINTERRUPT(uint32_t *registros, uint32_t *memory, uint32_t *res)	39
4.11.2.9	PUSH(uint32_t *registros, uint32_t *memory, uint32_t *res)	39
4.11.2.10	PUSHINTERRUPT(uint32_t *registros, uint32_t *memory, uint32_t *res)	39
4.11.2.11	STR(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	39
4.11.2.12	STRB(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	39
4.11.2.13	STRH(uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)	40
4.12	Referencia del Archivo salto.c	40
4.12.1	Documentación de las funciones	41
4.12.1.1	B(uint32_t *pc, uint32_t valor)	41
4.12.1.2	BAL(uint32_t *pc, uint32_t valor)	41
4.12.1.3	BCC(uint32_t *pc, uint32_t valor)	41
4.12.1.4	BCS(uint32_t *pc, uint32_t valor)	41
4.12.1.5	BEQ(uint32_t *pc, uint32_t valor)	42
4.12.1.6	BGE(uint32_t *pc, uint32_t valor)	42
4.12.1.7	BGT(uint32_t *pc, uint32_t valor)	42
4.12.1.8	BHI(uint32_t *pc, uint32_t valor)	42
4.12.1.9	BL(uint32_t *pc, uint32_t valor, uint32_t *LR)	42
4.12.1.10	BLE(uint32_t *pc, uint32_t valor)	42
4.12.1.11	BLS(uint32_t *pc, uint32_t valor)	43
4.12.1.12	BLT(uint32_t *pc, uint32_t valor)	43
4.12.1.13	BMI(uint32_t *pc, uint32_t valor)	43
4.12.1.14	BNE(uint32_t *pc, uint32_t valor)	43
4.12.1.15	BPL(uint32_t *pc, uint32_t valor)	43
4.12.1.16	BVC(uint32_t *pc, uint32_t valor)	43
4.12.1.17	BVS(uint32_t *pc, uint32_t valor)	44
4.12.1.18	BX(uint32_t *pc, uint32_t *LR)	44
4.12.2	Documentación de las variables	44
4.12.2.1	banderas	44
4.12.2.2	LR	44
4.13	Referencia del Archivo salto.h	44
4.13.1	Documentación de las funciones	45
4.13.1.1	B(uint32_t *pc, uint32_t valor)	45
4.13.1.2	BAL(uint32_t *pc, uint32_t valor)	45
4.13.1.3	BCC(uint32_t *pc, uint32_t valor)	45
4.13.1.4	BCS(uint32_t *pc, uint32_t valor)	45
4.13.1.5	BEQ(uint32_t *pc, uint32_t valor)	46
4.13.1.6	BGE(uint32_t *pc, uint32_t valor)	46
4.13.1.7	BGT(uint32_t *pc, uint32_t valor)	46
4.13.1.8	BHI(uint32_t *pc, uint32_t valor)	46

4.13.1.9 BL(uint32_t *pc, uint32_t valor, uint32_t *LR)	46
4.13.1.10 BLE(uint32_t *pc, uint32_t valor)	46
4.13.1.11 BLS(uint32_t *pc, uint32_t valor)	47
4.13.1.12 BLT(uint32_t *pc, uint32_t valor)	47
4.13.1.13 BMI(uint32_t *pc, uint32_t valor)	47
4.13.1.14 BNE(uint32_t *pc, uint32_t valor)	47
4.13.1.15 BPL(uint32_t *pc, uint32_t valor)	47
4.13.1.16 BVC(uint32_t *pc, uint32_t valor)	47
4.13.1.17 BVS(uint32_t *pc, uint32_t valor)	48
4.13.1.18 BX(uint32_t *pc, uint32_t *LR)	48

Capítulo 1

Índice de estructura de datos

1.1. Estructura de datos

Lista de estructuras con una breve descripción:

ins_t	5
instruction_t	5

Capítulo 2

Indice de archivos

2.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

alu.c	7
alu.h	16
decoder.c	24
decoder.h	26
flags.c	29
flags.h	29
main.c	30
NVIC.c	32
NVIC.h	33
ram.c	33
ram.h	36
salto.c	40
salto.h	44

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](#)

Capítulo 4

Documentación de archivos

4.1. Referencia del Archivo alu.c

```
#include <stdint.h>
#include <stdbool.h>
#include "flags.h"
#include <curses.h>
```

Funciones

- void **ADD** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que suma los valores de dos direcciones.
- void **SUB** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que resta los valores de dos direcciones.
- void **AND** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que multiplica los valores de dos direcciones.
- void **ORR** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion de suma logica entre los valores de dos direcciones.
- void **EOR** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion logica exclusiva entre dos direcciones.
- void **ADCS** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que realiza suma con carry.
- void **CMN** (uint32_t *Rm, uint32_t *Rn)
Funcion que suma pero no guarda el resultado solo modifica banderas.
- void **SBC** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion de resta con carry.
- void **MOVS** (uint32_t *Rm, uint32_t Rn)
Funcion que escribe un valor inmediato en un registro.
- void **SUBS** (uint32_t *Rd, uint32_t *Rm, uint32_t Rn)
Funcion que resta un valor inmediato con el valor de direccion Rm.
- void **ADDS** (uint32_t *Rd, uint32_t *Rm, uint32_t Rn)
Funcion que suma un valor inmediato con el valor de direccion Rm.
- void **LSL** (uint32_t *Rm, uint32_t *Rn)
Funcion de desplazamiento logico a la izquierda.
- void **LSR** (uint32_t *Rm, uint32_t *Rn)
Funcion de desplazamiento logico a la izquierda.

- void **RORS** (uint32_t *Rm, uint32_t Rn)
Funcion de rotacion a la derecha.
- void **ASRS** (uint32_t *Rm, uint32_t Rn)
Funcion de desplazamiento aritmetico logico a la derecha.
- void **MOV** (uint32_t *Rm, uint32_t *Rn)
Funcion que escribe un valor en un resgistro.
- void **CMP** (uint32_t *Rm, uint32_t *Rn)
Funcion que resta pero no guarda el resultado solo modifica banderas.
- void **REV** (uint32_t *Rm, uint32_t *Rn)
Funcion que cambia el orden de los Bytes.
- void **REV16** (uint32_t *Rm, uint32_t *Rn)
Funcion que cambia de orden los bytes en cada halfword de 16 bits.
- void **BICS** (uint32_t *Rm, uint32_t Rn)
Funcion que ingresa un registro y el complementeto de otro.
- void **MVN** (uint32_t *Rm, uint32_t *Rn)
Funcion que guarda el complementeto de un registro.
- void **RSBS** (uint32_t *Rm, uint32_t *Rn)
Funcion que obtiene el complemento a dos de un numero.
- void **NOP** (void)
Funcion que no ejecuta ninguna operacion o mas bien no hace nada.
- void **TST** (uint32_t *Rm, uint32_t *Rn)
Funcion que multiplica bit a bit pero no guarad el resultado solo modifica banderas.
- void **LSLS** (uint32_t *Rd, uint32_t *Rm, uint32_t Rn)
Funcion que realiza un desplazamiento logico a la izquierda dependiendo del valor inmediato.
- void **LSRS** (uint32_t *Rd, uint32_t *Rm, uint32_t Rn)
Funcion que realiza un desplazamiento logico a la derecha dependiendo del valor inmediato.
- void **obtenerBandera** (bool *bands)
Funcion que obtiene el valor de las banderas.
- void **SalvarBanderas** (bool *bands)
Funcion que almacena el valor de las banderas.
- void **MULS** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion de multiplicacion y solo guarda los 32 bits menos significativos.

Variables

- bool **banderas** [4]
- uint8_t **comp** =0

4.1.1. Documentación de las funciones

4.1.1.1. void ADCS (uint32_t * Rd, uint32_t * Rm, uint32_t * Rn)

Funcion que realiza suma con carry.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.2. void ADD (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que suma los valores de dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.3. void ADDS (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t *Rn*)

Funcion que suma un valor inmediato con el valor de direccion Rm.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.4. void AND (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que multiplica los valores de dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.5. void ASRS (uint32_t * *Rm*, uint32_t *Rn*)

Funcion de desplazamiento aritmetico logico a la derecha.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

4.1.1.6. void BICS (uint32_t * *Rm*, uint32_t *Rn*)

Funcion que ingresa un registro y el complemento de otro.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
-----------	--

<i>Rn</i>	Operador sin signo de 32 bits
-----------	-------------------------------

4.1.1.7. void CMN (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que suma pero no guarda el resultado solo modifica banderas.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.1.8. void CMP (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que resta pero no guarda el resultado solo modifica banderas.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.1.9. void EOR (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion logica exclusiva entre dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.10. void LSL (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de desplazamiento logico a la izquierda.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.1.11. void LSLS (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que realiza un desplazamiento logico a la izquierda dependiendo del valor inmediato.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	<i>Rn</i> Operador sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.12. void LSR (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de desplazamiento logico a la izquierda.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.1.13. void LSRS (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t *Rn*)

Funcion que realiza un desplazamiento logico a la derecha dependiendo del valor inmediato.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Rn Operador sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.14. void MOV (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que escribe un valor en un resgistro.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.1.15. void MOVS (uint32_t * *Rm*, uint32_t *Rn*)

Funcion que escribe un valor inmediato en un registro.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

4.1.1.16. void MULS (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de multiplicacion y solo guarda los 32 bits menos significativos.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.17. void MVN (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que guarda el complementeto de un registro.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.1.18. void NOP (void)

Funcion que no ejecuta ninguna operacion o mas bien no hace nada.

4.1.1.19. void obtenerBandera (bool * *bands*)

Funcion que obtiene el valor de las banderas.

Parámetros

<i>bands</i>	direccion que obtiene el valor de las banderas
--------------	--

4.1.1.20. void ORR (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de suma logica entre los valores de dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.21. void REV (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que cambia el orden de los Bytes.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.1.22. void REV16 (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que cambia de orden los bytes en cada halfword de 16 bits.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.1.23. void RORS (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de rotacion a la derecha.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

4.1.1.24. void RSBS (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que obtiene el complemento a dos de un numero.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.1.25. void SalvarBanderas (bool * *bands*)

Funcion que almacena el valor de las banderas.

Parámetros

<i>bands</i>	direccion que obtiene el valor de las banderas
--------------	--

4.1.1.26. void SBC (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de resta con carry.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.27. void SUB (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que resta los valores de dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.28. void SUBS (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que resta un valor inmediato con el valor de direccion Rm.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.1.1.29. void TST (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que multiplica bit a bit pero no guarad el resultado solo modifica banderas.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.1.2. Documentación de las variables

4.1.2.1. bool banderas[4]

4.1.2.2. uint8_t comp =0

4.2. Referencia del Archivo alu.h

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

Funciones

- void **ADD** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que suma los valores de dos direcciones.
- void **SUB** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que resta los valores de dos direcciones.
- void **AND** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que multiplica los valores de dos direcciones.
- void **ORR** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion de suma logica entre los valores de dos direcciones.
- void **EOR** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion logica exclusiva entre dos direcciones.
- void **LSL** (uint32_t *Rm, uint32_t *Rn)
Funcion de desplazamiento logico a la izquierda.
- void **LSR** (uint32_t *Rm, uint32_t *Rn)
Funcion de desplazamiento logico a la izquierda.
- void **RORS** (uint32_t *Rm, uint32_t Rn)
Funcion de rotacion a la derecha.
- void **ASRS** (uint32_t *Rm, uint32_t Rn)
Funcion de desplazamiento aritmetico logico a la derecha.
- void **REV** (uint32_t *Rm, uint32_t *Rn)
Funcion que cambia el orden de los Bytes.

- void **REV16** (uint32_t *Rm, uint32_t *Rn)
Funcion que cambia de orden los bytes en cada halfword de 16 bits.
- void **BICS** (uint32_t *Rm, uint32_t *Rn)
Funcion que ingresa un registro y el complemento de otro.
- void **MVN** (uint32_t *Rm, uint32_t *Rn)
Funcion que guarda el complemento de un registro.
- void **RSBS** (uint32_t *Rm, uint32_t *Rn)
Funcion que obtiene el complemento a dos de un numero.
- void **NOP** (void)
Funcion que no ejecuta ninguna operacion o mas bien no hace nada.
- void **MOV** (uint32_t *Rm, uint32_t *Rn)
Funcion que escribe un valor en un registro.
- void **CMP** (uint32_t *Rm, uint32_t *Rn)
Funcion que resta pero no guarda el resultado solo modifica banderas.
- void **ADCS** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que realiza suma con carry.
- void **CMN** (uint32_t *Rm, uint32_t *Rn)
Funcion que suma pero no guarda el resultado solo modifica banderas.
- void **SBC** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion de resta con carry.
- void **MOVS** (uint32_t *Rm, uint32_t *Rn)
Funcion que escribe un valor inmediato en un registro.
- void **SUBS** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que resta un valor inmediato con el valor de direccion Rm.
- void **ADDS** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que suma un valor inmediato con el valor de direccion Rm.
- void **TST** (uint32_t *Rm, uint32_t *Rn)
Funcion que multiplica bit a bit pero no guarda el resultado solo modifica banderas.
- void **LSLS** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que realiza un desplazamiento logico a la izquierda dependiendo del valor inmediato.
- void **LSRS** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion que realiza un desplazamiento logico a la derecha dependiendo del valor inmediato.
- void **obtenerBandera** (bool *bands)
Funcion que obtiene el valor de las banderas.
- void **SalvarBanderas** (bool *bands)
Funcion que almacena el valor de las banderas.
- void **MULS** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn)
Funcion de multiplicacion y solo guarda los 32 bits menos significativos.

4.2.1. Documentación de las funciones

4.2.1.1. void ADCS (uint32_t * Rd, uint32_t * Rm, uint32_t * Rn)

Funcion que realiza suma con carry.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
-----------	--

<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits
-----------	--

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.2. void ADD (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que suma los valores de dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.3. void ADDS (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t *Rn*)

Funcion que suma un valor inmediato con el valor de direccion Rm.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.4. void AND (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que multiplica los valores de dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.5. void ASRS (uint32_t * *Rm*, uint32_t *Rn*)

Funcion de desplazamiento aritmetico logico a la derecha.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

4.2.1.6. void BICS (uint32_t * *Rm*, uint32_t *Rn*)

Funcion que ingresa un registro y el complemento de otro.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

4.2.1.7. void CMN (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que suma pero no guarda el resultado solo modifica banderas.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.2.1.8. void CMP (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que resta pero no guarda el resultado solo modifica banderas.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.2.1.9. void EOR (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion logica exclusiva entre dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.10. void LSL (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de desplazamiento logico a la izquierda.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.2.1.11. void LSLS (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t *Rn*)

Funcion que realiza un desplazamiento logico a la izquierda dependiendo del valor inmediato.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Rn Operador sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.12. void LSR (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de desplazamiento logico a la izquierda.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.2.1.13. void LSRS (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t *Rn*)

Funcion que realiza un desplazamiento logico a la derecha dependiendo del valor inmediato.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Rn Operador sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.14. void MOV (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que escribe un valor en un resgistro.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.2.1.15. void MOVS (uint32_t * *Rm*, uint32_t *Rn*)

Funcion que escribe un valor inmediato en un registro.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

4.2.1.16. void MULS (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de multiplicacion y solo guarda los 32 bits menos significativos.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.17. void MVN (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que guarda el complementado de un registro.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.2.1.18. void NOP (void)

Funcion que no ejecuta ninguna operacion o mas bien no hace nada.

4.2.1.19. void obtenerBandera (bool * *bands*)

Funcion que obtiene el valor de las banderas.

Parámetros

<i>bands</i>	direccion que obtiene el valor de las banderas
--------------	--

4.2.1.20. void ORR (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de suma logica entre los valores de dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.21. void REV (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que cambia el orden de los Bytes.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.2.1.22. void REV16 (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que cambia de orden los bytes en cada halfword de 16 bits.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.2.1.23. void RORS (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de rotacion a la derecha.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

4.2.1.24. void RSBS (uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que obtiene el complemento a dos de un numero.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.2.1.25. void SalvarBanderas (bool * *bands*)

Funcion que almacena el valor de las banderas.

Parámetros

<i>bands</i>	direccion que obtiene el valor de las banderas
--------------	--

4.2.1.26. void SBC (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion de resta con carry.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.27. void SUB (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*)

Funcion que resta los valores de dos direcciones.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
-----------	--

<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits
-----------	--

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.28. void SUBS (uint32_t * Rd, uint32_t * Rm, uint32_t Rn)

Funcion que resta un valor inmediato con el valor de direccion Rm.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Operador sin signo de 32 bits

Devuelve

Rd Direccion que contiene el valor a retornar

4.2.1.29. void TST (uint32_t * Rm, uint32_t * Rn)

Funcion que multiplica bit a bit pero no guarad el resultado solo modifica banderas.

Parámetros

<i>Rm</i>	Direccion que contiene el valor sin signo de 32 bits
<i>Rn</i>	Direccion que contiene el valor sin signo de 32 bits

4.3. Referencia del Archivo decoder.c

```
#include "decoder.h"
#include "alu.h"
#include "salto.h"
#include <stdint.h>
#include "screen.h"
#include "ram.h"
#include "NVIC.h"
```

Funciones

- void **iniciaram** (void)
Funcion que inicia la ram con FFFF..
- void **decodeInstruction** (**instruction_t** instruction)
- void **obtenerPC** (uint32_t *pcount)
- void **obtener_registros** (uint32_t *pcount)
Funcion que obtiene de las direcciones el valor de los registro.
- void **obtener_memoria** (uint32_t *pcount)
Funcion que obtiene los registros de memoria.
- **instruction_t** **getInstruction** (char *instStr)
Obtiene la instrucción separada por partes.
- int **readFile** (char *filename, **ins_t** *instructions)
- int **countLines** (FILE *fp)

Variables

- uint32_t `registers` [15]
- uint32_t `memoria` [MEMORIA]
- uint8_t `guardar` [16] = {1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,1}
- uint8_t `i`
- uint8_t `indicador` =0
- uint8_t `exnum` [16] = {0}
- uint16_t `bin` =0

4.3.1. Documentación de las funciones

4.3.1.1. `int countLines (FILE * fp)`

4.3.1.2. `void decodeInstruction (instruction_t instruction)`

4.3.1.3. `instruction_t getInstruction (char * instStr)`

Obtiene la instrucción separada por partes.

Parámetros

<code>instStr</code>	cadena que contiene la instrucción
----------------------	------------------------------------

Devuelve

`instruction_t` la instrucción separada por partes

4.3.1.4. `void iniciaram (void)`

Funcion que inicia la ram con FFFF..

4.3.1.5. `void obtener_memoria (uint32_t * pcount)`

Funcion que obtiene los registros de memoria.

Parámetros

<code>fp</code>	Direccion de la memoria
-----------------	-------------------------

4.3.1.6. `void obtener_registros (uint32_t * pcount)`

Funcion que obtiene de las direcciones el valor de los registro.

Parámetros

<code>pcount</code>	Direccion de la memoria
---------------------	-------------------------

4.3.1.7. `void obtenerPC (uint32_t * pcount)`

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

4.3.2. Documentación de las variables

4.3.2.1. uint16_t bin =0

4.3.2.2. uint8_t exnum[16] ={0}

4.3.2.3. uint8_t guardar[16] ={1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,1}

4.3.2.4. uint8_t i

4.3.2.5. uint8_t indicador =0

4.3.2.6. uint32_t memoria[MEMORIA]

4.3.2.7. uint32_t registers[15]

4.4. 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

- [instruction_t getInstruction](#) (char *instStr)
Obtiene la instrucción separada por partes.
- int [readFile](#) (char *filename, [ins_t](#) *instructions)
- int [countLines](#) (FILE *fp)
- void [obtenerPC](#) (uint32_t *pcount)
- void [decodeInstruction](#) ([instruction_t](#) instruction)
- void [iniciaram](#) (void)
Funcion que inicia la ram con FFFF.
- void [obtener_registros](#) (uint32_t *pcount)
Funcion que obtiene de las direcciones el valor de los registro.
- void [obtener_memoria](#) (uint32_t *pcount)
Funcion que obtiene los registros de memoria.
- void [PUSHINTERRUPT](#) ([registers](#), [memoria](#), [guardar](#))
Funcion PUSH dentro de la interrupcion.
- void [POPHINTERRUPT](#) ([registers](#), [memoria](#), [guardar](#))
Funcion POP dentro de la interrupcion.

4.4.1. Documentación de las funciones

4.4.1.1. int countLines (FILE * fp)

4.4.1.2. void decodeInstruction ([instruction_t](#) instruction)

4.4.1.3. `instruction_t getInstruction (char * instStr)`

Obtiene la instrucción separada por partes.

Parámetros

<i>instrStr</i>	cadena que contiene la instrucción
-----------------	------------------------------------

Devuelve

[instruction_t](#) la instrucción separada por partes

4.4.1.4. void iniciaram (void)

Funcion que inicia la ram con FFFF.

4.4.1.5. void obtener_memoria (uint32_t * pcount)

Funcion que obtiene los registros de memoria.

Parámetros

<i>fp</i>	Direccion de la memoria
-----------	-------------------------

4.4.1.6. void obtener_registros (uint32_t * pcount)

Funcion que obtiene de las direcciones el valor de los registro.

Parámetros

<i>pcount</i>	Direccion de la memoria
---------------	-------------------------

4.4.1.7. void obtenerPC (uint32_t * pcount)

4.4.1.8. void POPHINTERRUPT (registers , memoria , guardar)

Funcion POP dentro de la interrupcion.

Parámetros

<i>registers</i>	
<i>memoria</i>	
<i>guardar</i>	

4.4.1.9. void PUSHINTERRUPT (registers , memoria , guardar)

Funcion PUSH dentro de la interrupcion.

Parámetros

<i>registers</i>	
<i>memoria</i>	
<i>guardar</i>	

4.4.1.10. int readFile (char * filename, ins_t * instructions)

4.5. Referencia del Archivo flags.c

```
#include <stdint.h>
#include <stdio.h>
#include "flags.h"
#include "alu.h"
#include <curses.h>
```

Funciones

- void **flag** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn, bool *banderas, uint8_t *compar)

Función que determina las banderas (C,Z y N).

Variables

- uint32_t **H** =2147483648UL

4.5.1. Documentación de las funciones

4.5.1.1. void **flag** (uint32_t * *Rd*, uint32_t * *Rm*, uint32_t * *Rn*, bool * *banderas*, uint8_t * *compar*)

Función que determina las banderas (C,Z y N).

Parámetros

<i>Rd</i>	Dirección que contiene el operador sin signo de 32 bits.
<i>Rm</i>	Dirección que contiene el operador sin signo de 32 bits.
<i>Rn</i>	Dirección que contiene el operador sin signo de 32 bits.

Devuelve

banderas Un booleano con el resultado (0 y 1) dependiendo el caso.

4.5.2. Documentación de las variables

4.5.2.1. uint32_t **H** =2147483648UL

4.6. Referencia del Archivo flags.h

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

'defines'

- #define **C** 0
- #define **Z** 1
- #define **N** 2
- #define **V** 3

Funciones

- void **flag** (uint32_t *Rd, uint32_t *Rm, uint32_t *Rn, bool *banderas, uint8_t *compar)

Función que determina las banderas (C,Z y N).

4.6.1. Documentación de los 'defines'

4.6.1.1. #define C 0

4.6.1.2. #define N 2

4.6.1.3. #define V 3

4.6.1.4. #define Z 1

4.6.2. Documentación de las funciones

4.6.2.1. void flag (uint32_t * Rd, uint32_t * Rm, uint32_t * Rn, bool * banderas, uint8_t * compar)

Función que determina las banderas (C,Z y N).

Parámetros

<i>Rd</i>	Dirección que contiene el operador sin signo de 32 bits.
<i>Rm</i>	Dirección que contiene el operador sin signo de 32 bits.
<i>Rn</i>	Dirección que contiene el operador sin signo de 32 bits.

Devuelve

banderas Un booleano con el resultado (0 y 1) dependiendo el caso.

4.7. Referencia del Archivo main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <curses.h>
#include "screen.h"
#include "alu.h"
#include "flags.h"
#include "decoder.h"
#include "ram.h"
#include "io.h"
```

'defines'

- #define NORMAL 0
- #define BRILLO 1
- #define SEMIBRILLO 2
- #define INTERMITENTE 5
- #define INVERSO 7
- #define NEGRO 30
- #define ROJO 31
- #define VERDE 32

- #define MARRON 33
- #define AZUL 34
- #define ROSA 35
- #define CELESTE 36
- #define BLANCO 37
- #define MAXLINEAS 15
- #define MAXCAD 70

Funciones

- int Disp (void)
- void Inivideo (void)
- void Exit (void)
- void IniVideo (void)
- int main (void)

Variables

- WINDOW * win
- WINDOW * wine
- uint32_t memoria [MEMORIA]
- uint8_t data

4.7.1. Documentación de los 'defines'

4.7.1.1. #define AZUL 34

4.7.1.2. #define BLANCO 37

4.7.1.3. #define BRILLO 1

4.7.1.4. #define CELESTE 36

4.7.1.5. #define INTERMITENTE 5

4.7.1.6. #define INVERSO 7

4.7.1.7. #define MARRON 33

4.7.1.8. #define MAXCAD 70

4.7.1.9. #define MAXLINEAS 15

4.7.1.10. #define NEGRO 30

4.7.1.11. #define NORMAL 0

4.7.1.12. #define ROJO 31

4.7.1.13. #define ROSA 35

4.7.1.14. #define SEMIBRILLO 2

4.7.1.15. #define VERDE 32

4.7.2. Documentación de las funciones

4.7.2.1. `int Disp (void)`

4.7.2.2. `void Exit (void)`

4.7.2.3. `void Inivideo (void)`

4.7.2.4. `void IniVideo (void)`

4.7.2.5. `int main (void)`

4.7.3. Documentación de las variables

4.7.3.1. `uint8_t data`

4.7.3.2. `uint32_t memoria[MEMORIA]`

4.7.3.3. `WINDOW* win`

4.7.3.4. `WINDOW * wine`

4.8. Referencia del Archivo NVIC.c

```
#include <stdint.h>
#include "NVIC.h"
```

Funciones

- void [NVIC_EnableIRQ](#) (uint8_t *Exnumb, uint8_t numb)
Funcion que activa las interrupciones.
- void [NVIC_DisableIRQ](#) (uint8_t *Exnumb, uint8_t numb)
Funcion que ejecuta la regresion al codigo principal.

4.8.1. Documentación de las funciones

4.8.1.1. `void NVIC_DisableIRQ (uint8_t * Exnumb, uint8_t numb)`

Funcion que ejecuta la regresion al codigo principal.

Parámetros

<i>Exnum</i>	Direccion a la cual regresar al codigo principal
<i>numb</i>	numero sin signo de 8 bits

4.8.1.2. `void NVIC_EnableIRQ (uint8_t * Exnumb, uint8_t numb)`

Funcion que activa las interrupciones.

Parámetros

<i>Exnum</i>	Direccion a la cual accede para iniciar la interrupcion
<i>numb</i>	numero entero sin signo de 8 bits

4.9. Referencia del Archivo NVIC.h

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

Funciones

- void [NVIC_EnableIRQ](#) (uint8_t *Exnumb, uint8_t numb)
Funcion que activa las interrupciones.
- void [NVIC_DisableIRQ](#) (uint8_t *Exnumb, uint8_t numb)
Funcion que ejecuta la regresion al codigo principal.

4.9.1. Documentación de las funciones

4.9.1.1. void [NVIC_DisableIRQ](#) (uint8_t * *Exnumb*, uint8_t *numb*)

Funcion que ejecuta la regresion al codigo principal.

Parámetros

<i>Exnum</i>	Direccion a la cual regresar al codigo principal
<i>numb</i>	numero sin signo de 8 bits

4.9.1.2. void [NVIC_EnableIRQ](#) (uint8_t * *Exnumb*, uint8_t *numb*)

Funcion que activa las interrupciones.

Parámetros

<i>Exnum</i>	Direccion a la cual accede para iniciar la interrupcion
<i>numb</i>	numero entero sin signo de 8 bits

4.10. Referencia del Archivo ram.c

```
#include <stdint.h>
#include "ram.h"
#include <curses.h>
#include "flags.h"
```

Funciones

- uint32_t [bitcount](#) (uint32_t *R)
- void [PUSH](#) (uint32_t *registros, uint32_t *memory, uint32_t *res)
Funcion que permite realizar el PUSH en la memoria.
- void [POP](#) (uint32_t *registros, uint32_t *memory, uint32_t *res)

Funcion que permite realizar el POP en la memoria.

- void **mostrar_memoria** (uint32_t *memoria, int tama)

Funcion que permite mostrar los valores almacenados en la memoria.

- void **inimemoria** (uint32_t *memoria, int tama)

Funcion que inicia la memoria, es decir comienza a desde el valor maximo y va disminuye (descendente)

- void **PUSHINTERRUPT** (uint32_t *registros, uint32_t *memory, uint32_t *res)

Funcion POP dentro de la interrupcion.

- void **POPINTERRUPT** (uint32_t *registros, uint32_t *memory, uint32_t *res)

Funcion POP dentro de la interrupcion.

- void **LDR** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que extrae 4 valores de la pila dependiendo de la suma de las direcciones.

- void **LDRB** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que extrae un valor de la pila dependiendo de la suma de la direccion con otra direccion de registro.

- void **LDRH** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que extrae dos valores de la pila dependiendo de la suma de la direccion con un valor inmediato adicionalmente se realiza extension de ceros.

- void **STR** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que almacena 4 valores en la pila.

- void **STRB** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que almacena un valore en la pila ya sea los primeros 8 bit o los 8 bit ultimos.

- void **STRH** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que almacena dos valores en la pila ya sea los 16 primeros 16bit o los ultimos 16 bits.

4.10.1. Documentación de las funciones

4.10.1.1. uint32_t bitcount (uint32_t * R)

4.10.1.2. void inimemoria (uint32_t * memoria, int tama)

Funcion que inicia la memoria, es decir comienza a desde el valor maximo y va disminuye (descendente)

Parámetros

<i>memoria</i>	Direccion en la cual se almacenan los datos en la memoria
<i>tama</i>	Es el tamaño de los datos a almacenar

4.10.1.3. void LDR (uint32_t * Rt, uint32_t Rn, uint32_t Rm, uint32_t * memory)

Funcion que extrae 4 valores de la pila dependiendo de la suma de las direcciones.

Parámetros

<i>Rt</i>	Es la direccion de donde se extrae los valores
<i>Rn</i>	Dirección que contien el valor del registro a operar
<i>Rm</i>	Dirección que contien el valor del registro a operar
<i>memory</i>	Direccion donde se ubicara en memoria para obtener el valor

4.10.1.4. void LDRB (uint32_t * Rt, uint32_t Rn, uint32_t Rm, uint32_t * memory)

Funcion que extrae un valor de la pila dependiendo de la suma de la direccion con otra direccion de registro.

Parámetros

<i>Rt</i>	Es la direccion de donde se extrae el valor
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memory</i>	Dirección donde se ubicará en memoria para obtener el valor

4.10.1.5. void LDRH (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memory*)

Función que extrae dos valores de la pila dependiendo de la suma de la dirección con un valor inmediato adicionalmente se realiza extensión de ceros.

Parámetros

<i>Rt</i>	Es la direccion de donde se extrae los valores
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memory</i>	Dirección donde se ubicará en memoria para obtener el valor

4.10.1.6. void mostrar_memoria (uint32_t * *memoria*, int *tama*)

Función que permite mostrar los valores almacenados en la memoria.

Parámetros

<i>memoria</i>	Dirección en la cual se almacenan los datos en la memoria
<i>tama</i>	Es el tamaño de los datos a almacenar

4.10.1.7. void POP (uint32_t * *registros*, uint32_t * *memory*, uint32_t * *res*)

Función que permite realizar el POP en la memoria.

Parámetros

<i>registros</i>	Dirección de los registros que van del 0 al 15
<i>memoria</i>	Dirección en la cual se almacenan los datos en la memoria
<i>res</i>	Dirección de registro

4.10.1.8. void POPINTERRUPT (uint32_t * *registros*, uint32_t * *memory*, uint32_t * *res*)

Función POP dentro de la interrupción.

Parámetros

<i>registros</i>	Registros del 0 al 15
<i>memoria</i>	Dirección de memoria
<i>rest</i>	Registros en memoria

4.10.1.9. void PUSH (uint32_t * *registros*, uint32_t * *memory*, uint32_t * *res*)

Función que permite realizar el PUSH en la memoria.

Parámetros

<i>registros</i>	Dirección de los registros que van del 0 al 15
<i>memoria</i>	Dirección en la cual se almacenan los datos en la memoria
<i>res</i>	Dirección de registro

4.10.1.10. void PUSHINTERRUPT (uint32_t * *registros*, uint32_t * *memoria*, uint32_t * *res*)

Función POP dentro de la interrupción.

Parámetros

<i>registros</i>	Registros del 0 al 15
<i>memoria</i>	Dirección de memoria
<i>res</i>	Registros en memoria

4.10.1.11. void STR (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memoria*)

Función que almacena 4 valores en la pila.

Parámetros

<i>Rt</i>	Es la dirección donde se almacenarán los datos
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memoria</i>	Dirección donde se ubicará en memoria para almacenar el valor

4.10.1.12. void STRB (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memoria*)

Función que almacena un valor en la pila ya sea los primeros 8 bits o los 8 bits últimos.

Parámetros

<i>Rt</i>	Es la dirección donde se almacenarán los datos
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memoria</i>	Dirección donde se ubicará en memoria para almacenar el valor

4.10.1.13. void STRH (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memoria*)

Función que almacena dos valores en la pila ya sea los primeros 16 bits o los últimos 16 bits.

Parámetros

<i>Rt</i>	Es la dirección donde se almacenarán los datos
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memoria</i>	Dirección donde se ubicará en memoria para almacenar el valor

4.11. Referencia del Archivo ram.h

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

'defines'

- #define **MEMORIA** 64
- #define **DIRMAXMEM** 255

Funciones

- void **inimemoria** (uint32_t ***memoria**, int tama)

Funcion que inicia la memoria, es decir comienza a desde el valor maximo y va disminuye (descendente)
- void **mostrar_memoria** (uint32_t ***memoria**, int tama)

Funcion que permite mostrar los valores almacenados en la memoria.
- void **PUSH** (uint32_t *registros, uint32_t *memory, uint32_t *res)

Funcion que permite realizar el PUSH en la memoria.
- void **POP** (uint32_t *registros, uint32_t *memory, uint32_t *res)

Funcion que permite realizar el POP en la memoria.
- uint32_t **bitcount** (uint32_t *R)
- void **POPINTERRUPT** (uint32_t *registros, uint32_t *memory, uint32_t *res)

Funcion POP dentro de la interrupcion.
- void **PUSHINTERRUPT** (uint32_t *registros, uint32_t *memory, uint32_t *res)

Funcion POP dentro de la interrupcion.
- void **LDR** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que extrae 4 valores de la pila dependiendo de la suma de las direcciones.
- void **LDRB** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que extrae un valor de la pila dependiendo de la suma de la direccion con otra direccion de registro.
- void **LDRH** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que extrae dos valores de la pila dependiendo de la suma de la direccion con un valor inmediato adicionalmente se realiza extension de ceros.
- void **STR** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que almacena 4 valores en la pila.
- void **STRB** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que almacena un valore en la pila ya sea los primeros 8 bit o los 8 bit ultimos.
- void **STRH** (uint32_t *Rt, uint32_t Rn, uint32_t Rm, uint32_t *memory)

Funcion que almacena dos valores en la pila ya sea los 16 primeros 16bit o los ultimos 16 bits.

4.11.1. Documentación de los 'defines'

4.11.1.1. #define **DIRMAXMEM** 255

4.11.1.2. #define **MEMORIA** 64

4.11.2. Documentación de las funciones

4.11.2.1. uint32_t **bitcount** (uint32_t * *R*)

4.11.2.2. void **inimemoria** (uint32_t * *memoria*, int *tama*)

Funcion que inicia la memoria, es decir comienza a desde el valor maximo y va disminuye (descendente)

Parámetros

<i>memoria</i>	Dirección en la cual se almacenan los datos en la memoria
<i>tama</i>	Es el tamaño de los datos a almacenar

4.11.2.3. void LDR (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memory*)

Funcion que extrae 4 valores de la pila dependiendo de la suma de las direcciones.

Parámetros

<i>Rt</i>	Es la dirección de donde se extrae los valores
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memory</i>	Dirección donde se ubicará en memoria para obtener el valor

4.11.2.4. void LDRB (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memory*)

Funcion que extrae un valor de la pila dependiendo de la suma de la dirección con otra dirección de registro.

Parámetros

<i>Rt</i>	Es la dirección de donde se extrae el valor
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memory</i>	Dirección donde se ubicará en memoria para obtener el valor

4.11.2.5. void LDRH (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memory*)

Funcion que extrae dos valores de la pila dependiendo de la suma de la dirección con un valor inmediato adicionalmente se realiza extensión de ceros.

Parámetros

<i>Rt</i>	Es la dirección de donde se extrae los valores
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memory</i>	Dirección donde se ubicará en memoria para obtener el valor

4.11.2.6. void mostrar_memoria (uint32_t * *memoria*, int *tama*)

Funcion que permite mostrar los valores almacenados en la memoria.

Parámetros

<i>memoria</i>	Dirección en la cual se almacenan los datos en la memoria
<i>tama</i>	Es el tamaño de los datos a almacenar

4.11.2.7. void POP (uint32_t * *registros*, uint32_t * *memory*, uint32_t * *res*)

Funcion que permite realizar el POP en la memoria.

Parámetros

<i>registros</i>	Direccion de los registros que van del 0 al 15
<i>memoria</i>	Direccion en la cual se almacenan los datos en la memoria
<i>res</i>	Direccion de registro

4.11.2.8. void POPINTERRUPT (uint32_t * *registros*, uint32_t * *memory*, uint32_t * *res*)

Funcion POP dentro de la interrupcion.

Parámetros

<i>registros</i>	Registros del 0 al 15
<i>memoria</i>	Direccion de memoria
<i>rest</i>	Registros en memoria

4.11.2.9. void PUSH (uint32_t * *registros*, uint32_t * *memory*, uint32_t * *res*)

Funcion que permite realizar el PUSH en la memoria.

Parámetros

<i>registros</i>	Direccion de los registros que van del 0 al 15
<i>memoria</i>	Direccion en la cual se almacenan los datos en la memoria
<i>res</i>	Direccion de registro

4.11.2.10. void PUSHINTERRUPT (uint32_t * *registros*, uint32_t * *memory*, uint32_t * *res*)

Funcion POP dentro de la interrupcion.

Parámetros

<i>registros</i>	Registros del 0 al 15
<i>memoria</i>	Direccion de memoria
<i>res</i>	Registros en memoria

4.11.2.11. void STR (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memory*)

Funcion que almacena 4 valores en la pila.

Parámetros

<i>Rt</i>	Es la direccion donde se almacenaran los datos
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memory</i>	Direccion donde se ubicara en memoria para almacenar el valor

4.11.2.12. void STRB (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memory*)

Funcion que almacena un valor en la pila ya sea los primeros 8 bit o los 8 bit ultimos.

Parámetros

<i>Rt</i>	Es la direccion donde se almacenaran los datos
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memory</i>	Dirección donde se ubicará en memoria para almacenar el valor

4.11.2.13. void STRH (uint32_t * *Rt*, uint32_t *Rn*, uint32_t *Rm*, uint32_t * *memory*)

Función que almacena dos valores en la pila ya sea los 16 primeros 16bit o los últimos 16 bits.

Parámetros

<i>Rt</i>	Es la direccion donde se almacenaran los datos
<i>Rn</i>	Dirección que contiene el valor del registro a operar
<i>Rm</i>	Dirección que contiene el valor del registro a operar
<i>memory</i>	Dirección donde se ubicará en memoria para almacenar el valor

4.12. Referencia del Archivo salto.c

```
#include <stdint.h>
#include <stdio.h>
#include "flags.h"
#include "alu.h"
#include <stdbool.h>
#include <urses.h>
```

Funciones

- void **B** (uint32_t *pc, uint32_t valor)
Función que realiza un salto en una dirección específica.
- void **BEQ** (uint32_t *pc, uint32_t valor)
Función que realiza un salto teniendo en cuenta que la bandera sea Z == 1.
- void **BNE** (uint32_t *pc, uint32_t valor)
Función que realiza un salto teniendo en cuenta que la bandera sea Z == 0.
- void **BCS** (uint32_t *pc, uint32_t valor)
Función que realiza un salto teniendo en cuenta que la bandera sea C == 1.
- void **BCC** (uint32_t *pc, uint32_t valor)
Función que realiza un salto teniendo en cuenta que la bandera sea C == 0.
- void **BMI** (uint32_t *pc, uint32_t valor)
Función que realiza un salto teniendo en cuenta que la bandera sea N == 1.
- void **BPL** (uint32_t *pc, uint32_t valor)
Función que realiza un salto teniendo en cuenta que la bandera sea N == 0.
- void **BVS** (uint32_t *pc, uint32_t valor)
Función que realiza un salto teniendo en cuenta que la bandera sea V == 1.
- void **BVC** (uint32_t *pc, uint32_t valor)
Función que realiza un salto teniendo en cuenta que la bandera sea V == 0.
- void **BHI** (uint32_t *pc, uint32_t valor)
Función que realiza un salto teniendo en cuenta que las banderas sean C == 1 y Z == 0.
- void **BLS** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que las banderas sean $C == 0$ o $Z == 1$.

- void **BGE** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que la bandera sea $N == V$.

- void **BLT** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que la bandera sea $N != V$.

- void **BGT** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que las banderas sean $Z == 1$ y $N == V$.

- void **BLE** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que las banderas sean $Z == 0$ o $N != V$.

- void **BAL** (uint32_t *pc, uint32_t valor)

Función Función que realiza siempre un salto no se tiene condición.

- void **BL** (uint32_t *pc, uint32_t valor, uint32_t *LR)

Función que llama a una subrutina que se encuentra en una dirección relativa al pc .

- void **BX** (uint32_t *pc, uint32_t *LR)

Función Función que realiza un salto a una dirección específica por un registro.

Variables

- bool **banderas** [4]
- uint32_t **LR**

4.12.1. Documentación de las funciones

4.12.1.1. void B (uint32_t * pc, uint32_t valor)

Función que realiza un salto en una dirección específica.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.2. void BAL (uint32_t * pc, uint32_t valor)

Función Función que realiza siempre un salto no se tiene condición.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.3. void BCC (uint32_t * pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que la bandera sea $C == 0$.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.4. void BCS (uint32_t * pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que la bandera sea $C == 1$.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.5. void BEQ (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea Z == 1.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.6. void BGE (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea N == V.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.7. void BGT (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que las banderas sean Z == 1 y N == V.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.8. void BHI (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que las banderas sean C == 1 y Z == 0.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.9. void BL (uint32_t * *pc*, uint32_t *valor*, uint32_t * *LR*)

Función que llama a una subrutina que se encuentra en una dirección relativa al pc .

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.10. void BLE (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que las banderas sean Z == 0 o N != V.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.11. void BLS (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que las banderas sean C == 0 o Z==1.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.12. void BLT (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea N != V.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.13. void BMI (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea N == 1.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.14. void BNE (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea Z == 0.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.15. void BPL (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea N == 0.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.16. void BVC (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea V == 0.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.17. void BVS (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea $V == 1$.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.12.1.18. void BX (uint32_t * *pc*, uint32_t * *LR*)

Función Función que realiza un salto a una dirección específica por un registro.

Parámetros

<i>pc</i>	Contador del programa.
-----------	------------------------

4.12.2. Documentación de las variables

4.12.2.1. bool banderas[4]

4.12.2.2. uint32_t LR

4.13. Referencia del Archivo salto.h

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

Funciones

- void **B** (uint32_t **pc*, uint32_t *valor*)
Función que realiza un salto en una dirección específica.
- void **BEQ** (uint32_t **pc*, uint32_t *valor*)
Función que realiza un salto teniendo en cuenta que la bandera sea $Z == 1$.
- void **BNE** (uint32_t **pc*, uint32_t *valor*)
Función que realiza un salto teniendo en cuenta que la bandera sea $Z == 0$.
- void **BCS** (uint32_t **pc*, uint32_t *valor*)
Función que realiza un salto teniendo en cuenta que la bandera sea $C == 1$.
- void **BCC** (uint32_t **pc*, uint32_t *valor*)
Función que realiza un salto teniendo en cuenta que la bandera sea $C == 0$.
- void **BMI** (uint32_t **pc*, uint32_t *valor*)
Función que realiza un salto teniendo en cuenta que la bandera sea $N == 1$.
- void **BPL** (uint32_t **pc*, uint32_t *valor*)
Función que realiza un salto teniendo en cuenta que la bandera sea $N == 0$.
- void **BVS** (uint32_t **pc*, uint32_t *valor*)
Función que realiza un salto teniendo en cuenta que la bandera sea $V == 1$.
- void **BVC** (uint32_t **pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea $V == 0$.

- void **BHI** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que las banderas sean $C == 1$ y $Z == 0$.

- void **BLS** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que las banderas sean $C == 0$ o $Z == 1$.

- void **BGE** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que la bandera sea $N == V$.

- void **BLT** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que la bandera sea $N != V$.

- void **BGT** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que las banderas sean $Z == 1$ y $N == V$.

- void **BLE** (uint32_t *pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que las banderas sean $Z == 0$ o $N != V$.

- void **BAL** (uint32_t *pc, uint32_t valor)

Función Función que realiza siempre un salto no se tiene condición.

- void **BL** (uint32_t *pc, uint32_t valor, uint32_t *LR)

Función que llama a una subrutina que se encuentra en una dirección relativa al pc .

- void **BX** (uint32_t *pc, uint32_t *LR)

Función Función que realiza un salto a una dirección específica por un registro.

4.13.1. Documentación de las funciones

4.13.1.1. void B (uint32_t * pc, uint32_t valor)

Función que realiza un salto en una dirección específica.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.2. void BAL (uint32_t * pc, uint32_t valor)

Función Función que realiza siempre un salto no se tiene condición.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.3. void BCC (uint32_t * pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que la bandera sea $C == 0$.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.4. void BCS (uint32_t * pc, uint32_t valor)

Función que realiza un salto teniendo en cuenta que la bandera sea $C == 1$.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.5. void BEQ (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea Z == 1.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.6. void BGE (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea N == V.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.7. void BGT (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que las banderas sean Z == 1 y N == V.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.8. void BHI (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que las banderas sean C == 1 y Z == 0.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.9. void BL (uint32_t * *pc*, uint32_t *valor*, uint32_t * *LR*)

Función que llama a una subrutina que se encuentra en una dirección relativa al pc .

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.10. void BLE (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que las banderas sean Z == 0 o N != V.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.11. void BLS (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que las banderas sean C == 0 o Z==1.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.12. void BLT (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea N != V.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.13. void BMI (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea N == 1.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.14. void BNE (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea Z == 0.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.15. void BPL (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea N == 0.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.16. void BVC (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea V == 0.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.17. void BVS (uint32_t * *pc*, uint32_t *valor*)

Función que realiza un salto teniendo en cuenta que la bandera sea V == 1.

Parámetros

<i>pc</i>	Contador del programa.
<i>valor</i>	Es un valor inmediato sin signo de 32 bits.

4.13.1.18. void BX (uint32_t * *pc*, uint32_t * *LR*)

Función Función que realiza un salto a una dirección específica por un registro.

Parámetros

<i>pc</i>	Contador del programa.
-----------	------------------------