

Microcontrolador ESP32

Arquitectura del Conjunto de Instrucciones de Xtensa - Descripción General

Datos básicos sobre Xtensa ISA

Xtensa es tiene post-RISC ISA (*Instruction Set Architecture*), por lo que la mayoría de sus características derivan de los RISC (*Reduced Instruction Set Computer*) pero también incorpora ciertas características convenientes ha sido tomadas de los CISC (*Complex Instruction Set Computer*).

Los procesadores Xtensa generalmente son configurables. Los diseñadores de CPU (*Central Processor Unit*) pueden habilitar funciones como: instrucciones adicionales (predefinidas y personalizadas), interrupciones, coprocesadores, administración de memoria y otras. Algunas de estas funciones afectan la ABI (*Application Binary Interface*) y el código generado por el compilador.

Las instrucciones estándar de Xtensa son de 24 bits. La opción de densidad de código se puede habilitar para manejar instrucciones de 16 bits. Además también son posibles instrucciones más amplias en algunas configuraciones.

Los procesadores Xtensa emplean la arquitectura Harvard, lo que significa que tienen buses de instrucción y de datos separados. Según el diseño del SoC (*System on Chip*), estos buses se pueden conectar a memorias de instrucciones y datos separadas, o a una memoria compartida.

PC:

Contador de programa, contiene la dirección de la instrucción que se está ejecutando. Este registro no se puede escribir directamente. Se puede modificar como efecto secundario de llamadas, saltos y excepciones. El PC tampoco es legible directamente, sin embargo, Xtensa proporciona instrucciones para realizar cargas y saltos relativos a la PC, facilitando el acceso a valores contantes y a la generación de código independiente de la posición.

a0 - a15 :

Un total de 16 registros de propósito general de 32 bits.

AR[n]

Registros físicos de propósito general. En configuraciones de CPU sin la opción de ventanas de registros, estos son los mismos que los registros de propósito general (a0 - a15). En configuraciones de CPU donde la opción de ventanas de registros está habilitada, existen más registros físicos que los registros de propósito general. El número de registros físicos puede ser 32 o 64. Se asignan 16 registros físicos a los registros de propósito general (únicamente 16 a la vez).

Special Registers:

Los procesadores Xtensa contienen una serie de registros que se utilizan para controlar el funcionamiento del procesador, realizar el manejo de interrupciones y excepciones, etc. Solo unos pocos registros especiales son relevantes para la generación de código por parte del compilador. Los registros especiales no se pueden utilizar como operandos de ALU e instrucciones de bifurcación. Deben leerse y escribirse utilizando instrucciones RSR (*Read Special Register*) y WSR (*Write Special Register*).

SAR :

El registro SAR (*Shift Amount Register*) es un registro especial que se utiliza para almacenar el número de bits para instrucciones de corrimientos subsecuentes. La ISA de Xtensa no proporciona instrucciones de corrimiento que contengan la cantidad de corrimientos dada en un operando mediante un registro de propósito general (a0 – a15).

User Registers:

Estos registros son agregados por varias opciones de configuración del procesador o por los diseñadores de procesadores que definen instrucciones personalizadas. Solo unos pocos registros especiales son relevantes para la generación de código por parte del compilador. Al igual que los registros especiales, los registros de usuario no se pueden utilizar como operandos de ALU e instrucciones de bifurcación. Deben leerse y escribirse utilizando instrucciones RUR (*Read User Register*) y WUR (*Write User Register*).

THREADPTR:

El registro apuntador de hilo un registro de usuario. El software del sistema normalmente escribe en este registro un apuntador al TCB (*Task Control Block*) del hilo en ejecución. El compilador utiliza el registro cuando accede a variables locales de hilos.

Ventana de Registros

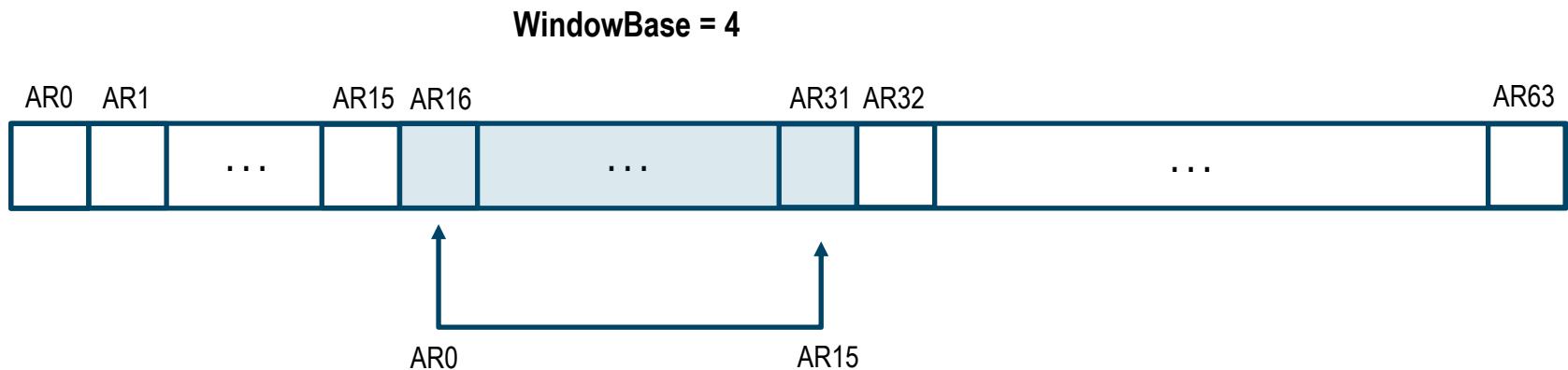
Los registros de propósito general (GPR) se utilizan para almacenar datos temporalmente en el CPU mientras se realizan varias operaciones. Estos registros son de alta velocidad pero tienen un número limitado (8 a 32 registros).

Normalmente, el número de registros presentes en el conjunto de registro es igual a los registros directamente accesibles por el núcleo. El núcleo Xtensa solo puede acceder a 16 GPR, es decir, a0 - a15.

Xtensa tiene una opción para ventanas de registros que cuando se habilita, se amplía el conjunto de registro para que contenga 64 registros. Esencialmente, el conjunto de registros (a0 – a15) actúa como una ventana a través y solo se pueden acceder 16 registros, que se deslizan sobre el conjunto registros total que se amplía a 64 registros. Y de ahí el nombre: registro en ventana.

Ventana de Registros

Los 16 registros que son visibles están controlados por el registro WindowBase. El registro WindowBase indica dónde comienza la ventana del conjunto de registro. Además, el corrimiento/rotación de esta ventana ocurre en unidades de 4. Esto significa que la ventana comienza en la posición ($\text{WindowBase} \times 4$) del conjunto de registro ampliado.



Convención a llamadas con ventana de registros

- La dirección de retorno se almacena en **a0** y el apuntador de pila se almacena en **a1**.
- Los argumentos de las funciones (procedimientos) se pasan tanto en los registros como en la memoria (pila). Los primeros 6 argumentos se pasan en los registros y los restantes en la pila.

Registro	Uso
a0	Dirección de Retorno
a1	Apuntador de Pila
a2 - a7	Argumentos de entrada

- Los valores de retorno se devuelven en registros que iniciando de a2 hasta a5. Si hay más de 4 valores a retornar, el invocador pasa un apuntador que luego el procedimiento llamado completa con todos los valores a retornar

Convención a llamadas con ventana de registros

En Xtensa, las llamadas a procedimientos se inician usando las instrucciones **CALLn** y **CALLXn**, donde n especifica la cantidad en la que se debe mover/rotar la ventana de registros para el destinatario de la llamada (n puede ser igual a 4, 8 o 12).

Se debe tener en cuenta que las instrucciones **CALL0/CALLX0** no siguen la convención de llamada con ventana de registro.

Cuando se llama a un procedimiento usando **CALLn/CALLXn**, el registro de **WindowBase** se incrementa en **n/4**, por lo que los registros visibles por el procedimiento llamado son diferentes de los visibles por el invocador de la llamada puesto que la ventana de registro (a0 – a15) se ha movido.

En general, para una llamada de registro en ventana **CALLn/CALLXn**:

- a_n del invocador será a0 en el procedimiento llamado, a_{n+1} del invocador será a1 el procedimiento llamado y así sucesivamente.

Entonces, el invocador debe poner el 1^{er} argumento del procedimiento a llamar en a_{n+2} , el 2^{do} en a_{n+3} y así sucesivamente.

Al retornar al procedimiento invocador utilizando la instrucción **RETW**, el registro de **WindowBase** se decrementa en $n/4$. Esto restaura la ventana de registro del invocador que llamó al procedimiento.

Uso de la Pila

```
/*
 * void bar(int x, int y);
 *
 * void func(void)
 * {
 *     ...
 *     foo = bar(x, y);
 *     ...
 * }
 */

func:
    ...
    mov    a10, x    // a10 is bar's a2
    mov    a11, y    // a11 is bar's a3
    call8 bar
    mov    foo, a10 // a10 is bar's a2 (return value)
    ...
```

Uso de la Pila

Como ya se mencionó el apuntador de pila reside en el registro **a1**. El apuntador de pila siempre apunta a la cima de la pila.

Por lo general, el inicio de la función configura la pila para si misma.

En Xtensa, la instrucción **ENTRY** es realiza esa función.

La instrucción **ENTRY** principalmente hace dos cosas:

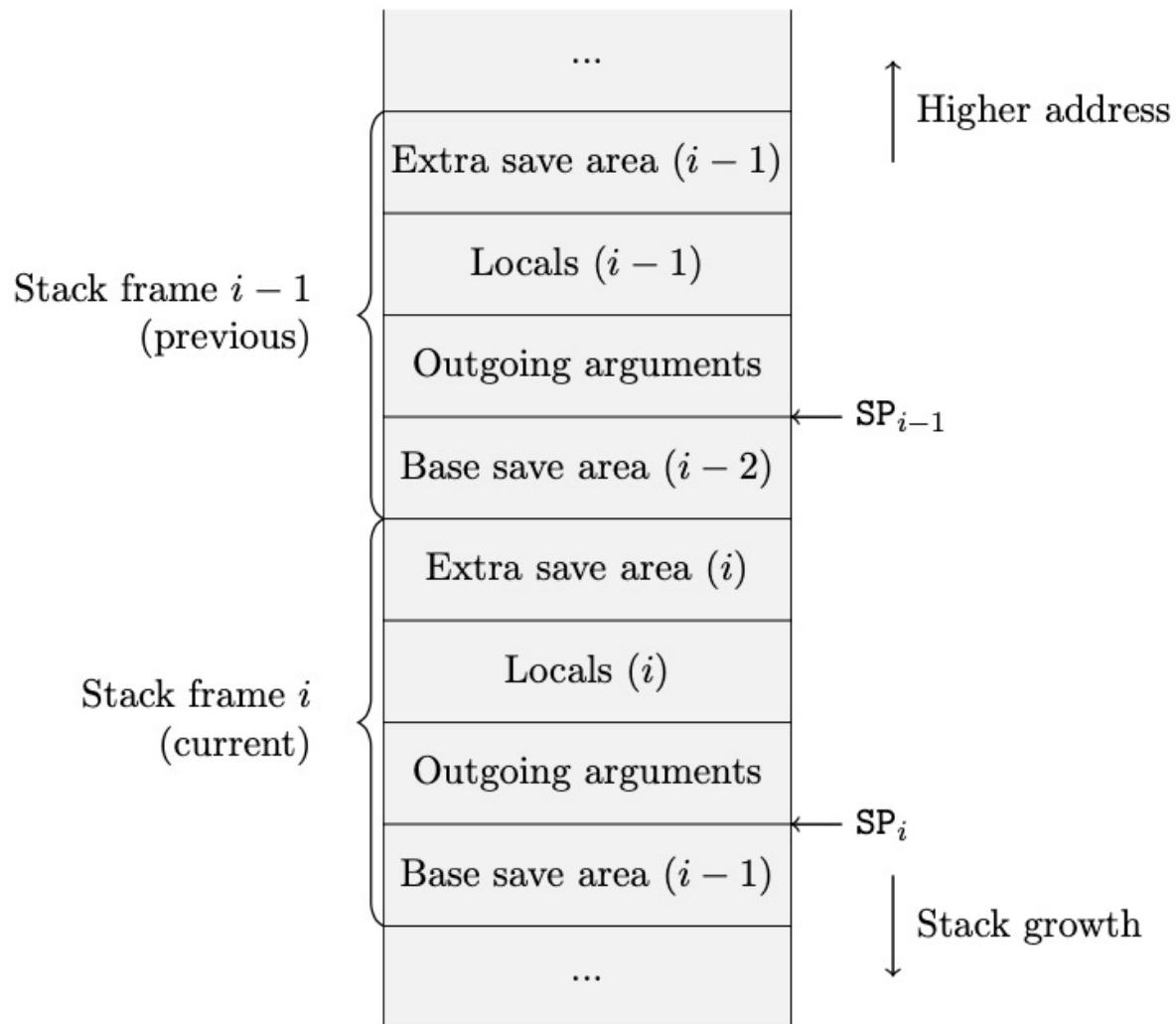
1. Asigna el marco de pila para la función y establece el apuntador de pila.
2. Mueve/rota la ventana de registros por **n** como se especifica en la instrucción CALLn/CALLXn.

Para mayor claridad, se usa **sp** como apuntador de pila en lugar de **a1**

Como la mayoría de las arquitecturas, en Xtensa también la pila crece hacia direcciones menores. Si hay argumentos salientes, además de los primeros 6 argumentos, se desplazarán positivamente desde **sp**, es decir, el 7^{mo} argumento en **sp**, el 8^{vo} en **sp + 4** y así sucesivamente. Por encima de los argumentos salientes, se almacenan las variables locales de la función.

La región debajo del apuntador de pila, llamada **Base Save Area**, es de 16 bytes y está reservada para salvar **a0 – a3** del invocador (marco de pila anterior) cuando ocurre la excepción de desbordamiento de la ventana. Si se requiere guardar más registros del invocador, entonces se almacenan en el **Extra Save Area** en la parte superior del marco de pila del invocador (anterior). La ubicación de los registros salvados del invocador (i-1) marco se resalta en la imagen.

Uso de la Pila



Uso de la Pila

```
// In procedure g, the call
// z=f(x,y)
// would compile into
    mov    a6, x           // a6 is f's a2(x)
    mov    a7, y           // a7 is f's a3(y)
    call4  f               // put return address in f's a0,
                           // goto f
    mov    z, a6            // a6 is f's a2 (return value)

// The function
// int f(int a, int *b) { return a + *b ;}
// would compile into
f:   entry  sp, framesize // allocate stack frame, rotate regs
      // on entry, a0/ return address, a1/ stack pointer,
      // a2/ a, a3/ *b
    132i  a3, a3, 0        // *b
    add   a2, a2, a3        // *b + a
    retw
```

Conjunto de Instrucciones

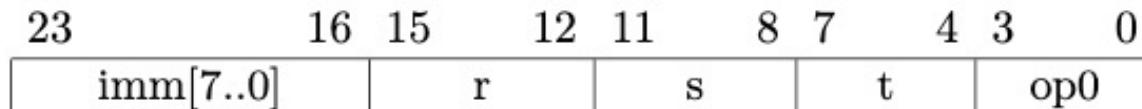
Instruction Category	Instructions ¹
Load	L8UI, L16SI, L16UI, L32I, L32R
Store	S8I, S16I, S32I
Memory ordering	MEMW, EXTW
Jump, Call	CALLO, CALLX0, RET J, JX
Conditional branch	BALL, BNALL, BANY, BNONE BBC, BBCI, BBS, BBSI BEQ, BEQI, BEQZ BNE, BNEI, BNEZ BGE, BGEI, BGEU, BGEUI, BGEZ BLT, BLTI, BLTU, BLTUI, BLTZ
Move	MOVI, MOVEQZ, MOVGEZ, MOVLTZ, MOVNEZ

Conjunto de Instrucciones

Instruction Category	Instructions ¹
Arithmetic	ADDI, ADDMI, ADD, ADDX2, ADDX4, ADDX8, SUB, SUBX2, SUBX4, SUBX8, NEG, ABS
Bitwise logical	AND, OR, XOR
Shift	EXTUI, SRLI, SRAI, SLLI SRC, SLL, SRL, SRA SSL, SSR, SSAI, SSA8B, SSA8L
Processor control	RSR, WSR, XSR, RUR, WUR, ISYNC, RSYNC, ESYNC, DSYNC, NOP

Instrucciones LOAD

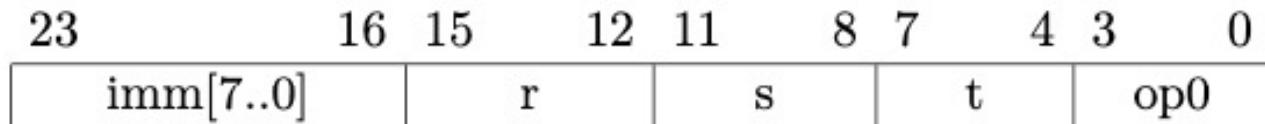
Instruction	Format	Definition
L8UI	RRI8	8-bit unsigned load (8-bit offset)
L16SI	RRI8	16-bit signed load (8-bit shifted offset)
L16UI	RRI8	16-bit unsigned load (8-bit shifted offset)
L32I	RRI8	32-bit load (8-bit shifted offset)
L32R	RI16	32-bit load PC-relative (16-bit negative word offset)

RRI8 Instruction Format**L8UI Load 8-bit Unsigned**

imm[7..0]	0000	s	t	0010	<i>L8UI</i>	$offset \leftarrow sign_extend(imm)$ $vAddr \leftarrow AR[s] + offset$ $mem \leftarrow LoadMemory(vAddr, 8)$ $AR[t] \leftarrow 0^{24} mem_{7..0}$
-----------	------	---	---	------	-------------	--

```
18ui    a3, a5, 0      ; carga en el registro a3 el byte de la dirección
          ; apuntada por registro a5+0
          ; (el byte de extiende a 32 bits)
```

RRI8 Instruction Format



L32I Load 32-bit

imm[7..0]	0010	s	t	0010	<i>L32I</i>	$offset \leftarrow sign_extend(imm)$ $vAddr \leftarrow AR[s] + offset$ $mem \leftarrow LoadMemory(vAddr, 32)$ $AR[t] \leftarrow mem_{31..0}$
-----------	------	---	---	------	-------------	--

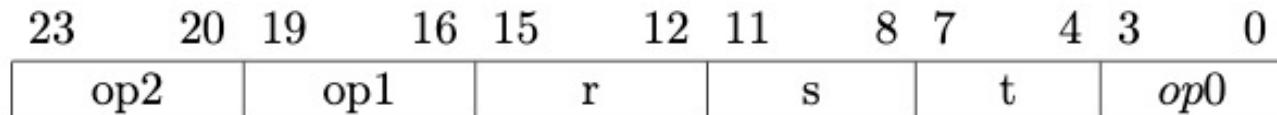
L32i a3, a5, 0 ; carga en el registro a3 el dato de 32 bits de la dirección
; apuntada por el registro a5 + 0 (sin desplazamiento)

L32i a3, a7, 4 ; carga en el registro a3 el dato de 32 bits de la dirección
; apuntada por el registro a7 + 4 (con desplazamiento)

Instrucciones Aritméticas

Instruction	Format	Definition
ADD	RRR	Add two registers $AR[r] \leftarrow AR[s] + AR[t]$
ADDX2	RRR	Add register to register shifted by 1 $AR[r] \leftarrow (AR[s]_{30..0} \parallel 0) + AR[t]$
ADDX4	RRR	Add register to register shifted by 2 $AR[r] \leftarrow (AR[s]_{29..0} \parallel 0^2) + AR[t]$
ADDX8	RRR	Add register to register shifted by 3 $AR[r] \leftarrow (AR[s]_{28..0} \parallel 0^3) + AR[t]$
SUB	RRR	Subtract two registers $AR[r] \leftarrow AR[s] - AR[t]$
SUBX2	RRR	Subtract register from register shifted by 1 $AR[r] \leftarrow (AR[s]_{30..0} \parallel 0) - AR[t]$
SUBX4	RRR	Subtract register from register shifted by 2 $AR[r] \leftarrow (AR[s]_{29..0} \parallel 0^2) - AR[t]$
SUBX8	RRR	Subtract register from register shifted by 3 $AR[r] \leftarrow (AR[s]_{28..0} \parallel 0^3) - AR[t]$
NEG	RRR	Negate $AR[r] \leftarrow 0 - AR[t]$

RRR Instruction Format



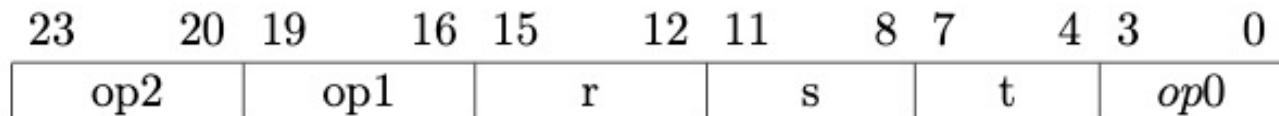
1000	0000	r	s	t	0000	ADD	$AR[r] \leftarrow AR[s] + AR[t]$
1001	0000	r	s	t	0000	ADDX2	$AR[r] \leftarrow AR[s] + (AR[t] * 2)$
1010	0000	r	s	t	0000	ADDX4	$AR[r] \leftarrow AR[s] + (AR[t] * 4)$
1011	0000	r	s	t	0000	ADDX8	$AR[r] \leftarrow AR[s] + (AR[t] * 8)$

```
add a5, a4, a3 ; suma el registro a3 y a4 y coloca el resultado
; en el registro a5
```

Instrucciones Lógicas

Instruction	Format	Definition
AND	RRR	Bitwise logical AND $AR[r] \leftarrow AR[s] \text{ and } AR[t]$
OR	RRR	Bitwise logical OR $AR[r] \leftarrow AR[s] \text{ or } AR[t]$
XOR	RRR	Bitwise logical exclusive OR $AR[r] \leftarrow AR[s] \text{ xor } AR[t]$

RRR Instruction Format



AND

Bitwise Logical AND

0001	0000	r	s	t	0000	AND	$AR[r] \leftarrow AR[s] \& AR[t]$
------	------	---	---	---	------	-----	-----------------------------------

OR

Bitwise Logical OR

0010	0000	r	s	t	0000	OR	$AR[r] \leftarrow AR[s] AR[t]$
------	------	---	---	---	------	----	----------------------------------

XOR

Bitwise Logical XOR

0011	0000	r	s	t	0000	XOR	$AR[r] \leftarrow AR[s] \oplus AR[t]$
------	------	---	---	---	------	-----	---------------------------------------

```
xor    a3, a3, a2      ; realiza la operación XOR entre el registro a2 y a3
          ; y coloca el resultado en el registro a3
```

Instrucciones de Salto y Llamadas

Instruction	Format	Definition
CALL0	CALL	Call subroutine, PC-relative
CALLX0	CALLX	Call subroutine, address in register
J	CALL	Unconditional jump, PC-relative
JX	CALLX	Unconditional jump, address in register
RET	CALLX	Subroutine return—jump to return address. Used to return from a routine called by CALL0 / CALLX0.

Instrucciones de Salto y Llamadas

CALL Instruction Format



J

Unconditional Jump

23	5	4	3	0	
imm[17..0]	00	0110	J	$offset \leftarrow sign_extend(imm)$ $PC \leftarrow PC + offset + 4$	

:
j <label> ; **salta a <label>**
:
<label>:

Instrucciones de Salto y Llamadas

CALL Instruction Format



CALL0

Non-windowed Call

23	5 4 3 0			
imm[17..0]	00	0101	CALL0	$AR[0] \leftarrow next(PC)$ $offset \leftarrow sign_extend(imm)$ $PC \leftarrow (PC_{31..2} + offset_{31..0} + 1)_{31..2} 0^2$

```
:  
call0 <label> ; invoca al procedimiento <label>  
:  
<label>:
```

Instrucciones de Salto y Llamadas

CALLX Instruction Format

23	20	19	16	15	12	11	8	7	6	5	4	3	0
op2	op1		r		s		m	n		op0			

JX Unconditional Jump Register

0000	0000	0000	s	10	10	0000	JX	$PC \leftarrow AR[s]$
------	------	------	---	----	----	------	----	-----------------------

```
:  
jx  a5    ; salta a la dirección de programa dada por  
; el registro a5  
:
```

Instrucciones de Salto y Llamadas

CALLX Instruction Format

23	20	19	16	15	12	11	8	7	6	5	4	3	0
op2		op1		r		s		m		n		op0	

CALLX0 Non-windowed Call Register

23	20	19	16	15	12	11	8	7	6	5	4	3	0
0000	0000	0000		s	11	00	0000	CALLX0	AR[0] \leftarrow next(PC) PC \leftarrow AR[s]				

:
callx0 a3 ; invoca al procedimiento que se encuentra en
; la dirección dada por el registro a3
:

Instrucciones de Salto y Llamadas

CALLX Instruction Format

23	20	19	16	15	12	11	8	7	6	5	4	3	0
op2		op1		r		s		m		n		op0	

RET

Non-windowed Return

23	20	19	16	15	12	11	8	7	6	5	4	3	0
0000		0000		0000		0000		10	00	00	0000	RET	PC \leftarrow AR[0]

:

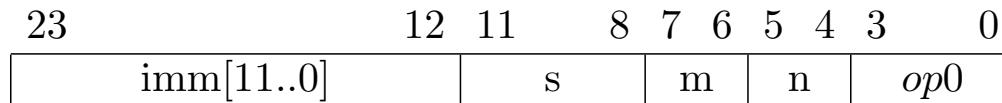
ret ; regresar a la sección que invocó al procedimiento

Instrucciones de Saltos Condicionados

Instruction	Format	Definition
BEQZ	BRI12	Branch if equal to zero
BNEZ	BRI12	Branch if not equal to zero
BGEZ	BRI12	Branch if greater than or equal to zero
BLTZ	BRI12	Branch if less than zero
BEQI	BRI8	Branch if equal immediate ¹
BNEI	BRI8	Branch if not equal immediate ¹
BGEI	BRI8	Branch if greater than or equal immediate ¹
BLTI	BRI8	Branch if less than immediate ¹
BGEUI	BRI8	Branch if greater than or equal unsigned immediate ²
BLTUI	BRI8	Branch if less than unsigned immediate ²
BBCI	RRI8	Branch if bit clear immediate
BBSI	RRI8	Branch if bit set immediate
BEQ	RRI8	Branch if equal
BNE	RRI8	Branch if not equal
BGE	RRI8	Branch if greater than or equal
BLT	RRI8	Branch if less than
BGEU	RRI8	Branch if greater than or equal unsigned
BLTU	RRI8	Branch if less than Unsigned
BANY	RRI8	Branch if any of masked bits set
BNONE	RRI8	Branch if none of masked bits set (All Clear)
BALL	RRI8	Branch if all of masked bits set
BNALL	RRI8	Branch if not all of masked bits set
BBC	RRI8	Branch if bit clear
BBS	RRI8	Branch if bit set

Instrucciones de Saltos condicionados

BRI12 Instruction Format



BEQZ

Branch if Equal to Zero

23	12	11	8	7	4	3	0		
imm[11..0]	s	0001	0110	BEQZ	$offset \leftarrow sign_extend(imm)$ $condition \leftarrow (AR[s] = 0^{32})$ if condition then $PC \leftarrow PC + offset + 4$ endif				

:
beqz a3,<label> ; salta a <label> si a3 es cero
:

Instrucciones de Saltos condicionados

BRI8 Instruction Format

23	16 15	12 11	8 7	6 5	4	3	0
imm[7..0]	r	s	m	n	<i>op0</i>		

BEQI

Branch if Equal Immediate

23	16	15	12	11	8	7	4	3	0
imm[7..0]		r		s	0010	0110	<i>BEQI</i>		<i>offset ← sign.extend(imm)</i> <i>condition ← (AR[t] = B4Const[r])</i> <i>if condition then</i> <i>PC ← PC + offset + 4</i> <i>endif</i>

:

beqi a3, **64** <label> ; salta a <label> si a3 igual a **64**
; (B4Const[13] es 64)

Instrucciones de Saltos condicionados

B4const

Encoding	Decimal Value of Immediate	Hex Value of Immediate
0	-1	32'hFFFFFFF
1	1	32'h00000001
2	2	32'h00000002
3	3	32'h00000003
4	4	32'h00000004
5	5	32'h00000005
6	6	32'h00000006
7	7	32'h00000007
8	8	32'h00000008
9	10	32'h0000000A
10	12	32'h0000000C
11	16	32'h000000010
12	32	32'h000000020
13	64	32'h000000040
14	128	32'h000000080
15	256	32'h000000100

```
int B4const(uint imm) {  
    const int B4constValues[16] = {-1,1,2,3,4,5,6,7,8,10,12,16,32,64,128,256};  
    return B4constValues[imm];  
}
```

Instrucciones de Saltos condicionados

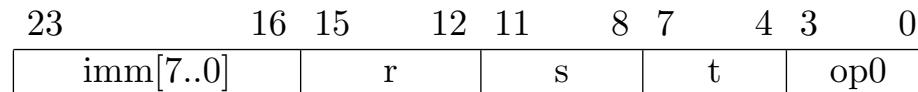
B4constu

Encoding	Decimal Value of Immediate	Hex Value of Immediate
0	32768	32'h00008000
1	65536	32'h00010000
2	2	32'h00000002
3	3	32'h00000003
4	4	32'h00000004
5	5	32'h00000005
6	6	32'h00000006
7	7	32'h00000007
8	8	32'h00000008
9	10	32'h0000000A
10	12	32'h0000000C
11	16	32'h00000010
12	32	32'h00000020
13	64	32'h00000040
14	128	32'h00000080
15	256	32'h00000100

```
uint B4constu(uint imm) {
    const int B4constuValues[16] = {32768,65536,2,3,4,5,6,7,8,10,12,16,32,64,128,256};
    return B4constuValues[imm];
}
```

Instrucciones de Saltos condicionados

RRI8 Instruction Format



BEQ

Branch if Equal

23	16	15	12	11	8	7	4	3	0
imm[7..0]		0001		s		t	0111	BEQ	<i>offset ← sign_extend(imm)</i> <i>condition ← (AR[t] = AR[s])</i> if condition then <i>PC ← PC + offset + 4</i> endif

:

beq a3, a4, <label> ; salta a <label> si registro a3
; es igual al registro a4

Instrucciones de Saltos condicionados

RRI8 Instruction Format

23	16 15	12 11	8 7	4 3	0
imm[7..0]	r	s	t	op0	

BBC Branch if Bit Clear

23	16 15	12 11	8	7	4 3	0
imm[7..0]	0101	s	t	0111	BBC	$offset \leftarrow sign_extend(imm)$ $bit \leftarrow AR[t]_{4..0}$ $condition \leftarrow AR[s]_{bit} = 0$ if condition then $PC \leftarrow PC + offset + 4$ endif

```
:  
bbc a3,a4, <label> ; salta a <label> si el bit del registro a3  
; que está en la posición dada por registro a4  
; es cero
```

Instrucciones de Saltos condicionados

RRI8 Instruction Format

23	16 15	12 11	8 7	4 3	0
imm[7..0]	r	s	t	op0	

BBS

Branch if Bit Set

23	16 15	12 11	8 7	4 3	0
imm[7..0]	1101	s	t	0111	<i>BBS</i>

offset \leftarrow sign_extend(*imm*)
bit \leftarrow *AR*[*t*]_{4..0}
condition \leftarrow *AR*[*s*]_{*bit*} \neq 0
if condition then
 PC \leftarrow *PC* + *offset* + 4
endif

:

bbs a3,a4, <label> ; salta a <label> si el bit del registro a3
; que está en la posición dada por registro a4
; es uno

Instrucciones de Saltos condicionados

RRI8 Instruction Format

23	16	15	12	11	8	7	4	3	0
imm[7..0]	r	s	t	op0					

BBCI Branch if Bit Clear Immediate

23	16	15	12	11	8	7	4	3	0	
imm[7..0]	011b[4]	s	b[3..0]	0111	BBCI	$offset \leftarrow signe_extend(imm)$ $condition \leftarrow AR[s]_b = 0$ if condition then $PC \leftarrow PC + offset + 4$ endif				

```
:  
bbei a3,20, <label> ; salta a <label> si el bit del registro a3  
    : ; que está en la posición 20 es cero  
    : ; (el número de bit estar en el rango [0..31])
```

Instrucciones de Saltos condicionados

RRI8 Instruction Format

23	16 15	12 11	8 7	4 3	0
imm[7..0]	r	s	t	op0	

BBSI

Branch if Bit Set Immediate

23	16 15	12 11	8 7	4 3	0	
imm[7..0]	111b[4]	s	b[3..0]	0111	BBSI	$offset \leftarrow sign_extend(imm)$ $condition \leftarrow AR[s]_b \neq 0$ if condition then $PC \leftarrow PC + offset + 4$ endif

:

```
bbsi a3,4, <label> ; salta a <label> si el bit del registro a3
: ; que está en la posición 4 es uno
: ; (el número de bit den estar en el rango [0..31])
```

Instrucciones de Saltos condicionados

RRI8 Instruction Format

23	16	15	12	11	8	7	4	3	0
imm[7..0]		r		s		t		op0	

BALL Branch if All Bits Set

23	16	15	12	11	8	7	4	3	0
imm[7..0]	0100		s		t	0111	BALL		$offset \leftarrow sign_extend(imm)$ $condition \leftarrow (\text{NOT AR}[s]) \text{ AND } \text{AR}[t] \neq 0^{32}$ if condition then $PC \leftarrow PC + offset + 4$ endif

:

```
ball a3,a4,<label> ; salta a <label> si el todos los bits indicados
; activados en el registro a4 está activados en
; el registro a4.
```

Instrucciones de Saltos condicionados

RRI8 Instruction Format

23	16 15	12 11	8 7	4 3	0
imm[7..0]	r	s	t	op0	

BALL Branch if All Bits Set

23	16 15	12 11	8	7	4 3	0	
imm[7..0]	0100	s		t	0111	BALL	$offset \leftarrow sign_extend(imm)$ $condition \leftarrow (\text{NOT } AR[s]) \text{ AND } AR[t] = 0^{32}$ if condition then $PC \leftarrow PC + offset + 4$ endif

```
ball a3,a4,<label> ; salta a <label> si todos los bits indicados
; (activados) en el registro a4 estan activados
; en el registro a3
```

Instrucciones de Saltos condicionados

RRI8 Instruction Format

23	16 15	12 11	8 7	4 3	0
imm[7..0]	r	s	t	op0	

BANY Branch if Any Bit Set

23	16 15	12 11	8 7	4 3	0	
imm[7..0]	1000	s	t	0111	BANY	$offset \leftarrow sign_extend(imm)$ $condition \leftarrow (AR[s] \text{ AND } AR[t]) \neq 0^{32}$ if condition then $PC \leftarrow PC + offset + 4$ endif

```
bany a3,a4,<label> ; salta a <label> si cualquier de los bits
; indicados (activados) en el registro a4
; está activado en el registro a3.
```

Instrucciones de Saltos condicionados

RRI8 Instruction Format

23	16 15	12 11	8 7	4 3	0
imm[7..0]	r	s	t	op0	

BNALL

Branch if Not-All Bits Set

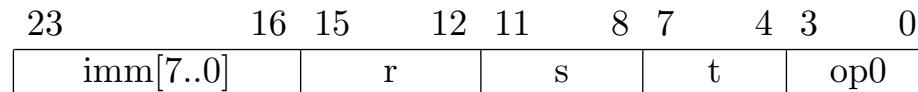
23 16 15 12 11 8 7 4 3 0

imm[7..0]	1100	s	t	0111	<i>BNALL</i>	$offset \leftarrow sign_extend(imm)$ $condition \leftarrow (\text{NOT AR}[s]) \text{ AND AR}[t] \neq 0^{32}$ if condition then $PC \leftarrow PC + offset + 4$ endif
-----------	------	---	---	------	--------------	---

```
bnall a5,a4,<label> ; salta a <label> si no todos los bits
; indicados (activados) en el registro a4
; están activados en el registro a5
```

Instrucciones de Saltos condicionados

RRI8 Instruction Format



BNONE

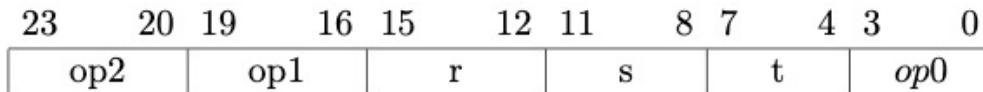
Branch if No Bits Set

23	16 15	12 11	8 7	4 3	0	
imm[7..0]	0000	s	t	0111	BNONE	$offset \leftarrow sign_extend(imm)$ $condition \leftarrow (AR[s] AND AR[t]) = 0^{32}$ if condition then $PC \leftarrow PC + offset + 4$ endif

```
bnone a5,a4,<label> ; salta a <label> si ninguno de los bits
; indicados (activados) en el registro a4
; están activados en el registro a5
```

Instrucciones de Corrimiento

RRR Instruction Format



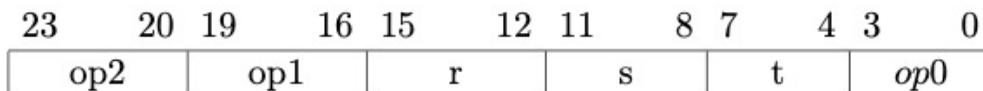
SRAI Shift Right Arithmetic Immediate

23	20	19	16	15	12	11	8	7	4	3	0
001sh[4]	0001		r	sh[3..0]		t	0000	SRAI		AR[r] \leftarrow AR[t]_31^sh AR[t]_31..sh	

```
srai a5,a4,3      ; corrimiento aritmético a la derecha de 3 lugares  
; sobre el registro a4 y el resultado se asigna  
; al registro a5
```

Instrucciones de Corrimiento

RRR Instruction Format



SSR

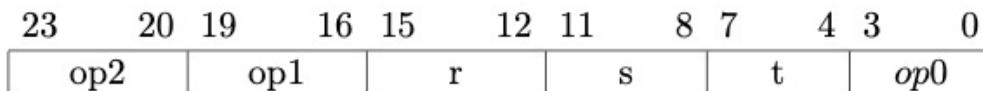
Set Shift Amount for Right Shift

23	20	19	16	15	12	11	8	7	4	3	0
0100	0000	0000		s	0000	0000	SSR		$sh \leftarrow 0 AR[s]_{4..0}$		$SAR \leftarrow sh$

```
ssr a3 ; copia el valor formado por los 5 bits menos significativos
; de a3 al registro SAR con el fin de que el registro SAR
; contenga la cantidad de corrimientos para futuras
; instrucciones de corrimientos a la derecha.
;
; Esto es equivalente a:
;                                     SA = a3 % 32
```

Instrucciones de Corrimiento

RRR Instruction Format



SSL

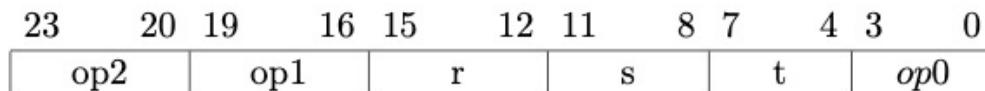
Set Shift Amount for Left Shift

23	20	19	16	15	12	11	8	7	4	3	0
1001	0001		r		0000		t		0000	SRL	$sh \leftarrow SAR_{5..0}$ $AR[r] \leftarrow 0^{sh} AR[t]_{31..sh}$

```
ssl a4 ; copia el valor formado por los 5 bits menos significativos  
; de a4 al registro SAR con el fin de que el registro SAR  
; contenga la cantidad de corrimientos para futuras  
; instrucciones de corrimientos a la izquierda.  
  
; Esto es equivalente a:  
;  $SAR = a4 \% 32$ 
```

Instrucciones de Corrimiento

RRR Instruction Format



SRA Shift Right Arithmetic

23	20	19	16	15	12	11	8	7	4	3	0
1011	0001		r	0000		t	0000		SRA	sh $\leftarrow SAR_{5..0}$	AR[r] $\leftarrow AR[t]^{sh}_{31} AR[t]_{31..sh}$

```
sra a4,a2      ; corrimiento aritmético a la derecha de SAR lugares  
; sobre el registro a2 y el resultado se asigna  
; al registro a4
```