Práctica 1:

Arquitectura y programación del procesador NIOS2/e

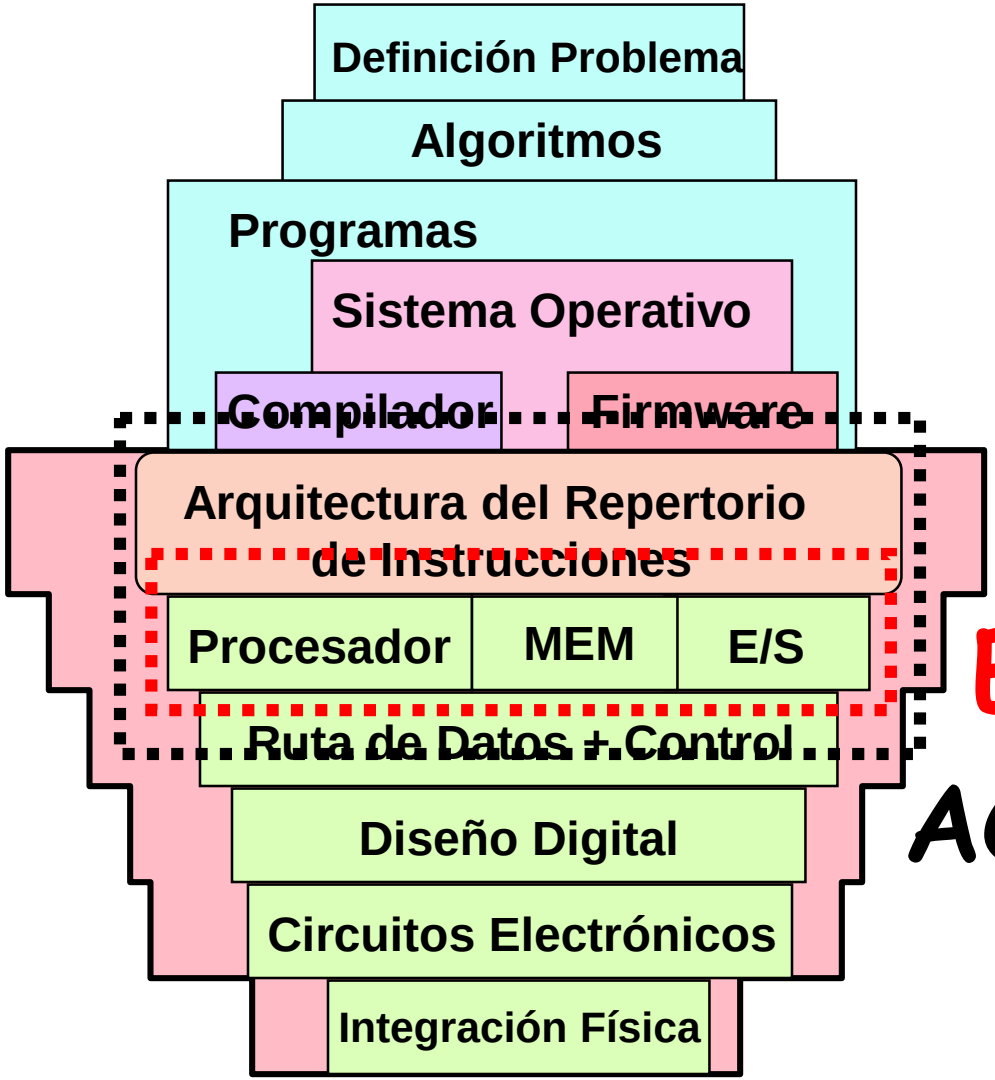
Estructura de Computadores

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Sumario

- Jerarquía de los niveles de abstracción del computador
- Elementos de la arquitectura del repertorio de instrucciones
- Modos de funcionamiento del procesador
- Registros de propósito general y de control
- Acceso al espacio de direccionamiento
- Tipos de instrucciones
- Ejemplo de programa en lenguaje ensamblador
- Subrutinas

Jerarquía de los niveles de abstracción del computador



Nivel de Abstracción

Ejemplo de la Descripción Jerárquica

Lenguaje de Programación

Programa de Lenguaje de alto Nivel

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compilador

Modelo del Programador de la Arquitectura Abstracta

Programa en Lenguaje Ensamblador

```
lw  $15,0($2)
lw  $16,4($2)
sw  $16,0($2)
sw  $15,4($2)
```

Ensamblador

Modelo Hardware de la Arquitectura Abstracta

Programa en Lenguaje Máquina

P

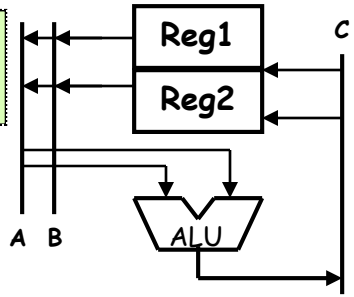
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

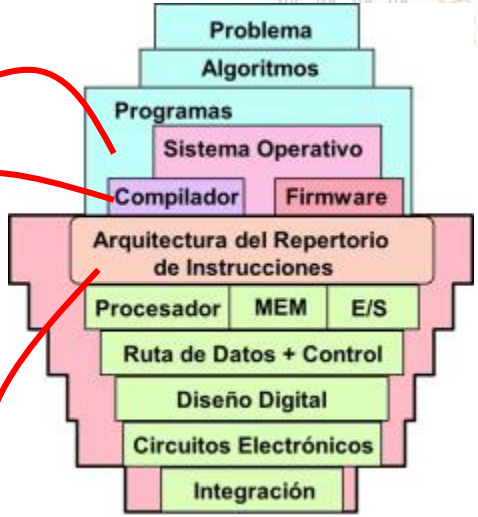
Arquitectura ISA del Repertorio de Instrucciones

Interpretación máquina → ALUOP[0:3] <= InstReg[9:11] & MASK

Microarquitectura Modelo Hardware de la Arquitectura Concreta

Especificación de la ruta de datos y el control

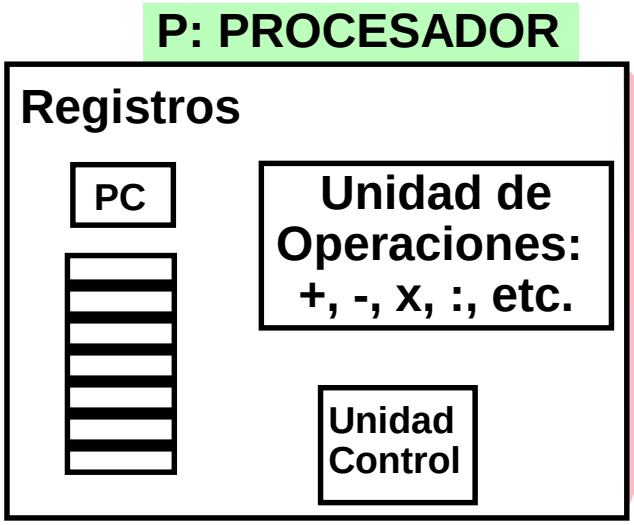


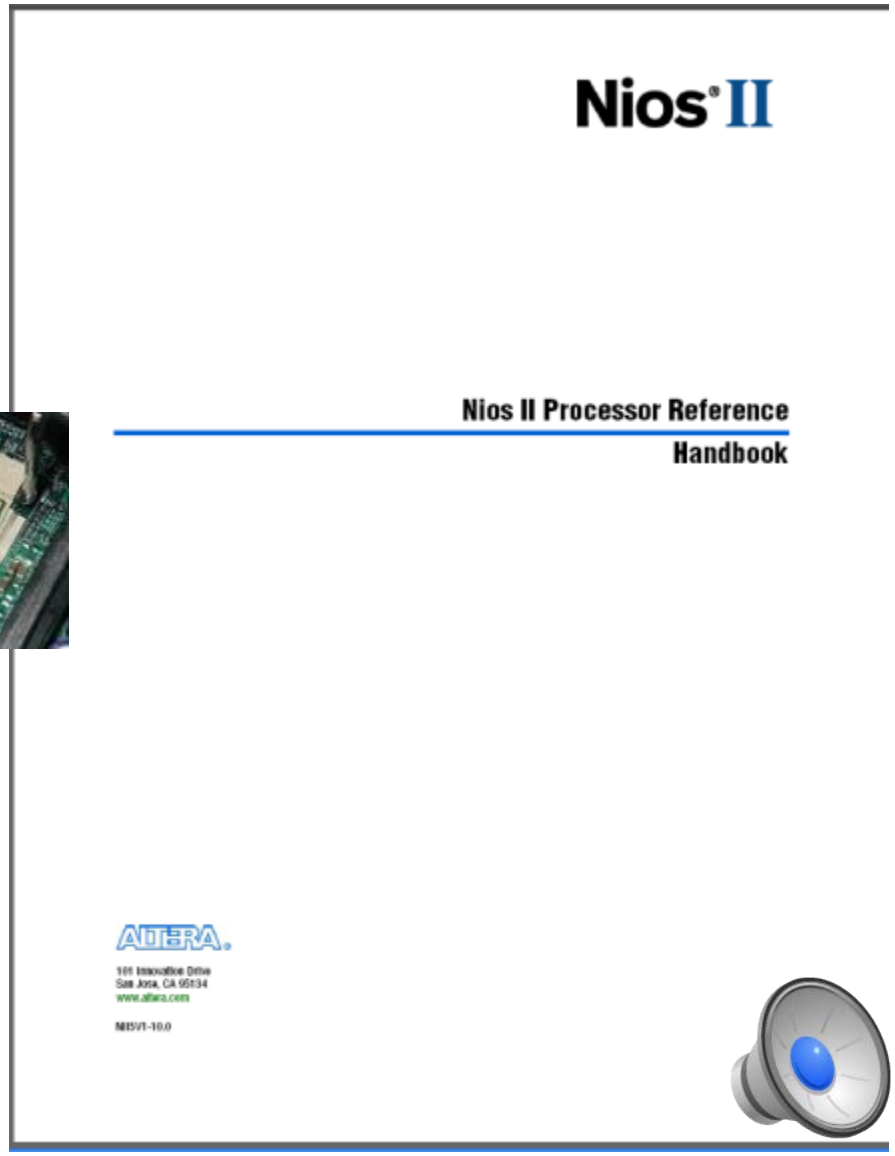


Arquitectura del Repertorio de Instrucciones (ISA)

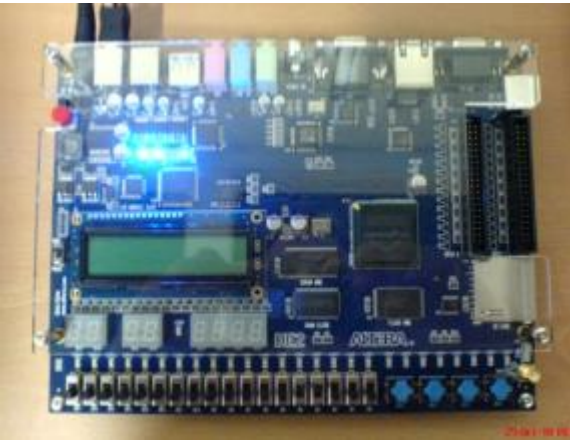
Elementos ISA, manejables por el Programador

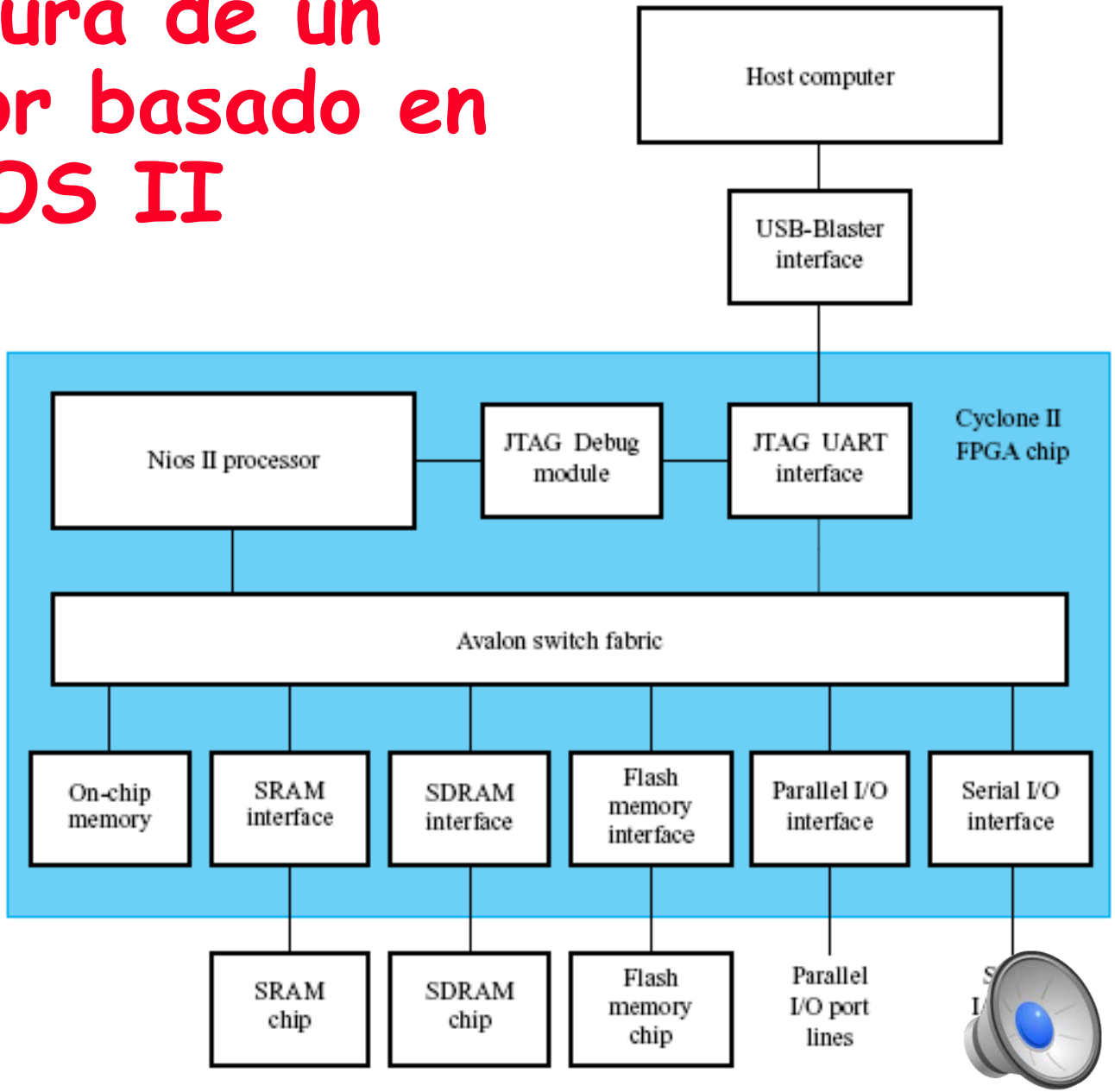
- Tipo de datos
- Estado de la máquina: Espacio de direccionamiento, Registros de datos, Registros de estados
- Acceso a los datos: tipo de datos, modo de direccionamiento
- Manipulación del estado: Inicialización de flags, Control (beq, ...), Manipulación del registro de estados
- 2 niveles: usuario, sistema
- Operaciones: tipos (suma, resta, punto flotante, etc.), instrucciones, su codificación, sus latencias, y lenguaje máquina
- Interrupciones hardware y software
- Operaciones de Entrada/Salida


P: PROCESADOR
Registros
PC
Unidad de Operaciones: +, -, x, :, etc.
Unidad Control


Nios II
Nios II Processor Reference Handbook
ALTERA
101 Innovation Drive
San Jose, CA 95134
www.altera.com
NIISV1-10.0

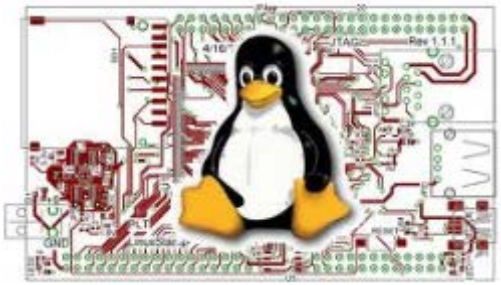
Estructura de un computador basado en NIOS II





Modos de funcionamiento NIOS II

Los "modos de funcionamiento" controlan cómo el procesador realiza todas sus funciones, gestiona el sistema de memoria, y accede a los dispositivos periféricos.



- Supervisor – puede ejecutar todas las funciones disponibles. Cuando reset=1 entra en este modo
- Usuario – ciertas funciones que pueden afectar al funcionamiento del procesador no están permitidas. Sólo cuando existe una MMU o MPU
- Depuración – permite utilizar breakpoints y watchpoints. Es un modo especial del modo supervisor

Registros de propósito general

Table 3–5. The Nios II General Purpose Registers

Register	Name	Function	Register	Name	Function
r0	zero	0x00000000	r16		
r1	at	Assembler temporary	r17		
r2		Return value	r18		
r3		Return value	r19		
r4		Register arguments	r20		
r5		Register arguments	r21		
r6		Register arguments	r22		
r7		Register arguments	r23		
r8		Caller-saved register	r24	et	Exception temporary
r9		Caller-saved register	r25	bt	Breakpoint temporary *(1)*
r10		Caller-saved register	r26	gp	Global pointer
r11		Caller-saved register	r27	sp	Stack pointer
r12		Caller-saved register	r28	fp	Frame pointer
r13		Caller-saved register	r29	ea	Exception return address
r14		Caller-saved register	r30	ba	Breakpoint return address *(2)*
r15		Caller-saved register	r31	ra	Return address

Registros de control: informan sobre el estado del procesador y cambian su comportamiento

SÓLO las instrucciones rdctl y wrctl leen y escriben en registros de control (ejecutables sólo en modo supervisor)

Table 3–6. Control Register Names and Bits

Register	Name	Register Contents
0	status	Refer to Table 3–7 on page 3–12
1	estatus	Refer to Table 3–9 on page 3–14
2	bstatus	Refer to Table 3–10 on page 3–15
3	ienable	Internal interrupt-enable bits *(3)*
4	ipending	Pending internal interrupt bits *(3)*
5	cpuid	Unique processor identifier
6	Reserved	Reserved
7	exception	Refer to Table 3–11 on page 3–16
8	pteaddr *(1)*	Refer to Table 3–13 on page 3–16
9	tlbacc *(1)*	Refer to Table 3–15 on page 3–17
10	tlbmisc *(1)*	Refer to Table 3–17 on page 3–18
11	Reserved	Reserved
12	badaddr	Refer to Table 3–19 on page 3–21
13	config *(2)*	Refer to Table 3–21 on page 3–21
14	mpubase *(2)*	Refer to Table 3–23 on page 3–22
15	mpuacc *(2)*	Refer to Table 3–25 on page 3–23
16–31	Reserved	Reserved

Memory Management Unit, MMU

Memory Protection Unit, MPU

Nombres reconocidos por el ensamblad (excepto "reserved")

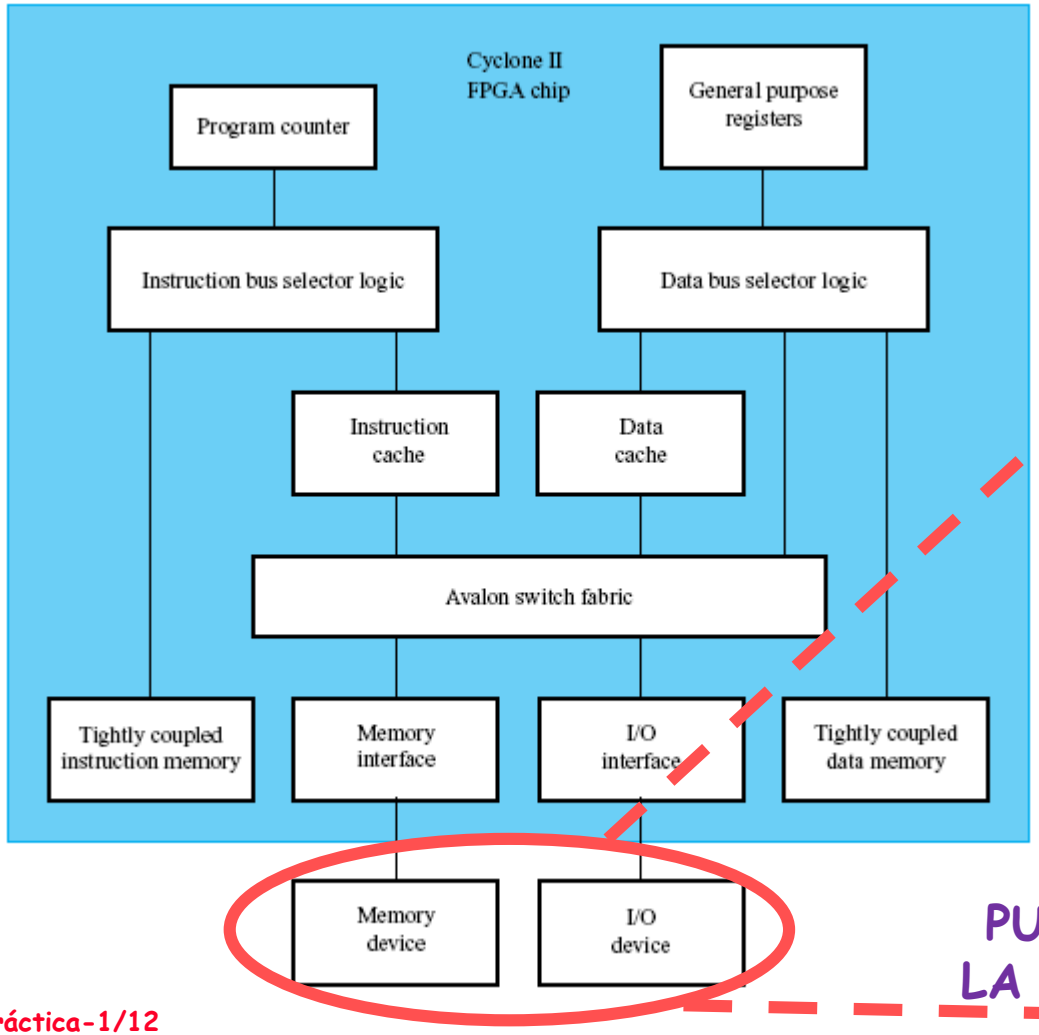
Registro status: estado del procesador NIOS II

Table 3–7. status Control Register Fields

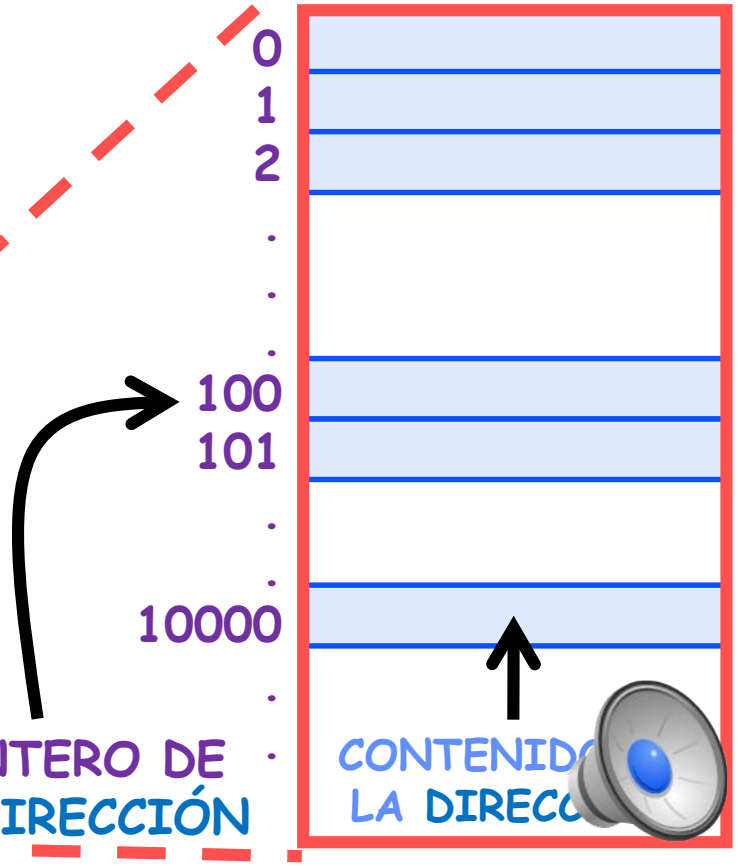
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								RSIE	NMI	PRS MMU						CRS						IL						IH	EH MMU	U	PIE

U: modo usuario (1) o supervisor (0)
PIE: habilitación de interrupciones
RSIE, NMI, IL, IH: control de interrupciones
PRS, EH: control de la MMU

Conexión con la memoria y los dispositivos de entrada/salida



ESPACIO DE DIRECCIONAMIENTO COMPARTIDO: MEMORIA Y ENTRADA/SALIDA



Modos de direccionamiento

- **Inmediato**: dato codificado en la propia instrucción
- **Registro**: dirección en un registro de propósito general
- NIOS II ⇨ **Desplazamiento**: dirección de memoria es la suma de inmediato 16-bit y contenido de registro
- **Indirecto**: posición de memoria que es apuntada por un registro de propósito general

Tipos de instrucciones en NIOS II según su formato

- **I-type**

31 30 29 28 27	26 25 24 23 22	21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6	5 4 3 2 1 0
A	B	IMM16	OP

- **R-type**

31 30 29 28 27	26 25 24 23 22	21 20 19 18 17	16 15 14 13 12 11 10 9 8 7 6	5 4 3 2 1 0
A	B	C	OPX	OP

- **J-type**

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6	5 4 3 2 1 0
IMMED26	OP

Códigos de operación (OP)

Table 8–1. OP Encodings

OP	Instruction	OP	Instruction	OP	Instruction	OP	Instruction
0x00	call	0x10	cmplti	0x20	cmpeqi	0x30	cmpltui
0x01	jmpi	0x11		0x21		0x31	
0x02		0x12		0x22		0x32	custom
0x03	ldbu	0x13	initda	0x23	ldbuio	0x33	initd
0x04	addi	0x14	ori	0x24	muli	0x34	orhi
0x05	stb	0x15	stw	0x25	stbio	0x35	stwio
0x06	br	0x16	blt	0x26	beq	0x36	bltu
0x07	ldb	0x17	ldw	0x27	ldbio	0x37	ldwio
0x08	cmpgei	0x18	cmpnei	0x28	cmpgeui	0x38	rdprs
0x09		0x19		0x29		0x39	
0x0A		0x1A		0x2A		0x3A	R-type
0x0B	ldhu	0x1B	flushda	0x2B	ldhuio	0x3B	flushd
0x0C	andi	0x1C	xori	0x2C	andhi	0x3C	xorhi
0x0D	sth	0x1D		0x2D	sthio	0x3D	
0x0E	bge	0x1E	bne	0x2E	bgeu	0x3E	
0x0F	ldh	0x1F		0x2F	ldhio	0x3F	

Table 8–2. OPX Encodings for R-Type Instructions (Part 1 of 2)

OPX	Instruction	OPX	Instruction	OPX	Instruction	OPX	Instruction
0x00		0x10	cmplt	0x20	cmpeq	0x30	cmpltu
0x01	eret	0x11		0x21		0x31	add
0x02	roli	0x12	slli	0x22		0x32	
0x03	rol	0x13	sll	0x23		0x33	
0x04	flushp	0x14	wrprs	0x24	divu	0x34	break
0x05	ret	0x15		0x25	div	0x35	
0x06	nor	0x16	or	0x26	rdctl	0x36	sync
0x07	mulxuu	0x17	mulxsu	0x27	mul	0x37	
0x08	cmpge	0x18	cmpne	0x28	cmpgeu	0x38	
0x09	bret	0x19		0x29	initi	0x39	sub
0x0A		0x1A	srli	0x2A		0x3A	srai
0x0B	ror	0x1B	srl	0x2B		0x3B	sra
0x0C	flushi	0x1C	nextpc	0x2C		0x3C	
0x0D	jmp	0x1D	callr	0x2D	trap	0x3D	
0x0E	and	0x1E	xor	0x2E	wrctl	0x3E	
0x0F		0x1F	mulxss	0x2F		0x3F	

Pseudoinstrucciones

Table 8–3. Assembler Pseudo-Instructions

Pseudo-Instruction	Equivalent Instruction
bgt rA, rB, label	blt rB, rA, label
bgtu rA, rB, label	bltu rB, rA, label
ble rA, rB, label	bge rB, rA, label
bleu rA, rB, label	bgeu rB, rA, label
cmpgt rC, rA, rB	cmplt rC, rB, rA
cmpgti rB, rA, IMMED	cmpgei rB, rA, (IMMED+1)
cmpgtu rC, rA, rB	cmpltu rC, rB, rA
cmpgtui rB, rA, IMMED	cmpgeui rB, rA, (IMMED+1)
cmple rC, rA, rB	cmpge rC, rB, rA
cmplei rB, rA, IMMED	cmplti rB, rA, (IMMED+1)
cmpleu rC, rA, rB	cmpgeu rC, rB, rA
cmpleui rB, rA, IMMED	cmpltui rB, rA, (IMMED+1)
mov rC, rA	add rC, rA, r0
movhi rB, IMMED	orhi rB, r0, IMMED
movi rB, IMMED	addi, rB, r0, IMMED
movia rB, label	orhi rB, r0, %hiadj(label) addi, rB, r0, %lo(label)
movui rB, IMMED	ori rB, r0, IMMED
nop	add r0, r0, r0
subi rB, rA, IMMED	addi rB, rA, (-IMMED)

Tipos de instrucciones según su operación: Acceso a memoria

Operaciones de 32 bits

Table 3–38. Wide Data Transfer Instructions

Instruction	Description
`ldw` `stw`	The `ldw` and `stw` instructions load and store 32-bit data words from/to memory. The effective address is the sum of a register's contents and a signed immediate value contained in the instruction. Memory transfers can be cached or buffered to improve program performance. This caching and buffering might cause memory cycles to occur out of order, and caching might suppress some cycles entirely. Data transfers for I/O peripherals should use `ldwio` and `stwio`.
`ldwio` `stwio`	`ldwio` and `stwio` instructions load and store 32-bit data words from/to peripherals without caching and buffering. Access cycles for `ldwio` and `stwio` instructions are guaranteed to occur in instruction order and are never suppressed.

The data transfer instructions in Table 3–39 support byte and half-word transfers.

Operaciones de 8 ó 16 bits

Table 3–39. Narrow Data Transfer Instructions

Instruction	Description

`ldb`
`ldbu`
`stb`
`ldh`
`ldhu`
`sth`
	`ldb`, `ldbu`, `ldh` and `ldhu` load a byte or half-word from memory to a register. `ldb` and `ldh` sign-extend the value to 32 bits, and `ldbu` and `ldhu` zero-extend the value to 32 bits.
	`stb` and `sth` store byte and half-word values, respectively.
	Memory accesses can be cached or buffered to improve performance. To transfer data to I/O peripherals, use the "io" versions of the instructions, described below.

`ldbio`
`ldbuio`
`stbio`
`ldhio`
`ldhuio`
`sthio`
	These operations load/store byte and half-word data from/to peripherals without caching or buffering.

Operación:

$rB \leftarrow Mem32[rA + \sigma(IMM16)]$

Sintaxis:

```
ldw rB, byte_offset(rA)

ldwio rB, byte_offset(rA)
```

Programa:

```
ldw r6, 100(r5)
```

Tipos de instrucciones según su operación: Aritmético-lógicas

Table 3–40. Arithmetic and Logical Instructions (Part 1 of 2)

Instruction	Description Operaciones lógicas
and or xor nor	These are the standard 32-bit logical operations. These operations take two register values and combine them bit-wise to form a result for a third register. Formato R
andi ori xori	These operations are immediate versions of the and, or, and xor instructions. The 16-bit immediate value is zero-extended to 32 bits, and then combined with a register value to form the result. Formato I
andhi orhi xorhi	In these versions of and, or, and xor, the 16-bit immediate value is shifted logically left by 16 bits to form a 32-bit operand. Zeroes are shifted in from the right. Formato I

Table 3–40. Arithmetic and Logical Instructions (Part 2 of 2)

Instruction	Description Operaciones aritméticas
add sub mul div divu	These are the standard 32-bit arithmetic operations. These operations take two registers as input and store the result in a third register. Formato R
addi subi muli	These instructions are immediate versions of the add, sub, and mul instructions. The instruction word includes a 16-bit signed value. Formato I
mulxss mulxuu	These instructions provide access to the upper 32 bits of a 32x32 multiplication operation. Choose the appropriate instruction depending on whether the operands should be treated as signed or unsigned values. It is not necessary to precede these instructions with a mul.
mulxsu	This instruction is used in computing a 128-bit result of a 64x64 signed multiplication.

Operación:

$rC \leftarrow rA + rB$

Sintaxis:

add rC, rA, rB

Programa:

add r6, r7, r8

Tipos de instrucciones según su operación: tipo es especial de aritmético-lógica, desplazamiento de datos

Table 3–43. Shift and Rotate Instructions

Instruction	Description

`rol`
`ror`
`roli`
	The `rol` and `roli` instructions provide left bit-rotation. `roli` uses an immediate value to specify the number of bits to rotate. The `ror` instructions provides right bit-rotation.
	There is no immediate version of `ror`, because `roli` can be used to implement the equivalent operation.

`sll`
`slli`
`sra`
`srl`
`srai`
`srli`
	These shift instructions implement the << and >> operators of the C programming language. The `sll`, `slli`, `srl`, `srli` instructions provide left and right logical bit-shifting operations, inserting zeros. The `sra` and `srai` instructions provide arithmetic right bit-shifting, duplicating the sign bit in the most significant bit. `slli`, `srli` and `srai` use an immediate value to specify the number of bits to shift.

Sintaxis:

rol rC, rA, rB

Programa:

rol r6, r7, r8

Tipos de instrucciones según su operación: copia entre registros

Table 3–41. Move Instructions

Instruction	Description

`mov`
`movhi`
`movi`
`movui`
`movia`
	`mov` copies the value of one register to another register. `movi` moves a 16-bit signed immediate value to a register, and sign-extends the value to 32 bits. `movui` and `movhi` move an immediate 16-bit value into the lower or upper 16-bits of a register, inserting zeros in the remaining bit positions. Use `movia` to load a register with an address.

Operación: rC ← rA

Sintaxis: mov rC, rA

Programa: mov r6, r7

Operación: rC ← label

Sintaxis: movia rC, label

Programa: movia r6, funcion

Tipos de instrucciones según su operación: comparación de datos

Table 3–42. Comparison Instructions (Part 1 of 2)

Instruction	Description
cmpeq	==
cmpne	!=
cmpge	signed >=
cmpgeu	unsigned >=
cmpgt	signed >
cmpgtu	unsigned >
cmple	unsigned <=
cmpleu	unsigned <=
cmplt	signed <

Operación: if (rA == rB)
then rC ← 1
else rC ← 0

Sintaxis: cmpeq rC, rA, rB

Programa: cmpeq r6, r7, r8

Table 3–42. Comparison Instructions (Part 2 of 2)

Instruction	Description
cmpltu	unsigned <
cmpeqi cmpnei cmpgei cmpgeui cmpgti cmpgtui cmplei cmpleui cmplti cmpltui	These instructions are immediate versions of the comparison operations. They compare the value of a register and a 16-bit immediate value. Signed operations sign-extend the immediate value to 32-bits. Unsigned operations fill the upper bits with zero.

Tipos de instrucciones según su operación: saltos incondicionales

Table 3–44. Unconditional Jump and Call Instructions (Part 1 of 2)

Instruction	Description
`call`	This instruction calls a subroutine using an immediate value as the subroutine's absolute address, and stores the return address in register `ra`.
`callr`	This instruction calls a subroutine at the absolute address contained in a register, and stores the return address in register `ra`. This instruction serves the roll of dereferencing a C function pointer.
`ret`	The `ret` instruction is used to return from subroutines called by `call` or `callr`. `ret` loads and executes the instruction specified by the address in register `ra`.
`jmp`	The `jmp` instruction jumps to an absolute address contained in a register. `jmp` is used to implement switch statements of the C programming language.

Table 3–44. Unconditional Jump and Call Instructions (Part 2 of 2)

Instruction	Description
`jmpi`	The `jmpi` instruction jumps to an absolute address using an immediate value to determine the absolute address.
~~`br`~~ **jr**	This instruction branches relative to the current instruction. A signed immediate value gives the offs next instruction to execute.

Tipos de instrucciones según su operación: saltos condicionales

Table 3–45. Conditional-Branch Instructions

Instruction	Description
bge bgeu bgt bgtu ble bleu blt bltu beq bne	These instructions provide relative branches that compare two register values and branch if the expression is true. Refer to "Comparison Instructions" on page 3–56 for a description of the relational operations implemented.

Operación: if ((signed) rA >= (signed) rB)
then PC← PC+4+σ(IMM16)
else PC← PC+4

Sintaxis: bge rA, rB, label

Programa: bge r6, r7, top_of_loop

Ejemplo de programa en lenguaje ensamblador

Zona de programa: 2 partes

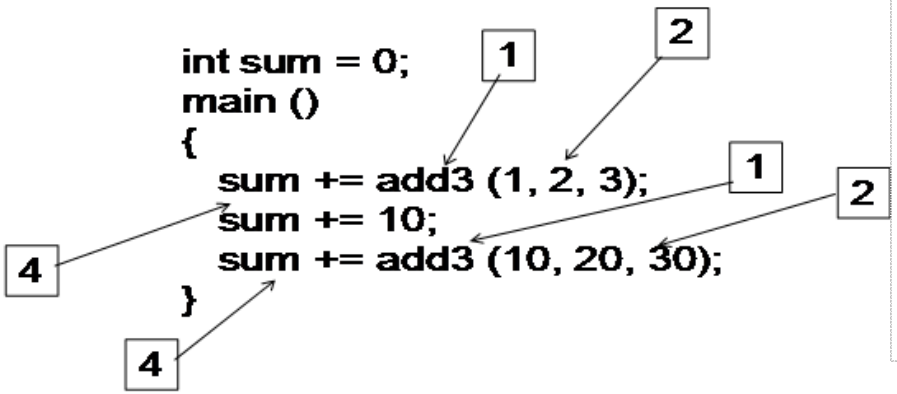
```
.include "nios_macros.s"
.global  _start
_start:
         movia  r2, AVECTOR          /* Register r2 is a pointer to vector A */
         movia  r3, BVECTOR          /* Register r3 is a pointer to vector B */
         movia  r4, N
         ldw    r4, 0(r4)            /* Register r4 is used as the counter for loop iterations */
         add    r5, r0, r0           /* Register r5 is used to accumulate the product */
LOOP:    ldw    r6, 0(r2)            /* Load the next element of vector A */
         ldw    r7, 0(r3)            /* Load the next element of vector B */
         mul    r8, r6, r7           /* Compute the product of next pair of elements */
         add    r5, r5, r8           /* Add to the sum */
         addi   r2, r2, 4            /* Increment the pointer to vector A */
         addi   r3, r3, 4            /* Increment the pointer to vector B */
         subi   r4, r4, 1            /* Decrement the counter */
         bgt    r4, r0, LOOP         /* Loop again if not finished */
         stw    r5, DOT_PRODUCT(r0)  /* Store the result in memory */
STOP:    br     STOP
```

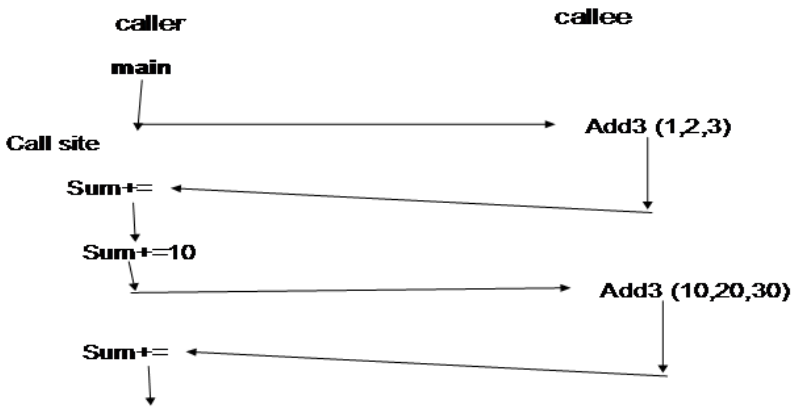
Zona de datos: 4 partes

```
N:
.word    6                           /* Specify the number of elements */
AVECTOR:
.word    5, 3, −6, 19, 8, 12         /* Specify the elements of vector A */
BVECTOR:
.word    2, 14, −3, 2, −5, 36        /* Specify the elements of vector B */
DOT_PRODUCT:
.skip    4
```

Subrutinas

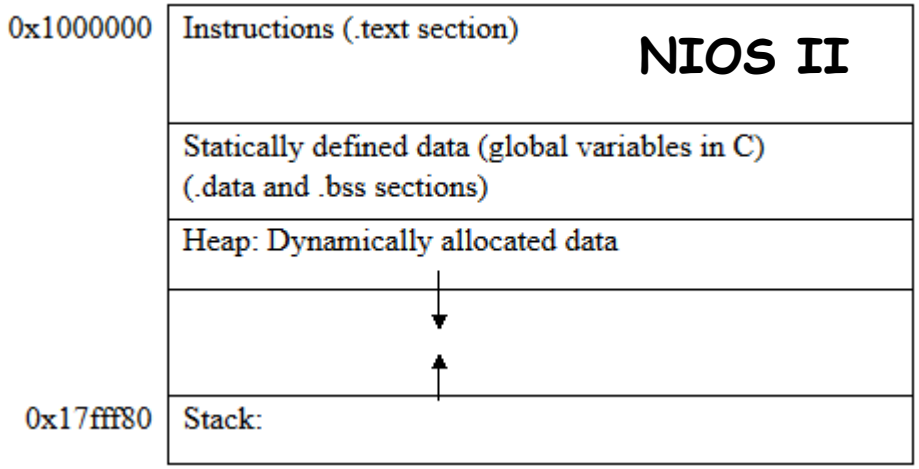
```
int add3 (int a, int b, int c)
{
return a  +  b +c;
}

int sum = 0;
main ()
{
   sum += add3 (1, 2, 3);
   sum += 10;
   sum += add3 (10, 20, 30);
}
```





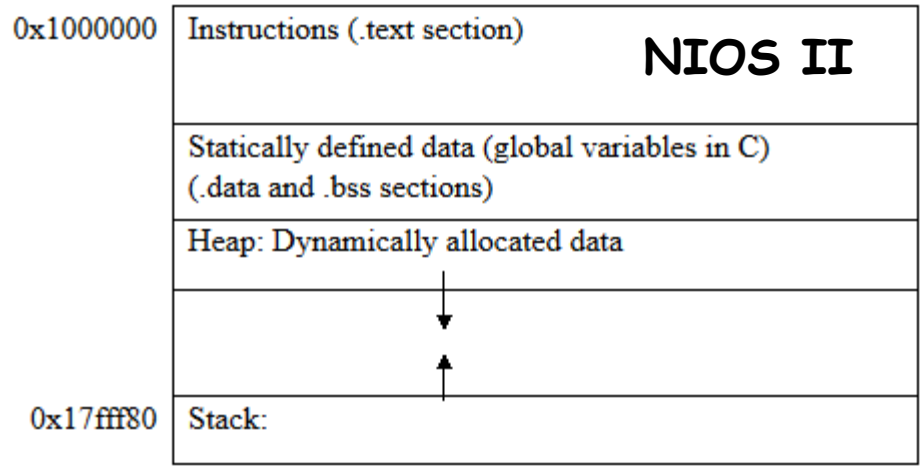
Pila (stack)

- Estructura de datos LIFO
- Puntero de pila: sp←0x017fff80
  - movia sp, 0x017fff80



Pila (stack)

- Operaciones:
  - Push
    - » stw r31, 0(sp)
    - » subi sp, sp, 4
  - Pop
    - » addi sp, sp, 4
    - » ldw r31, 0(sp)



Ejemplo

SP --> 0x6fff0	01	02	03	04
0x6fff4	10	20	30	40
0x6fff8	11	22	33	44
0x6fffc	55	66	77	88

ldw r9, 8(sp) -> r9 <-Mem32[sp+8] = 0x44332211

ldb r10, 0xd(sp) -> r10<-Mem8[sp+0xd]=

Subrutinas en ensamblador

```
.text
 boo:
        call coo
        ...
coo:
        ret
```

Llamada a subrutina;
equivalente a:
r31 = PC + 4
PC = coo

Retorno desde subrutina;
equivalente a salto jmp r31:
PC = r31

Llamadas anidadas a subrutinas

```
boo() {
   coo ();
   doo ();
}
void coo () {
   doo ();
   return;
}
void doo() {
   return;
}
```

Llamada SIMPLE a subrutina

Llamada SIMPLE a subrutina

Llamada ANIDADA a subrutina

Retorno de subrutina

```
.text
boo:        subi sp, sp, 4
            stw ra,0(sp)
            call coo
boo_ret1:
            call doo
boo_ret2:
            ret
coo:
            subi  sp, sp, 4
            stw   ra,0(sp)
            call  doo
coo_ret:
            ldw  ra, 0(sp)
            addi sp, sp, 4
            ret
doo:
            ret
```

Llamadas anidadas a subrutinas

```
 .text
main:       subi sp, sp, 4
            stw ra,0(sp)
            call coo
boo_ret1:
            call doo
boo_ret2:   ret
coo:
            subi  sp, sp, 4
            stw   ra,0(sp)
            call  doo
coo_ret:
            ldw  ra, 0(sp)
            addi sp, sp, 4
            ret
 doo:
            ret
```

PC | Código Instrucción | Instrucción

```
01000000 <main>:
 1000000:   deffff04   addi   sp,sp,-4
 1000004:   dfc00035   stw    ra,0(sp)
 1000008:   100001c0   call   100001c <coo>

0100000c <boo_ret1>:
 100000c:   10000340   call   1000034 <doo>

01000010 <boo_ret2>:
 1000010:   dfc00037   ldw    ra,0(sp)
 1000014:   dec00104   addi   sp,sp,4
 1000018:   f800683a   jmp    ra

0100001c <coo>:
 100001c:   deffff04   addi   sp,sp,-4
 1000020:   dfc00035   stw    ra,0(sp)
 1000024:   10000340   call   1000034 <doo>

01000028 <coo_ret>:
 1000028:   dfc00037   ldw    ra,0(sp)
 100002c:   dec00104   addi   sp,sp,4
 1000030:   f800683a   jmp    ra

01000034 <doo>:
 1000034:   f800683a   jmp    ra
```

Llamada a subrutina

Retorno de subrutina

Llamadas anidadas a subrutinas

movia sp, 0x20000c

Suponemos que la zona de pila empieza en 0x20000c

```
01000000 <boo>:
PC → 1000000:    defffff04    addi    sp,sp,-4
 1000004:        dfc00035     stwio   ra,0(sp)
 1000008:        100001c0     call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340     call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037     ldwio   ra,0(sp)
 1000014:        dec00104     addi    sp,sp,4
 1000018:        f800683a     jmp     ra
0100001c <coo>:
 100001c:        defffff04    addi    sp,sp,-4
 1000020:        dfc00035     stwio   ra,0(sp)
 1000024:        10000340     call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037     ldwio   ra,0(sp)
 100002c:        dec00104     addi    sp,sp,4
 1000030:        f800683a     jmp     ra
01000034 <doo>:
 1000034:        f800683a     jmp     ra
```

Suponemos que a boo se salta desde otra subrutina y ra tiene la dirección actual de retorno desde boo

0x200000	0x0
0x200004	0x0
0x200008	0x0
SP → 0x20000c	0x0

ra: 0x1000068

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
PC → 1000000:    defffff04    addi    sp,sp,-4
 1000004:        dfc00035     stwio   ra,0(sp)
 1000008:        100001c0     call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340     call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037     ldwio   ra,0(sp)
 1000014:        dec00104     addi    sp,sp,4
 1000018:        f800683a     jmp     ra
0100001c <coo>:
 100001c:        defffff04    addi    sp,sp,-4
 1000020:        dfc00035     stwio   ra,0(sp)
 1000024:        10000340     call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037     ldwio   ra,0(sp)
 100002c:        dec00104     addi    sp,sp,4
 1000030:        f800683a     jmp     ra
01000034 <doo>:
 1000034:        f800683a     jmp     ra
```

Se actualiza el puntero de la pila: SP

	0x200000	0x0
	0x200004	0x0
SP →	0x200008	0x0
	0x20000c	0x0

ra: 0x1000068

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:        defffff04        addi    sp,sp,-4
 1000004:        dfc00035         stwio   ra,0(sp)
PC ---> 1000008: 100001c0         call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340         call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037         ldwio   ra,0(sp)
 1000014:        dec00104         addi    sp,sp,4
 1000018:        f800683a         jmp     ra
0100001c <coo>:
 100001c:        defffff04        addi    sp,sp,-4
 1000020:        dfc00035         stwio   ra,0(sp)
 1000024:        10000340         call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037         ldwio   ra,0(sp)
 100002c:        dec00104         addi    sp,sp,4
 1000030:        f800683a         jmp     ra
01000034 <doo>:
 1000034:        f800683a         jmp     ra
```

Se guarda la dirección de retorno de ra en la pila: SP

Se llama a coo y se actualiza ra con la dirección de retorno cuando se retorne desde coo

0x200000	0x0	
0x200004	0x0	
SP → 0x200008	0x1000068	
0x20000c	0x0	

ra: 0x100000c

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:       defffff04       addi    sp,sp,-4
 1000004:       dfc00035        stwio   ra,0(sp)
 1000008:       100001c0        call    100001c <coo>
0100000c <boo_ret1>:
 100000c:       10000340        call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:       dfc00037        ldwio   ra,0(sp)
 1000014:       dec00104        addi    sp,sp,4
 1000018:       f800683a        jmp     ra
0100001c <coo>:
PC → 100001c:   defffff04       addi    sp,sp,-4
 1000020:       dfc00035        stwio   ra,0(sp)
 1000024:       10000340        call    1000034 <doo>
01000028 <coo_ret>:
 1000028:       dfc00037        ldwio   ra,0(sp)
 100002c:       dec00104        addi    sp,sp,4
 1000030:       f800683a        jmp     ra
01000034 <doo>:
 1000034:       f800683a        jmp     ra
```

Se actualiza el puntero de la pila: SP dentro de coo

	0x200000	0x0
SP →	0x200004	0x0
	0x200008	0x1000068
	0x20000c	0x0

ra: 0x100000c

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:        defffff04        addi    sp,sp,-4
 1000004:        dfc00035         stwio   ra,0(sp)
 1000008:        100001c0         call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340         call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037         ldwio   ra,0(sp)
 1000014:        dec00104         addi    sp,sp,4
 1000018:        f800683a         jmp     ra
0100001c <coo>:
 100001c:        defffff04        addi    sp,sp,-4
PC → 1000020:    dfc00035         stwio   ra,0(sp)
 1000024:        10000340         call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037         ldwio   ra,0(sp)
 100002c:        dec00104         addi    sp,sp,4
 1000030:        f800683a         jmp     ra
01000034 <doo>:
 1000034:        f800683a         jmp     ra
```

Se guarda la dirección de retorno de ra en la pila: SP

	0x200000	0x0
SP →	0x200004	0x100000c
	0x200008	0x1000068
	0x20000c	0x0

ra: 0x100000c

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:        defffff04        addi    sp,sp,-4
 1000004:        dfc00035         stwio   ra,0(sp)
 1000008:        100001c0         call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340         call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037         ldwio   ra,0(sp)
 1000014:        dec00104         addi    sp,sp,4
 1000018:        f800683a         jmp     ra
0100001c <coo>:
 100001c:        defffff04        addi    sp,sp,-4
 1000020:        dfc00035         stwio   ra,0(sp)
PC → 1000024:    10000340         call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037         ldwio   ra,0(sp)
 100002c:        dec00104         addi    sp,sp,4
 1000030:        f800683a         jmp     ra
01000034 <doo>:
 1000034:        f800683a         jmp     ra
```

Se llama a doo y se actualiza ra con la dirección de retorno cuando se retorne desde doo

	0x200000	0x0
SP →	0x200004	0x100000c
	0x200008	0x1000068
	0x20000c	0x0

ra: 0x1000028

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:       defffff04       addi    sp,sp,-4
 1000004:       dfc00035        stwio   ra,0(sp)
 1000008:       100001c0        call    100001c <coo>
0100000c <boo_ret1>:
 100000c:       10000340        call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:       dfc00037        ldwio   ra,0(sp)
 1000014:       dec00104        addi    sp,sp,4
 1000018:       f800683a        jmp     ra
0100001c <coo>:
 100001c:       defffff04       addi    sp,sp,-4
 1000020:       dfc00035        stwio   ra,0(sp)
 1000024:       10000340        call    1000034 <doo>
01000028 <coo_ret>:
 1000028:       dfc00037        ldwio   ra,0(sp)
 100002c:       dec00104        addi    sp,sp,4
 1000030:       f800683a        jmp     ra
01000034 <doo>:
 1000034:       f800683a        jmp     ra
```

PC → 1000034

Se retorna desde doo utilizando ra actual

0x200000	0x0
0x200004	0x100000c
0x200008	0x1000068
0x20000c	0x0

SP → 0x200004

ra: 0x1000028

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:        defffff04        addi    sp,sp,-4
 1000004:        dfc00035         stwio   ra,0(sp)
 1000008:        100001c0         call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340         call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037         ldwio   ra,0(sp)
 1000014:        dec00104         addi    sp,sp,4
 1000018:        f800683a         jmp     ra
0100001c <coo>:
 100001c:        defffff04        addi    sp,sp,-4
 1000020:        dfc00035         stwio   ra,0(sp)
 1000024:        10000340         call    1000034 <doo>
01000028 <coo_ret>:
PC → 1000028:    dfc00037         ldwio   ra,0(sp)
 100002c:        dec00104         addi    sp,sp,4
 1000030:        f800683a         jmp     ra
01000034 <doo>:
 1000034:        f800683a         jmp     ra
```

Se actualiza ra cuando se va a retornar desde coo

	0x200000	0x0
SP →	0x200004	0x100000c
	0x200008	0x1000068
	0x20000c	0x0

ra: 0x100000c

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:        defffff04        addi    sp,sp,-4
 1000004:        dfc00035         stwio   ra,0(sp)
 1000008:        100001c0         call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340         call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037         ldwio   ra,0(sp)
 1000014:        dec00104         addi    sp,sp,4
 1000018:        f800683a         jmp     ra
0100001c <coo>:
 100001c:        defffff04        addi    sp,sp,-4
 1000020:        dfc00035         stwio   ra,0(sp)
 1000024:        10000340         call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037         ldwio   ra,0(sp)
PC → 100002c:    dec00104         addi    sp,sp,4
 1000030:        f800683a         jmp     ra
01000034 <doo>:
 1000034:        f800683a         jmp     ra
```

Se actualiza el puntero de pila SP

	0x200000	0x0
	0x200004	0x100000c
SP →	0x200008	0x1000068
	0x20000c	0x0

ra: 0x100000c

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:       defffff04       addi    sp,sp,-4
 1000004:       dfc00035        stwio   ra,0(sp)
 1000008:       100001c0        call    100001c <coo>
0100000c <boo_ret1>:
 100000c:       10000340        call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:       dfc00037        ldwio   ra,0(sp)
 1000014:       dec00104        addi    sp,sp,4
 1000018:       f800683a        jmp     ra
0100001c <coo>:
 100001c:       defffff04       addi    sp,sp,-4
 1000020:       dfc00035        stwio   ra,0(sp)
 1000024:       10000340        call    1000034 <doo>
01000028 <coo_ret>:
 1000028:       dfc00037        ldwio   ra,0(sp)
 100002c:       dec00104        addi    sp,sp,4
PC → 1000030:   f800683a        jmp     ra
01000034 <doo>:
 1000034:       f800683a        jmp     ra
```

Se retorna desde coo

	0x200000	0x0
	0x200004	0x100000c
SP →	0x200008	0x1000068
	0x20000c	0x0

ra: 0x100000c

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:        defffff04        addi    sp,sp,-4
 1000004:        dfc00035         stwio   ra,0(sp)
 1000008:        100001c0         call    100001c <coo>
0100000c <boo_ret1>:
PC → 100000c:    10000340         call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037         ldwio   ra,0(sp)
 1000014:        dec00104         addi    sp,sp,4
 1000018:        f800683a         jmp     ra
0100001c <coo>:
 100001c:        defffff04        addi    sp,sp,-4
 1000020:        dfc00035         stwio   ra,0(sp)
 1000024:        10000340         call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037         ldwio   ra,0(sp)
 100002c:        dec00104         addi    sp,sp,4
 1000030:        f800683a         jmp     ra
01000034 <doo>:
 1000034:        f800683a         jmp     ra
```

Se llama a doo y se guarda dirección de retorno en ra

	0x200000	0x0
	0x200004	0x100000c
SP →	0x200008	0x1000068
	0x20000c	0x0

ra: 0x1000010

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:        defffff04        addi    sp,sp,-4
 1000004:        dfc00035         stwio   ra,0(sp)
 1000008:        100001c0         call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340         call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037         ldwio   ra,0(sp)
 1000014:        dec00104         addi    sp,sp,4
 1000018:        f800683a         jmp     ra
0100001c <coo>:
 100001c:        defffff04        addi    sp,sp,-4
 1000020:        dfc00035         stwio   ra,0(sp)
 1000024:        10000340         call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037         ldwio   ra,0(sp)
 100002c:        dec00104         addi    sp,sp,4
 1000030:        f800683a         jmp     ra
01000034 <doo>:
PC → 1000034:    f800683a         jmp     ra
```

Se retorna desde doo a boo utilizando ra actual que ahora es distinto de cuando se retornó desde doo a coo

0x200000	0x0	
0x200004	0x100000c	
SP → 0x200008	0x1000068	
0x20000c	0x0	

ra: 0x1000010

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:        deffff04        addi    sp,sp,-4
 1000004:        dfc00035        stwio   ra,0(sp)
 1000008:        100001c0        call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340        call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037        ldwio   ra,0(sp)
 1000014:        dec00104        addi    sp,sp,4
 1000018:        f800683a        jmp     ra
0100001c <coo>:
 100001c:        deffff04        addi    sp,sp,-4
 1000020:        dfc00035        stwio   ra,0(sp)
 1000024:        10000340        call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037        ldwio   ra,0(sp)
 100002c:        dec00104        addi    sp,sp,4
 1000030:        f800683a        jmp     ra
01000034 <doo>:
 1000034:        f800683a        jmp     ra
```

PC → 1000014

Se actualiza ra y SP

0x200000	0x0
0x200004	0x100000c
0x200008	0x1000068
SP → 0x20000c	0x0

ra	0x1000068

Llamadas anidadas a subrutinas

movia sp, 0x20000c

```
01000000 <boo>:
 1000000:        defffff04        addi    sp,sp,-4
 1000004:        dfc00035         stwio   ra,0(sp)
 1000008:        100001c0         call    100001c <coo>
0100000c <boo_ret1>:
 100000c:        10000340         call    1000034 <doo>
01000010 <boo_ret2>:
 1000010:        dfc00037         ldwio   ra,0(sp)
 1000014:        dec00104         addi    sp,sp,4
PC → 1000018:    f800683a         jmp     ra
0100001c <coo>:
 100001c:        defffff04        addi    sp,sp,-4
 1000020:        dfc00035         stwio   ra,0(sp)
 1000024:        10000340         call    1000034 <doo>
01000028 <coo_ret>:
 1000028:        dfc00037         ldwio   ra,0(sp)
 100002c:        dec00104         addi    sp,sp,4
 1000030:        f800683a         jmp     ra
01000034 <doo>:
 1000034:        f800683a         jmp     ra
```

Se retorna a la rutina que llamó a boo

	0x200000	0x0
	0x200004	0x100000c
	0x200008	0x1000068
SP →	0x20000c	0x0

ra	0x1000068