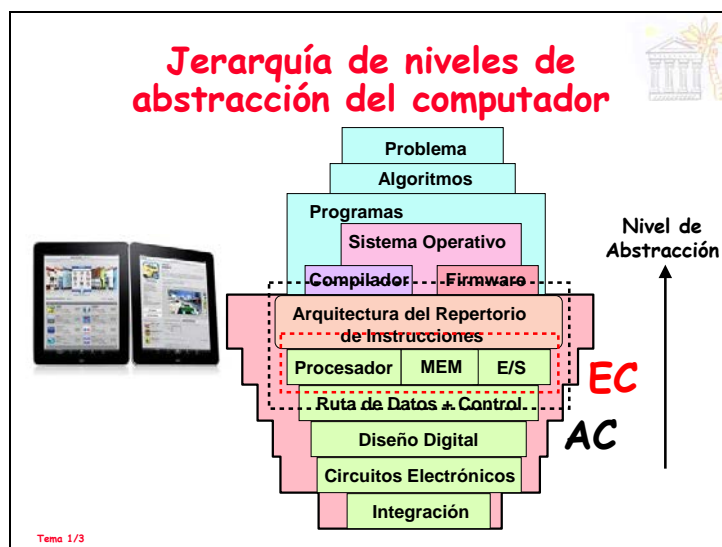


# Práctica 1: Arquitectura y programación del procesador NIOS2/e

## 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

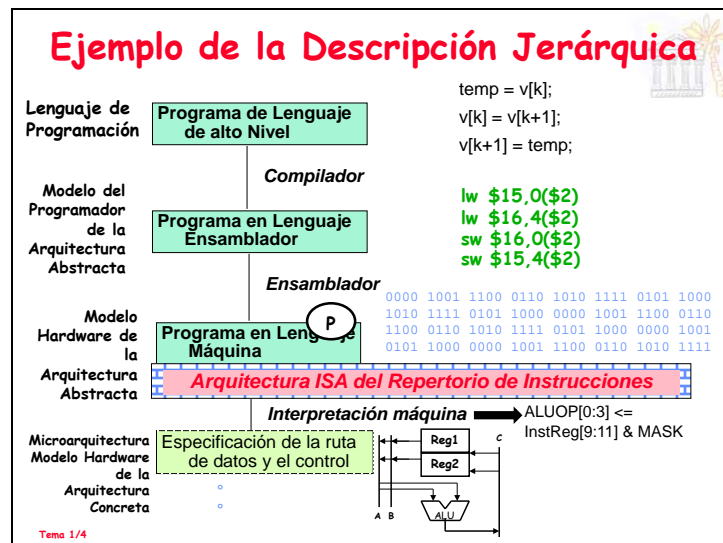


En el concepto de **diseño y construcción de computadores** se toma en consideración desde los tipos de problemas a los que va destinada la utilización del computador hasta la construcción física de cada una de las partes del computador, de mayor nivel de abstracción (definición del problema) hasta el menor nivel de abstracción (integración física). En la figura de esta transparencia se pueden observar los distintos niveles que se identifican en la representación de los elementos tanto software como hardware que forman parte de un computador genérico; casi todos los

nombres de los niveles corresponden a materias que se estudian en el Grado en Ingeniería Informática.

La materia relacionada con Arquitectura de Computadores (AC) incluye desde el diseño de la microarquitectura (nombre que recibe la unión de “ruta de datos + unidad de control”) hasta el software que gestiona directamente el hardware. Este software incluye: la generación de código del compilador, el propio sistema operativo, el sistema básico de rutinas de entrada/salida (denominado “BIOS”), y los programas de alto nivel que gestionan el hardware sin intervención del sistema operativo como por ejemplo VTune. Dentro de la materia AC, la parte que denominamos “Estructura de Computadores (EC)” incluye la denominada “arquitectura del repertorio de instrucciones (ISA)” del procesador, la organización y gestión de la jerarquía de memoria, y la organización y gestión del subsistema de entrada/salida del computador.

En esta lección se describen los principales elementos de la Arquitectura del Repertorio de Instrucciones con especial énfasis en la que implementa el procesador NIOS II de Altera, el cual se utiliza en las clases prácticas de EC.



De la jerarquía de niveles mostrada en la transparencia anterior se extrae en esta transparencia los niveles involucrados en EC y que vamos a definir a continuación:

Lenguaje de alto nivel: es un conjunto de expresiones y directivas expresadas en un lenguaje de programación que se utilizan para representar a programas fuentes que posteriormente se transforman en programas

codificados en lenguaje ensamblador. En la transparencia se observan tres expresiones distintas que pueden formar parte de un programa expresado en un lenguaje de alto nivel.

Compilador: es un programa que permite codificar en lenguaje ensamblador un determinado programa fuente descrito en un lenguaje de alto nivel.

Lenguaje ensamblador: es conjunto de expresiones formado por las agrupaciones de siglas asignadas a las instrucciones del repertorio de instrucciones de un procesador, y además un conjunto de directivas que facilitan la generación del código máquina de los programas descritos en este lenguaje ensamblador. Este lenguaje ensamblador forma parte de la arquitectura del repertorio de instrucciones. Algunos autores le asignan también el nombre de “modelo de programador de la arquitectura abstracta”. En la transparencia se observan cuatro expresiones distintas que pueden formar parte de un programa expresado en un lenguaje ensamblador.

Ensamblador: es un programa que permite codificar en lenguaje máquina un determinado programa fuente descrito en lenguaje ensamblador.

Lenguaje máquina o Repertorio de instrucciones: conjunto de instrucciones en formato binario que representa a las órdenes que ejecuta un determinado procesador. Algunos autores le asignan también el nombre de “modelo hardware de la arquitectura abstracta”.

Instrucción: es el conjunto de bits que representa a una orden que puede ejecutar el hardware del procesador, a partir de la cual se construyen los programas descritos en lenguaje máquina. En la transparencia se observan cuatro instrucciones distintas que pueden formar parte de un programa expresado en un lenguaje máquina.

Arquitectura del repertorio de instrucciones: conjunto de elementos informáticos que se utiliza para implementar la interfaz entre el software y el hardware de un procesador. En inglés recibe el nombre de ISA (Instruction Set Architecture).

Microoperaciones: son aquellas actividades que se producen en el hardware del procesador cuyo objetivo es la realización de la orden asociada a una instrucción. En la transparencia se observa una microoperación descrita en lo que se denomina “lenguaje de transferencia de registros”: `ALUOP[0:3] ←`

InstReg[9:11] & MASK, la cual está indicando que la salida de la unidad funcional ALU del procesador produce un resultado de cuatro bits (ALUOP[0:3]) que corresponde al resultado de la operación lógica “and” sobre dos operandos, uno que consta de los bits 9 á 11 del registro interno del procesador denominado de “instrucción” (InstReg) y un patrón de bits fijo almacenado en un registro constante de 4 bits denominado “MASK”.


Ruta de datos: es la parte del hardware del procesador que se encarga de procesar la información contenida en las instrucciones de un programa.

Unidad de control: es la parte del hardware del procesador que en carga de dirigir a la ruta de datos para que realice las microoperaciones.

Microarquitectura: al conjunto formado por la ruta de datos y la unidad de control del procesador.

### 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



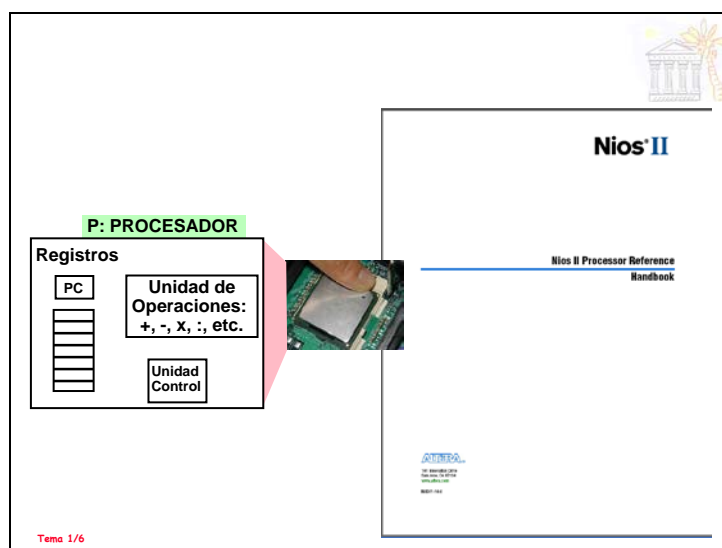
Tema 1/5

Se denomina **arquitectura del repertorio de instrucciones** o **arquitectura ISA** al conjunto formado por: repertorio de instrucciones del procesador, estructuras de datos que codifican la información dentro del procesador, registros del procesador accesibles desde el programa (tanto de propósito general como los de propósito específico, como por ejemplo los registros de estado), el espacio de direccionamiento del procesador, y los modos de direccionamiento de la memoria.

La arquitectura ISA proporciona un modelo del computador que el Programador utiliza para construir los programas, e incluye los siguientes elementos:

- Descripción del tipo de datos a manejar que codifican la información

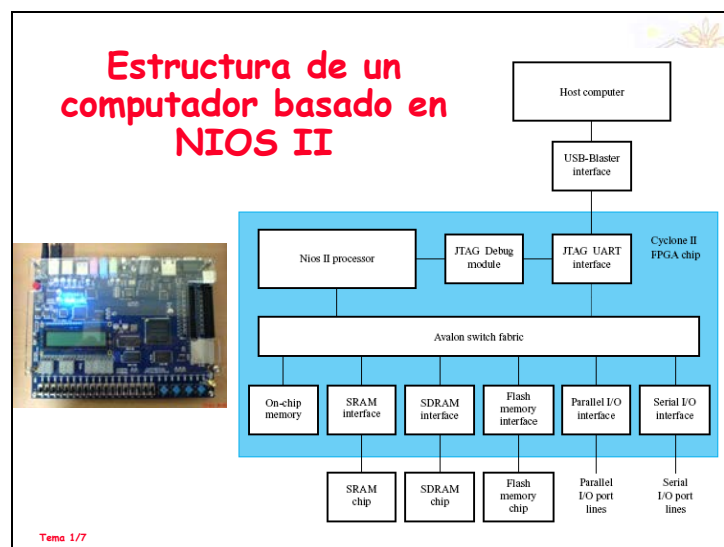
- Descripción del espacio de estados de la máquina que incluye a: espacio de direccionamiento del procesador que se corresponde con la memoria, espacio de direccionamiento del procesador que se corresponde con el sistema de entrada/salida, registros de datos, y registros de estados y control
- Descripciones de cómo se realiza el acceso a los datos, que incluye: tipo de datos sobre los que opera, modo de direccionamiento, codificación binaria de la instrucción, forma de acceder a los dos espacios de direccionamiento (direccionamiento real, direccionamiento virtual)
- Descripción de la manipulación del estado, que consiste en la inicialización de los bits de estado y control dependiendo del tipo de operación. En esta parte se incluyen las instrucciones de transferencia de control (beq, ...), instrucciones de manipulación del registro de estados, y las instrucciones privilegiadas y no privilegiadas que permiten utilizar el procesador en dos modos o niveles distintos: no privilegiado o también llamado usuario, y privilegiado o también llamado sistema.
- Descripción de las órdenes que ejecuta el procesador: instrucciones del lenguaje máquina, su codificación (que sigue unos determinados patrones denominados “formatos de instrucción”), y su latencia (o tiempo en el que se realiza una determinada instrucción dentro del procesador).
- Interrupciones hardware e interrupciones software que son las posibles causas que originan la interrupción de la ejecución de un determinado programa.
- Operaciones de Entrada/Salida que son las órdenes que ejecuta el procesador que están relacionadas con los controladores de entrada/salida a los que está conectado físicamente el procesador.



NIOS II es el nombre comercial de un procesador real que cuya propiedad intelectual pertenece a la empresa Altera. Se utiliza frecuentemente en computadores empotrados dado que es un procesador pequeño y proporciona las suficientes prestaciones para aplicaciones software empotradas como por ejemplo las destinadas a sistemas de control (motores, automóviles, etc.).

NIOS II es el procesador que se utiliza en los programas ensamblador que se manejan en las prácticas de la asignatura EC y que se encuentra en el computador de prácticas denominado DE2.

En las siguientes transparencias se describe la arquitectura ISA del procesador NIOS II.



En las clases prácticas se va a interactuar con un computador real, no simulado. En esta transparencia se puede observar un diagrama de bloques que representa a la estructura u organización del computador de prácticas DE2. En ella se distinguen los siguientes elementos:

Procesador NIOS II: dispositivo electrónico que ejecuta los programas codificados en el lenguaje máquina de NIOS II.

Interconexión múltiple Avalon: permite la comunicación entre el procesador y todos los elementos conectados al bus Avalon. Avalon permite que el procesador pueda acceder a la memoria (on-chip, SRAM, SDRAM, flash), o a los controladores de las interfaces externas (serie, paralela) a través de la ejecución de las instrucciones que la arquitectura ISA destinadas a realizar estas tareas.

Memoria interna on-chip: memoria volátil que se encuentra en el mismo circuito electrónico del procesador y que se utiliza como memoria principal.

Controladores de las memorias externas SRAM, SDRAM y Flash: circuitos que se utilizan para controlar tres tipos de memoria que se encuentran en la parte externa del chip donde se encuentra el procesador. SRAM y SDRAM son memorias de tipo volátil, y Flash es una memoria no volátil.

Controladores de interfaces paralela y serie: circuitos que se utilizan para acceder a los dispositivos que se conectan a las interfaces paralela y serie. Estos controladores también están integrados en el chip donde se encuentra el procesador.

Memoria externa SRAM y SDRAM: memorias volátiles que se utilizan como memoria principal del procesador. En ellas se pueden almacenar las instrucciones y los datos de los programas que ejecuta el procesador.

Memoria Flash: memoria no volátil que también se puede utilizar como memoria principal del procesador. El contenido de esta memoria no se pierda cuando se desconecta la alimentación eléctrica de la placa.

Dispositivos conectados a las interfaces serie y paralelo: algunos de estos dispositivos permiten que un usuario externo pueda intervenir y modificar el flujo de la ejecución de los programas. Otros dispositivos cambian dependiendo de los resultados que proporciona la ejecución de un programa.

Controladores UART serie para interfaces USB: este es otro controlador que está integrado en el mismo chip del procesador y se utiliza para interconectar la interfaz USB de un computador convencional con el computador DE2; se utiliza por ejemplo para que programas desarrollados en un ordenador convencional se puedan cargar en el ordenador basado DE2 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



Tema 1/8

Los “modos de funcionamiento” controlan cómo el procesador realiza todas sus funciones, gestiona el sistema de memoria, y accede a los dispositivos periféricos. El procesador NIOS II puede funcionar de tres formas distintas, y que se denominan: supervisor, usuario, y depuración.

Supervisor: modo de funcionamiento en el que todas las funciones disponibles del procesador se pueden ejecutar a través de un programa. Cuando se activa físicamente la señal de reset del chip del procesador o se activa la tensión de alimentación de DE2 después de estar apagado, el procesador entra en este modo supervisor. Este es el modo de funcionamiento que acostumbra a ser utilizado por el núcleo de un sistema operativo; cualquier instrucción puede ser ejecutada, cualquier operación de entrada/salida puede ser iniciada desde el procesador, y cualquier zona del espacio de direccionamiento de memoria puede ser accedida.

Usuario: la mayoría de las funciones del procesador están disponibles por programa en este modo, y sólo ciertas funciones que pueden afectar al funcionamiento del procesador no están permitidas. Este es el modo que se acostumbra a activar cuando el usuario de un sistema operativo interacciona con el computador. Este modo sólo es activado en NIOS II cuando el procesador incluye una MMU (Unidad de Gestión de memoria, en inglés: Memory Management Unit) o una MPU (Unidad de Protección de Memoria, en inglés: Memory Protección Unit). El sistema operativo determina qué direcciones de memoria son accesibles en este modo. Si la aplicación que se ejecuta en modo usuario intenta acceder a posiciones de memoria que no están permitidas, este modo genera una excepción o interrupción interna en el procesador, lo cual hace que se ejecute un programa de manejo de excepciones del propio sistema operativo, el cual normalmente aborta la



ejecución de la aplicación. La aplicación que se ejecuta en modo usuario puede hacer uso de ciertas rutinas del sistema operativo haciendo lo que se llama “llamada al sistema” para hacer por ejemplo operaciones de entrada/salida, o para acceder a ciertas funcionalidades del modo supervisor.

Depuración: Este es un modo especial del modo supervisor. En este modo de funcionamiento se permite utilizar breakpoints que obligan a que el programa se pare una vez terminada ciertas instrucciones del programa, lo cual permite visualizar el estado de los registros y memoria en ese momento. Este modo también dispone de los llamados “watchpoints” que son alarmas que obligan a parar la ejecución del programa cuando se cumplen una serie de condiciones, sobre todo asociadas a errores. Este es el modo de funcionamiento que se utiliza cuando se desarrolla un programa para analizar los posibles fallos de funcionamiento.

**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

Tema 1/9

El procesador NIOS II dispone de 32 registros internos de propósito general, es decir, que no se destinan a ningún propósito predeterminado, simplemente guardan información. Los registros tienen asignado un nombre que es reconocido por el ensamblador: r0, r1,..., r31. Estos registros son accedidos a través de las instrucciones del repertorio de instrucciones. Sólo el registro r0 es un registro que se utiliza para proporcionar siempre el valor 0, la información que se escribe en él, no se guarda.

El ensamblador a veces establece un tipo de utilización para cada uno de los registros de propósito general. En la transparencia se puede observar las funciones que el ensamblador de NIOS II establece para cada registro:

- Registro temporal de propósito general (r1)
- Registros para almacenar los resultados que devuelve la llamada a una subrutina (r2, r3)
- Registros para almacenar los argumentos que se pasan a funciones y subrutinas (r4 á r7)
- Registros que guardan los contenidos de los registros que se van a modificar cuando se ejecute una función o subrutina (r8 á r15)
- Registro que guarda la causa de una excepción (r24)
- Registro utilizado en el modo de depuración (r25 y r30)
- Registro para almacenar el principio de una zona de memoria donde se encuentran las variables globales del programa (r26)
- Registros involucrados en las llamadas anidadas a subrutinas (r27 y r28)
- Registro que guarda la dirección de memoria de retorno después de ejecutar la rutina de manejo de excepciones (r29)
- Registro que guarda la dirección de memoria donde se retorna cuando se termina de ejecutar una subrutina (r31)

**SÓLO las instrucciones rdctl y wrctl leen y escriben en registros de control**

### Registros de control

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	Uniqua 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	npubase (2)	Refer to Table 3-23 on page 3-22
15	npuacc (2)	Refer to Table 3-25 on page 3-23
16-31	Reserved	Reserved



Tema 1/10

Los registros de control de un procesador dan información de su estado y permiten también modificar su comportamiento. Se accede a ellos de forma distinta a la de los registros de propósito general. Por ejemplo, en NIOS II sólo las instrucciones rdctl y wrctl pueden acceder a los registros de control. Para conocer el contenido de un registro de control se realiza lo que se denomina “operación de lectura” (instrucción rdctl). Para modificar el contenido de un registro de control se realiza una “operación de escritura” (instrucción wrctl).

NIOS II dispone de 32 registros de control como se puede observar en la transparencia. Se dividen en “reservados” para uso interno y “no reservados” que son reconocidos por el ensamblador.

## Registro status: estado del procesador NIOS II



**Table 3-7. status Control Register Fields**

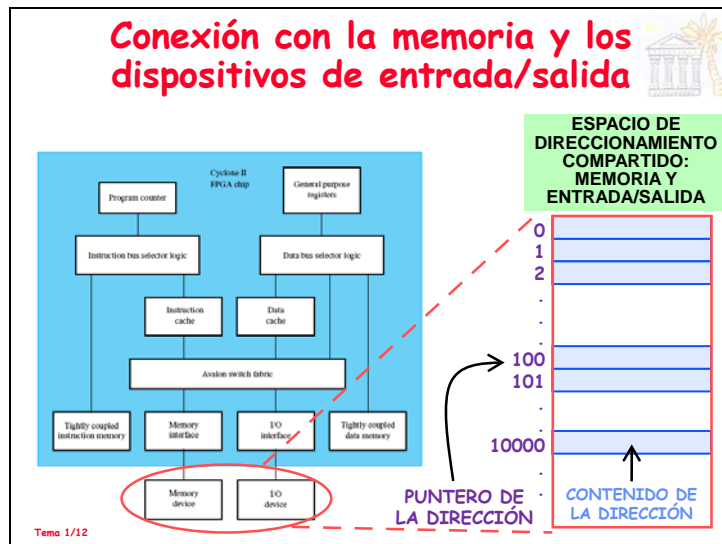
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								RSIE	NMI	PRS				CRS								IL				IH	EH	U	PIE		

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

Tema 1/11

El contenido del Registro de Estado 0 del procesador determina su estado. Cuando el procesador se resetea, su contenido está predefinido.

Este registro se divide en campos o grupos de bits como se puede observar en la transparencia. Algunos de ellos son utilizados sólo por ciertas funciones del procesador como por ejemplo la MMU (campos PRS, EH), o el acceso externo a las interrupciones (campos RSIE, NMI, IL, IH). Uno de los campos importantes de este registro de control es el que indica el modo de funcionamiento; indica si el procesador se encuentra en modo usuario o supervisor (campo U).



El acceso a la memoria y a los dispositivos de entrada/salida por parte del procesador son dos de sus principales funciones que requieren que el interior del procesador se comuniquen con un dispositivo externo, bien sea memoria o periférico. En la transparencia se observa en azul un diagrama de bloques con algunos de los módulos hardware que se encuentran en el interior del procesador NIOS II. Los elementos que están fuera del rectángulo azul son los dispositivos externos al procesador con los que se puede comunicar: la memoria y los dispositivos de entrada/salida.

Denominamos “dirección” a la unidad de almacenamiento de la información en un dispositivo externo al procesador. Esta dirección tiene un identificador o puntero que permite diferenciarla de las demás direcciones.

Denominamos “direccionamiento” a la forma que tiene el procesador de indicar una dirección de la memoria o qué dispositivo de entrada/salida se quiere acceder. La arquitectura del repertorio de instrucciones indica cómo se direcciona la memoria o un dispositivo de entrada/salida.

Denominamos “espacio de direccionamiento” a la cantidad de direcciones a la que puede acceder un procesador.

En el procesador NIOS II, la memoria y los dispositivos de entrada/salida comparten el mismo espacio de direccionamiento.

## Modos de direccionamiento



- **Inmediato:** dato codificado en la propia instrucción
- **Registro:** dirección en un registro de propósito general
- **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

Tema 1/13

Los “modos de direccionamiento” son las distintas formas que tiene el procesador para calcular una dirección del espacio de direccionamiento. Los modos de direccionamiento más frecuentes son:

Inmediato: la dirección está codificada dentro de la propia instrucción, por ejemplo utilizando uno de los campos en los que se divide el formato de instrucción.

Registro: la dirección se encuentra en un registro de propósito general.

Desplazamiento: la dirección se calcula como la suma de un dato inmediato codificado en la propia instrucción y el contenido de un registro de propósito general.

Indirecto: la dirección está contenida en una posición de memoria que es apuntada por un registro de propósito general.

NIOS II utiliza el modo de direccionamiento por desplazamiento.

## Tipos de instrucciones



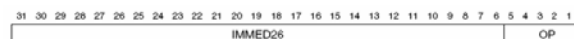
### • I-type



### • R-type



### • J-type



Tema 1/14

Las instrucciones son los elementos del repertorio de instrucciones que se representan por un patrón de bits. Cada patrón de bits codifica una orden que el procesador recibe en un momento determinado. A continuación, el procesador realiza la tarea asociada a la instrucción coordinando su ruta de datos y su unidad de control.

Estos patrones de bits se organizan en “campos de bits” que son subconjuntos de bits que forman el patrón. Un “formato de instrucción” es la combinación del número de campos en los que se puede dividir una determinada instrucción junto con los bits destinados a cada campo.

NIOS II codifica las instrucciones con 3 formatos distintos denominados: I, R, J.

I: con este formato la instrucción se divide en 4 campos. OP representa al denominado “código de operación” que identifica a la orden de la instrucción. A y B identifican a los números de los registros de propósito general que utiliza la operación como operandos. IMM16 representa y codifica a un número de 16 bits denominado “inmediato”.

R: con este formato la instrucción se divide en 5 campos. OP identifica al denominado “código de operación” que identifica a la orden de la instrucción. A y B identifican a los números de los registros de propósito general que utiliza la operación como operandos. C identifica al número de registro de propósito general que utiliza la operación para guardar el resultado de la instrucción. OPX es un código que se puede utilizar para dirigir de una forma determinada a la unidad funcional donde se realizan las operaciones dentro del procesador.

J: con este formato la instrucción se divide en 2 campos. OP identifica al denominado “código de operación” que identifica a qué orden corresponde la instrucción. IMM26 representa y codifica a un número denominado “inmediato” de 26 bits.

Table 8-1. OP Encodings

OP	Instruction	OP	Instruction	OP	Instruction	OP	Instruction
0x00	call	0x00	cmplti	0x20	cmpeqi	0x30	cmptu
0x01	jmpi	0x01		0x21		0x31	
0x02		0x02		0x22		0x32	outcom
0x03	ldbu	0x03	initda	0x23	ldbuio	0x33	initd
0x04	addi	0x04	ori	0x24	mul1	0x34	orhi
0x05	stb	0x05	stw	0x25	stb10	0x35	stw10
0x06	br	0x06	blt	0x26	beq	0x36	bltu
0x07	ldb	0x07	ldw	0x27	ldbuio	0x37	ldw10
0x08	cmpeqi	0x08	cmptui	0x28	cmpeui	0x38	rdprw
0x09		0x09		0x29		0x39	
0x0A		0x0A		0x2A		0x3A	R-type
0x0B	ldbu	0x0B	flushda	0x2B	ldbuio	0x3B	flushd
0x0C	andi	0x0C	xori	0x2C	andhi	0x3C	xorhi
0x0D	sth	0x0D		0x2D	sth10	0x3D	
0x0E	bge	0x0E	bse	0x2E	bgeu	0x3E	
0x0F	ldh	0x0F		0x2F	ldbuio	0x3F	

**Códigos de  
operación  
(OP)**

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	cmptu
0x01	eret	0x11		0x21		0x31	add
0x02	roll	0x12	sl1i	0x22		0x32	
0x03	rol	0x13	sll	0x23		0x33	
0x04	flushbp	0x14	wrprw	0x24	divu	0x34	break
0x05	ret	0x15		0x25	div	0x35	
0x06	nor	0x16	or	0x26	rdctl	0x36	sync
0x07	mulxwu	0x17	mulxwu	0x27	mul	0x37	
0x08	cmpe	0x18	cmpe	0x28	cmpeu	0x38	
0x09	bret	0x19		0x29	init1	0x39	sub
0x0A		0x1A	srl1	0x2A		0x3A	sra1
0x0B	xor	0x1B	srl	0x2B		0x3B	sra
0x0C	flushi	0x1C	nextpc	0x2C		0x3C	
0x0D	jmp	0x1D	callr	0x2D	trap	0x3D	
0x0E	and	0x1E	xor	0x2E	wrc1	0x3E	
0x0F		0x1F	mulxwu	0x2F		0x3F	

En esta transparencia se pueden observar los códigos de operación (OP) de las instrucciones NIOS II.

## 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	cmpe rC, rB, rA
cmplei rB, rA, IMMED	cmplti rB, rA, (IMMED+1)
cmpleu rC, rA, rB	cmpeu 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)

Las pseudoinstrucciones se utilizan en el código fuente ensamblador de la misma forma que las instrucciones ensamblador. Cada pseudoinstrucción se implementa a nivel de código máquina utilizando una instrucción ensamblador equivalente. En la transparencia se pueden observar estas equivalencias. La única excepción es la pseudoinstrucción movia que se implementa con dos instrucciones

## Tipos de instrucciones: Acceso a memoria

**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.

**Table 3-39.** Narrow Data Transfer Instructions

Instruction	Description
ldb ldbu atb lsh ldhu ath	ldb, ldbu, lsh and ldhu load a byte or half-word from memory to a register. ldb and lsh sign-extend the value to 32 bits, and ldbu and ldhu zero-extend the value to 32 bits. atb and ath 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 atbio lshio ldhuio athio	These operations load/store byte and half-word data from/to peripherals without caching or buffering.

Tema 1/15

**Operación:**

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

**Sintaxis:**

ldw rB, byte\_offset(rA)

ldwio rB, byte\_offset(rA)

**Programa:**

ldw r6, 100(r5)

Otro tipo de clasificación que se establece sobre las instrucciones del repertorio de instrucciones se basa en el tipo de operación que realiza cada instrucción. De esta forma, en el caso del procesador NIOS II podemos encontrar los siguientes tipos de instrucciones:

- Acceso a memoria
- Aritmético-lógicas
- Copia entre registros
- Comparación de datos
- Llamada a funciones y subrutinas
- Saltos condicionales e incondicionales

Las instrucciones de acceso a memoria permiten realizar 4 tipos distintos de instrucciones:


- Almacenar en una dirección de la memoria principal el contenido de un registro interno del procesador (también llamado almacenamiento). Su código nemónico es stw.
- Almacenar temporalmente en un registro interno del procesador el contenido de una dirección de memoria (también llamado carga). Su código nemónico es ldw.
- Almacenar en un registro de un periférico externo el contenido de un registro interno del procesador. Su código nemónico es stwio
- Almacenar temporalmente en un registro interno del procesador el contenido de un registro de un periférico externo. Su código nemónico es ldwio

Las instrucciones anteriores utilizan para codificar la información el tamaño típico de palabra de la arquitectura como el tamaño de la información que se traspa en cada instrucción ldw, stw, ldwio, stwio (32 bits en el caso de



NIOS II). Existe otro conjunto de instrucciones en el que la unidad de la información intercambiada entre los registros y la memoria es inferior: 1 byte (instrucciones que incluyen una “b” en su nemónico: *ldb*, *ldbu*, *stb*, *ldbio*, *ldbuio*, *stbio*), o 2 bytes (instrucciones que incluyen una “h” en su nemónico: *ldh*, *ldhu*, *ldhio*, *sthio*). Las instrucciones que incluyen una “u” en su nemónico consideran que el dato representa a un número y está codificado sin signo.

Por ejemplo, en la transparencia se observa la sintaxis de la instrucción *ldw* que carga un dato en un registro “rB” desde una posición de memoria cuya dirección se obtiene como la suma de “byte\_offset” (extensión en signo de IMM16:  $\sigma$  (IMM16)) y el contenido del registro “rA”:  $rB \leftarrow \text{Mem32}[rA + \sigma \text{ (IMM16)}]$ . La sintaxis de esta instrucción es la siguiente: *ldw* rB, byte\_offset(rA). Un ejemplo de esta instrucción tal y como aparecería en un programa es (rA= r5, rB= r6, byte\_offset= 100): *ldw* r6, 100(r5).

Tipos de instrucciones: Aritmético-lógicas	
	
<b>Table 3-40. Arithmetic and Logical Instructions (Part 1 of 2)</b>	
Instruction	Description
<i>and</i> <i>or</i> <i>xor</i> <i>nor</i>	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.
<i>andi</i> <i>ori</i> <i>xori</i>	These operations are immediate versions of the <i>and</i> , <i>or</i> , and <i>xor</i> instructions. The 16-bit immediate value is zero-extended to 32 bits, and then combined with a register value to form the result.
<i>andhi</i> <i>orhi</i> <i>xorhi</i>	In these versions of <i>and</i> , <i>or</i> , and <i>xor</i> , 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.
<b>Table 3-40. Arithmetic and Logical Instructions (Part 2 of 2)</b>	
Instruction	Description
<i>add</i> <i>sub</i> <i>muli</i> <i>div</i> <i>divu</i>	These are the standard 32-bit arithmetic operations. These operations take two registers as input and store the result in a third register.
<i>addi</i> <i>subi</i> <i>muli</i> <i>divi</i>	These instructions are immediate versions of the <i>add</i> , <i>sub</i> , and <i>muli</i> instructions. The instruction word includes a 16-bit signed value.
<i>mull</i> <i>mullu</i> <i>mullu</i>	These instructions provide access to the upper 32 bits of a 64-bit 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 <i>muli</i> .
<i>mullu</i>	This instruction is used in computing a 128-bit result of a 64-bit signed multiplication.

**Operación:**  
 $rC \leftarrow rA + rB$

**Sintaxis:**  
*add* rC, rA, rB

**Programa:**  
*add* r6, r7, r8

Las instrucciones aritmético-lógicas realizan alguna operación bien aritmética (suma, resta, multiplicación o división) o bien lógica (and, or, xor, etc.) sobre dos operandos, de forma tal que los combinan obteniéndose un resultado que se almacena en un registro interno del procesador.


Existen dos versiones de instrucciones aritmético-lógicas:

- Aquellas en las que cada uno de los dos operandos se encuentran en un registro distinto del procesador; versión denominada “registro-registro” o “formato R”. Los nemónicos de estas instrucciones NO terminan en “i”.
- Aquellas en las que un operando se encuentra en un registro y el otro es un inmediato que está codificado en la propia instrucción; versión

denominada "registro-inmediato" o "formato I". Los nemónicos de estas instrucciones SI terminan en "i".

Por ejemplo, en la transparencia se observa la sintaxis de la instrucción add que suma un dato que está almacenado en el registro "rA" con otro dato que está en el registro "rB"; el resultado se guarda temporalmente en el registro "rC":  $rC \leftarrow rA + rB$ . La sintaxis de esta instrucción es la siguiente: add rC, rA, rB. Un ejemplo de esta instrucción tal y como aparecería en un programa es (rA= r7, rB= r8, rC= r6): add r6, r7, r8.

### Tipos de instrucciones: 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

Tema 1/17

Un tipo especial de instrucción aritmético lógica es aquel cuyo resultado consiste en desplazar un patrón de bits a la derecha o a la izquierda. Se diferencian aquellos desplazamientos que rotan los bits (instrucciones que empiezan por "r") de aquellos que no rotan los bits (instrucciones que empiezan por "s"). Como se puede observar en la transparencia, también existen versiones registro-registro (sin terminación en "i") y registro-inmediato (con terminación en "i"). Igualmente, se puede observar que existen rotaciones y desplazamiento tanto a derecha como a izquierda. Los desplazamientos pueden ser aritméticos como sra y srai (sólo desplazamientos aritméticos a la derecha que salvaguardan el signo del patrón a desplazar), o pueden ser desplazamientos lógicos donde no se salvaguarda el signo del dato y que corresponden al resto de instrucciones de este tipo.

Por ejemplo, en la transparencia se observa la sintaxis de la instrucción rol que rota a la izquierda el patrón de bits almacenado en rA tantos bits como indican los bits 0-4 de rB. Los bits que salen por la izquierda al realizar el desplazamiento son introducidos por la derecha en la posición menos

significativa del resultado, el cual se guarda temporalmente en el registro "rC":  $rC \leftarrow rA \text{ rotates left } rB$ . La sintaxis de esta instrucción es la siguiente: `rol rC, rA, rB`. Un ejemplo de esta instrucción tal y como aparecería en un programa es ( $rA = r7$ ,  $rB = r8$ ,  $rC = r6$ ): `rol r6, r7, r8`.

Tipos de instrucciones: copia entre registros



**Table 3-41. Move Instructions**

Instruction	Description
<code>mov</code>	<code>mov</code> copies the value of one register to another register. <code>movl</code> moves a 16-bit signed immediate value to a register, and sign-extends the value to 32 bits. <code>movsl</code> and <code>movshl</code> move an immediate 16-bit value into the lower or upper 16-bits of a register, inserting zeros in the remaining bit positions. Use <code>movia</code> to load a register with an address.
<code>movhi</code>	
<code>movl</code>	
<code>movsl</code>	
<code>movshl</code>	
<code>movia</code>	

**Operación:**  $rC \leftarrow rA$

**Sintaxis:** `mov rC, rA`

**Programa:** `mov r6, r7`

**Operación:**  $rC \leftarrow \text{label}$

**Sintaxis:** `movia rC, label`

**Programa:** `movia r6, funcion`

Tema 1/18

Este tipo de instrucción copia el contenido de un registro o un valor inmediato en otro registro.

Por ejemplo, en la transparencia se observa la sintaxis de la instrucción `mov` que copia el contenido del registro  $rA$  en el registro  $rC$ :  $rC \leftarrow rA$ . El nemónico de esta instrucción es la siguiente: `mov rC, rA`. Un ejemplo de esta instrucción tal y como aparecería en un programa es ( $rA = r7$ ,  $rC = r6$ ): `mov r6, r7`.

Una instrucción de este tipo que se utiliza frecuentemente es aquella que sirve para inicializar un registro con la dirección de una posición de memoria: `movia`. Su operación es la siguiente:  $rB \leftarrow \text{label}$ ; su sintaxis: `movia rB, label`; y un ejemplo en programa es: `movia r6, function_address`.

## Tipos de instrucciones: 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	signed <=
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 <
cmpeq	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.
cmpnei	
cmpgei	
cmpgeui	
cmpgti	
cmpgtui	
cmplei	
cmpleui	
cmplti	
cmpltui	

Tema 1/19

Las instrucciones de comparación de datos comparan el contenido de un registro con otro registro o un inmediato, y como resultado inicializan otro registro con un 1 o un 0 dependiendo del resultado de la comparación. Los tipos de comparaciones son:

- eq: compara si es igual
- ne: compara si es no igual
- ge: compara si es mayor o igual
- gt: compara si es mayor
- le: compara si es menor o igual
- lt: compara si es menor

Por ejemplo, en la transparencia se observa la sintaxis de la instrucción cmpeq que compara el contenido del registro rA y el contenido del registro rB para analizar si son iguales o no. En el registro "rC" se guarda un 1 si son iguales, o un 0 en caso contrario: if (rA == rB) then rC ← 1; else rC ← 0. La sintaxis de esta instrucción es la siguiente: cmpeq rC, rA, rB. Un ejemplo de esta instrucción tal y como aparecería en un programa es (rA= r7, rB= r8, rC= r6): cmpeq r6, r7, r8

Las instrucciones que termina en "u" realizan comparaciones con números sin signo; las que terminan en "i" realizan comparaciones con inmediatos.

## Tipos de instrucciones: saltos incondicionales



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

Instruction	Description
<code>call</code>	This instruction calls a subroutine using an immediate value as the subroutine's absolute address, and stores the return address in register <code>ra</code> .
<code>callr</code>	This instruction calls a subroutine at the absolute address contained in a register, and stores the return address in register <code>ra</code> . This instruction serves the roll of dereferencing a C function pointer.
<code>ret</code>	The <code>ret</code> instruction is used to return from subroutines called by <code>call</code> or <code>callr</code> . <code>ret</code> loads and executes the instruction specified by the address in register <code>ra</code> .
<code>jmp</code>	The <code>jmp</code> instruction jumps to an absolute address contained in a register. <code>jmp</code> 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
<code>jmp.i</code>	The <code>jmp.i</code> instruction jumps to an absolute address using an immediate value to determine the absolute address.
<code>jr</code>	This instruction branches relative to the current instruction. A signed immediate value gives the offset of the next instruction to execute.

Tema 1/20

Los procesadores están preparados para ejecutar instrucciones que se alojan en memoria en direcciones consecutivas. Cuando en un punto del programa se requiere ejecutar una instrucción que no se encuentra alojada en memoria de forma consecutiva a la anterior, se requiere realizar lo que se denomina un “salto”. Las operaciones de salto se realizan con instrucciones denominadas de “salto”.

Existen 2 tipos de instrucciones de salto, denominados “incondicionales” y “condicionales”.

Los saltos incondicionales producen un salto en la ejecución del programa desde el punto donde se encuentra la instrucción de salto hasta la instrucción denominada “destino del salto”. Posteriormente, el programa sigue ejecutando secuencialmente las instrucciones a partir del destino de salto.

Los saltos incondicionales se utilizan en los siguientes casos:

- Llamadas a subrutinas y funciones
- Retorno desde las subrutinas y funciones
- Modificación del flujo secuencial del programa a través de sentencias de alto nivel del tipo “goto”.

## Tipos de instrucciones: 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

Tema 1/21

Los saltos condicionales producen un salto en la ejecución del programa desde el punto donde se encuentra la instrucción de salto hasta la instrucción denominada "destino del salto" SÓLO SI se cumple una condición. Posteriormente, el programa sigue ejecutando secuencialmente las instrucciones a partir del destino de salto.

Las instrucciones para saltar comparan 2 registros y en NIOSII pueden ser las siguientes:

- beq: instrucción salta si igual
- bne: instrucción salta si no igual
- bge: instrucción salta si mayor o igual
- bgt: instrucción salta si mayor
- ble: instrucción salta si menor o igual
- blt: instrucción salta si menor

Por ejemplo, en la transparencia se observa la sintaxis de la instrucción bge que compara el contenido del registro rA y el contenido del registro rB para ver si el primero es mayor o igual que el segundo. En el caso de cumplirse la condición "mayor o igual", el registro interno del procesador NIOSII denominado "PC" guarda la dirección de instrucción destino del salto. Esta dirección se calcula como la suma del PC actual, 4, y el inmediato con signo. Si no se cumple la condición, el PC siguiente a la instrucción actual de salto se aumenta en 4 respecto al valor actual: if ((signed) rA >= (signed) rB) then PC ← PC + 4 + σ(IMM16) else PC ← PC + 4. La sintaxis de esta instrucción es la siguiente: bge rA, rB, label. Un ejemplo de esta instrucción tal y como aparecería en un programa es (rA= r6, rB= r7, label= top\_of\_loop): bge r6, r7, top\_of\_loop.

## Ejemplo de programa

```

#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             /* Register r4 is used as the counter for loop iterations */
    ldw r5, 0(r4)          /* 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 */
    mtl 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

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

```

Tema 1/22

En esta transparencia se muestra un programa fuente que está descrito en el lenguaje ensamblador de NIOS II. Este programa calcula el producto escalar de 2 vectores con 6 elementos cada uno de ellos. El programa se estructura en 2 partes:

- Zona de datos: a partir de la etiqueta N hasta el final de DOT\_PRODUCT
- Zona de programa: a partir de la etiqueta \_start hasta la etiqueta STOP

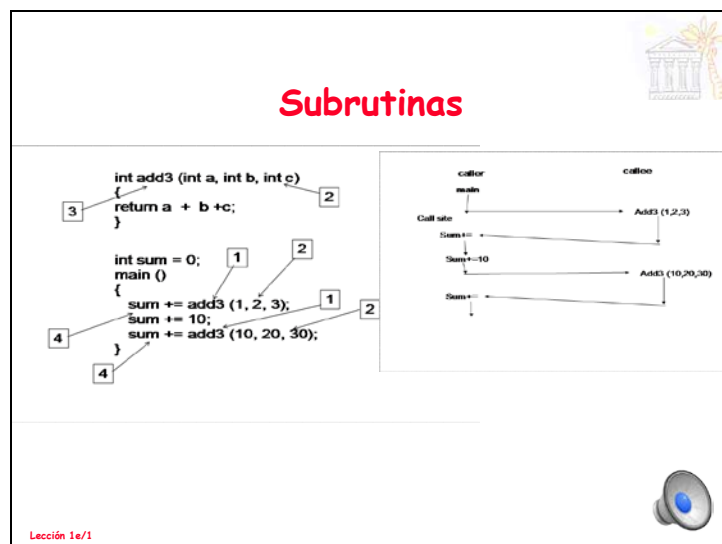
La zona de datos se divide en 4 partes:

- la etiqueta N identifica la zona de memoria donde se guarda el número de componentes de los vectores, para lo cual se reserva una palabra de 4 bytes; esta reserva se indica con la palabra clave del ensamblador “.word”
- la etiqueta AVECTOR identifica a la zona de memoria donde se guardan los 6 elementos del primer vector, cada elemento se guarda en 4 bytes; por ello se utiliza la palabra clave “.word”. Estos 6 números se inicializan a los valores: 5, 3, -6, 19, 8, 12
- la etiqueta BVECTOR identifica a la zona de memoria donde se guardan los 6 elementos del segundo vector, para lo cual también se reservan 4 bytes para cada elemento, y que también están inicializados a los valores: 2, 14, -3, 2, -5, 36
- la etiqueta DOT\_PRODUCT identifica a la zona de memoria donde se reservan 4 posiciones de memoria (4 bytes) donde se almacena el resultado del producto escalar. Para esta reserva de memoria se utiliza la palabra clave del ensamblador “.skip” junto con el número de posiciones de memoria de 1 byte: 4.

La zona de programa se divide en 2 partes.



- La primera parte consta de las 5 primeras instrucciones
  - las tres primeras (instrucciones movia) inicializan los registros r2, r3 y r4 a los punteros de una determinada posición de memoria;
  - las dos instrucciones restantes inicializan r4 con el número almacenado en la etiqueta N (instrucción ldw), y r5 con un 0 (instrucción add), ya que va a guardar la acumulación del producto escalar en la segunda parte del programa.
- La segunda parte de la zona de programa consiste en un bucle donde se realiza el producto escalar
  - primero se cargan dos componentes (instrucciones ldw), una de cada vector
  - luego, estas componentes se multiplican utilizando la instrucción mul, y su resultado se acumula en r5 (instrucción add)
  - seguidamente, se actualizan los punteros de los vectores A y B (instrucciones addi)
  - se reduce en 1 el registro r4 que cuenta el número de componentes que quedan por procesar (instrucción subi)
  - se ejecuta una instrucción de salto condicional para que en el caso que el bucle no haya iterado en 6 ocasiones comience una nueva iteración (instrucción de salto condicional “mayor que”: bgt)
  - se ejecuta un almacenamiento del producto escalar después de salir del bucle (instrucción stw)
  - y por último se encuentra una instrucción que salta sobre sí misma para terminar el programa (br)



Los programas estructurados se basan en las denominadas “subrutinas”. En la transparencia se puede observar un diagrama de flujo del programa principal “main()” y una subrutina “add3(...)”. La subrutina add3 puede ser

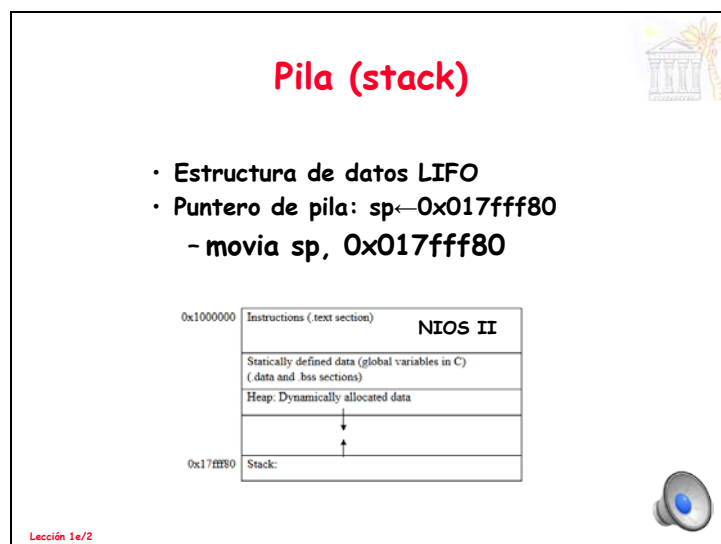


llamada desde cualquier parte del programa principal main. Cuando la subrutina termina su procesamiento, se sigue ejecutando el programa principal main. Observar en la transparencia que la subrutina add3 es llamada dos veces desde el programa principal.

Utilizando el lenguaje de programación C , a la izquierda se puede observar un programa que consta de su rutina principal main y la subrutina add3 que suma tres valores que recibe como argumentos. El programa principal utiliza la subrutina para actualizar la variable global sum que se va acumulando con los resultados de dos llamadas a la subrutina add3 y el valor 10.

Para que estas llamadas a las subrutinas se puedan realizar con fiabilidad, es necesario que el procesador sea capaz de realizar las siguientes acciones:

- El procesador debe cambiar el flujo del programa hacia una subrutina desde cualquier parte del programa. Esto se hace con la instrucción “call”
- La subrutina debe recibir distintos parámetros que pueden tomar cualquier valor para luego poderlos procesar dentro de la subrutina
- La subrutina puede devolver uno o varios resultados
- Cuando termina la subrutina, el procesador debe cambiar el flujo del programa hacia el punto inmediatamente después de la llamada a la subrutina desde el programa principal.



Para guardar las direcciones de retorno cuando la ejecución de las subrutinas finalicen se utiliza una zona del espacio de direccionamiento del procesador denominada “pila”. Esta zona de memoria se encuentra en una parte del espacio de direccionamiento del procesador que se asigna a cada programa en su conjunto incluyendo las subrutinas. Esta memoria se

considera de tipo LIFO (last in first out): lo último que entra es lo primero que sale.

La zona de memoria destinada al programa se divide en cuatro partes:

- Zona de instrucciones que se identifica en el programa ensamblador de NIOS II por “.text”.
- Zona de estática de memoria que se destina a guardar datos y que se identifica en el programa ensamblador de NIOS II por “.data”.
- Zona variable destinada a memoria dinámica que crece hacia posiciones de memoria superiores.
- Zona de la pila que crece a posiciones de memoria inferiores a partir del denominado “puntero de pila” que en la transparencia tiene el valor 0x017fff80. El registro r27 del procesador NIOS II se acostumbra a utilizar para guardar la última posición del puntero de pila. En ensamblador, r27 se representa por “sp”.

## Pila (stack)

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

**Ejemplo**

SP -->	01	02	03	04
0x000000	10	20	30	40
0x000004	11	22	33	44
0x000008	55	66	77	88

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

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

Lección 1e/3

Se pueden realizar dos tipos de operaciones con la pila, denominadas push y pop. La operación push guarda en la pila el contenido de un registro del interior del procesador, y además actualiza el puntero de la pila restándole 4 posiciones de memoria. Esto se realiza con dos instrucciones: stw que es un almacenamiento del registro (en la transparencia es r31) en la posición de memoria apuntada por el contenido de sp, y subi que es una instrucción que resta 4 al puntero de pila sp.

La operación pop obtiene desde la zona de memoria de la pila un valor que guarda en un registro del procesador. Esto se realiza también con dos instrucciones: addi suma el contenido del registro que guarda el puntero

actual de la pila con 4, y `ldw` que es la carga en un registro (en la transparencia r31) el contenido de la dirección de memoria apuntada por el puntero de pila `sp`.

Observemos ahora un ejemplo del uso de la pila. En la transparencia se puede observar una zona de memoria inicializada con distintos valores de byte. Supongamos que corresponde a la zona de la pila de un determinado programa.

Con la instrucción `ldw r9, 8(sp)` se guardan en el registro `r9` 4 bytes que se almacenan en cuatro posiciones de memoria consecutivos a partir del puntero `sp+8`, es decir las direcciones `sp+8=0x6fff8` que guarda el valor `0x11`, `sp+9=0x6fff9` que guarda el valor `0x22`, `sp+10=0x6fffa` que guarda el valor `0x33` y `sp+11=0x6fffb` que guarda el valor `0x44`. Por tanto en `r9` se guarda el valor de 4 bytes `0x44332211`.

Con la instrucción `ldb r10, 0xd(sp)` se guardan en el registro `r10` 1 byte que se almacena en una posición de memoria apuntada por el puntero `sp+0xd`, es decir la dirección `sp+0xd=0x6fffd` guarda el valor `0x66`. Por tanto en `r10` se guarda el valor de 1 byte `0x66`.

## Subrutinas en ensamblador

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

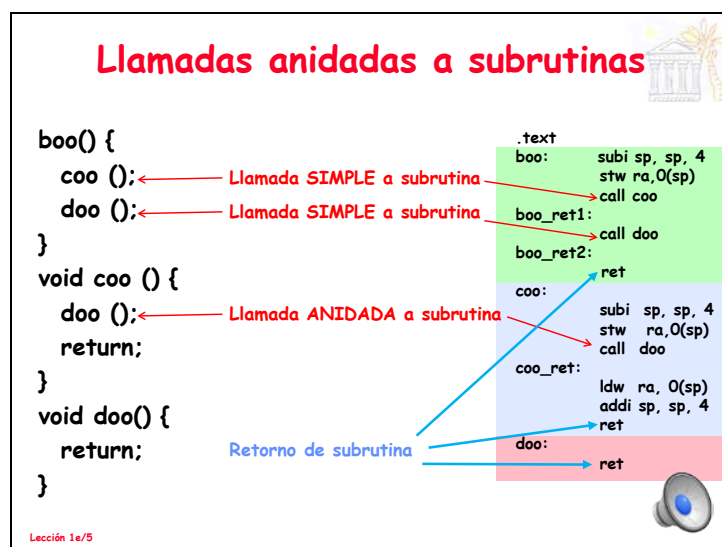
Llamada a subrutina;  
equivalente a:  
 $r31 = PC + 4$   
 $PC = coo$

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

Siguiendo con las subrutinas, nos vamos a centrar ahora en el flujo de control de la ejecución de las instrucciones de un programa con subrutinas. En esta transparencia, suponemos que las subrutinas no aceptan argumentos, ni devuelven valores resultados.

En la transparencia se puede observar una parte de un programa en ensamblador de NIOS II que llama a la subrutina denominada coo, utilizando la instrucción “call”. Cuando se produce la ejecución de call, el registro r31 del procesador se inicializa con la dirección de retorno de la subrutina, es decir el contador de programa actual (PC) que está apuntando a la instrucción de salto incondicional “call” al que además se le suma el valor 4; y a continuación, el contador de programa se inicializa con la dirección de memoria donde se encuentra la primera instrucción de la subrutina, en este caso la que corresponde a la etiqueta “coo:”

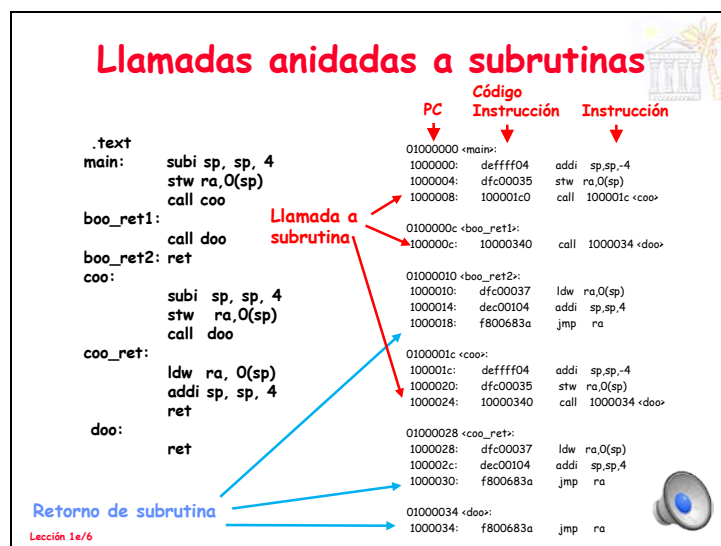
Cuando la subrutina coo termina, se ejecuta la instrucción de salto incondicional denominada “ret” que restaura el contador de programa con el valor guardado en el registro r31. De esta forma, la ejecución del programa sigue con la instrucción siguiente a la instrucción call que llamó a la subrutina coo.



Otro aspecto de las subrutinas es el denominado “llamada anidada” de las subrutinas que consiste en llamar a una subrutina desde dentro de otro subrutina. En la transparencia se puede observar un ejemplo. La rutina boo hace una llamada simple desde la subrutina boo a otras dos subrutinas: coo y doo. A su vez, coo llama a la subrutina doo en una llamada anidada.

En la parte derecha de la transparencia se puede observar el correspondiente programa en ensamblador de NIOS II. Con distintos colores se delimitan las zonas de código de las rutinas boo, coo y doo. Las flechas rojas de la derecha indican las instrucciones call que realizan las llamadas a las subrutinas. Las flechas azules indican las instrucciones que producen el retorno de las subrutinas.

Cuando se produce llamadas anidadas a subrutinas, el retorno de las distintas subrutinas se debe dirigir a la instrucción siguiente de la instrucción de llamada. En este ejemplo, el retorno de la subrutina doo se dirige a la dirección etiquetada “coo\_ret” cuando se retorna de la llamada “call doo” que se ejecuta dentro de la subrutina “coo”. Y el retorno de la subrutina coo se dirige a la dirección de la etiqueta “boo\_ret1” cuando se retorna de la llamada “call coo” dentro de la subrutina “boo”.

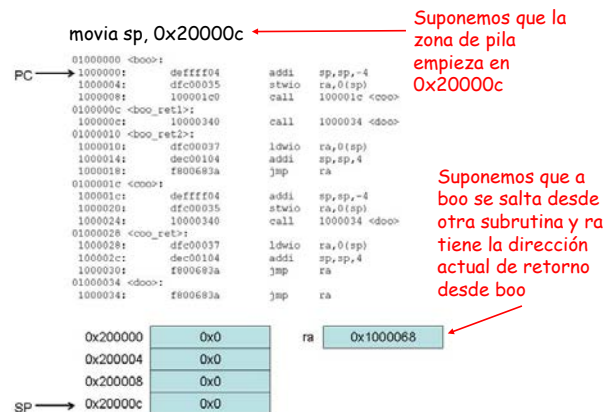


En esta transparencia se puede observar a la derecha las direcciones que suponemos asigna el ensamblador a cada instrucción. Cuando cada instrucción se vaya a ejecutar, previamente el contador de programa PC ha sido inicializado con dicha dirección. A la derecha de cada dirección de memoria se puede observar el código de la correspondiente instrucción. En la última columna se encuentra la instrucción que ha seleccionado el ensamblador.

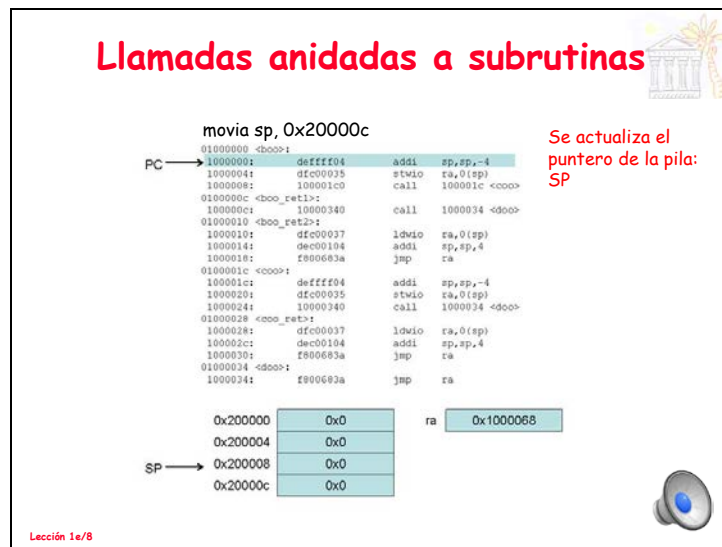
Observar que las llamadas a subrutinas utilizan la instrucción call en las que se ha sustituido la etiqueta de la subrutina por su dirección de memoria.

Observar también que todos los retornos de subrutinas se implementan con la instrucción “jmp ra”, es decir, con un salto incondicional a la dirección que contiene el registro ra que es el registro r31.

## Llamadas anidadas a subrutinas

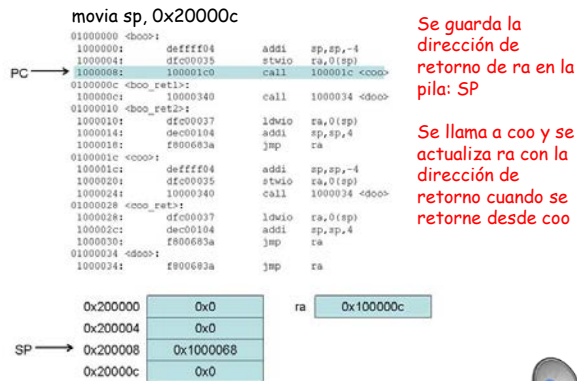


A continuación vamos a describir una simulación de la ejecución de este programa. Suponemos que la pila comienza en la dirección 0x20000c, que el contador de programa PC apunta a la primera instrucción de la subrutina boo: 0x1000000, y que la dirección de retorno de la subrutina boo es ra=0x1000068.



Como resultado de la ejecución de la instrucción addi, se actualiza el contenido del registro sp que ahora apunta a la dirección 0x200008. Al terminar esta ejecución, el contador de programa apunta a la siguiente instrucción que se encuentra cuatro posiciones de memoria más adelante.

## Llamadas anidadas a subrutinas



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

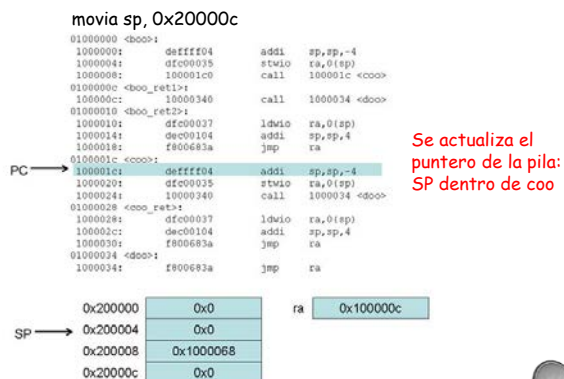
Se llama a coo y se actualiza ra con la dirección de retorno cuando se retorne desde coo

Lección 1a/9

Como resultado de la ejecución de la segunda instrucción de almacenamiento stwio, la dirección de memoria apuntada por el puntero de pila sp guarda la dirección de retorno que tenía el registro ra. Al terminar esta ejecución, el contador de programa apunta a la siguiente instrucción que se encuentra cuatro posiciones de memoria más adelante.

Como resultado de la ejecución de la tercera instrucción, se produce el salto a la rutina coo, y el registro ra guarda la dirección de retorno que sería cuatro posiciones más adelante: 0x100000c. Al terminar esta ejecución, el contador de programa apunta a la primera instrucción de la subrutina coo que está en la dirección 0x100001c.

## Llamadas anidadas a subrutinas



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

Lección 1a/10

Ahora se ejecuta la primera instrucción de la subrutina coo que es de tipo addi. Esta instrucción actualiza el puntero de la pila que ahora apunta a la dirección 0x200004.



Al terminar esta ejecución, el contador de programa apunta a la siguiente instrucción que se encuentra cuatro posiciones de memoria más adelante.

### Llamadas anidadas a subrutinas

```

movia sp, 0x20000c

01000000: <boo>:
10000001: dffff04      addl    sp,sp,-4
10000004: dfc00035     stwio   ra,0(sp)
10000008: 100001c0     call   100001c <coo>
0100000c: <boo_ret1>:
1000000c: 10000340     call   1000034 <doo>
01000010: <boo_ret2>:
10000010: dfc00037     ldwio   ra,0(sp)
10000014: dec00104     addl    sp,sp,4
10000018: f800683a     jmp     ra
0100001c: <coo>:
1000001c: dffff04      addl    sp,sp,-4
10000020: dfc00035     stwio   ra,0(sp)
10000024: 10000340     call   1000034 <doo>
01000028: <coo_ret1>:
10000028: dfc00037     ldwio   ra,0(sp)
1000002c: dec00104     addl    sp,sp,4
10000030: f800683a     jmp     ra
01000034: <doo>:
10000034: f800683a     jmp     ra

```

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

PC → 1000020: dfc00035 stwio ra,0(sp)

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

SP → 0x200004

ra 0x100000c

Lección 1e/11

Ahora se ejecuta la segunda instrucción de la subrutina coo que es de tipo almacenamiento stwio. Esta instrucción guarda la dirección de retorno de ra en la pila, en la dirección apuntada por sp. Al terminar esta ejecución, el contador de programa apunta a la siguiente instrucción que se encuentra cuatro posiciones de memoria más adelante.

### Llamadas anidadas a subrutinas

```

movia sp, 0x20000c

01000000: <boo>:
10000001: dffff04      addl    sp,sp,-4
10000004: dfc00035     stwio   ra,0(sp)
10000008: 100001c0     call   100001c <coo>
0100000c: <boo_ret1>:
1000000c: 10000340     call   1000034 <doo>
01000010: <boo_ret2>:
10000010: dfc00037     ldwio   ra,0(sp)
10000014: dec00104     addl    sp,sp,4
10000018: f800683a     jmp     ra
0100001c: <coo>:
1000001c: dffff04      addl    sp,sp,-4
10000020: dfc00035     stwio   ra,0(sp)
10000024: 10000340     call   1000034 <doo>
01000028: <coo_ret1>:
10000028: dfc00037     ldwio   ra,0(sp)
1000002c: dec00104     addl    sp,sp,4
10000030: f800683a     jmp     ra
01000034: <doo>:
10000034: f800683a     jmp     ra

```

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

PC → 1000024: 10000340 call 1000034 <doo>

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

SP → 0x200004

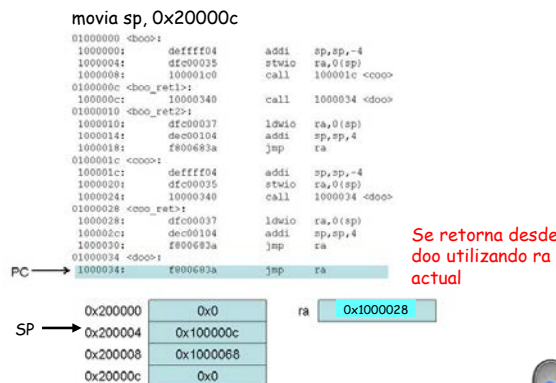
ra 0x1000028

Lección 1e/12

Como resultado de la ejecución de la tercera instrucción de la subrutina coo, se produce el salto a la rutina doo, y el registro ra guarda la dirección de retorno que sería cuatro posiciones más adelante: 0x1000028. Al terminar esta ejecución, el contador de programa apunta a la primera instrucción de la subrutina doo que está en la dirección 0x1000034.



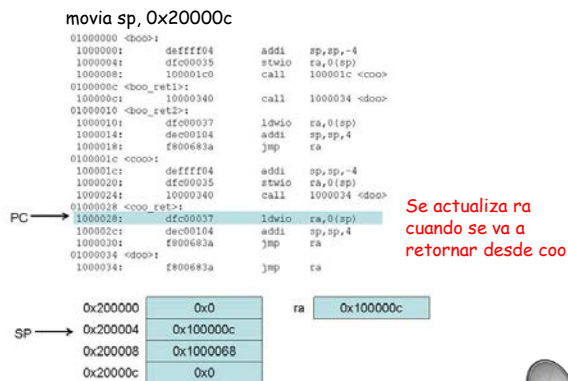
## Llamadas anidadas a subrutinas



Lección 1e/13

La subrutina doo consta de una sola instrucción que es un retorno de subrutina. En este caso se retorna a la dirección que guarda el registro ra y cuyo valor es 0x1000028 que corresponde a la etiqueta coo\_ret. Al terminar esta ejecución, el contador de programa apunta a la instrucción de la subrutina coo que está en la dirección 0x1000028, una carga ldwio.

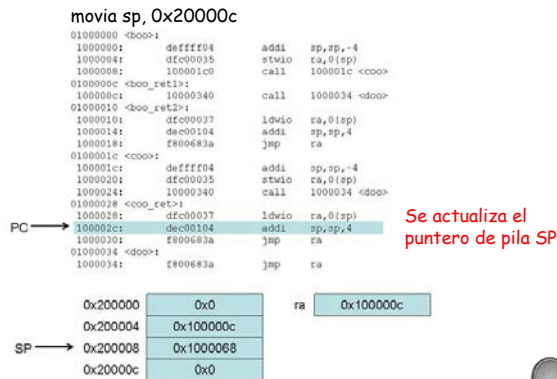
## Llamadas anidadas a subrutinas



Lección 1e/14

Esta instrucción de carga actualiza el registro ra con el contenido de la dirección apuntada por el puntero de pila sp; en este caso 0x100000c. Esta dirección de retorno es la que corresponde al retorno de la subrutina coo. Al terminar esta ejecución, el contador de programa apunta a la siguiente instrucción que se encuentra cuatro posiciones de memoria más adelante.

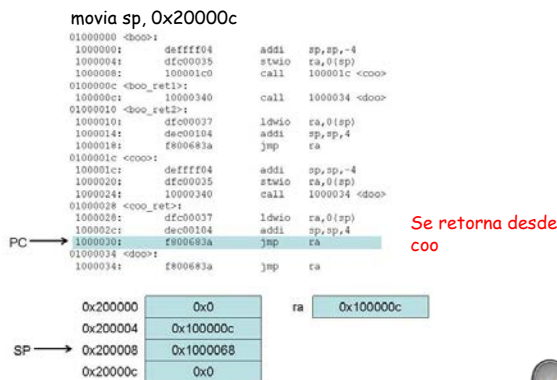
## Llamadas anidadas a subrutinas



Lección 1e/15

La siguiente instrucción actualiza el puntero de pila sp (0x200008), cuyo contenido indica la siguiente dirección de retorno anidada. Al terminar esta ejecución, el contador de programa apunta a la siguiente instrucción que se encuentra 4 posiciones de memoria más adelante.

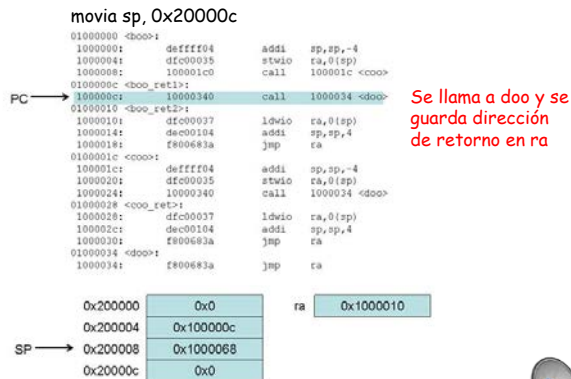
## Llamadas anidadas a subrutinas



Lección 1e/16

La siguiente instrucción produce el retorno a la dirección que guarda el registro ra y cuyo valor es 0x100000c que corresponde a la etiqueta boo\_ret1. Al terminar esta ejecución, el contador de programa apunta a la instrucción de la subrutina boo que está en la dirección 0x100000c, una llamada a la subrutina doo.

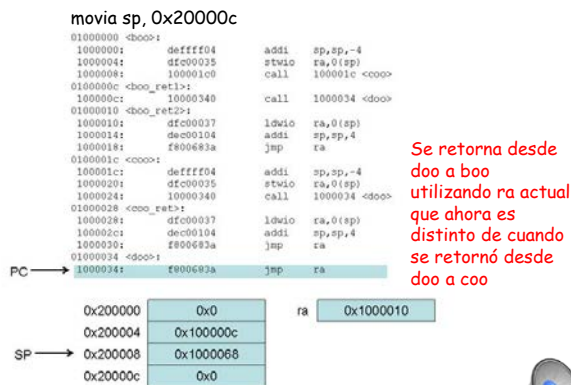
## Llamadas anidadas a subrutinas



Lección 1e/17

Como resultado de la ejecución de esta instrucción de llamada a la subrutina doo desde la subrutina boo, se produce por segunda vez el salto a la rutina doo, y el registro ra guarda la dirección de retorno que sería cuatro posiciones más adelante: 0x1000010. Al terminar esta ejecución, el contador de programa apunta a la primera instrucción de la subrutina doo que está en la dirección 0x1000034.

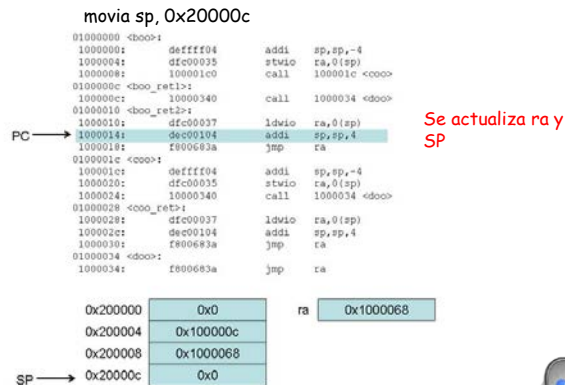
## Llamadas anidadas a subrutinas



Lección 1e/18

Se vuelve a ejecutar la subrutina doo. La subrutina doo consta de una sola instrucción que es un retorno de subrutina. En este caso se retorna a la dirección que guarda el registro ra y cuyo valor es 0x1000010 que corresponde a la etiqueta boo\_ret2. Al terminar esta ejecución, el contador de programa apunta a la instrucción de la subrutina boo que está en la dirección 0x1000010, una carga ldwio.

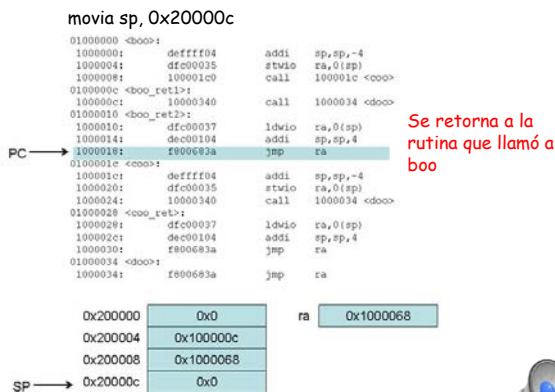
## Llamadas anidadas a subrutinas



Lección 1e/19

Se ejecuta ahora una instrucción de carga del registro ra con el contenido de la memoria apuntado por el puntero de pila sp, ra= 0x1000068. La siguiente instrucción produce una actualización del puntero de pila al valor inicial que es 0x20000c. Al terminar esta ejecución, el contador de programa apunta a la siguiente instrucción que se encuentra 4 posiciones de memoria más adelante.

## Llamadas anidadas a subrutinas



Lección 1e/20

Por último, la última instrucción de la subrutina boo hace retornar la ejecución del programa a la subrutina que llamó inicialmente a boo.

## EJERCICIOS DE AUTOEVALUACIÓN

- ¿Qué es el lenguaje de alto nivel?
- ¿Qué es el lenguaje ensamblador?
- ¿Qué es el lenguaje máquina?
- ¿Qué es una instrucción?
- ¿Qué es una microoperación?
- ¿Qué es la arquitectura del repertorio de instrucciones de un procesador? ¿Cuáles son los principales elementos que forman parte de la arquitectura ISA?
- ¿Qué representa NIOS II?
- ¿Cuáles son los principales elementos de la estructura del un computador basado en NIOSII
- ¿Qué son los modos de funcionamiento de un procesador?
- ¿Para qué se utilizan los registros de propósito general de un procesador?
- ¿Para qué se utilizan los registros de control de un procesador? ¿Cómo se accede a ellos?
- ¿Qué función realiza el Registro de Estado del procesador?
- ¿Qué es el espacio de direccionamiento de un procesador?
- ¿Qué son los modos de direccionamiento?
- ¿Cuáles son los tipos más frecuentes de modos de direccionamiento?
- ¿Qué modo de direccionamiento utiliza NIOSII?
- ¿Qué son los formatos de instrucción?
- ¿Cuántos tipos de formatos de instrucción tiene NIOSII? Describe los campos en los que se divide cada uno de ellos
- ¿Qué son los tipos de instrucciones?
- ¿Cuántos tipos de instrucciones tiene NIOS II?
- ¿En qué se caracterizan las instrucciones de acceso a memoria?
- ¿En qué se caracterizan las instrucciones aritmético-lógicas?
- ¿En qué se caracterizan las instrucciones de copia entre registros?
- ¿En qué se caracterizan las instrucciones de comparación?
- ¿En qué se caracterizan las instrucciones de salto incondicional?
- ¿En qué se caracterizan las instrucciones de salto condicional?
- Realiza un programa en el lenguaje ensamblador de NIOS II que calcule una serie Fibonacci de 16 elementos.

**Bibliografía:**

- NIOS II Processor Reference Handbook, Altera, n2cpu\_nii5v1.pdf
- Programming Model (Chapter 3: General-Purpose Registers, Control Registers) of the Nios II Processor Reference Handbook (n2cpu\_nii51003.pdf, [http://www.altera.com/literature/hb/nios2/n2cpu\\_nii51003.pdf](http://www.altera.com/literature/hb/nios2/n2cpu_nii51003.pdf))
- Instruction Set Reference of the Nios II Processor Reference Handbook (n2cpu\_nii51017.pdf)
- IPAD 2: <http://t0.gstatic.com/images?q=tbn:ANd9GcRfDrtFPzrS1krzLZNNK0OBOIScjKqMXFwyBsSQ76j9jY7EYVhjQQ>
- Altera, Basic Computer System for the Altera DE2 Board (DE2\_Basic\_Computer.pdf)
- <http://www.eecg.toronto.edu/~moshovos/ECE243-2013/lec9%20-%20subroutines.htm>