MICROPROCESADOR DIDÁCTICO DE ARQUITECTURA RISC IMPLEMENTADO EN UN FPGA

DIDACTIC MICROPROCESSOR OF RISC ARCHITECTURE IMPLEMENTED IN A FPGA

Víctor H. García Ortega[1], Julio C. Sosa Savedra[1], Susana Ortega S.[2] y Rubén H. Tovar.[1]
vgarciao@ipn.mx / jcsosa@ipn.mx / susana.ortega@cucei.udg.mx / rhtovar@ipn.mx

Recibido: 02 marzo, 2009 / Aceptado: 01 mayo, 2009 / Publicado: 31 diciembre, 2009

RESUMEN. En este trabajo se presenta el diseño e implementación de un microprocesador de 16 bits de arquitectura RISC tipo MIPS. Las características principales que presenta el microprocesador son: una arquitectura Harvard con una memoria de programa de 64K x 25 bits y una memoria de datos de 64k x 16 bits, una pila de ocho niveles implementada en hardware y la Unidad Lógica-Aritmética (ALU) con esquema de acarreo anticipado por propagación y generación para el aumento de velocidad en la ejecución de las operaciones aritméticas. La unidad de control decodifica y maneja las señales de control del microprocesador para ejecutar las instrucciones en un solo ciclo de reloj. La implementación se realizó en un dispositivo lógico programable del tipo FPGA (*Field Programmable Gate Array*) de la familia Spartan-3A de Xilinx. El diseño de cada elemento del microprocesador se desarrolló con el lenguaje de descripción de hardware VHDL.

PALABRAS CLAVE: microprocesador RISC, MIPS, VHDL.

ABSTRACT. This paper presents the design and implementation of a RISC architecture microprocessor. The main microprocessor features are: Harvard Architecture, 64k x 25 bits program memory, 64kB x 16 bits data memory, an eight levels hardware stack and Aritmetic Logic Unit (ALU). The ALU was designed with a carry anticipate scheme by propagation and generation for improving the performance in arithmetic operations. The control unit decodes and drives the microprocessor's control signals to execute each instrucction in one clock cicle. The implementation was made in a Field Programmable Gate Array (FPGA) of Xilinx's Sparta-3A family. Each microprocessor component was designed using the VHSIC Hardware Description Language (VHDL).

KEYWORDS: RISC microprocessor, MIPS, FPGA, VHDL

Introducción

Uno de los aspectos relacionados con los computadores que han evolucionado de manera más visible es el de los lenguajes de programación [1]. Los lenguajes de programación de alto nivel permiten al programador expresar algoritmos de manera más concisa, pero ocasionan otro problema conocido como el *salto semántico*: la diferencia entre las operaciones que proporcionan los lenguajes de alto nivel y las que proporciona la arquitectura del computador. De esta manera, se puede diferenciar entre dos filosofías de diseño de arquitecturas de unidades de procesamiento: Computador de un Conjunto de Instrucciones Reducido (RISC) y Computador de un Conjunto de Instrucciones Complejo (CISC).

Por otro lado, en los últimos años los sistemas integrados han venido incrementando su potencial, tanto de procesado como de integración en un mismo chip, para dar posibilidad a sistemas más complejos. De esta manera es posible utilizar un procesador concreto o hacer uso de la lógica programable. Los *system on chip* (SoC) permiten implementar en un único dispositivo sistemas complejos que antes empleaban varios

[1] Centro Universitario de Ciencias Exactas e Ingenierías (CUCEI), Blvd. Marcelino García Barragán #1421, Guadalajara, 44430, Jalisco, México - www.cucei.udg.mx
[2] Escuela Superior de Cómputo (ESCOM) del Instituto Politécnico Nacional, Unidad Profesional Adolfo López Mateos

circuitos integrados en un mismo circuito impreso. Al emplear un FPGA se tiene la ventaja de poder emplearse para el prototipado de un ASIC o bien ser un sistema final. Además se cuenta con la capacidad de expandir el sistema en caso de ser necesario.

Existen varios microprocesadores RISC empleados en SoC, como son: LEON [**2**], OpenRISC [**3**], MicroBlaze [**4**], Nios II [**5**], Cortex-M1 [**6**], entre otros. En este trabajo se presenta el diseño de una arquitectura RISC, dejando abierta la arquitectura para un futuro desarrollo de un SoC, embebiendo un sistema operativo.

Las principales características que presenta una arquitectura RISC [**7**] son:

- Un conjunto limitado y simple de instrucciones. Se cuenta con un conjunto constituido por instrucciones capaces de ejecutarse en un ciclo de reloj.
- Instrucciones orientadas a los registros con acceso limitado a memoria. Un conjunto de tipo RISC ofrece pocas instrucciones básicas (*Load* y *Store*) que pueden ingresar datos en la memoria. El resto de ellas operan exclusivamente con registros
- Modos limitados de direccionamiento. Muchas computadoras de tipo RISC ofrecen sólo un modo para direccionar la memoria, generalmente un direccionamiento directo o indirecto de registros con un desplazamiento.
- Un gran banco de registros. Los procesadores de tipo RISC contienen muchos registros de manera que las variables y los resultados intermedios usados durante la ejecución del programa no requieran utilizar la memoria. Con ello se evitan muchas instrucciones del tipo *Load* y *Store*.
- Palabra de la instrucción con extensión y formatos fijos. Al hacer idénticos el tamaño y el formato de todas las instrucciones, es posible obtenerlas y decodificarlas por separado. No hay que esperar hasta conocer la extensión de una instrucción anterior a fin de obtener y decodificar la siguiente. Por tanto, esas dos acciones pueden llevarse a cabo en paralelo. En pocas palabras, la decodificación se simplifica.

Muchos procesadores han adquirido esta arquitectura por ser una de las más eficientes. Las arquitecturas más didácticas usadas en los ambientes universitarios son las basadas en MIPS [**8**, **9**]. Se conoce como MIPS (*Microprocessor without Interlocked Pipeline Stages*) a toda una familia de procesadores de arquitectura RISC desarrollados por MIPS Technologies, Inc. [**10**]. Estos procesadores han permitido enseñar los elementos básicos de una arquitectura de cómputo a estudiantes de un nivel universitario, sentando las bases para el aprendizaje de arquitecturas más complejas.

Arquitectura del Microprocesador

El microprocesador desarrollado en este trabajo tiene arquitectura MIPS de 16 Bits para la ejecución de instrucciones con punto fijo. El procesador fue llamado ESCOMIPS, puesto que fue desarrollado en la Escuela Superior de Cómputo (ESCOM) del IPN. El procesador, mostrado en la **Figura 1**, presenta las siguientes características:

- Formato de instrucción de 25 bits para todas las instrucciones.
- Cada instrucción se ejecuta en un ciclo de reloj.
- Archivo de 16 registros de trabajo.
- Memoria de programa y memoria de datos separada, es decir, Arquitectura Harvard.
- El contador de programa puede direccionar hasta 64 kwords.
- En memoria de datos se puede direccionar hasta 64 kwords + 4 kwords.

Conjunto de instrucciones

El conjunto de instrucciones tiene un formato de 25 bits con el que podemos realizar instrucciones con 3 operandos. Básicamente se tienen 3 formatos de instrucciones que son: formato tipo R, formato tipo I y formato tipo J. Los formatos de instrucción, que se muestran en la **Tabla 1**, permiten manejar los modos de direccionamiento inmediato, por registro, directo e indirecto.

Los formatos de instrucción están constituidos por varios campos: **OP**: código de operación; **Rd**: registro operando destino; **Rt**: primer registro operando fuente; **Rs**: segundo registro operando fuente; **SHAMT**: No. de bits a desplazar en las instrucciones de corrimiento; **FUNCT**: campo que selecciona una variante de la instrucción especificada en el campo OP, también llamado código de función; **Constante o dirección**: dato de 16 o 12 bits que representa un número inmediato o dirección.

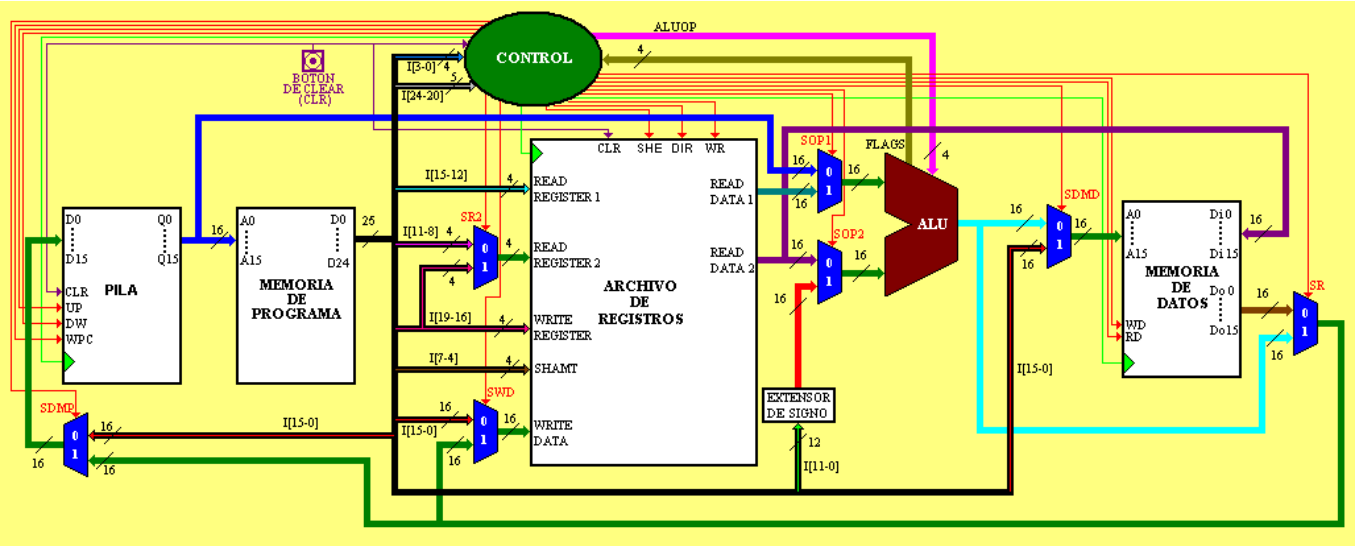

Figura 1. Arquitectura del microprocesador MIPS

Tabla 1. Formatos de instrucción

Formato tipo R	24.....20	19.....16	15.....12	11.....8	7.....4	3.....0
	OP	Rd	Rt	Rs	SHAMT	FUNCT
	5 bits	4 bits	4 bits	4 bits	4 bits	4 bits
Formato tipo I (16 bits)	24.....20	19.....16	15.....0			
	OP	Rd	Constante o Dirección			
	5 bits	4 bits	16 bits			
Formato tipo I (12 bits)	24.....20	19.....16	15.....12	11.....0		
	OP	Rd	Rt	Constante o Dirección		
	5 bits	4 bits	4 bits	12 bits		
Formato tipo J	24.....20	19.....16	15.....0			
	OP	Sin uso	Constante o dirección			
	5 bits	4 bits	16 bits			

Con los formatos de instrucción propuestos se formó un conjunto con 32 instrucciones, las cuales se muestran en la **Figura 2**.

Las instrucciones de brinco condicionales inmediatas (*BEQI, BNEI, BLTI, BGTI*) realizan la comparación de los registros y su respectivo salto en un solo ciclo de reloj. En estas instrucciones la dirección de salto es de 12 bits, permitiendo lo que conocemos como saltos relativos en una ventana de 4K hacia adelante y hacia atrás de la posición actual del registro contador de programa. Para poder hacer saltos condicionales sobre todo el mapa de la memoria de programa usamos las instrucciones *BEQ, BNE, BLT y BGT* en conjunto con la instrucción de comparación *CMP*, tomando dos ciclos para realizar tanto la comparación como el brinco condicional.

Memoria de programa

Esta memoria contiene todas las instrucciones que ejecuta el microprocesador. Su organización es de 64kx25; es decir, tenemos capacidad para 65,536 instrucciones y cada una está formada por un dato de 25 bits. Este dato corresponde con el código de 25 bits de la instrucción a ejecutar por el procesador. La instrucción que sale de la memoria se distribuye hacia la unidad de control, el archivo de registros, ALU y memoria de datos.

Pila

El procesador tiene una pila en hardware de 8 niveles con la que podemos implementar las instrucciones de saltos, o llamadas a subrutinas, de forma rápida al usar 8 registros contadores de programa (PC) los cuales son manejados por un contador llamado Apuntador de pila.

Instr.	Ejemplo	Significado	Instr.	Ejemplo	Significado
LWI	LWI Rd, #num16	Rd=num	*SLL*	SLL Rd,Rt, #num4	Rd=Rt<<num
LW	LW Rd, D(Rt)	Rd=Mem[Rt+D]	*SRL*	SRL Rd,Rt, #num4	Rd=Rt>>num
SWI	SWI Rd, D	Mem[D] = Rd	*BEQI*	BEQI Rd,Rt,D	If(Rd==Rt)goto D
SW	SW Rd, D(Rt)	Mem[Rt+D]=Rd	*BNEI*	BNEI Rd,Rt,D	If(Rd!=Rt) goto D
ADD	ADD Rd,Rt,Rs	Rd=Rt+Rs	*BLTI*	BLTI Rd,Rt,D	If(Rd<Rt) goto D
SUB	SUB Rd,Rt,Rs	Rd=Rt-Rs	*BGTI*	BGTI Rd,Rt,D	If(Rd>Rt) goto D
ADDI	ADDI Rd,Rt, #num12	Rd=Rt+num	*CMP*	CMP Rt, Rs	Rt – Rs
SUBI	SUBI Rd,Rt, #num12	Rd=Rt-num	*BEQ*	BEQ D	If(Rd==Rt) goto D
AND	AND Rd,Rt,Rs	Rd=Rt&Rs	*BNE*	BNE D	If(Rd!=Rt) goto D
OR	OR Rd,Rt,Rs	Rd=Rt | Rs	*BLT*	BLT D	If(Rd<Rt) goto D
NAND	NAND Rd,Rt,Rs	Rd=~(Rt & Rs)	*BGT*	BGT D	If(Rd>Rt) goto D
NOR	NOR Rd,Rt,Rs	Rd=~(Rt | Rs)	*SLT*	SLT Rd,Rt,Rs	If(Rt<Rs)Rd=1, Else Rd=0
ANDI	ANDI Rd,Rt, #num12	Rd=Rt & num	*SLTI*	SLTI Rd,Rt, #num12	If(Rt<num)Rd=1, Else Rd=0
ORI	ORI Rd,Rt, #num12	Rd=Rt | num	*B*	B D	PC = D
NANDI	NANDI Rd,Rt, #num12	Rd=~(Rt & num)	*CALL*	CALL D	PC(n+1) = D
NORI	NORI Rd,Rt, #num12	Rd=~(Rt | num)	*RET*	RET	PC = PC(n-1)

Figura 2. Conjunto de instrucciones

Archivo de Registros

Esta sección del procesador está formada por 16 registros de 16 bits cada uno, los cuales llamaremos R0, R1, R2…R15. Estos registros son los que contienen los datos que usan las instrucciones del procesador. En este archivo se pueden acceder hasta dos registros de forma simultánea para la ejecución de las instrucciones tipo R y tipo I. Estos 16 registros tienen la función de un registro Barrel Shifter, con lo que podemos ejecutar las instrucciones de corrimiento sobre cualquiera de los registros en un solo ciclo de reloj.

ALU

La ALU es de 16 bits y está implementada con un esquema de acarreo anticipado para tener un tiempo de respuesta de 2 retardos de propagación de forma constante para la obtención de los acarreos. De cada resultado se obtienen los valores de las banderas de C (*carry*), N (*Negative*), Z (*Zero*) y OV (*overflow*). Estas banderas son almacenadas en el registro de estado o banderas para poder ser usadas con las instrucciones de comparación y brincos condicionales. Esta unidad, además de realizar las operaciones aritméticas (suma y resta) y lógicas (*AND*, *OR*, *NAND*, *NOR*) del conjunto de instrucciones, también realiza el cálculo de las direcciones para el manejo de bloques de datos o arreglos en la memoria de datos.

Memoria de datos

La memoria de datos tiene una organización de 64Kx16. En esta memoria se guardan todos los datos que usamos para las variables y arreglos en un programa. En la implementación de esta memoria, la operación de escritura se realizó de manera síncrona usando la señal de control WD (*Write Data*), mientras que la operación de lectura funciona de manera asíncrona usando la señal de control RD (*Read Data*).

Unidad de control

La unidad de control es el cerebro del microprocesador. Esta unidad decodifica los códigos de operación y los de función para poder determinar la instrucción que se va a ejecutar. Una vez determinada la instrucción, la unidad de control activa o no cada una de las señales de control de todo el microprocesador. De esta manera se forma un código que emite la unidad de control al que llamamos micro-instrucción. Por lo tanto, podemos definir a **las micro-instrucciones como los códigos que emite la unidad de control para ejecutar cada instrucción del ensamblador**. Los bloques que contiene la unidad de control se muestran en la **Figura 3**.

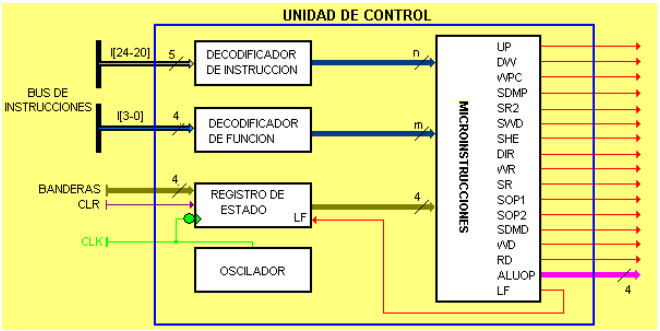

Figura 3. Unidad de control del microprocesador MIPS

```
        LWI     R0,   #0        ; R0 = 0
        LWI     R1,   #0        ; R1 = 0      contador i
        LWI     R9,   #4        ; R9 = 4      limite del contador j
        LWI     RA,   #3        ; RA = 3      limite del contador i
CICLO_I:
        ADDI    R2, R1, 1       ; R2 = R1 +1  contador j
        ADDI    R8, R1, 0       ; R8 = R1
CICLO_J:
        LW      R3, 0(R1)       ; R3 = MEM[R1]
        ADDI    R7, R2, 0       ; R7 = R2
        LW      R4, 0(R2)       ; R4 = MEM[R2]
        CMP     R3, R4
        BLT     INCREMENTO_J    ; IF(R3 < R4 ) GOTO INCREMENTO_J
INTERCAMBIO:
        ADDI    R6, R3, 0       ; R6 = R3
        ADDI    R3, R4, 0       ; R3 = R4
        ADDI    R4, R6, 0       ; R4 = R6
        SW      R4, 0(R2)       ; MEM[R2] = R4
        SW      R3, 0[R1]       ; MEM[R1] = R3
INCREMENTO_J:
        ADDI    R2, R7, 1       ; R2 = R7 + 1
        CMP     R2, R9
        BNE     CICLO_J         ; IF(R2 = ! R9) GOTO CICLO_J
INCREMENTO_I:
        ADDI    R1, R8, 1       ; R1 = R8 + 1
        CMP     R1, RA
        BNE     CICLO_I         ; IF(R1 != RA) GOTO CICLO_I
CICLO:
        NOP
        B       CICLO
```

Figura 4. Programa de la burbuja

Resultados

Cada uno de los componentes del microprocesador fue programado con VHDL usando el ambiente de desarrollo ISE WEBPACK 10.1 de Xilinx. También cada componente fue simulado con el software ISE Simulator incluido en el ambiente ISE WEBPACK. Una vez simulado cada componente, se realizaron varios programas usando el conjunto de instrucciones propuesto para el ESCOMIPS. Uno de ellos es el que realiza el ordenamiento de varios números con el algoritmo clásico de ordenamiento por burbuja. Con este programa se probó el manejo de arreglos usando el modo de direccionamiento indirecto, la ejecución de las instrucciones de carga, almacenamiento, brinco incondicional, brinco condicional y la ruta de datos por el microprocesador. Este programa se muestra en la **Figura 4**. La simulación del programa de ordenamiento por burbuja se muestra en la **Figura 5**.

Las pruebas se realizaron en una tarjeta de desarrollo que posee el FPGA XC3S700A, de la familia Spartan 3A. Se trabajó a una frecuencia de reloj de 50 MHz y cada instrucción se ejecuta en un ciclo de reloj; por tanto, el procesador puede ejecutar 50 MIPS.

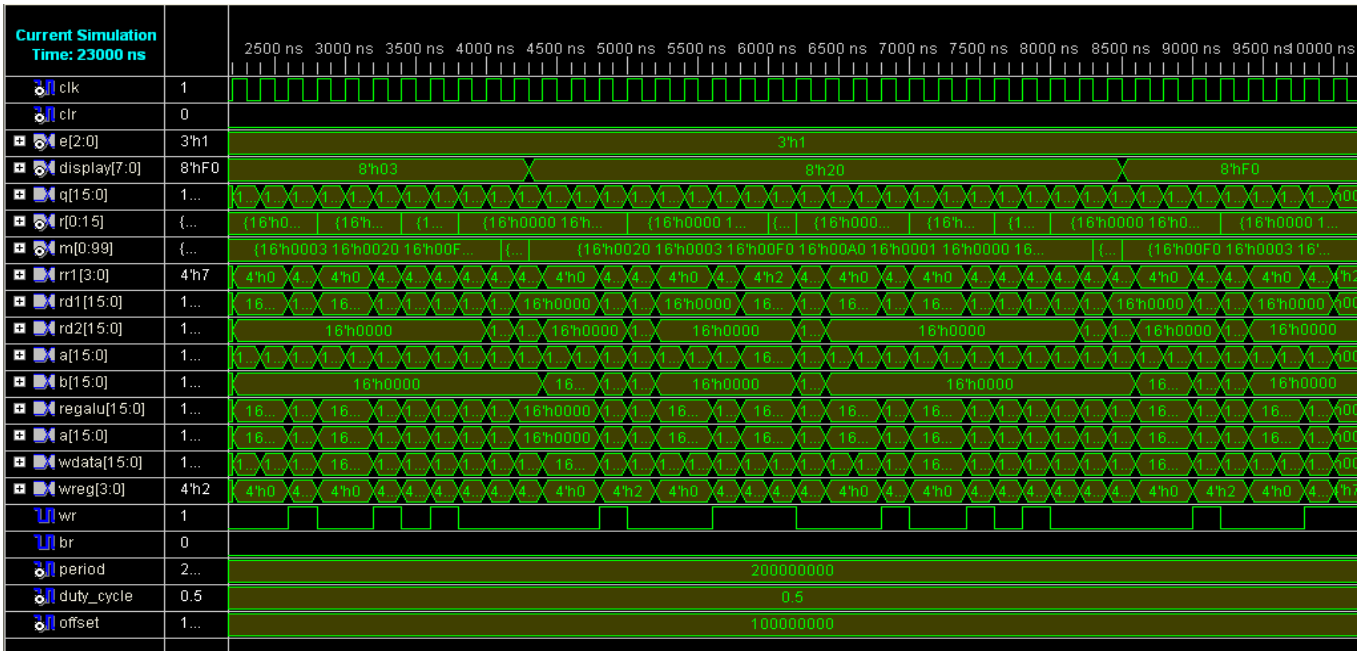

Figura 5. Simulación del programa de la burbuja

Conclusiones y perspectivas

En este trabajo se presentó el diseño, implementación y pruebas de un microprocesador RISC cuyo conjunto de instrucciones permite poder ejecutar cualquier algoritmo computable. Cada instrucción se ejecuta en un ciclo de reloj haciendo al procesador eficiente en la ejecución de algoritmos. Dentro de las aportaciones de la investigación educativa se pueden mencionar: el diseño didáctico de 16 bits del microprocesador, lo que lo hace aplicable para los cursos de Arquitectura de computadoras, microprocesadores, entre otros a nivel licenciatura. Con este diseño se sientan las bases para el estudio de arquitecturas más complejas como los procesadores escalares, vectoriales y otras arquitecturas.

El trabajo a futuro inmediato será la segmentación de la ruta de datos del microprocesador (*pipeline*) para lograr reducir el ciclo de instrucción y, así, aumentar la frecuencia del reloj y el rendimiento del procesador. Posteriormente se plantea implantar en un SoC incorporando un sistema operativo de código abierto. Existen trabajos [**11**] en donde se implementa el sistema Linux, basado en el procesador LEON 2. De esta manera, con este trabajo se inicia una etapa de diseño para implantar todo un sistema de desarrollo propio.

Agradecimientos

Este trabajo ha sido financiado por los proyectos de investigación SIP: 20091669 y SIP: 20091661.

Referencias

1. William Stallings. 2006. *Computer Organization and Architecture*, Seventh Edition. Prentice Hall.
2. http://1-core.com/resources/
3. http://www.opencores.org/?do=project&who=or1k
4. http://www.xilinx.com/products/design_resources/proc_central/microblaze.htm
5. http://www.altera.com/nios2
6. http://www.arm.com/products/CPUs/ARM_Cortex-M1.html
7. Englander Irv. 2002. *Arquitectura Computacional*. Editorial CECSA.
8. Patterson David, Hennessy John. 2008. *Computer Organization and Design*. Elsevier, The Morgan Kaufmann Series.
9. Behrooz Parhami. 2005. *Computer Architecture: From Microprocessors to Supercomputers*. Oxford Series in electrical and computer engineering.
10. http://www.mips.com/
11. Hipólito Guzmán Miranda, Jonathan Noel Tombs, Miguel Angel Aguirre Echanove(2006). *Implementación de Sistemas Integrados Linux Basados en el Procesador Leon 2*. FPGAS, Metodologías, Herramientas y Experiencias: VI Jornadas Sobre Computación Reconfigurable y Aplicaciones. JCRA'06. Cáceres, España. pp. 11-15. ISBN: 84-611-1314-4.