LA MÁQUINA RUDIMENTARIA UN PROCESADOR PEDAGÓGICO

Enric Pastor y Fermín Sánchez
Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya

Resumen

En este artículo se presenta un procesador pedagógico denominado Máquina Rudimentaria. Se incluye la definición de la arquitectura del procesador, la especificación de los lenguajes máquina y ensamblador y una implementación de la unidad de control y de la unidad de proceso. El entorno de trabajo de este procesador permite la creación de programas en lenguaje ensamblador, su posterior traducción a lenguaje máquina y una simulación lógica interactiva de su ejecución.

1 Introducción

La *Máquina Rudimentaria* (MR) [1] es un procesador pedagógico. Su principal objetivo es servir como herramienta para enseñar los conceptos básicos sobre estructura y arquitectura de computadores a un estudiante de primer nivel de unos estudios de ingeniería. El diseño de la MR se ha realizado utilizando conocimientos básicos sobre sistemas digitales, por lo cual puede utilizarse como ejemplo de un sistema complejo en un curso de lógica digital, permitiendo realizar una transición natural entre el estudio de los circuitos digitales y el del lenguaje máquina. La MR ha sido diseñada en el Departament d'Arquitectura de Computadores (DAC) de la Universitat Politècnica de Catalunya.

A mediados de los 80, los profesores del DAC Ayguadé, Navarro y Valero diseñaron la *Máquina Sencilla* [2], un procesador pedagógico que se explicaba en el primer curso de los estudios de la Facultat d'Informática de Barcelona. La *Máquina Sencilla* es un procesador de tipo von Neumann, con 128 palabras de memoria de 16 bits e instrucciones de 16 bits de longitud. El acceso a los operandos se realiza con el modo de direccionamiento absoluto. La *Máquina Sencilla* tiene cuatro instrucciones: *sumar, mover, comparar y saltar si igual*, y un indicador de cero. Las instrucciones son de dos operandos, y usan el segundo operando además como operando destino. La arquitectura de la *Máquina Sencilla* permite introducir conceptos básicos sobre el funcionamiento de las instrucciones, pero carece de un juego de instrucciones variado y no dispone de distintos métodos de direccionamiento ni de un banco de registros. Por ello, el DAC se planteó utilizar un procesador más complejo. Por diversas razones, se descartó usar un procesador comercial (por ejemplo, el VAX [3] por tener un conjunto demasiado amplio de instrucciones y métodos de direccionamiento, o el Intel 8086 [4] por ser muy poco ortogonal). También se pensó en usar un procesador pedagógico ya diseñado, como el DLX [5]. Este procesador está bien documentado y permite la posterior introducción de conceptos más avanzados. Sin embargo, es todavía demasiado complejo para ser asimilado completamente por un estudiante de primer curso. Por ello, a principios de los 90 algunos profesores del DAC comenzaron a trabajar en un nuevo procesador pedagógico de tipo RISC lo más sencillo posible: la MR.

La arquitectura y el lenguaje máquina de la MR se introducen en la Sección 2. El diseño de la Unidad de Proceso y la Unidad de Control se presentan en las Secciones 3 y 4. El lenguaje ensamblador y el entorno diseñado para el ensamblaje y simulación de programas se describen en las Secciones 5 y 6. Finalmente, las conclusiones de este trabajo se discuten en la Sección 7.

2 Arquitectura de la MR

2.1 Estructura básica de la MR

La MR es un procesador de tipo von Neumann. La memoria es de 256 posiciones de 16 bits; tiene dos buses de datos de 16 bits, uno de entrada y otro de salida, y una señal de control de lectura/escritura. Por simplicidad, la MR no tiene unidad de Entrada/Salida. La unidad central de proceso (UCP) está formada por la Unidad de Proceso (UP) y la Unidad e Control (UC), descritas en las Secciones 3 y 4.

Las instrucciones de la MR trabajan con números enteros de 16 bits codificados en complemento a dos. La UCP dispone de dos indicadores (*flags*) de condición: el indicador de cero (Z) y el indicador de signo (N). También tiene un banco de 8 registros de 16 bits, numerados desde R0 hasta R7. El registro R0 es un registro especial que siempre contiene el valor 0.

2.2 Lenguaje Máquina de la MR

Las instrucciones de la MR son de longitud fija de 16 bits. El código de operación se encuentra en los dos bits de mayor peso. Se dividen en tres grupos:

- Instrucciones aritmético-lógicas.
- Instrucciones de acceso a memoria.
- Instrucciones de salto.

La MR dispone de cuatro modos de direccionamiento:

- Modo registro.
- Modo inmediato.
- Modo desplazamiento (o base + desplazamiento).
- Modo absoluto.

El repertorio de instrucciones de la MR se describe a continuación.

2.2.1 Instrucciones aritmético-lógicas

Las instrucciones aritmético-lógicas usan el modo registro y el modo inmediato. En el modo registro se accede a un registro del banco de registros. El operando inmediato (en aquellas instrucciones que lo utilizan) es un número de 5 bits codificado en complemento a dos y contenido dentro de la propia instrucción. Todas las instrucciones aritmético-lógicas escriben su resultado en un registro (Rd) y activan los indicadores de condición N y Z. Si el registro indicado es el R0 no se escribe el resultado, y solamente se activan los indicadores N y Z.

Se distinguen dos tipos de instrucciones aritmético-lógicas, en función de donde están sus operandos fuente: las que leen los operandos de registros y las que tienen un operando en un registro y el otro operando es inmediato. La MR dispone de seis instrucciones aritmético-lógicas:

- **Suma con registros**: suma (el contenido de) dos registros (Rf1 + Rf2).
- **Resta con registros**: resta dos registros (Rf1 - Rf2).
- **Desplazamiento aritmético**: desplaza 1 bit a la derecha con extensión de signo un registro (Rf >> 1). Esta instrucción tiene un solo operando fuente.
- **And**: *and* lógica bit a bit de dos registros (Rf1 *and* Rf2).
- **Suma con inmediato**: suma un registro y el operando inmediato extendido a 16 bits (Rf + #n).
- **Resta con inmediato**: resta el operando inmediato extendido a 16 bits de un registro (Rf - #n).

La Figura 1(a) presenta la codificación utilizada para cada una de estas instrucciones. Todas las instrucciones aritmético-lógicas usan el mismo código de operación (11). El campo OP (bits 2-0) permite identificar a cada instrucción, según se indica en la Figura 1(b). El bit OP_2 distingue los dos tipos de instrucciones aritmético-lógicas: registro-registro (OP_2=1) y registro-inmediato (OP_2=0).

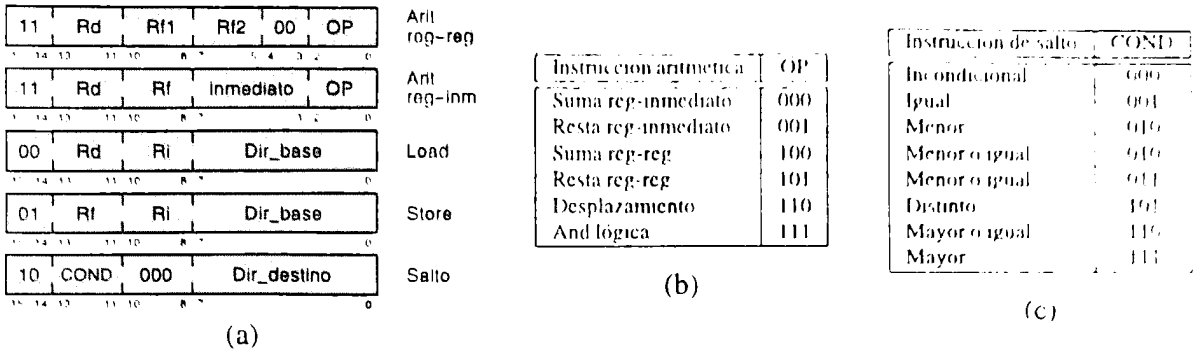

Figura 1: Formato de las instrucciones en el Lenguaje Máquina de la MR.

2.2.2 Instrucciones de acceso a memoria

Las instrucciones de acceso a memoria usan el modo registro y el modo desplazamiento. La dirección de memoria a la que acceden se calcula sumando los 8 bits de menor peso de un registro del banco de registros, denominado registro índice (Ri), con una dirección codificada en la propia instrucción (dir_base). Existen dos instrucciones de acceso a memoria (ver Figura 1(a)):

- **Load**: almacena en un registro (Rd) el contenido de una posición de memoria y activa los indicadores N y Z. Si el registro destino es R0, no se escribe. Su código de operación es 00.
- **Store**: almacena en una posición de memoria el contenido de un registro (Rf). Esta instrucción no altera el valor de los indicadores de condición. Su código de operación es 01.

2.2.3 Instrucciones de salto

Las instrucciones de salto usan el modo absoluto para indicar la dirección de salto. Se requieren 8 bits para identificar una dirección absoluta. Ninguna instrucción de salto altera el valor de los indicadores de condición. Existen seis instrucciones de salto condicional y una de salto incondicional:

- **Saltar si mayor**: se produce el salto si N=0 y Z=0.
- **Saltar si mayor o igual**: se produce el salto si N=0.
- **Saltar si menor**: se produce el salto si N=1.
- **Saltar si menor o igual**: se produce el salto si N=1 ó Z=1.
- **Saltar si igual**: se produce el salto si Z=1.
- **Saltar si distinto**: se produce el salto si Z=0.
- **Saltar incondicionalmente**: se produce el salto siempre.

La Figura 1(a) presenta la codificación utilizada para cada una de estas instrucciones. Todas las instrucciones de salto usan el código de operación (10). El campo COND (bits 13-11) permite identificar a cada instrucción, según se indica en la Figura 1(c).

3 Unidad de Proceso

La UP de la MR se muestra en la Figura 2. Es capaz de ejecutar las instrucciones descritas en la Sección 2, y ha sido diseñada con el doble objetivo de ser eficiente y de fácil comprensión. La UP tiene 6 registros especiales, cuya carga está gestionada por la UC, y su función es la siguiente:

- **Contador de programa (PC)**: almacena la dirección de la siguiente instrucción a ejecutar.
- **Registro de instrucción (IR)**: almacena la instrucción en ejecución.
- **Registro de direcciones (R@)**: almacena la dirección a la que accede a memoria una instrucción Load, Store o una instrucción de salto.
- **Registro de la UAL (RA)**: almacena el primer operando de una instrucción aritmético-lógica.
- **Registros de condición (RN y RZ)**: almacenan el valor de los indicadores de condición N y Z.

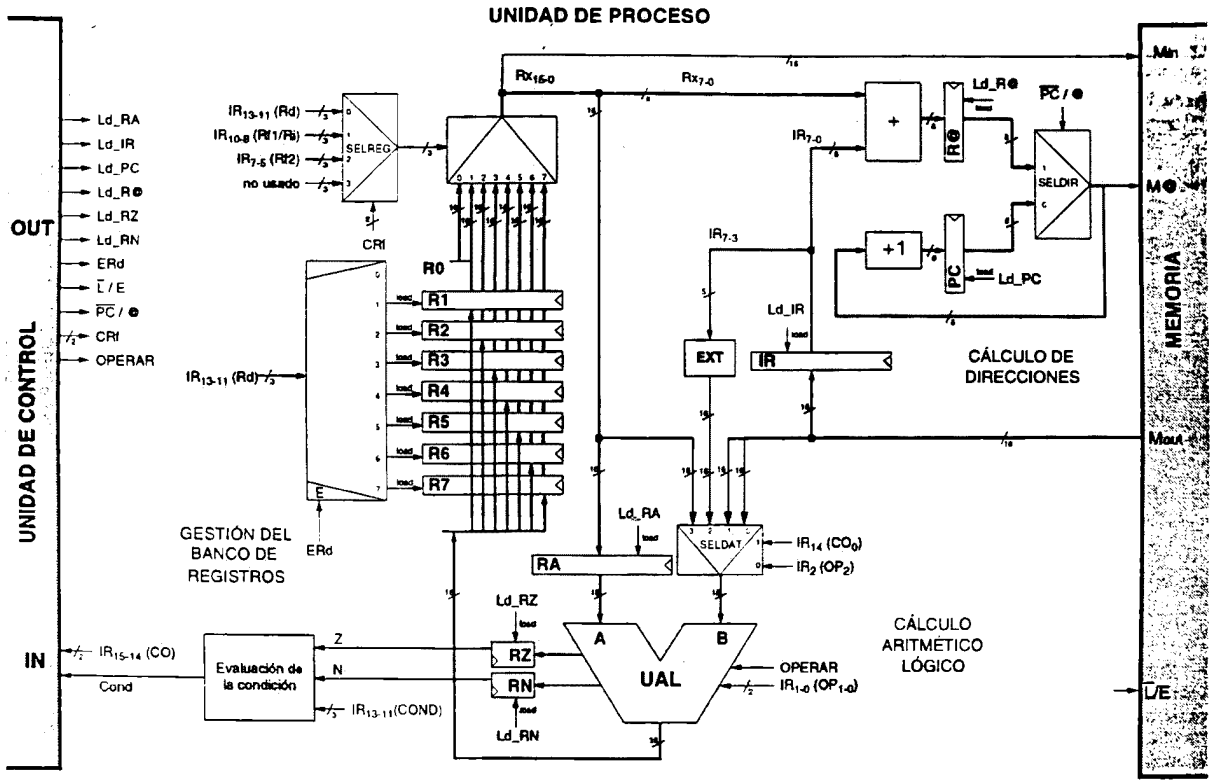

Figura 2: Unidad de Proceso de la MR.

En la UP se distinguen dos circuitos sencillos que realizan funciones específicas y tres grandes bloques:

- **Evaluación de la condición**: es un circuito combinacional que recibe como entrada el valor actual de los indicadores de condición N y Z y los tres bits que codifican el campo COND de una instrucción de salto. Envía un bit a la UC que indica si el salto debe o no producirse.
- **Extensión de signo (EXT)**: realiza la extensión de signo a 16 bits del operando inmediato.
- **Bloque de gestión del banco de registros**: El banco de registros contiene 8 registros de 16 bits (R0 siempre vale cero). Tiene un puerto de salida y un puerto de entrada, y permite lectura y escritura simultánea. La señal de escritura (ERd) está gobernada por la UC. La codificación del registro a escribir se lee directamente de los bits 13-11 de la instrucción (registro destino de las instrucciones aritmético-lógicas y Load). El registro a leer se selecciona entre los campos que codifican el registro fuente en las instrucciones aritmético-lógicas (bits 10-8 y 7-5), el registro índice en las instrucciones de acceso a memoria (bits 10-8) o el registro fuente en las instrucciones Store (bits 13-11). La selección se realiza mediante el multiplexor SELREG, cuyos bits de control (CRf) son gestionados por la UC.
- **Bloque de cálculo aritmético-lógico**: Este bloque incluye la Unidad Aritmético-Lógica (UAL), en la que se realizan las operaciones requeridas por las instrucciones y se calcula el valor de los indicadores de condición N y Z. La UAL también puede dejar pasar el operando B para activar los indicadores de condición durante la ejecución de las instrucciones Load. La señal OPERAR de la UAL es gestionada por la UC, e indica si debe realizarse una operación (OPERAR=1) o bien dejar pasar el operando B (OPERAR=0). La UAL ha sido diseñada de tal modo que los bits 1-0 que codifican el campo OP en las instrucciones aritmético-lógicas coincidan con la codificación de la operación a realizar, por lo que pueden conectarse directamente a los dos bits de control de menor peso de la UAL.

  El operando A de la UAL siempre proviene del banco de registros. El multiplexor SELDAT permite seleccionar el operando B entre el banco de registros y el operando inmediato para las instrucciones aritméticas, o de la memoria para las instrucciones Load. Esta selección puede hacerse sin intervención de la UC conectando los bits 14 (CO_0) y 2 (OP_2) de la instrucción a los bits de control del multiplexor SELDAT, tal y como se muestra en la Figura 2.

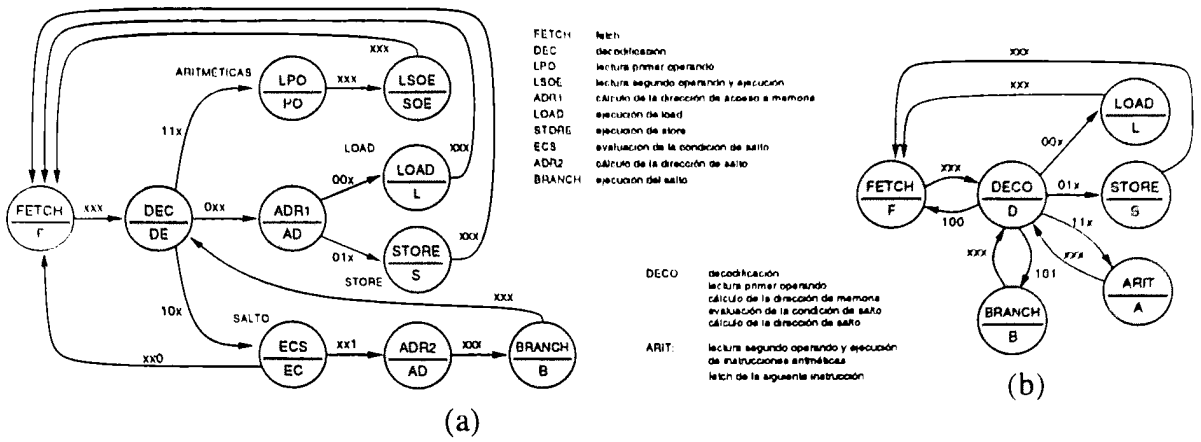

Figura 3: Máquina de estados de la Unidad de Control de la MR.

Salidas UC	F	DE	AD	L	S	PO	SOE	EC	B
Ld_RA	0	0	0	0	0	1	0	0	0
Ld_IR	1	0	0	0	0	0	0	0	1
Ld_PC	1	0	0	0	0	0	0	0	1
Ld_R@	0	0	1	0	0	0	0	0	0
Ld_RZ	0	0	0	1	0	0	1	0	0
Ld_RN	0	0	0	1	0	0	1	0	0
ERd	0	0	0	1	0	0	1	0	0
$\overline{L}$/E	0	0	0	0	1	0	0	0	0
$\overline{PC}$/@	0	x	x	1	1	x	x	x	1
CRf	x	x	1	x	0	1	2	x	x
OPERAR	x	x	x	0	x	x	1	x	x

(a)

Salidas UC	F	D	A	L	S	B
Ld_RA	0	1	0	0	0	0
Ld_IR	1	0	1	0	0	1
Ld_PC	1	0	1	0	0	1
Ld_R@	0	1	0	0	0	0
Ld_RZ	0	0	1	1	0	0
Ld_RN	0	0	1	1	0	0
ERd	0	0	1	1	0	0
$\overline{L}$/E	0	0	0	0	1	0
$\overline{PC}$/@	0	x	0	1	1	1
CRf	x	1	2	x	0	x
OPERAR	x	x	1	0	x	x

(b)

Figura 4: Tabla de salidas de la Unidad de Control de la MR.

- **Bloque de cálculo de direcciones**: En este bloque se realiza el cálculo de todas las direcciones de memoria a las que se accede durante la ejecución de un programa. Las direcciones se almacenan en el PC o en el R@, y la UC selecciona el registro adecuado mediante el multiplexor SELDIR. Los accesos a memoria pueden realizarse por tres causas distintas:
  - *Secuenciamiento implícito*: La dirección de la siguiente instrucción a ejecutar se lee del PC. Incluye el autoincremento del PC.
  - *Ejecución de una instrucción de acceso a memoria*: La dirección de acceso está en el registro R@, y se calcula sumando la dirección base (bits 7-0 de la instrucción) con los 8 bits de menor peso del registro índice. Cuando se realiza el cálculo de la dirección, la UC selecciona la entrada 1 del multiplexor SELREG para leer el registro índice.
  - *Ejecución de una instrucción de salto*: La dirección de acceso está en el registro R@. Para almacenar en R@ los bits 7-0 de la instrucción (dirección de salto) se suman éstos con 0. Para ello, la UC selecciona en el multiplexor SELREG la entrada 1. Esta entrada selecciona los bits 10-8 de la instrucción, que en las instrucciones de salto valen "000" , y por tanto se selecciona el registro R0 del banco de registros, poniendo un 0 en la entrada del sumador.

4 Unidad de Control

El funcionamiento de la UP esta gobernado por la UC. La UC es un sistema secuencial que determina el orden en que deben actuar los distintos elementos en la UP para completar la ejecución de cada una de las instrucciones. La UC gestiona la carga de los distintos registros (Ld_RA, Ld_IR, Ld_PC, Ld_R@, Ld_RZ y Ld_RN) y del banco de registros (ERd), la memoria ($\overline{L}$/E), la UAL (OPERAR) y los distintos multiplexores ($\overline{PC}$/@ y CRf). El control de estos bloques se realiza según el código de operación de la instrucción en ejecución (CO) y la evaluación de la condición de salto (Cond).

Instrucción	Operación
ADDI Rf, #inmediato, Rd	Rd := Rf + inmediato
SUBI Rf, #inmediato, Rd	Rd := Rf − inmediato
ADD Rf1, Rf2, Rd	Rd := Rf1 + Rf2
SUB Rf1, Rf2, Rd	Rd := Rf1 − Rf2
ASR Rf, Rd	Rd := Rf >> 1
AND Rf1, Rf2, Rd	Rd := Rf1 ∧ Rf2
LOAD dir_base(Ri), Rd	Rd := M[dir_base + Ri]
STORE Rf, dir_base(Ri)	M[dir_base + Ri] := Rf

Instrucción	Condición de Salto	
BR dir_destino	incondicional	1
BEQ dir_destino	igual	Z
BL dir_destino	menor	N
BLE dir_destino	menor o igual	N ∨ Z
BNE dir_destino	distinto	$\overline{Z}$
BGE dir_destino	mayor o igual	$\overline{N}$
BG dir_destino	mayor	$\overline{N \vee Z}$

Figura 5: Formato de las instrucciones en el Lenguaje Ensamblador de la MR.

La UC se especifica como una máquina de estados finita usando el modelo de *Moore*. En este modelo, las salidas están asociadas a los estados y las transiciones entre estados dependen de las entradas del sistema. Las operaciones a realizar en cada estado se especifican en una tabla de salidas. En las Figuras 3 y 4 se muestran dos ejemplos de máquina de estados finitos y sus respectivas tablas de salida.

La utilización de este modelo permite seguir un proceso didáctico de diseño. El estudiante parte de un diseño inicial, donde cada una de las operaciones requeridas para la ejecución de una instrucción se realiza en un estado diferente (ver las Figuras 3(a) y 4(a)). A continuación, se introducen distintas técnicas para aumentar el nivel de utilización por ciclo de los elementos de la UP, desarrollando un conjunto de UCs sucesivamente mas optimizadas, que concluye en la especificación seleccionada como definitiva (ver las Figuras 3(b) y 4(b)). Las distintas UCs generadas son ejemplos claros para ser minimizados y sintetizados como parte de un curso de lógica digital, consiguiendo una mayor integración entre los conceptos de arquitectura de computadores y los circuitos digitales.

5 Lenguaje Ensamblador

El lenguaje ensamblador (LE) se introduce para simplificar la programación en lenguaje máquina. El LE de la MR está constituido por instrucciones, etiquetas, directivas, expresiones y macros.

5.1 Instrucciones

El LE codifica, mediante la utilización de un código mnemotécnico, cada una de las distintas instrucciones del lenguaje máquina y la forma de acceder a los operandos. La Figura 5 muestra una lista de los códigos mnemotécnicos y de la especificación de los operandos.

5.2 Etiquetas

Las etiquetas son un mecanismo para establecer referencias a instrucciones y datos sin necesidad de conocer su dirección exacta en memoria. Una etiqueta se define al principio de una línea mediante un identificador seguido del símbolo ":". El proceso de ensamblaje encargado de la traducción a lenguaje máquina calcula la dirección de memoria exacta correspondiente a cada etiqueta, y la almacena en una *tabla de símbolos* que es utilizada para la traducción de cada instrucción.

5.3 Directivas

Las directivas son indicaciones que se introducen en un programa para controlar el proceso de ensamblaje. Existen dos tipos básicos de directivas en el LE de la MR:

- **Directivas de definición de variables y constantes**: se utilizan para reservar posiciones de memoria o inicializarlas con valores concretos. También permiten la asignación de valores a símbolos.

$.dw\ n_1\ \{, n_2, \ldots, n_N\}$	inicializa posiciones de memoria consecutivas con los valores indicados.
$.rw\ n$	reserva n palabras consecutivas de memoria sin valor inicial.
$identificador = n$	establece una equivalencia entre un identificador y un valor numérico.

- **Directivas de control de ejecución**: Permiten conocer la dirección de la primera y la última instrucción a ejecutar.

  .begin identificador — indica la dirección de la primera instrucción a ejecutar en un programa.
  .end — indica la finalización de un programa.

5.4 Expresiones Aritméticas

El lenguaje ensamblador permite el uso de expresiones aritméticas en lugar de valores numéricos. Las expresiones aritméticas son evaluadas durante el proceso de ensamblaje. Una expresión es: un valor numérico, una etiqueta, o bien la suma (+), la resta (−), el producto (×) o la división (/) de dos expresiones. Para realizar la evaluación de las expresiones se utiliza el orden de precedencia clásico de los operadores. No se permiten paréntesis.

5.5 Macros

Una *macro* o "pseudoinstrucción" es un módulo de instrucciones parametrizable que puede ser referenciado desde cualquier punto del programa. Para definir una macro se dispone de dos directivas especiales, entre las cuales debe escribirse el código (cuerpo) de la macro:

.def nombre {argumento{, argumento}}
{ cuerpo de la macro }
.enddef

La directiva (*.def*) señala el principio de la definición de la macro y le da nombre. La directiva (*.enddef*) indica el final de la macro. Los argumentos pueden ser de tres tipos, y van precedidos de los símbolos $\$d$, $\$i$ o \$, según el espacio de direcciones en el que está almacenado el dato al que hacen referencia:

- $\$dn$: el operando n debe ser interpretado como una dirección de memoria,
- $\$in$: el operando n debe ser interpretado como una constante inmediata, y
- $\$n$: el operando n está en el registro Rn de la UP.

6 Ensamblador y Simulador

Existe un programa ensamblador y un simulador gráfico de la arquitectura de la MR, ambos desarrollados en el DAC y de libre distribución. Pueden obtenerse a través de `ftp` en la dirección `ftp://ftp.ac.upc.es/pub/soft/mr`. Las siguientes secciones hacen referencia al funcionamiento de dicho entorno.

Un programa escrito en lenguaje ensamblador debe ser creado en un fichero con extensión ".asm". El proceso de ensamblaje genera un fichero con extensión ".cod", que contiene su traducción a lenguaje máquina. El proceso de ensamblaje de un programa se realiza en dos etapas. La primera etapa (preensamblaje) se ocupa de realizar la expansión de las macros. La ejecución del programa: `pren prog{.mr} {aux{.mr}}` lee el fichero "prog.mr" y genera el fichero "prog.asm". El fichero "aux.mr" es opcional y debe contener únicamente la definición de macros.

La segunda etapa (postensamblaje) genera el lenguaje máquina ejecutable por el simulador de la MR. Inicialmente se crea la tabla de símbolos, que contiene los identificadores de etiquetas y de constantes con sus valores asociados. Posteriormente se traducen las instrucciones de lenguaje ensamblador a lenguaje máquina, utilizando los valores almacenados en la tabla de símbolos. La ejecución del postensamblador: `posten prog{.asm}` lee el fichero "prog.asm", y genera el código en lenguaje máquina en el fichero "prog.cod". Las dos etapas del proceso de ensamblaje se realizan de forma ininterrumpida mediante el comando: `enmr prog{.mr} {aux{.mr}}`.

Una vez obtenido el fichero "prog.cod", se puede realizar la simulación del programa ejecutando el comando: `mr prog {.cod}`. La Figura 6 muestra las dos pantallas de este simulador: una con el

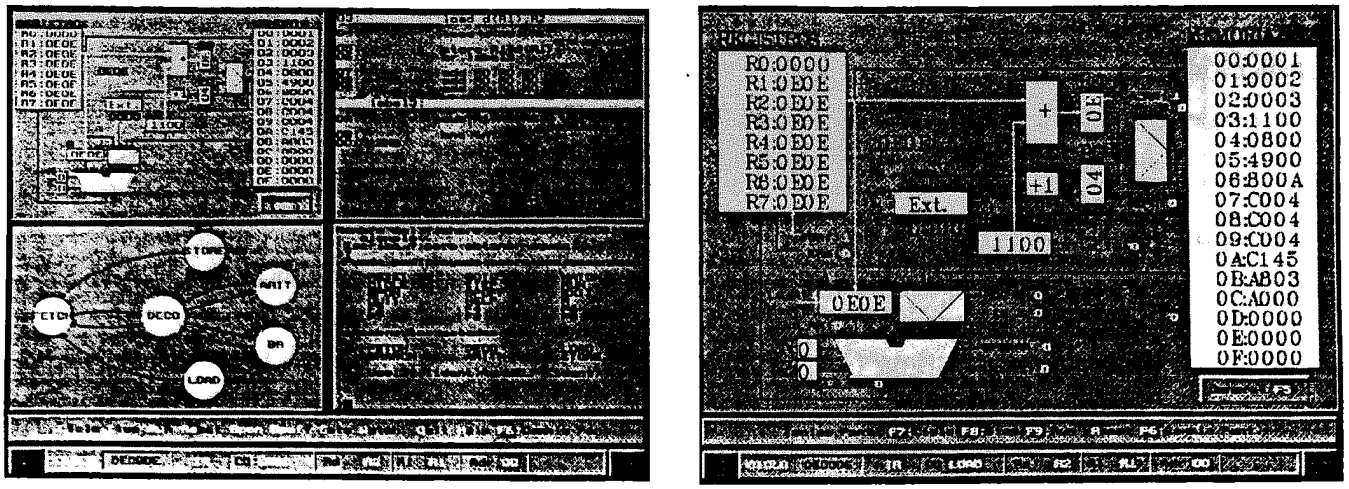

Figura 6: Pantalla principal y zoom de la UP en el simulador de la Máquina Rudimentaria.

conjunto UP–UC–Lenguaje Ensamblador, y la otra mostrando una ampliación detallada de la UP, con los valores de todas las señales generadas por la UC.

7 Conclusiones

En este artículo se presenta la *Máquina Rudimentaria*, un procesador pedagógico. La *Máquina Rudimentaria* es una herramienta útil para explicar los principios básicos sobre arquitectura de computadores a alumnos de un primer curso de Ingeniería. La experiencia adquirida por los autores en los dos últimos años, tiempo durante el cual este procesador se ha explicado en la asignatura " Introducció als Computadors" de primer curso de la Facultat d'Informática de Barcelona, así lo avala.

La *Máquina Rudimentaria* sirve de conexión natural al alumno para pasar del estudio de los circuitos digitales al estudio del lenguaje máquina. La disponibilidad de un simulador facilita al alumno la comprensión de su funcionamiento, permitiéndole experimentar y resolver por sí mismo las posibles dudas que pudiese tener a nivel de lenguaje ensamblador, lenguaje máquina y sistemas digitales.

Agradecimientos

En el diseño de la MR han intervenido, además de los autores, los profesores Jordi Cortadella, Anna M. del Corral, Eduard Elias, Agustín Fernández, Josep L. Larriba y Montse Peiron. En la especificación del lenguaje máquina intervinieron además Rosa M. Badia, Antonio González, Enrique Herrada, Miguel Valero y Mateo Valero. Quisiéramos agradecer a Asunción Pesado sus comentarios y sugerencias, y a los alumnos Daniel Jiménez, Joan Manuel Pérez, Mónica Pérez y Alex Ramírez las muchas horas invertidas en el diseño del simulador. Este trabajo no hubiera sido posible sin la colaboración de todos ellos y sin el soporte del resto de los miembros del DAC.

Referencias

[1] F. Tirado et al. *Fundamentos de Computadores*. Ed. Síntesis, 1997.

[2] E. Ayguadé et al. *La Máquina Sencilla: Introducción a la Estructura Básica de un Computador*. Apuntes CPDA, 1988. Disponible en catalán en Edicions UPC, Col.lecció Aula, Num. 26, 1992.

[3] E. F. Sowell. *Programming in Assembly Language VAX-11*. Addison-Wesley, 1987.

[4] C. L. Morgan and M. Waite. *Introducción al Microprocesador 8086/8088*. McGraw-Hill, 1984.

[5] J. L. Hennessy and D. A. Patterson. *Arquitectura de Computadores: un Enfoque Cuantitativo*. McGraw-Hill, 1993.