Diseño de Conjuntos de Instrucciones y su Emulación Usando MAE

Joe Miró Julià
Dept. de Matemàticas e Informàtica
Univ. de les Illes Balears
07071 Palma de Mallorca
e-mail: joe@ipc4.uib.es

1. Introducción

En las estudios de Ingeniería Técnica de Informática de Gestión e Ingeniería Técnica de Informática de Sistemas se han realizado prácticas de microprogramación desde que empezaron los nuevos planes de estudios [2]. Varios profesores del departamento de Matemáticas e Informática desarrollamos un emulador de la arquitectura *Mic-1* de Tanenbaum [4] para los laboratorios de la asignatura de Fundamentos de Computadores con buenos resultados. Al aparecer las asignaturas de Estructuras de Computadores y Laboratorio de Sistemas decidí crear un entorno de emulación de una arquitectura microprogramada para poder realizar prácticas en estas dos asignaturas. El resultado de esta decisión es MAE (*Microprogrammable Architecture for Education*).

MAE está basado en el Mic-1. Es una ampliación del mismo y solventa la mayoría de los problemas que éste presenta:

- La ALU puede realizar un conjunto más elevado de operaciones aritméticas y lógicas.
- Los valores constantes se guardan en un conjunto de máscaras no modificables (*Mask Pad*), dejando los registros (*Register Pad*) para guardar los valores que son variables.
- Se tienen cuatro *registros de indireccionmiento* que facilitan enormemente el decodificado de las instrucciones y el uso de registros de propósito general. También permiten una forma cruda de uso de micro subrutinas.
- Los bits de condición creados por la ALU pueden almacenarse directamente en un registro de códigos de condición.

MAE ha sufrido un proceso de revisión desde que se empezó a utilizar hace tres cursos. La versión actual es la 3.0. Está escrita en C y el código fuente está disponible vía WWW (http://dmi.uib.es/people/joe) o pidiéndolo por correo al autor. La versión 4.0 está en proyecto. En la Sección 5 se describe las carácterísticas que tendrá.

El objetivo de la práctica basada en MAE es el que los alumnos diseñen una arquitectura a nivel de conjunto de instrucciones. Se espera que los alumnos aprendan lo que *es* una arquitectura, pero también que aprendan lo que es el proceso de diseño. Que aprendan a especificar su diseño antes que empiecen a implementarlo, que aprendan a ver el diseño en su globalidad y sepan cuales son los *tradeoffs* que existen y sepan encontrar una solución que equilibre en lo posible los pros y los contras de sus decisiones.

2. Micro-arquitectura de MAE

MAE es una microarquitectura diseñada para facilitar la creación de arquitecturas a nivel de conjunto de instrucciones. Se puede ver su diseño en la Figura 1.

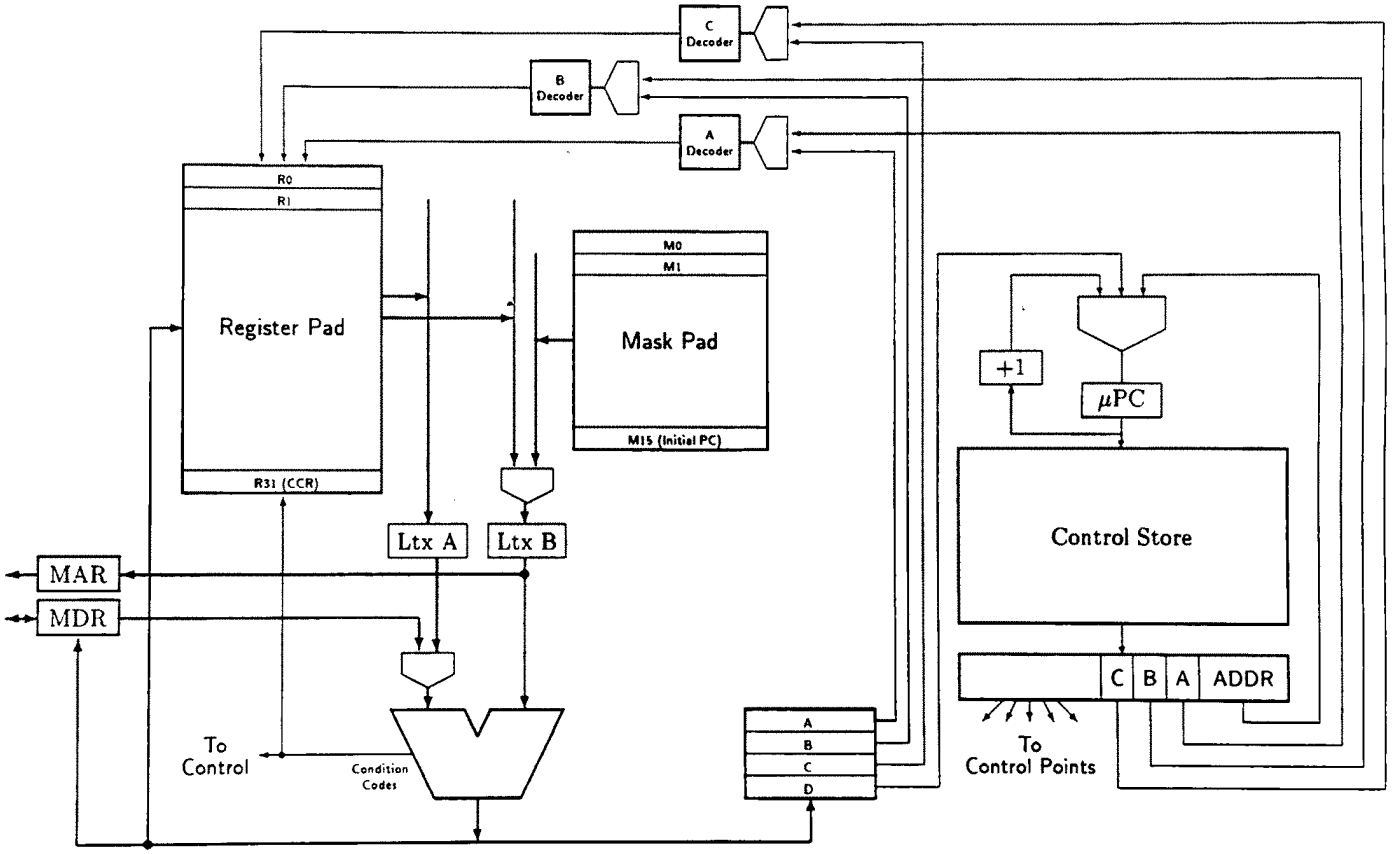
2.1 El *Data Path*

El *data path* consta de un conjunto de 32 registros de 32 bits (el *register pad*). En cada microinstrucción dos de estos registros son volcados a los buses A y B y un registro puede almacenar

lo que se encuentre en el bus C. De estos registros uno es especial: R31 puede servir como registro de códigos de condición (CCR). No existe ni Registro de Instrucciones ni Contador de Programa.

MAE también tiene un conjunto de 16 máscaras de 32 bits (el *mask pad*). Estos registros toman valor durante la fase de inicialización de la emulación y no puedes ser modificados. Aunque no es imprescindible hacerlo así se recomienda usar la máscara M15 como valor inicial del Contador de Programa

Además de estos registros, se disponen de cuatro registros especiales. Estos registros (A, B, C y D) permiten indicar indirectamente cuál es el registro que debe volcarse sobre los buses A y B (mediante los registros A y B, respectivamente), en qué registro debe almacenarse el resultado de la ALU (mediante el registro C), o a qué microinstrucción debe saltarse (mediante el registro D). El uso de estos registros facilita enormemente la decodificación de las instrucciones. El registro D puede usarse también para indicar la dirección de retorno de un salto a subrutina. Aunque los registros son de 32 bits, sólo se usan para la indirección los bits menos significativos (5 para indirección de registro y 12 para indirección de salto).



MAE Architecture

Figura 1.- Arquitectura del MAE

La ALU es también de 32 bits. Puede realizar 13 operaciones diferentes. Las dos aritméticas (*add* y *subtract*) se hacen sobre números enteros codificados en C2. Las cuatro lógicas (*and, or, not* y *xor*) se hacen bit a bit. Los cinco desplazamientos (aritméticas, lógicas y rotaciones, tanto a derecha como a izquierda) son de un bit. Las dos operaciones que faltan son el paso directo a través de la ALU de las entradas A y B. Todas las operaciones unarias (excepto el paso-B) se realizan sobre el dato de la entrada A. La ALU presenta cinco valores de condición: acarreo (**C**), extensión de operación (**X**), cero (**Z**), negativo (**N**) y desbordamiento (**V**). Estas condiciones pueden almacenarse individualmente en el CCR poniendo a 1 los bits adecuados en la microinstrucción. Z y N toman valor en todas las

operaciones, X y V sólo en las aritméticas. C toma valor en las operaciones aritméticas y de desplazamiento. En este caso representa el bit que se pierde tras el desplazamiento.

2.2 La memoria

MAE tiene dos registros para comunicarse con memoria: el *Memory Address Register* (MAR) y el *Memory Data Register* (MDR). MAR se carga desde el bus B, MDR puede cargarse desde memoria (en lecturas) o desde el bus C. El contenido de MDR sólo puede volcarse a la salida A de la ALU.

La memoria es direccionable a nivel de *bytes* pero debe accederse por palabras de 32 bits. La dirección debe ser un múltiplo de 4. Si no lo es los dos últimos bits son forzados a cero.

Existen tres lineas de control: dos para indicar el sentido de la transferencia de datos –*memory read* (mrd), y *memory write* (mwr)– y uno para indicar cuando se ha completado el acceso –*wait*–. Esta señal está manejada por el controlador de memoria. Se pone a uno si la operación de memoria no se completará en un ciclo, y permanece a uno hasta que finaliza la operación.

Para que una operación de memoria se lleve a cabo correctamente los valores de MAR, MDR, mrd y mwr no deben variar durante toda la operación. Si lo hacen, el resultado es impredecible. Otro caso de resultado impredecible es si se carga MDR simultáneamente desde memoria y desde el bus C.

2.3 Control

MAE tiene una memoria de control de 4096 palabras de 59 bits. Cada microinstrucción se divide en 21 campos. El formato de la microinstrucción puede verse en la Figura 2. Describamos los campos:

- HALT: 1: Detener la emulación
- AMUX: Multiplexor a la entrada A de la ALU. Valores: 0: Desde registros; 1: desde MDR
- BMUX: Multiplexor a la entrada B de la ALU. Valores: 0: Desde registros; 1: desde máscaras
- COND: Hay 12 valores posibles; 5 saltos sobre los valores de condición de la ALU; 5 sobre valores de condición del CCR; salto incondicional; no hay salto.
- ALU: Hay 13 valores posibles, cada uno indicando una operación de la ALU.
- MDR: 1: Carga MDR
- MAR: 1: Carga MAR
- MRD, MWR: Señales de control que deben mandarse al bus.
- FLAGS: Cada bit (C X N V Z) indica la carga del bit de condición de la ALU correspondiente en el CCR
- SPR: Indica la carga en uno (o ninguno) de los cuatro registros especiales.
- MASK: Volcado del registro de máscara correspondiente al bus B
- ENC: Habilitador de carga en registro. Valores: 0: No se carga ningún registro con el resultado de la ALU; 1: se carga un registro.
- INDA, INDB, INDC: Estos tres campos indican si el registro que debe volcarse/almacenarse de los buses A, B, y C son los que indican los campos A, B y C de la microinstrucción (valor 0) o son los que indican los registros especiales A, B y C respectivamente.
- A, B, C: indican los registros que deben volcarse a los buses A y B, y el registro que debe almacenar el contenido del bus C.
- INDD: En caso de producirse un salto la dirección a donde debe saltarse se toma del campo ADDR si el valor de INDD es 0, o se toma del registro especial D si el valor de INDD es 1.
- ADDR: Dirección de salto.

Para poder hacer prácticas de construcción de conjuntos de instrucciones con esta microarquitectura, se ha desarrolado un entorno de trabajo consistente en un compilador de microprogramas y un emulador del MAE, que paso a describir en la siguiente sección.

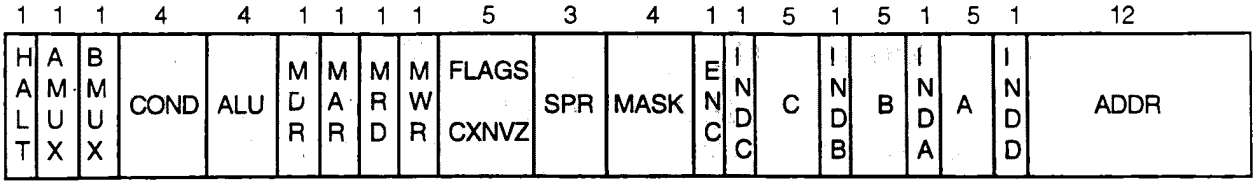


Fig. 2: Formato de la microinstrucción

3. Entorno de Trabajo de MAE

El entorno de trabajo de MAE consta de dos programas: un compilador de microprogramas y un emulador de la arquitectura. La descripción detallada de estos programas se encuentra en [3].

3.1 El compilador

Para facilitar la creación de conjuntos de instrucciones y la emulación de los mismos se ha intentado crear un lenguaje de microprogramación lo más fácil de usar posible. El resultado ha sido MAEL (*MAE Language*). Al igual que la arquitectura, este lenguaje es una evolución del lenguaje MAL de Tanenbaum [4].

Cada linea del programa que no esté en blanco, o que no sea un comentario representa una microinstrucción. Cada microinstrucción consta de una o más microoperaciones separadas por ';'. Se ha intentado que la sintaxis de las microoperaciones se parezca lo más posible a la de los lenguajes de alto nivel. Un ejemplo del MAEL puede verse en la Figura 3.

```
if ccn goto 61              # JNEG Cont'd
goto 81
r0 := r1 & m3; goto 81

if ccz goto 81              # JNZE Cont'd
r0 := r1 & m3; goto 81

mar := r4; mdr := r0;mwr;if wait goto 64 # CALL Cont'd
r0 := r1 & m3; goto 81

      # Decodification for 8-bit Opcodes
r2 := [ r2
r2 := [ r2
r2 := [ r2
d := r2 & m7; goto @d # Jump to execution
```

Figura 3: Ejemplo de MAEL

El compilador no sólo descubre errores sintácticos sino que también detecta conjuntos de microoperaciones que no pueden dar lugar a una microinstrucción. Por ejemplo indicará un error ante la microinstrucción "`mar := r5; r3 := r2 + r8`" ya que esto implica que dos registros diferentes deben volcar sus valores sobre el mismo bus (el B).

El resultado de la compilación es un fichero con el contenido del *control store* en hexadecimal. Este fichero es el que usará el emulador de la arquitectura.

3.2 El emulador

El emulador del MAE parte del fichero del contenido del *control store* resultado de la compilación, y de un fichero de inicialización en donde se indica el contenido inicial de la memoria, el valor inicial de los registros de máscara, y el valor inicial del micro contador de programa. Un ejemplo de fichero de inicialización puede verse en la Figura 4.

```
$M # Mask values
00000000 # m0 = 0
00000001 # m1 = 1
FFFFFFFF # m2 = -1

$S 00000050 # First mpc = 80

$A 00000100 # Memory address

0F0A0200 # Contens of position 100 Hex
10070205 # Contens of position 104 Hex
30000204 # Contens of position 108 Hex
10000208 # Contens of position 10C Hex

$A 0000020C # Memory address

00050000 # Contens of position 20C Hex
0FF0012C # Contens of position 210 Hex
```

Figura 4: Ejemplo de fichero de inicialización

Una vez en marcha el emulador permite dos modos de emulación: modo microinstrucción (M) y modo silencioso (Q). En el primero, el emulador se detiene tras cada microinstrucción y muestra el contenido de todos los elementos de la máquina –registros, *latches*, códigos de condición, etc.– al final de la ejecución de la microinstrucción. En el modo Q el emulador va ejecutando el microprograma sin mostrar nada en pantalla y se detiene al final de la ejecución. Todos los resultados aparecen en un fichero llamado *emul.log*. Para prevenir contra la posibilidad de bucles infinitos en el programa o microprograma, tras la ejecución de varios ciento de instrucciones se detiene y pregunta si se desea continuar.

Cada vez que el emulador se detiene es posible cambiar de modo, comprobar el estado de la memoria o finalizar la emulación.

3.3 *User friendliness*

El entorno de trabajo de MAE *no* es lo que se denomina *user friendly*. Esta característica del entorno no es un defecto ni es así por falta de tiempo o conocimiento, sino que se ha diseñado de esta manera a propósito. Lo he hecho así porque recuerdo lo mucho que he aprendido teniendo que trabajar en entornos que no daban un 'excesivo' soporte al usuario. Por ejemplo, programando con tarjetas perforadas, al tener que depurar el programa pudiendo ejecutarlo muy pocas veces, aprendí a diseñar entornos de depuración y conjuntos de *test* de una manera muy eficiente, cosa que hecho de menos entre los alumnos de hoy en día, con sus entornos gráficos y depuradores de todo nivel.

El objetivo de hacer un entorno poco amigable no es, evidentemente, el de dar más trabajo a los alumnos, sino el de obligarles a aprender con más profundidad y el de obligarles a diseñar con más cuidado. Se ha querido evitar casos como el de un entorno de microprogramación para el Mic-1 que no permitía al microprogramador cometer errores: Una vez escogido un dato que tenía que ir por el bus A, el entorno impedía escoger microoperaciones que permitiera que un dato diferente fuera al bus A. Esto no lo considero una ventaja, sino un grave inconveniente: el entrono *oculta* información al alumno, y el alumno no piensa, y por lo tanto no aprende, qué caminos cogen los datos, ni como funciona internamente la máquina. Esto es lo que se ha intentado impedir.

El hacer que el entorno sea poco permisivo con lo errores obliga a los alumnos a planificar mejor su trabajo. Los alumnos no pueden ponerse a codificar directamente, como están (por desgracia) acostumbrados a hacer. Quizá la mejor descripción de por qué el entorno es poco amigable la dio un alumno al quejarse: "Es que si no tienes las cosas muy bien pensadas no hay manera de trabajar con el programa". Exactamente.

4. La Práctica

El entorno MAE se utiliza en la asignatura Laboratorio de Sistemas en el tercer curso de la Ingeniería Técnica de Informática de Sistemas. Una de las prácticas que se ofrece a los alumnos de esta asignatura es la creación de un conjunto de instrucciones y su implementación en este entorno. Los alumnos que eligen esta práctica se supone han cursado o están cursando la asignatura Estructura de Computadores, en la cual se explican todos los aspectos que inciden sobre el diseño de una arquitectura.

El objetivo de la práctica es que los alumnos aprendan a ver de forma global todos los aspectos involucrados en el diseño de una arquitectura y cuales son los *tradeoffs* existentes en el diseño. Al acabar la práctica los alumnos deben haber diseñado una arquitectura a nivel de conjunto de instrucciones que sea consistente y que sea adecuado para el tipo de actividad que ellos pretendan.

Para ayudar en la consecución de estos objetivos se obliga a los alumnos a cumplir una serie de pasos en el desarrollo de la práctica, que paso a describir a continuación:

1. Decidir el tipo de procesador que se desea crear. El hardware de MAE impone unos límites superiores, pero no es necesario llegar a ellos. Los alumnos deciden si quieren crear una máquina de 32 bits con grandes capacidades de cálculo, un pequeño controlador de 8 bits, un coprocesador, etc. y qué características generales debe tener la arquitectura.
2. Diseñar el conjunto de instrucciones del procesador. El diseño presentado debe estar elaborado hasta el último detalle: deben detallar todas las instrucciones, qué hace cada una, todos los modos de direccionamiento, los formatos de instrucciones, codificación de cada una de ellas, etc.
3. Implementar en el entorno MAE (en lo posible) el conjunto de instrucciones diseñado. La implementación ha de ser lo suficientemente completa para poder asegurar que es factible. Por ejemplo, se debe demostrar que el número de registros de la arquitectura, más el número de registros temporales que se necesiten para la implementación no sobrepase las capacidades de MAE.

Tras haber completado cada paso los alumnos deben discutir la práctica con el profesor y haber obtenido el visto bueno antes de poder proseguir. Esto se hace por dos motivos: para que los alumnos deban seguir los pasos correctos de un diseño (especificación a alto nivel- especificación a bajo nivel-implementación) y no pongan el arado delante de los bueyes, y para obligarles a planificar el trabajo y que en su desmesurado optimismo no intenten completar la práctica en la última semana del curso.

En la primera revisión, tras la decisión del tipo de procesador que desean construir, se hace hincapié en que los alumnos sean conscientes de las consecuencias que van a tener las decisiones de diseño que hagan. Por ejemplo, si deciden crear un pequeño controlador de 8 bits, el espacio de memoria va a ser muy reducido y por lo tanto las instrucciones y modos de direccionamiento que escojan deben ser tales que minimicen el tamaño del código; si deciden por el contrario crear una arquitectura tipo RISC, deben ser conscientes que la implementación de esta arquitectura va a ser segmentada –aunque ellos la emulen mediante microprogramación– y que las instrucciones y modos de direccionamiento a crear deben ser consistentes con esta decisión. El patrón de discusión de esta primera reunión está adaptado de la primera mitad del capítulo 2 del libro de Hennessy y Patterson [1].

En la revisión que tiene lugar tras el segundo paso se dirige la atención principalmente hacia las instrucciones y modos de direccionamiento que se han escogido y cómo encajan en la idea general de la arquitectura. También se empieza a estudiar qué implicaciones van a tener las decisiones, sobre todo de formatos de instrucciones, a la hora de implementar en MAE la arquitectura diseñada.

Tras haber cumplimentado el tercer paso, con la entrega de la práctica, ya no se discute sobre la arquitectura –lo hecho, hecho está– sino que se les pregunta a los alumnos las dificultades principales que han tenido y cómo variarían MAE si la quisieran adaptar a su diseño.

Los resultados obtenidos en esta práctica pueden considerarse bastante positivos. Muchos alumnos lo escogen y quedan satisfechos con los resultados y aprendizaje obtenidos. A través de las frecuentes interacciones que se tiene con ellos se nota que van dándose cuenta de los *tradeoffs* sobre los que tienen que ir decidiendo durante el diseño y que se están enfrentando, quizá por primera vez, con la *globalidad* de un diseño, viendo como varias 'buenas' ideas consideradas aisladamente, no necesariamente dan lugar a un buen resultado.

Otra resultado positivo es la diversidad de los diseños presentados por los alumnos. Se han diseñado e implementado procesadores de propósito general de 16 y 32 bits con varias filosofías, pequeños controladores de 8 bits de propósito específico, y hasta un coprocesador gráfico.

5. El Futuro

En vista de los comentarios recibidos por los alumnos hay una serie de mejoras a realizar para la siguiente versión del entorno.

- Introducir el uso de microsubrutinas. Esto implicaría añadir una pequeña pila a la parte de control y dos tipo de salto nuevos, lo cual se puede hacer sin cambiar el formato de la microinstrucción. Por la experiencia de los alumnos parece ser que bastará una pila de 4 posiciones.
- Modificar la ALU de manera que los desplazamientos pudieran ser de un número arbitrario de bits. Esto se conseguiría haciendo que los desplazamientos sean operaciones binarias, indicando por la entrada B de la ALU el número de bits a desplazar. Este cambio es muy simple y tampoco implica hacer cambio alguno en el formato de la microinstrucción.
- Hacer variable el número de bits de la arquitectura. En el fichero de inicialización de la emulación se indicaría si se desea que los registros, ALU y buses sean de 8, 16 ó 32 bits. La necesidad de esta modificación es más sicológica que técnica: no resulta nada difícil montar una arquitectura de 8 ó 16 bits en MAE, pero esto implica que los alumnos deben luchar permanentemente con la tendencia a usar 32 bits "porque están ahí".
- Introducir direcciones simbólicas y etiquetas en MAEL. Esta es la dificultad más aparente que tienen los alumnos al escribir microprogramas, pero al introducir direcciones simbólicas no se elimina del todo, ya que en los saltos indirectos se tendría que introducir el valor numérico en el registro especial D. Estoy dudando sobre esta modificación ya que no estoy seguro si elimina un problema, o sólo lo oculta.

6. Conclusión

Se aprende mucho más haciendo, que viendo lo que otros han hecho antes. En la práctica que se ha presentado, los alumnos han de crear un conjunto de instrucciones para aprender tanto lo que significa hacer un diseño, como lo que es una arquitectura. Pero para poder hacer una práctica en condiciones es necesario disponer de una buena herramienta. MAE es un entorno que supone una evolución del Mic-1 y que facilita la creación y emulación de una arquitectura. Se ha estado usando durante tres años en los estudio de Ingeniería Técnica de Informática de Sistemas con buenos resultados.

Referencias

[1] HENNESY, J., PATTERSON, D.: *Computer Architecture, A Quantitative Approach. 2nd. Ed.* Morgan Kaufmann, San Francisco, 1996

[2] MIRO JULIA, J., PROENZA ARENAS, J., ORTIZ RODRIGUEZ, A.: *Una Estructura de Prácticas de Laboratorio para la Asignatura Fundamentos de Computadores.* Actas del I Congreso Universidad y Macintosh. Vol III, pags 451-457. Madrid, 1994

[3] MIRO JULIA, J.: *MAE: A Microprogrammable Architecture for Education. User Manual and Documentation.* 1996.

[4] TANENBAUM, A: *Structured Computer Organization. 3rd Ed.* Prentice Hall, England Cliffs, 1990