Manual de Set de Instrucioines MRA

CE4301: Arquitectura de Computadores

Versión de Documento 1

Editores: Fabián Montero Villalobos[1], José Julián Camacho Hernández[1],
José Alejandrro Soto Chacón[1]
[1]Tecnológico de Costa Rica
fmonterov@estudiantec.cr, jcamacho341@estudiantec.cr, soto@estudiantec.cr

3 de enero de 2024

Índice general

Capítulo 1

Introducción

Este documento describe la arquitectura MRA, la cual cubre todos los aspectos de sus sistemas.

Cualquier comentario acerca de decisiones de diseño tomadas por el equipo será formateado igual a este párrafo.

1.1. Terminología

Esta sección describe terminología usada para describir componentes del sistema.

- *pc: program counter*. Guarda la dirección de memoria de la instrucción que se está ejecutando.
- *lr: link register*. Guarda la dirección de memoria de la siguiente instrucción tras realizar el llamado a una función.
- *sp: stack pointer*. Guarda la dirección de memoria del tope de la pila.
- *src:* registro del cual se obtienen sus valores para ser operados.
- *dest:* registro en el cual se guarda el resultado de una operación.
- *immN:* valor inmediato, escalar, cuya representación binaria es de N bits.
- *label:* etiqueta que se coloca en el programa para designar un bloque.
- *target:* etiqueta a la cual se desea saltar.
- *offset:* cantidad de instrucciones que se desean saltar a partir del valor del *pc*.
- *Comportamiento indefinido:* corresponde a un estado donde no se tiene certeza de lo que pueda suceder.

1.2. Descripción de Arquitectura

El MRA es arquitectura de tipo RISC. Es *load/store*, ya que las operaciones de procesamiento de datos solo operan en el contenido del registro, no directamente en el contenido de la memoria.

Cuenta con campos de instrucciones uniformes y de longitud fija de 16 bits, para simplificar la decodificación de instrucciones.

El tamaño de una palabra es de 32 bits. Una media palabra se define de 16 bits.

1.2.1. Registros

MRA cuenta con 16 registros de propósito general, cada uno con 32 bits de ancho. Los registros de propósito general r1–r15 contienen valores que las instrucciones interpretan como una colección de valores booleanos o como enteros binarios sin signo o con signo en complemento a dos.

- El registro r0 está cableado directamente a cero.
- Los registros r1-r6 funcionan como registros temporales.
- Los registros r8-r11 corresponden a *scratch registers*, que son utilizados generalmente en la implementación de subrutinas.
- Como parte de la convención de llamada a subrutinas, los argumentos se pasan en los registros r12 y r13.
- En los registros r10 y r11 se almacenan los valores de retorno de subrutinas.
- El registro r14 funciona como puntero al *stack*. Es decir, almacena la dirección de memoria del valor en el *top* de la pila.
- El registro r15 está reservado como *link register*. Se encarga de guardar las direcciones de retorno tras el llamado a subrutinas. Almacena la siguiente instrucción después del llamado.

Los registros se definen de 32 bits por diversas razones. En primer lugar, registros de menor cantidad de bits limitarían la magnitud de los valores que se pueden operar. Adicionalmente, es el ancho adecuado para almacenar direcciones de memoria completas y así poder operarlas. Por el uso específico del ISA, los 32 bits son necesarios y suficientes para almacenar valores de píxeles en el formato RGBA que utiliza 4 bytes para su representación.

Por otro lado, debido a la convención de llamada definida, la cantidad óptima de registros en el ISA es de 16. Esto también se debe a que para facilitar la decodificación el número de registros debe ser una potencia de 2. Valores que cumplan esta condición y sean inferiores a 16 imposibilitarían implementar de manera correcta dicha convención y tener aún registros restantes para operar. Por el contrario, potencias de 2 mayores a 16 implicarían más bits en las decodificación de las instrucciones.

Los registros tienen los siguientes sinónimos:

31 0
r0 / zero
r1 / t0
r2 / t1
r3 / t2
r4 / t3
r5 / t4
r6 / t5
r7 / t6
r8 / s0
r9 / s1
r10 / s2 / o0
r11 / s3 / o1
r12 / a0
r13 / a1
r14 / sp
r15 / lr

1.2.2. Set de instrucciones

El orden de precedencia de los tipos de instrucciones se visualiza a continuación:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
offset[11:4]								0000				offset[3:0]				BAL
0000				imm4				10000000								EXT
imm4				000010000000												EXT
src				src/dest				10000000								MUL
0000				src				01000000								BIN
0000				imm4				11000000								SYS
dest				imm4				1000000								IMM
reg pair				offset						cond		0000				BCC
offset										11		dest				ADR
src1				src2				opcode			0	dest				ALU
00000				src				L		01		dest				MEM
imm5				0000				S		01		src/dest				INC
imm5				src				S		01		dest				SHI

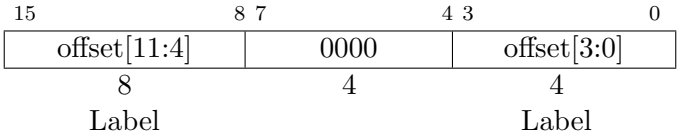
Capítulo 2

Conjunto de instrucciones de MRA

El presente capítulo describe las instrucciones que forman parte del *set* de MRA.

2.1. Instrucciones de control de flujo

2.1.1. Salto incondicional relativo



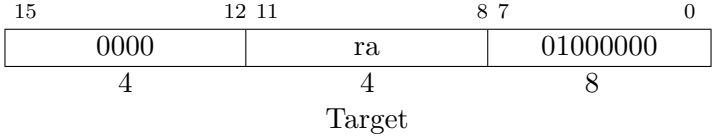
La instrucción BAL realiza un salto incondicional relativo hacia la dirección de *label*. El tamaño máximo del salto es de ±4KiB, o sea 2048 instrucciones.

Posibles permutaciones: 2^{12}

Uso:

```
bal <label>
```

2.1.2. Salto incondicional indirecto



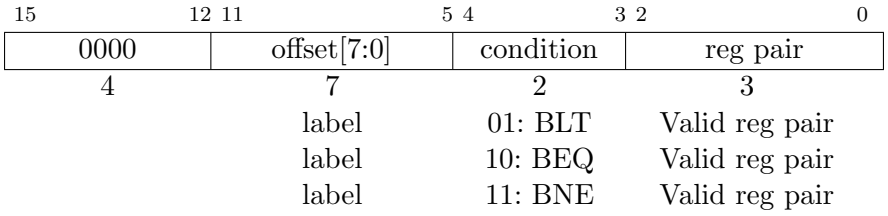
La instrucción BIN realiza un salto incondicional indirecto hacia la dirección que contiene el registro ra.

Posibles permutaciones: 2^4

Uso:

```
bin <ra>
```

2.1.3. Salto condicional



Las instrucciones BEQ, BNE y BLT definen saltos condicionales según el resultado de la comparación entre los dos registros que integran un par de registros válido.

Un par de registros válidos está compuesto por registros que cumplan lo siguiente: rb es un registro par y $rb = ra + 1$. Es decir, rb es par, y ra y rb son consecutivos.

BLT realiza el salto hacia el *label* si ra es menor que rb, considerando valores con signo. BEQ lo realiza si ra es igual a rb, mientras que BNE si ra es distinto de rb.

La condición debe ser diferente de 0, ya que este caso codifica otro tipo de instrucción.

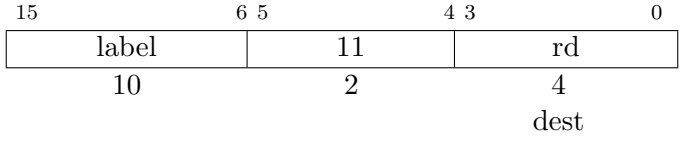
El rango de salto es de ±128 bytes, es decir, 64 instrucciones.

Posibles permutaciones: $2^{10} * 3$

Uso:

```
beq <ra>, <rb>, <label>
bne <ra>, <rb>, <label>
blt <ra>, <rb>, <label>
```

2.1.4. Dirección relativa a PC



La instrucción ADR almacena en rd la dirección de memoria correspondiente al *label*.

Como consideración rd debe ser distinto de r0.

Posibles permutaciones: $2^{10} * 15$

Uso:

```
adr <rd>, <label>
```

2.2. Instrucciones de computación de enteros

La mayoría de las instrucciones computacionales de enteros operan sobre los valores de 32 bits que se encuentran contenidos en los registros de propósito general. Estas instrucciones se codifican como operaciones de registro-inmediato o como operaciones registro-registro. En la mayoría de los casos el destino es el registro rd.

2.2.1. Operaciones de enteros registro-registro

15 12	11 8	7 5	4	3 0
ra	rb	opcode	0	rd
4	4	3	1	4
src1	src2	001: AND		dest
src1	src2	010: ORR		dest
src1	src2	011: XOR		dest
src1	src2	100: SHL		dest
src1	src2	101: SHR		dest
src1	src2	110: ADD		dest
src1	src2	111: SUB		dest

AND, ORR y XOR toman los registros ra y rb, y los operan lógicamente bit a bit mediante and, or y xor respectivamente. El resultado de cada una se guarda en rd.

SHL y SHR realizan desplazamientos a la izquierda y derecha respectivamente en el registro ra las veces indicadas en rb, y almacena el resultado en rd. Corresponden a desplazamientos variables.

ADD y SUB guardan en rd el resultado de la suma o resta, respectivamente, de los registros ra y rb.

En todos los casos rd debe ser distinto de r0.

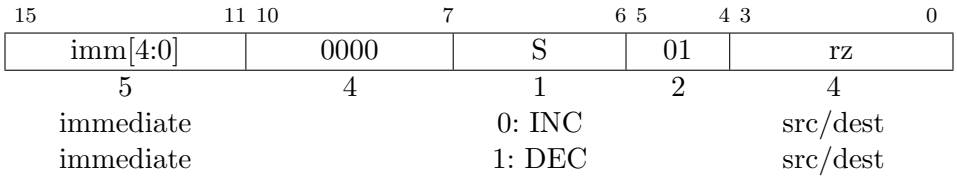
Posibles permutaciones: $2^8 * 7 * 15$

Uso:

```
and <rd>, <ra>, <rb>
orr <rd>, <ra>, <rb>
xor <rd>, <ra>, <rb>
shl <rd>, <ra>, <rb>
shr <rd>, <ra>, <rb>
```

```
add <rd>, <ra>, <rb>
sub <rd>, <ra>, <rb>
```

2.2.2. Operaciones de incremento y decremento de un registro por inmediato



La instrucción INC realiza un incremento de la cantidad especificada por el inmediato *imm* al valor en el registro rz. El resultado se almacena en el mismo rz.

DEC funciona de manera similar a INC, pero realiza un decremento del valor especificado por el inmediato.

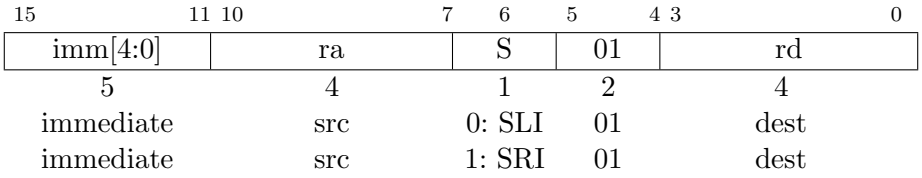
El registro rz debe ser distinto de r0. El inmediato es *zero-extended* y debe ser diferente de cero.

Posibles permutaciones: $(2^5 - 1) * 2 * 15$

Uso:

```
inc <rz>, <imm5>
dec <rz>, <imm5>
```

2.2.3. Operaciones de desplazamientos por inmediato



SLI realiza un desplazamiento lógico hacia la izquierda del registro ra las veces indicadas por el inmediato *imm* y guarda el resultado en el registro de destino rd.

SRI funciona de manera similar a SLI, pero el desplazamiento es hacia la derecha.

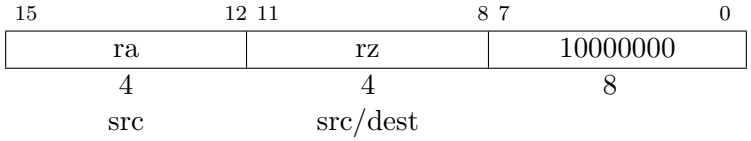
El registro de fuente y destino tienen que ser diferentes de r0. El valor del inmediato debe cumplir lo siguiente: $1 <= imm5 <= 31$, sin extensión de signo.

Posibles permutaciones: $31 * 15 * 2 * 15$

Uso:

```
sli <rd>, <ra>, <imm5>
shi <rd>, <ra>, <imm5>
```

2.3. Operaciones de multiplicación



MUL realiza la multiplicación sin signo de los valores en los registros ra y rz, y guarda el resultado en rz.

Tanto ra como rz deben ser distintos de 0.

Posibles permutaciones: 15^2

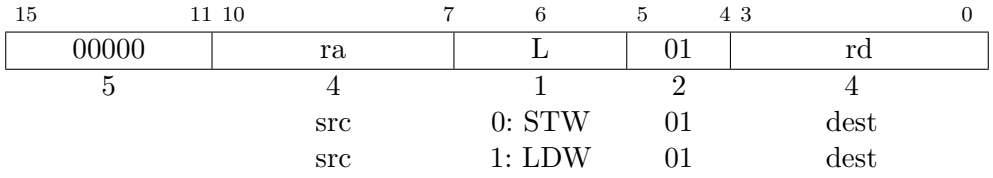
Uso:

```
mul <rz>, <rz>, <ra>
```

2.4. Instrucciones de acceso a memoria

2.4.1. Load/Store

MRA es una arquitectura *load-store*, en la cual solo estas instrucciones pueden acceder a memoria. El resto de las instrucciones operan únicamente sobre registros de propósito general.



La instrucción STW guarda el valor indicado en ra en la dirección de memoria especificada en rd.

LDW carga un valor desde la dirección de memoria especificada en ra en el registro rd.

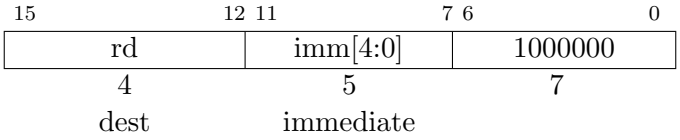
El registro ra debe ser distinto de cero.

Posibles permutaciones: $2^5 * 15$

Uso:

```
ldw <rd>, [<ra>]
stw [<rd>], <ra>
```

2.5. Carga de inmediato a registro



La instrucción IMM carga en el registro rd el inmediato *imm*. Este es *sign-extended* y debe estar en el rango: $-16 <= imm5 <= 15$.

Se debe considerar que rd debe ser distinto de r0.

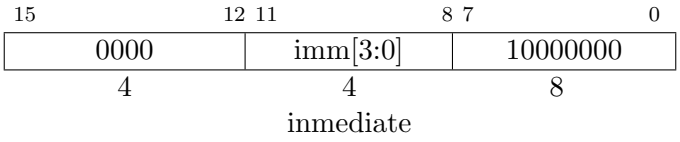
Para cargar inmediatos fuera del rango mencionado, existe una pseudo-instrucción para el caso general con imm32. La misma está realmente compuesta por un `ADR` *seguido de* `LDW`.

Posibles permutaciones: $2^5 * 15$

Uso:

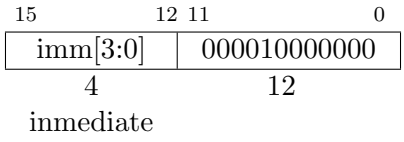
```
imm <rd>, <imm5>
```

2.6. Espacio de extensión



Instrucción que proporciona a quien implemente el ISA extender las instrucciones.

Para el caso donde $imm >= 16$ se tiene la siguiente codificación:

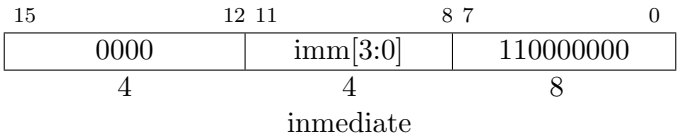


Posibles permutaciones: $2 * 2^4 - 1$

Uso:

```
ext <imm8>
```

2.7. Espacio de control



Esta instrucción permite la realización de operaciones estándares del sistema. Ejecutar un SYS no definido en la especificación genera un comportamiento indefinido.

Posibles permutaciones: 2^4

Uso:

```
sys <imm4>
```

Capítulo 3

Conjunto de pseudo-instrucciones de MRA

El presente capítulo describe las pseudo-instrucciones que el *set* de MRA proporciona. Estas no se consideran como parte del ISA, sino que se están compuestas por una o varias de las instrucciones del *set*.

3.1. Pseudo-instrucciones de control de flujo

3.1.1. Salto incondicional relativo

3.1.1.1. hlt

La instrucción HLT realiza saltos incondicionales a la misma dirección de tal manera que el programa se mantiene en un bucle infinito al llegar a esta instrucción.

Está compuesta por una instrucción BAL donde el *label* es la misma dirección de la instrucción.

Composición:

```
bal .
```

Uso:

```
hlt
```

3.1.1.2. nop

NOP define una instrucción que no implementa ninguna operación.

Uso:

```
nop
```

3.1.2. Salto incondicional indirecto

3.1.2.1. ret

Instrucción utilizada para retornar cuando se realiza el llamado a una función.

Está compuesta por una instrucción BIN donde el *target* es el *link register*.

Composición:

```
bin lr
```

Uso:

```
ret
```

3.1.2.2. rst

La instrucción RST se encarga de reiniciar la ejecución del programa. Para esto, hace uso de la instrucción BIN para realizar un salto incondicional a la posición cero, que es donde se encuentra el inicio del programa.

Composición:

```
bin zero
```

Uso:

```
rst
```

3.2. Pseudo-instrucción de llamada a funciones

3.2.0.1. blk

BLK se utiliza para realizar llamados a funciones dentro del programa.

Para llevarlo a cabo, en primer lugar utiliza la instrucción ADR para almacenar la dirección de retorno relativa el pc en el *link register* y seguidamente realiza un salto incondicional relativo con BAL a la dirección de la función.

Composición:

```
adr lr, <offset>
bal <offset>
```

Uso:

```
blk <label>
```

3.3. Pseudo-instrucciones de ALU

3.3.0.1. mov

La instrucción MOV realiza el movimiento de los contenidos de un registro ra, en el registro de destino rd.

Para realizar esto, utiliza la instrucción ADD, donde el destino es rd, y se suman los valores de r0 que es siempre cero y ra. Esto resulta en que el contenido de ra se copia a rd.

Composición:

```
add <rd>, zero, <ra>
```

Uso:

```
mov <rd>, <ra>
```

3.3.0.2. neg

NEG se encarga de obtener el valor negativo del valor contenido en el registro ra y lo almacena en rd

Está definida como una instrucción SUB donde rd se mantiene como el registro de destino, el primer operando corresponde a r0 y el segundo es ra. De esta manera se almacena en rd el resultado de restar el valor de ra a r0 (cero).

Composición:

```
sub <rd>, zero, <ra>
```

Uso:

```
neg <rd>, <ra>
```

3.4. Pseudo-instrucción de carga de inmediato a registro

Esta pseudo-instrucción proporciona la posibilidad de almacenar un valor superior a 4 bits en el registro rd. Para esto utiliza una composición de las instrucciones ADR y LDW.

Composición:

```
adr <ra>, <imm>
ldw <ra>, [<ra>]
```

Uso:

```
imm <rd>, <imm32>
```

3.5. Pseudo-instrucciones almacenamiento en el *stack*

3.5.0.1. psh

La instrucción PSH se encarga de almacenar los valores de los registros especificados en el espacio de direccionamiento del *stack*.

Para cada registro en esa lista, en primer lugar se utiliza la instrucción DEC para decrementar el *stack pointer* en 4 y posteriormente se almacena el valor del registro en la dirección del *sp* calculada mediante el uso de la instrucción STW.

Si se recibe más de un registro, se realiza la composición para cada uno de ellos en el orden especificado.

Composición:

```
dec sp, 4
stw [sp], <ra>
```

Uso:

```
psh <ra>, <rb>, <rc>, ...
```

3.5.0.2. pop

La instrucción POP se encarga de cargar en los registros especificados los contenidos del *stack*.

Para cada registro en esa lista, se carga en el mismo el valor que se encuentra en la dirección de memoria especificada en el *stack pointer*, mediante el uso de la instrucción LDW. Seguidamente, se incrementa el valor del *sp* en 4 para apuntar a la siguiente dirección. Para esto se hace uso de la instrucción INC.

Si se recibe más de un registro, se realiza la composición para cada uno de ellos en el orden especificado.

Composición:

```
ldw <ra>, [sp]
inc sp, 4
```

Uso:

```
pop <ra>, <rb>, <rc>, ...
```

3.6. Pseudo-instrucciones RAW

3.6.0.1. word

La instrucción WORD toma una inmediato del tamaño de una palabra (32 bits) y lo escribe directamente en el archivo binario del programa, es decir, la codificación de esta instrucción es el valor del inmediato.

Uso:

```
word <imm32>
```

3.6.0.2. hword

HWORD se encarga de escribir directamente en el archivo binario del programa el valor del inmediato de tamaño *halfword* (16 bits). Al igual que para WORD, la codificación de esta instrucción es el valor del inmediato.

Uso:

```
word <imm16>
```

Bibliografía