

CÁPITULO IV

Arquitectura de programación



El modelo de bus

el objetivo principal de este modelo es reducir el número de interconexiones entre la CPU y sus subsistemas. En lugar de tener rutas de comunicación separadas entre la memoria y cada dispositivo de entrada y salida, la CPU está interconectada con las otras dos unidades a través de un bus del sistema compartido.

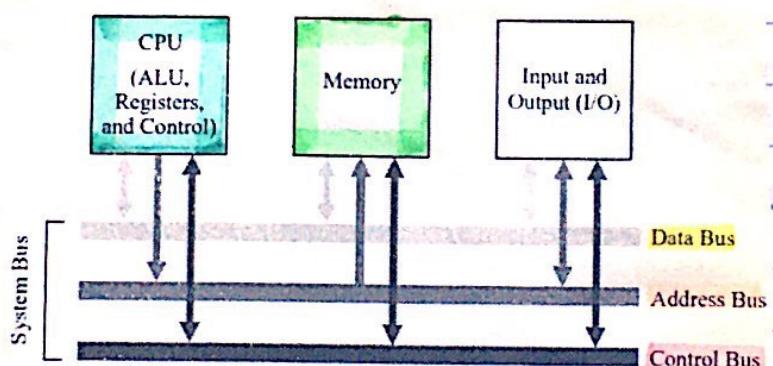


Figure 4-1 The system bus model of a computer system.

el bus de datos, lleva la información que se transmite. El bus de direcciones, identifica a donde se envía la info. En cuanto a las conexiones, la CPU genera direcciones, pero nunca recibe (con la unidad de memoria es al revés). El bus de control describe aspectos de cómo y de qué manera se envía la información.

MEMORIA

Consiste en una colección de registros numerados direccionalmente, contienen un byte y cada registro tiene una dirección mediante la cual los referenciamos. Un byte es una colección de 8 bits (1 bit es un pequeño comutador que puede ser 1 o 0).

Una palabra es una colección de N bytes, donde N está definido por el tipo de procesador. Un proc. de 32 bits tiene palabras de 32 bits (4 bytes) y

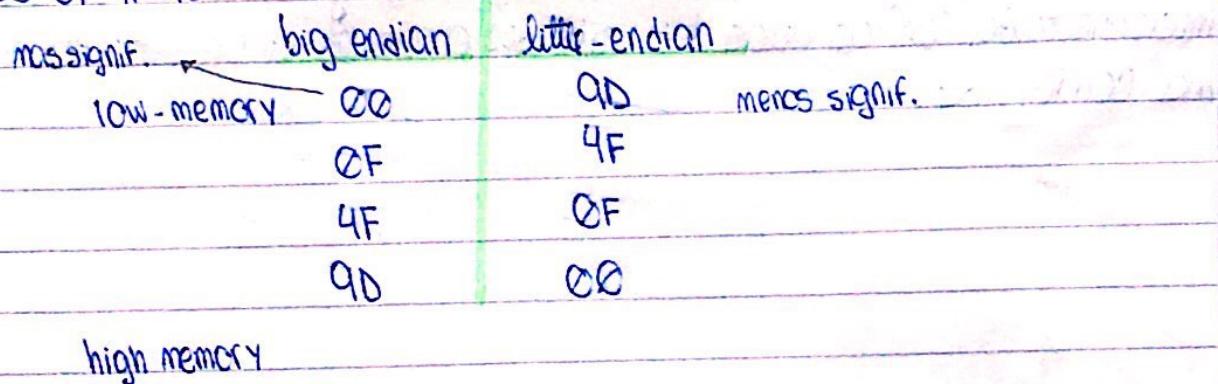
Uno de 64 bits de 64 bits (8 bytes)

Little endian vs big endian

Supongamos que tenemos la palabra `0x000000401`. Es una palabra de 4 bytes compuesta por los bytes `00, 00, 04, 01`. Para acceder a la palabra completa, se debe hacer direccionalo a por la dirección de byte más baja.

Se puede almacenar la palabra como la secuencia `00, 00, 04, 01` (como está escrita), colocando su byte más significativo en la dirección más baja conocida como big-endian. Otra forma es almacenarla como la secuencia `01, 04, 00, 00` colocando su byte menos significativo en la dirección más baja, conocida como little-endian.

$$X = 00\ 0F\ 4F\ 9D$$



Mapa de memoria

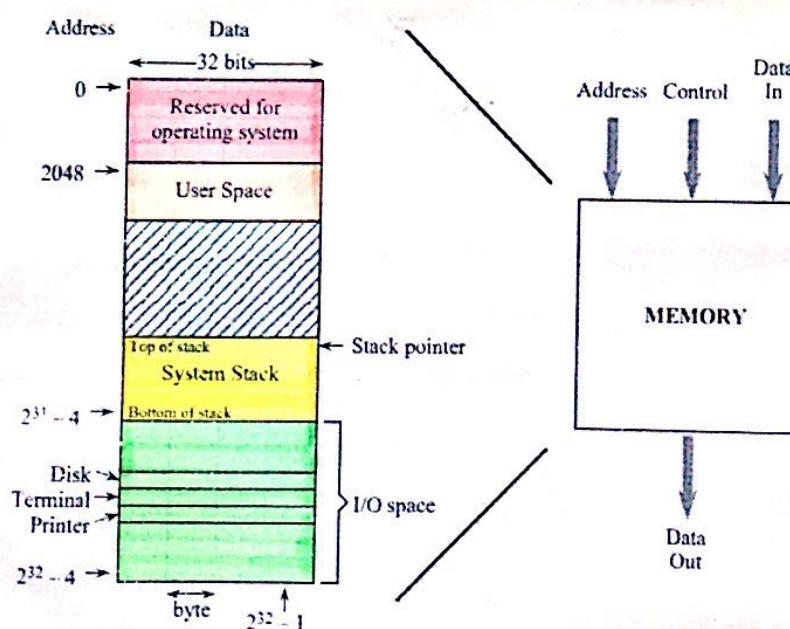


Figure 4-4 A memory map for an example architecture (not drawn to scale).

Al contarse las direcciones en secuencia comenzando en cero, la dirección más alta es una menos que el tamaño de la memoria. Entonces, una memoria de 2^{32} bytes (4 GB), el rango de direcciones va de 0 a $2^{32} - 1$.

Las primeras $2^{11} = 2048$ (de 0 a 2047) están reservadas para que las use el sistema operativo. El espacio para el usuario es donde se carga el programa ensamblado de un usuario y puede crecer durante la ejecución hasta encontrarse con la pila del sistema, la cual comienza en la ubicación $2^{31} - 4$ y crece hacia direcciones inferiores. El método para interactuar con los dispositivos de E/S es mediante un sistema de entrada-salida mapeado en memoria; en el que los dispositivos ocupan posiciones de memoria del espacio de direcciones entre 2^{31} y $2^{31} - 1$ y se leen y escriben como si fueran posiciones de memoria.

CPU o unidad central de procesos.

Ciclo de búsqueda - ejecución

La implementación del hardware ISA se expresa en términos de este ciclo. Los pasos del mismo son:

- ① Buscar la próxima instrucción. En este paso, los operandos se recuperan en memoria.
- ② Decodificar la instrucción. Este paso coloca los operandos en un formato manejable por la ALU.
- ③ Busqueda de operandos (si hay)
- ④ Ejecutarla y almacenar el resultado, realiza la operación seleccionada dentro de la ALU.

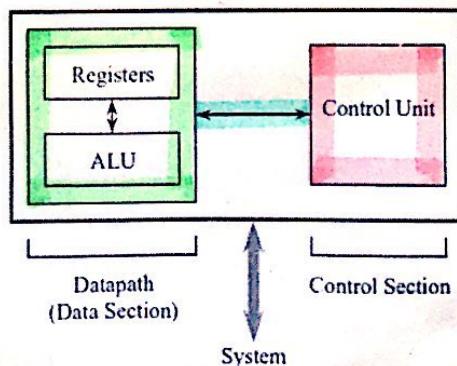


Figure 4-5 High level view of a CPU.

La unidad de control interpreta las instrucciones y efectúa transf. de registros. Consiste únicamente de la unidad de control. Es responsable de controlar las instrucciones del programa y luego dar instrucciones para que la ruta de datos execute realmente las distintas operaciones aritméticas y lógicas. La interfaz entre la unidad de control y el bloque de datos.

Hay 2 registros que forman la interfaz entre ambas secciones. Uno es el PC (Program Counter). Este registro contiene la dirección de la instrucción que se está ejecutando. La instrucción apuntada por PC se obtiene de memoria y se almacena en IR (instruction register) donde es interpretada.

Bloque de datos

el tránsito (bloque) de datos está formado por un conjunto de registros y la ALU que es la unidad que realmente ejecuta las operaciones.

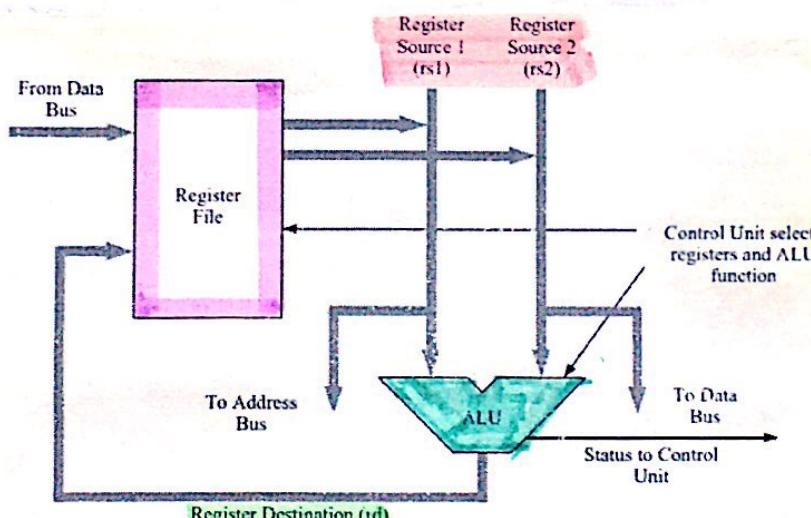


Figure 4-6 An example datapath.

el conjunto de registros puede pensarse como una memoria pequeña y rápida separada de la memoria del sistema, que se utiliza para almacenamiento temporal durante el cálculo. Como en la memoria del sistema, a cada registro se le asigna una dirección. Estas direcciones de registros son mucho más pequeñas que las de memoria principal. El conjunto de registros es mucho más rápido que la memoria principal.

La ALU implementa una variedad de operaciones: And, Not, Or. Estas operaciones (en la ALU) y los operandos que utilizan (los registros en el conj. de registros) son seleccionados por la unidad de control.

Los registros de origen son dos buses que admiten que los operandos sean buscados en el conj. de registros, que luego son operados por la ALU.

El registro de destino es a donde llega se devuelve el resultado de la operación de la ALU.

La arquitectura de la computadora ARC

el compilador se encarga de transformar el código de alto nivel en lenguaje simbólico (Assembly). En el momento de compilación. En el momento de ensamblaje se traduce a Código de máquina.

Memoria: contiene palabras de 32 bits y utiliza almacenamiento big endian.

CPU: es un subconjunto de la arquitectura SPARC, con un conjunto de instrucciones reducido. Tiene 32 registros numerados del %r0 al %r31 de 32 bits cada uno, así como el Program Counter (PC) y el reg de instrucción (IR)

Registro 00	%r0 [= 0]	Registro 11	%r11	Registro 22	%r22
Registro 01	%r1	Registro 12	%r12 Puntero de pila	Registro 23	%r23
Registro 02	%r2	Registro 13	%r13	Registro 24	%r24
Registro 03	%r3	Registro 14	%r14 [%sp]	Registro 25	%r25
Registro 04	%r4	Registro 15	%r15 [link]	Registro 26	%r26
Registro 05	%r5	Registro 16	%r16 ↓	Registro 27	%r27
Registro 06	%r6	Registro 17	%r17 Registro de enlace	Registro 28	%r28
Registro 07	%r7	Registro 18	%r18	Registro 29	%r29
Registro 08	%r8	Registro 19	%r19	Registro 30	%r30
Registro 09	%r9	Registro 20	%r20	Registro 31	%r31
Registro 10	%r10	Registro 21	%r21		
PSR		PC		32 bits	
32 bits		32 bits			

Figura 4.9 • Registros en ARC accesibles al programador.

el PSR es el registro de estado del procesador y es donde se indican los flags provenientes de la ALU. (Z, N, V, C)

ARC es una arq de load/store. Las instrucciones se deben cargar en el IH antes de ser ejecutadas, así como los operandos necesarios. Para ejecutarlas deben ser antes cargados en registros. No se opera con la memoria ppal, las únicas instrucciones permitidas para el acceso a memoria son:

- Cargar un valor de memoria en registro
- Almacenar en memoria el contenido de algún registro.

	Mnemónico	Función
Memoria	ld	Cargar un registro desde memoria
	st	Salvar un registro en memoria
	sethi	Cargar los 22 bits más significativos de un registro
	andcc	Producto lógico bit a bit (Y)
Lógicas	orcc	Suma lógica bit a bit (O)
	orncc	Suma lógica negada bit a bit (NOR)
	srl	Desplazar a derecha (lógico)
	addcc	Sumar
Aritméticas	call	Salto (llamado) a subrutina
	jmpl	Retorno de subrutina
	be	Bifurcación (salto por igual)
	bneq	Bifurcación (salto) por negativo
Control	bcs	Bifurcación (salto) por arrastre
	bvs	Bifurcación (salto) por desborde
	ba	Bifurcación (salto) incondicional

Figura 4.7 • Subconjunto de instrucciones para la arquitectura de programación ARC.

el sufijo 'CC' especifica que después de realizar la operación se deben actualizar los códigos de condición de Psh.

SRL desplaza un registro hacia la derecha y carga 0s en los bits más significativos

SRA desplaza hacia la derecha y almacena una copia del bit más significativo (extensión de signo)

Formato del lenguaje simbólico

Rótulo	Mnemónico	Operandos de origen	Operandos de destino	Comentario
lab_1:	addcc	\$r1, \$r2, \$r3		!Ejemplo de código simbólico

Figura 4.8 • Formato de una sentencia en el lenguaje simbólico SPARC (también ARC).

Formatos de instrucción en ARC

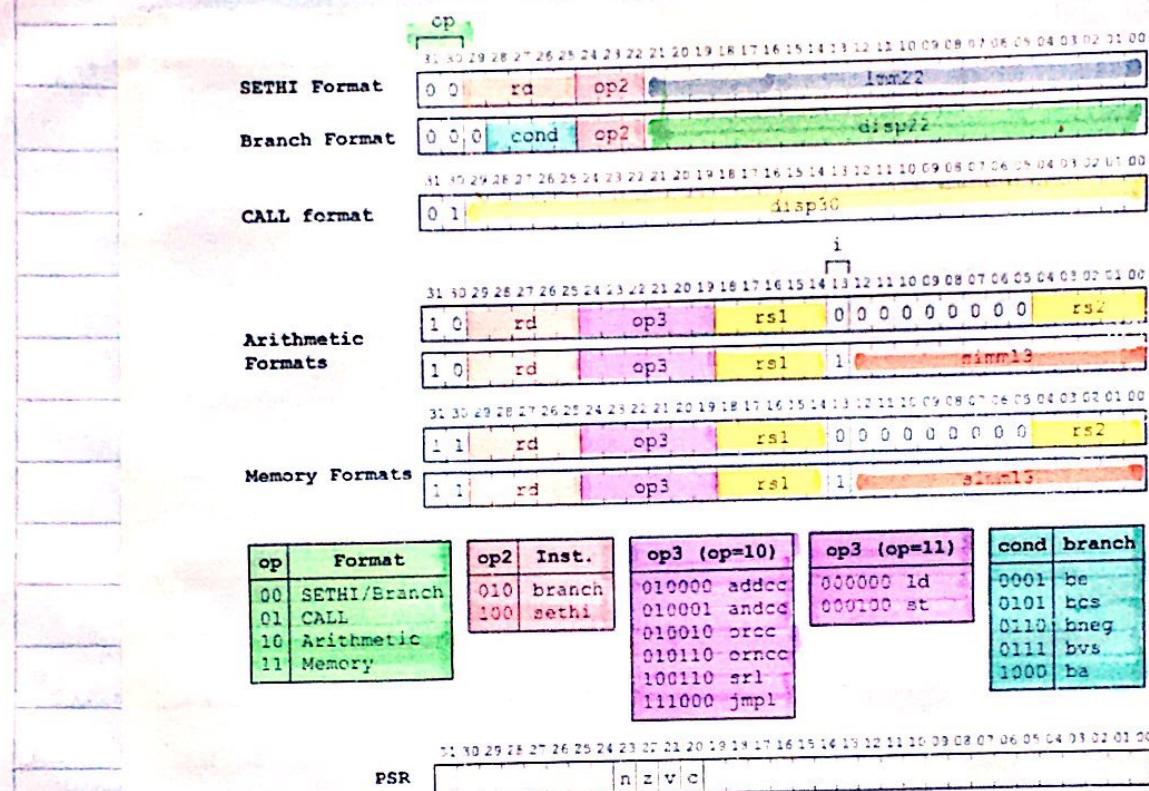


Figure 4-10 Instruction formats and PSR format for the ARC.

Los 2 bits más significativos forman el campo OP correspondiente al código de operación. El campo OP2 indica si es un formato SETHI o un salto. El campo rd de 5 bits identifica el registro al que se le aplicará SETHI. El campo cond identifica el tipo de salto condicional que se basa en los códigos de condición (n, z, c y v) del PSR. El campo op3 indica la operación a realizar. Los campos rs1 y rs2 son el primer y el segundo registro de origen respectivamente.

im22 → Cte de 22 bits que se utiliza como operando en SETHI

disp22 → " " " " para calcular la dirección de salto.

disp30 → desplazamiento de 30 bits utilizados para determinar la dir de la rutina invocada.

simm13 → Valor de 13 bits que se extiende para con signo para el segundo registro origen.

DIRECTIVAS

Directiva	Forma de uso	Función o significado
.equ	x .equ #10	Asignar a X el valor (10) ₁₆
.begin	.begin	Comienzo de traducción
.end	.end	Fin de traducción
.org	.org 2048	Cambiar el contador de posición a 2048
.dwb	.dwb 25	Reservar un bloque de 25 palabras
.global	.global Y	La variable Y se usa en otro módulo
.extern	.extern Z	La variable Z se define en otro módulo
.macro	.macro M a, b, ...	Definir macroinstrucción M. Parámetros formales: a, b, ...
.endmacro	.endmacro	Fin de definición de macroinstrucción
.if	.if <cond>	Ensamblar solo si <cond> es cierta
.endif	.endif	Fin de estructura condicional

Figura 4.12 • Directivas para el lenguaje ensamblador de ARC.

Variantes en las arquitecturas y en los direccionamientos

- Instrucciones de tres direcciones: opera con 2 de las dirs y guarda el resultado en la 3^º
- Instrucciones de dos direcciones: uno de los operandos se sobreescribe con el resultado.
- Instrucciones de una dirección/acumulador: utiliza un solo registro aritmético conocido como acumulador que sirve como operando y almacena el resultado.

El acceso a memoria

Modos de direccionamiento:

- Inmediato: referencia a un valor CTE, conocido al momento del ensamblaje
- Directo: dirección de memoria incluida en la instrucción.
- Indirecto: dirección de memoria donde está el puntero al dato (muy lento)
- Indirecto por registro: el registro tiene el puntero al dato (almac. de OPS en pila)
- Indexado: un registro da la dirección inicial y el otro un incremento.

PILAS Y SUBRUTINAS

Una subrutina (función/procedimiento) es una secuencia de instrucciones a la que se invoca como una única instrucción. Cuando un programa llama a una subrutina, se le transfiere a esta el control del programa.

La misma ejecuta la secuencia de instrucciones y luego vuelve a la posición siguiente a la que generó el llamado.

Convenciones de llamada (métodos para pasar args desde y hacia la rutina).

① Colocar los argumentos en registros. Es sencillo pero no funciona si la cantidad de elementos excede el número de registros o si la llamada a subrutinas están fuertemente encadenadas.

Se tiene que conocer el tamaño de la zona de datos. La directiva .dwb genera la zona de t. de X palabras siendo X la dirección de comienzo de la zona de datos que se le transfiere a la subrutina en un registro. Se cargan los argumentos en X, X+4 y la subrutina los toma de la dirección X y los guarda ^{la pos} en X+8. Permite el pasaje de bloques de cualquier tamaño sin tener que copiar más de un registro en el proceso de la llamada a la subrutina pero se puede complicar si una rutina es recursiva ya que se generarían varias zonas de transf.

② Parámetros por stack. La rutina invocante coloca a todos sus args en una pila. La rutina invocada extrae de la pila los args transferidos y coloca en la misma el valor de retorno. El registro de la CPU conocido como puntero a la pila contiene la dir de la cabeza de la pila. Ventaja: no se debe conocer el tamaño de la pila.

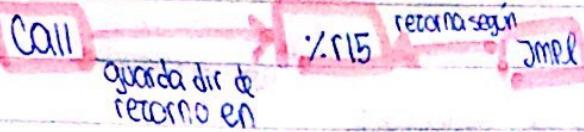
<pre> ! Rutina invocante : ld (x), @r1 ld (y), @r2 call add_1 st @r3, [z] : x: 53 y: 10 z: 0 </pre>	<pre> ! Rutina invocada ! @r3 ← @r1 + @r2 add_1: addcc @r1, @r2, @r3 jmpl @r15 + 4, @r0 </pre>
--	--

Figura 4.15 • Llamado a subrutina utilizando registros.

<pre> ! Rutina invocante : st @r1, [x] st @r2, [x+4] sethi x, @r5 srl @r5, 10, @r5 call add_2 ld [x+8], @r3 : !</pre>	<pre> ! Rutina invocada ! x[2] ← x[0] + x[1] add_2: ld @r5, @r8 ld @r5 + 4, @r9 addcc @r8, @r9, @r10 st @r10, @r5 + 8 jmpl @r15 + 4, @r0 </pre>
<i>! Zona de transferencia de datos</i> x: .dwb 3	

Figura 4.16 • Llamado a subrutina usando una zona para transferencia de datos.

dirección de retorno



Tipo de convención usada al pasar parámetros
Por registros
Por área reservada en memoria
en la pila

dónde se guarda .f15

registro no reservado

Incluido en el área reservada
pila

! Rutina invocante
:
.esp .equ \$r14
addcc \$sp, -4, \$sp
st \$r1, \$sp
addcc \$sp, -4, \$sp
st \$r2, \$sp
call add_3
ld \$sp, \$r3
addcc \$sp, 4, \$sp
:

! Rutina invocada
! Los argumentos están en la pila
! \$sp[0] ← \$sp[0] + \$sp[4]
\$sp .equ \$r14
add_3: ld \$sp, \$r8
addcc \$sp, 4, \$sp
ld \$sp, \$r9
addcc \$r8, \$r9, \$r10
st \$r10, \$sp
jmp \$r15 + 4, \$r0

Figura 4.17 • Llamado a subrutina utilizando una pila.

CAPÍTULO V

Los lenguajes y la máquina

Compilación: Proceso de traducción de un programa escrito en lenguaje de alto nivel a otro funcionalmente equivalente en lenguaje ensamblador. Traducir lenguaje ensamblador a lenguaje de máquina es muy directo ya que hay una relación de uno a uno entre las sentencias.

Pasos de la compilación:

- Análisis lexicográfico: Reconocer simbolos básicos del lenguaje (identif, decls y defs)
- Análisis sintáctico: Con los simbolos del paso anterior, reconocer las sentencias y verificar identificadores, expresiones, constantes,
- Análisis semántico:

Análisis de nombres: Asociar los nombres con variables y luego con posiciones de memoria.

Análisis de tipo: Determinar el tipo de todos los datos requeridos y compatibilidad.

Asignación de acciones y generación de código: Asociar cada sentencia del programa con una (o más) secuencia(s) del lenguaje ensamblador.

Especificación del mapeo:

El compilador debe conocer la arquitectura de programación del procesador y máquina en la que se desarrolla ej: como ubicar variables en recursos de la máquina. Es posible que el compilador soporte compilar un programa para una arq diferente donde corre el mismo compilador, este proceso se llama cross-compiling.

Almacenamiento de variables en memoria

- Variables globales: Son accesibles durante todo el curso de programa y están fijas en memoria en una ubicación conocida al momento de compilación.
- Variables locales: declaradas dentro de funciones, nacen cuando se ingresa a dicha función y dejan de existir al finalizar. Se usan pilas para el almacenamiento de estas.

Direcccionamiento base: se accede a las variables usando una copia del %.SP (stack pointer), llamada %.FP (frame pointer) más un offset.

Movimiento de datos

- Estructuras (structs) : el compilador acomoda la estructura en direcciones de memoria consecutivas, pudiendo acceder por base.
- Vectores (arrays) : la longitud es conocida al momento de compilación
la posición del elemento i -ésimo se calcula como:
dirección del $i^{\text{º}}$ elem \leftarrow base + ($i - \text{comienzo}$) * tamaño.

Instrucciones aritméticas

es posible que el compilador encuentre alguna instrucción aritmética que requiere mayor cantidad de registros de los que dispone. En este caso almacenan temporalmente algunas variables en la Pila.

Control de secuencia

(d)

• GOTO : se implementa con una instrucción de salto incondicional, branch always : ba rótulo ! Salto a rótulo

• IF-ELSE : si se cumple la condición se ejecuta un código, y sino el otro.

subcc %r1,%r2,%r0 ! restar los reg y fija los flags

bne over ! bne verifica el flag Z; si es 0 salta a over.

if: ! Código que se ejecuta si son iguales y no salta a over

ba End ! Cuando el código ya se ejecutó salta aca.

else: over: ! Código que se ejecuta si no son iguales.

End: ! Código después del if.

(a)

• While:

ba Test ! Salto incondicional a Test

True: add %r3,1,%r3 ! le suma 1 a r3

Test: subcc %r1,%r2,%r0 ! resta r1 y r2

be True ! Si Z está en 1, salta a True.

• do-while :

True: << código del while >>

Test: << cond >>

be True.

Ensamblado.

Traducción del lenguaje simbólico (assembly) a código binario. Es un proceso lineal ya que las direcciones de ASM se corresponden directamente con instrucciones del procesador.

Tabla de símbolos.

Consta de dos campos:

- Símbolo: lozulo o nombre simbólico que se refiere a un valor utilizado durante el proceso de ensamblado.
- Valor: la posición de cada símbolo, cada línea le suma 4 a la pos.
Las pseudo instrucciones no generan código (no se suma nada)

Primera pasada.

Se determinan las direcciones de todos los datos e instrucciones, mediante un contador llamado location counter. Si bien comienza en cero se puede modificar su valor directamente con la directiva .org. Expande las macros e inserta en la tabla de símbolos cualquier definición de identificadores junto con su ubicación en memoria según el location counter.

Segunda pasada.

Se usa para permitir el uso de símbolos dentro de un programa, con anterioridad a ser definidos. Cada instrucción es convertida a código de máquina. Cada identificador es reemplazado por su ubicación en memoria según la tabla de símbolos y se procesa por completo cada línea antes de pasar a la siguiente. Si al final no hay ningún error (ej: rótulos sin definir) se genera el código binario.

Tareas finales.

Después del proceso de traducción, el ensamblador debe agregar información adicional para el uso de enlace y carga.

- Nombre y tamaño del módulo.
- Dirección del símbolo de comienzo.
- Información acerca de símbolos globales y externos.
- Información acerca de las rutinas de biblioteca a las que el módulo hace ref.
- Valores de cualquier constante que deba cargarse en memoria

- Información de reubicación.

Enlace y Carga

Linker (Enlace)	Loader (Carga)
Unir en un programa único distintos módulos que fueron ensamblados de forma separada.	Traslado del programa a memoria y su preparación para ser ejecutada.
Resuelve todas las referencias globales y externas y reubica las direcciones de los distintos módulos.	el módulo de carga puede ser cargado en memoria por medio de un cargador que también puede necesitar modificar direcciones si el programa se carga en una distinta a la de origen usada por el linker.

DLL → biblioteca de enlace dinámico que pospone el enlace de algunos componentes hasta que sean requeridos efectivamente.

Enlace (linking)

el programa de enlace debe:

- resolver referencias de direcciones externas.
- reubicar los módulos
- Especificar el símbolo de comienzo del módulo de carga.

Reubicación.

Dos módulos con la misma dirección de comienzo no pueden ocupar la misma dirección en memoria. Si se ensamblan por separado, no hay forma de que un programa ensamblador pueda descubrir el conflicto. Por esto, se definen como reubicables a los símbolos que puedan admitir que su dirección se modifique durante el proceso de enlace.

No son relocables:

- Direcciones de entrada/salida

• Rótinas del sistema

Son relocalizables:

• Posiciones de memoria relativas a un .org

Carga (loader)

Carga los diversos segmentos de memoria con los valores apropiados e inicializa ciertos registros como el %.SP y el %.PC a sus valores iniciales. Cuando hay varios módulos en ejecución, el loader debe reubicar estos módulos en el momento de la carga, para lo cual suma un desplazamiento a todo el código reubicable del módulo en cuestión. También se puede usar la MMU (memory management unit) que realiza la reubicación durante la carga, utilizando un registro base con RPO a la cual todas las demás instrucciones se reubican.

Macroinstrucciones

• Asignar nombre a la secuencia de instrucciones que llevan a cabo una operación. Se acceden en tiempo de ensamblado convirtiéndola en su código equivalente (expansión).

Macro vs subrutina

Macroinstrucción	Subrutina
Se accede en tiempo de ensamblado convirtiendo su código equivalente.	Se accede en tiempo de ejecución por un CALL y termina con JMP en t. de ejecución
sus parámetros son interpretados por el ensamblador y reemplazados por lo que corresponde en el lugar que es invocada.	sus parámetros le llegan en tiempo de ejecución (fila o registros).
Código de máquina repetido tantas veces como sea invocada la macro	Código de máquina localizado en un lugar específico y único.

CÁPITULO VI

Trayecto de datos y control.

Microarquitectura

La microarquitectura es un nivel más profundo que la arquitectura y está compuesta por las mismas unidades (sección de control, Trayecto de datos).

Consiste en la unidad de control, las unidades funcionales (ALU), los registros accesibles al programador y todo otro registro adicional que la unidad de control pueda requerir. La idea general es que la sección de control es la encargada de seleccionar la microinstrucción a ejecutar mientras que la sección de datos es la encargada de ejecutarla.

Cuando la microarquitectura comienza a funcionar, un circuito de reinicio coloca el microcódigo en la ubicación 0 en el control store en el MIR y lo ejecuta. A partir de ahí comienza el ciclo fetch y cada unidad hace su trabajo para que el ciclo continve.

Trayecto de datos

está compuesto por:

- 32 registros (%f0 a %f31) accesibles por el usuario.
- Program Counter (%PC): solo se puede acceder a través de CALL y JMPI y contiene la próxima instrucción.
- registros temporales (%temp0 a %temp3) no accesibles por el usuario, para interpretar instrucciones AFC.
- Registro de instrucciones (%ir) no accesible por el usuario que contiene la instrucción en ejecución.
- La ALU que puede realizar 16 operaciones sobre los buses A y B y el resultado de cada operación se coloca en el bus C (a menos que este bloqueado debido a que el multiplexor que maneja el bus C haya colocado una palabra proveniente de la memoria de la máquina).

Ciclo fetch

- ① Busqueda prox instrucción
- ② Decodificación del código
- ③ Busqueda de operandos en mem
- ④ Ejecución y almacenamiento
- ⑤ I

Operaciones sobre los buses.

- ANDCC(A,B) : modifica los flags, producto lógico bit a bit.
- ORCC(A,B) : suma lógica bit a bit
- NORCC(A,B) : Or negado
- ADDCC(A,B) : suma aritmética en complemento a 2.
- SRL(A,B) : desplazar A a derecha tantos bits como B indique
- AND (A,B) : producto lógico bit a bit
- OR (A,B) : suma lógica bit a bit.
- NOR (A,B) : Or negado
- ADD (A,B) : suma aritmética comp a 2.
- LSHIFT2(A) : desplaza a la izquierda el contenido de A 2 bits
- LSHIFT10(A) : igual pero 10 bits.
- SIMM13(A) : recupera los 13 bits menos significativos de A
- SEXT13(A) : extiende el signo de los 13 bits menos significativos de A.
- INC (A) : incrementa en 1 el valor de A
- INCPC (A) : incrementa en 4 el valor de A (para el %PC)
- RSHIFT5(A) : desplaza a derecha 5 bits, y completa a izquierda extendiendo el signo.

CC

→ Tocan los flags (códigos de condición) (los otros no)

$F_3 \ F_2 \ F_1 \ F_0$	Operación	Modifica códigos de condición
0 0 0 0	ANDCC (A, B)	sí
0 0 0 1	ORCC (A, B)	sí
0 0 1 0	NORCC (A, B)	sí
0 0 1 1	ADDCC (A, B)	sí
0 1 0 0	SRL (A, B)	no
0 1 0 1	AND (A, B)	no
0 1 1 0	OR (A, B)	no
0 1 1 1	NOR (A, B)	no
1 0 0 0	ADD (A, B)	no
1 0 0 1	LSHIFT2 (A)	no
1 0 1 0	LSHIFT10 (A)	no
1 0 1 1	SIMM13 (A)	no
1 1 0 0	SEXT13 (A)	no
1 1 0 1	INC (A)	no
1 1 1 0	INCPC (A)	no
1 1 1 1	RSHIFT5 (A)	no

Figura 6.4 • Operaciones aritméticas en ARC.

10

Este registro contiene siempre el valor 0 y no se sabe combinar por lo que no tiene entradas del bus C ni decodificador C y no necesita FFs.)

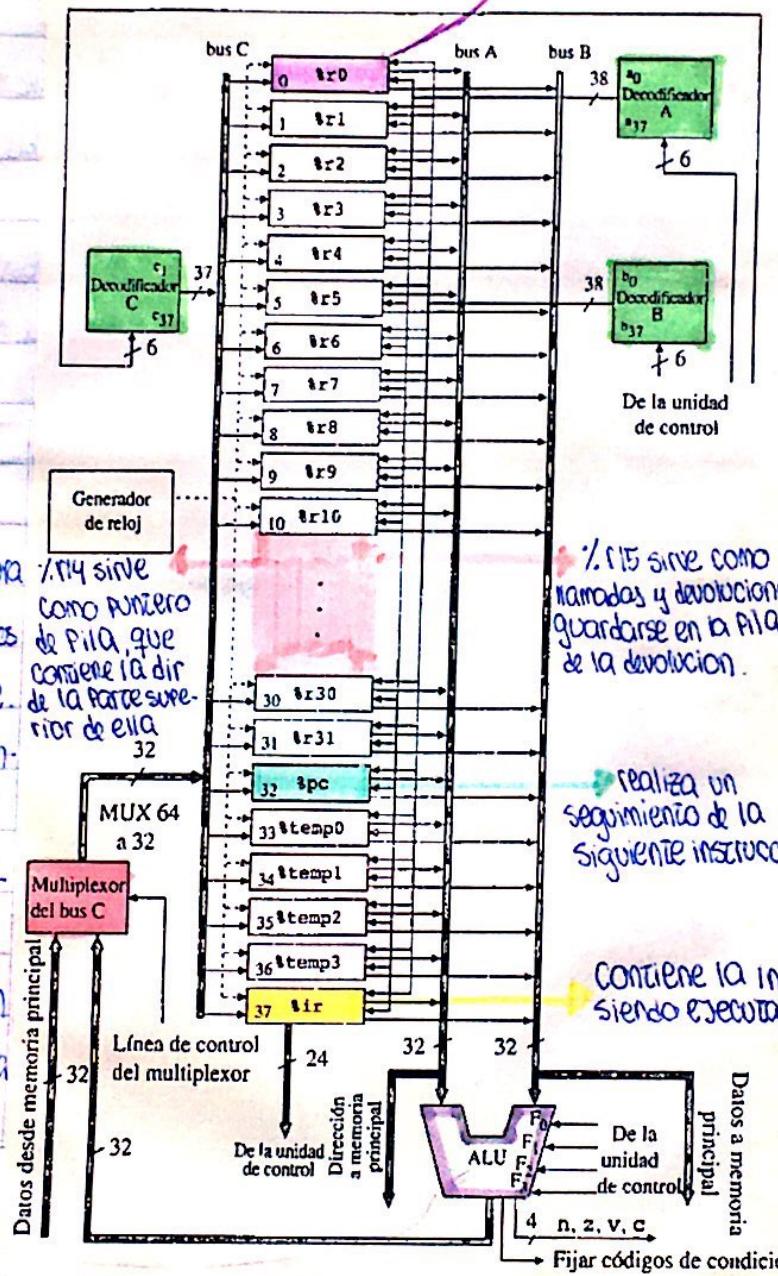


Figura 6.3 • Trayecto de datos en ARC.

LA ALU

Entradas:

- Dos entradas de datos A y B de 32 bits cada una
 - Una entrada F de control de 4 bits para elegir la operación a realizar

Solídos:

- Una salida de datos C de 32 bits que es el resultado de la operación entre A y B (que se alimenta a un multiplexor del bus y de ahí se reinsera al ir)

LOS decodificadores simplifican la selección de registros.

INPUTS: 6 bits que representan el nº de registro, procedente de AMUX/DMUX/CMUX.

OUTPUT: 38 bits que son 1xregistro con el correspondiente activado, que permitirá la lectura /escritura de los registros a los diferentes buses. Hay 64 salidas posibles para 38 registros.

Multiplexor del bus C. Su propósito es seleccionar qué 32 bits se colocarán en el bus C.

INPUTS: resultado de 32 bits de la operación de la ALU, otra palabra de 32 bits procedente de la memoria y un bit del RD del MDR. Este último indica que se está ejecutando una lectura de memoria y la palabra ingresada se coloca en el bus C. Si el bit no está establecido se está ejecutando una operación de la ALU entonces los 32 bits recibidos se colocan en el bus.

OUTPUT: la palabra de 32 bits seleccionada que se coloca en el bus C. Estos bits funcionan como la entrada para el registro seleccionado como destino para almacenar una palabra dentro del registro.

- Los códigos de condición de 4 bits (C, n, Z, V) con la señal de que deben actualizarse, que sirven como entrada de %PSR.

Las funciones de la ALU se pueden dividir en 2 tipos:

- Las operaciones aritméticas y lógicas: La implementación de hardware más simple de estas funciones es un enfoque de tabla de búsqueda (LUT). La ALU se puede descomponer en una cascada de 32 LUT que se implementan de manera casi idéntica.
- Las funciones de Combinación: La implementación del hardware implica un control de desplazamiento, con una LUT de control correspondiente (construido de manera similar a las otras LUT pero con diferentes entradas).

La ALU también debe lidar con los códigos de condición (donde funciones aritméticas con sufijo cc deben marcar si el resultado es negativo, cero, etc), N, Z, C y V que se implementan directamente y una señal que le indica a %PSR que actualice estos códigos.

Las entradas de LUT son solo un bit de cada entrada de datos ALU (4 y 8 de 32 bits), la entrada de control completa F de 4 bits (para buscar la ejecución de su LUT derecha que sirve como carry). Esta LUT resuelve las operaciones buscando las combinaciones de 4 bits en la tabla de verdad. Las salidas son: el resultado de la operación. Esto es, tanto el resultado (Z) como la ejecución que se envía al barrel shifter.

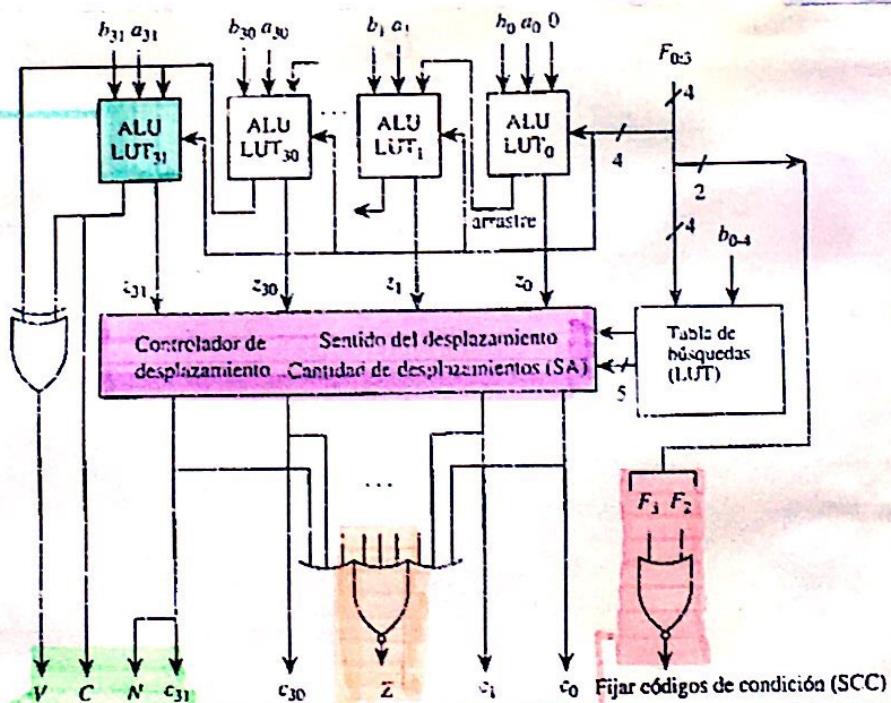


Figura 6.5 • Diagrama en bloques de la unidad aritmético-lógica de 32 bits.

La implementación de los códigos de condición es el bit más significativo de la salida del barrel shifter. C es directamente la ejecución de la LUT más significativa. El bit V se establece si los dos bits de carry más significativos. La implementación es un XOR entre la ejecución y el bit de la ejecución más significativa de la LUT.

El código de condición Z es recopilado los operaciones (cc) que serán los 2 bits más significativos de la salida de una LUT. Los flags son las únicas donde los 2 bits más significativos del input devuelven que Z es 1. La implementación - F son 0, el SCC es 1 solo si C es un NOH. Ambas señales ambas son 0. (NOH).

de un bit de cada LUT.

el control de desplazamiento.

INPUTS:

- * Entrada de control F (para saber si hay un desplazamiento)
- * Los primeros 5 bits del bus B (porque ese es el operando que indica cuantos cambios (de 0 a 31) habrá).

OUTPUTS:

- * Dirección (izq o derecho) a la que se debe aplicar el desplazamiento. 3 entradas de shifter
- * Cantidad de bits a desplazar

Incluso si no se produce ningún desplazamiento los resultados de las LUT deben pasar por el controlador de desplazamiento.

Finalmente, el output del control... es el resultado de la operación y, por lo tanto, la salida principal de la ALU: la salida C de 32 bits. Este resultado es luego utilizado por la ALU para establecer los flags.

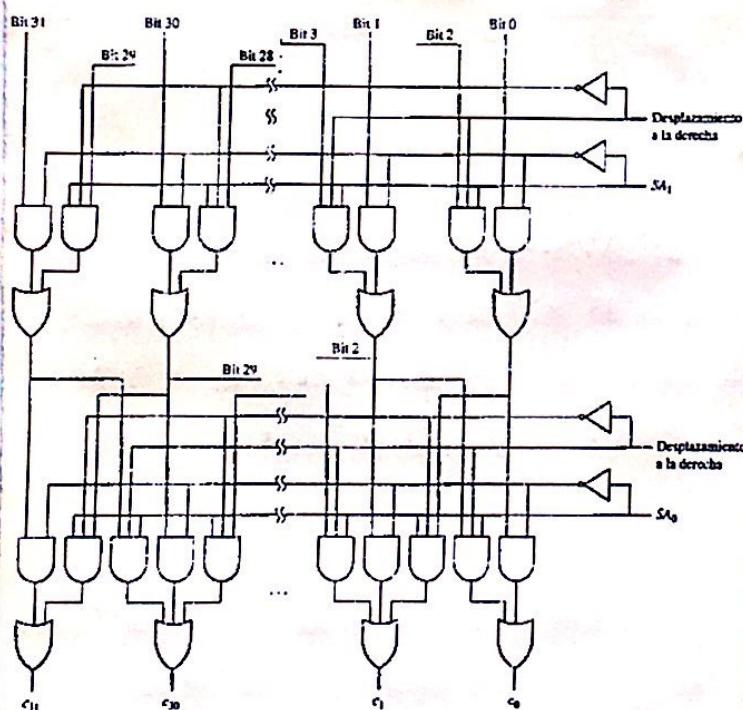


Figura 6.6 • Esquema circuital del control de desplazamiento.

Los desplazamientos se realizan en niveles en los que se observa un bit diferente del input shift amount en cada nivel.

En la parte inferior, las salidas son las mismas que las entradas si SA0 es 0. Si es 1, entonces la posición de salida tomará el valor del vecino izq o derecho (de acuerdo a la dir.).

En el siguiente nivel superior es igual solo que ahora se ve SA1 y la cantidad de desplazamiento es el doble. Concluye esto hacia SA4. Si no hay input → 0

Registros

Casi todos están diseñados de la misma forma. Las entradas son CLK su cambio se dará únicamente cuando la señal de sincronismo sufra una transición.

El propósito del registro es escribir el resultado de la operación de la ALU desde el bus C en el registro destino, o leer que registro fue seleccionado y cargarlo en el bus.

NOTA

Los output son 32 bits para escritura en los buses A y B que serán alimentados a la ALU para que se lleve a cabo la operación.

Todos los registros están implementados con flip-flops D activados por flanco descendiente. La salida del f-f no cambia hasta que el reloj hace una trans de alto a bajo. Todos los reg tienen 32 bits de ancho por lo que se utilizan 32 f-f.

INPUTS: las entradas de datos al registro se toman directamente del bus C.

se realiza un AND entre el input de CLK y C_1 lo que asegura que el registro solo cambie cuando se lo indique la sección de control (SOI).

Ochí tiene errores

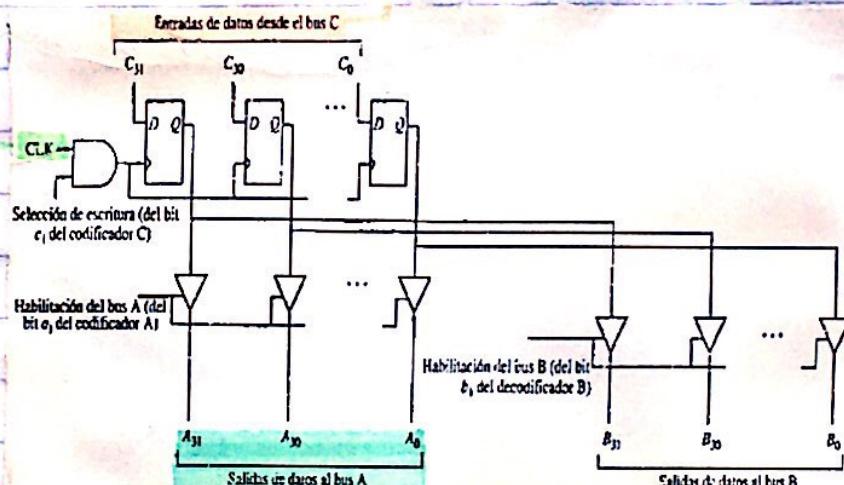


Figura 6.8 • Diseño del registro de 32 bits.

Las salidas se escriben en los buses A y B. La implementación es un buffer de 3 estados donde las salidas se habilitan en el los buses A y B y por los señales A_i y B_i de los decodificadores A y B donde i es el numero de registro. Si ni A_i ni B_i son 1 entonces las salidas se desconectan eléctricamente de los buses A y B.

Registro de instrucciones:

Contiene la instrucción a nivel usuario que se ejecutara. Se divide en varios campos y esto depende de la instrucción real ya que distintas instrucciones tienen distintos formatos. (arith + memo)

Las salidas de este registro son los distintos campos dispersos por todo el procesador.

Están los campos que representan los registros de origen y destino que son uno de los 32 posibles. Estos campos de 5 bits se alimentan a los decodificadores A, B y C para que se pueda llevar a cabo la selección de registro.

También están los códigos de operación de 8 bits que simbolizan la instrucción.

seleccionada. El mux de direcciones de memoria necesita estos códigos de operación únicos para que sean decodificados y transformados en la microinstrucción que tiene lugar.

Finalmente, el bit 13 simboliza si la instrucción tiene dos registros como operandos o si la operación usa un número entero. Este número se alimenta de la lógica de saltos de control que aplicará el salto necesario.

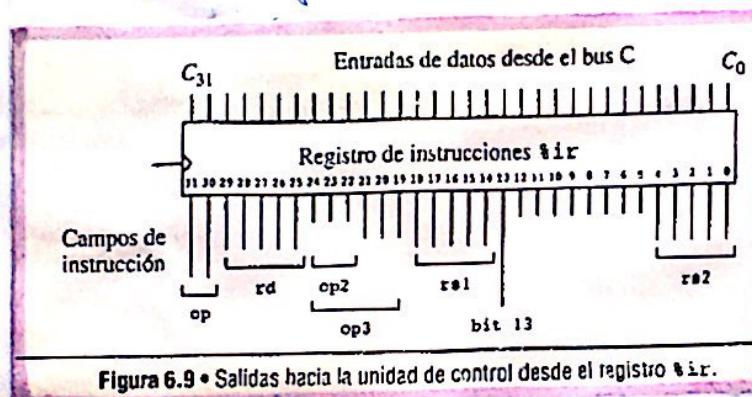


Figura 6.9 • Salidas hacia la unidad de control desde el registro RIR.

A diferencia de los otros registros, este tiene salidas adicionales que corresponden a los campos **rd**, **rs1**, **rs2**, **op**, **op2**, **op3** y el bit **13** de una instrucción. La sección de control usa estos bits para interpretar la ins.

Sección de control

Memoria de lectura (ROM) es el corazón de la unidad de control y contiene valores de 2048 palabras. Para todas las microinstrucciones que se pueden ejecutar. Hay 2048 microinstrucciones posibles, por lo que su dirección tiene una long de 11 bits y cada microinsc 41 bits. Su única entrada es la dirección de 11 bits que recibe el incrementador direcciones de la memo de control, que tiene la responsabilidad de seleccionar la instrucción a ejecutar. La función de la ROM es obtener la microinstrucción de 41 bits a la que se le hace referencia mediante la dirección recibida y colocarla en el MIR. Su única salida son estos 41 bits que alimentan al MIR.

Registro de instrucciones es donde se coloca la microinstrucción. Su propósito es de **microprograma (MIP)** derivar cada bit a su lugar correspondiente. La microinstrucción es una palabra de 41 bits dividida en 11 campos. Los inputs son la unidad de reloj y la microinstrucción proveniente de la ROM. Sus outputs son:

En primer lugar los campos A, B y C de 6 bits que representan los registros donde se operará la microinstrucción. La longitud de 6 bits es porque se puede seleccionar uno de 38 registros. Estos campos se fijan a los multiplexores A, B y C que deciden si los registros en los que opera la sección de datos proviene del MIP u otro lugar. El campo

A/B/C MUX establece cuando una instrucción opera en registros. Despues los campos RD y WR de 1 bit que se alimentan a la memòia PFM y establecen si es una operaciòn de carga o almacenamiento. RD tambièn controla el MUX 64-32 que se encarga de determinar si el destino de la operaciòn es el resultado o se carga desde memòia PFM.

Despues, el campo ALU de 4 bits indica la operaciòn y la envia a la ALU. Luego, el campo COND de 3 bits hace que el microcontrolador rescale la prox microinstrucciòn desde la siguiente pos en la ROM, la pos indicada en JUMP ADDR o lo almac en %.ir. Este campo se alimenta del CSEL que està a cargo de la lògica de determinar la prox microins.

Por ultimo el campo de 11 bits JUMP ADDR que va directo al MUX de dirs de memòia de control.

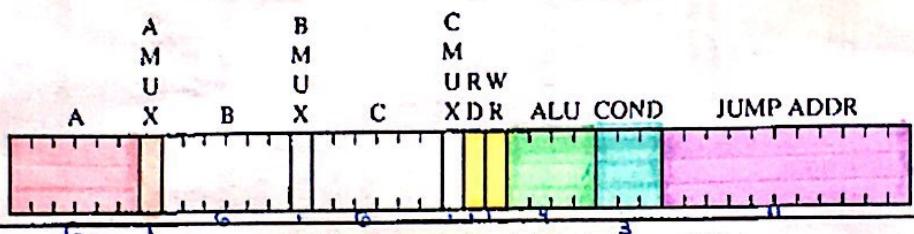
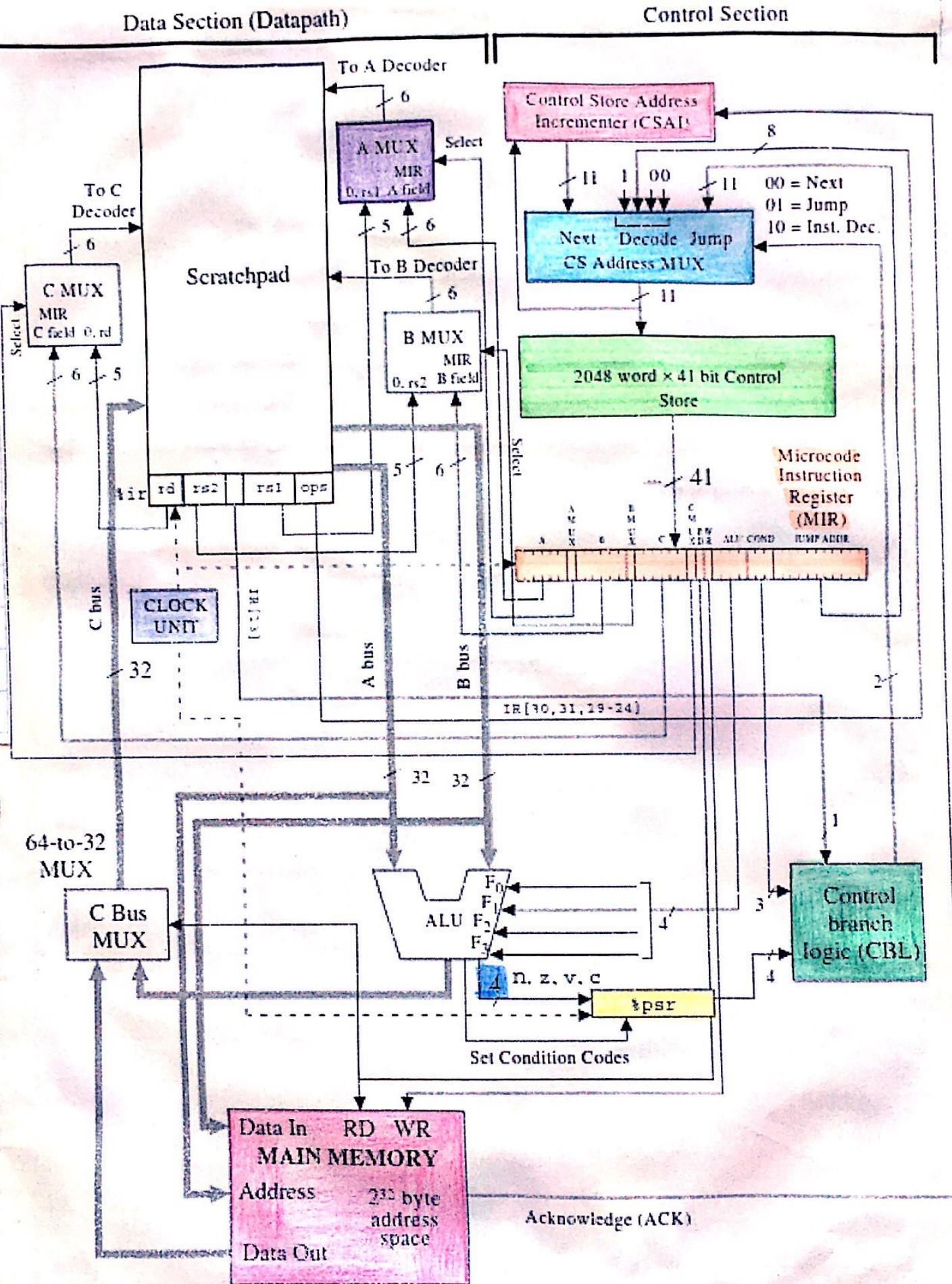


Figura 6.11 • El formato de la palabra de microcódigo.

Los campos A, B y C determinan qual registro del traeferro de datos se coloca en el bus (A y B) o C en cual registro se almacena el dato transferido. Los campos AMUX, BMUX y CMUX determinan si el decodificador obtiene sus entradas del MIR (MUX = 0) o desde el Campo rs1(A)/rs2(B) de %.ir o rd(C) (MUX = 1). Los bits RD y WR indican si la memoria se lee o escribe (no pueden ser 1 simultaneamente). El Campo ALU de 4 bits determina cual de las 16 operaciones se realizará. El Campo COND le indica al microcontrolador de donde rescalar la prox microinstruciòn. El Campo JUMP ADDR son 11 bits de direccionamiento para poder acceder a cualquier posicion de la memo de control.



M El propósito de los multiplexores A, B y C es decidir la fuente de donde seleccionar el registro del nivel de microarquitectura interno), tomando uno de los 32 del MIR o seleccionar un registro del nivel de usuario ISA, de los 32 posibles del IA.

El input de los decodificadores depende de si se lleva a cabo una instrucción de usuario o microinstructions. Desde el nivel de microarquitectura, los inputs provienen de la sección de control que tiene los campos A, B, YC de 6 bits del MIR (num entre 0 y 32) y un bit de AMUX/SMUX/CMUX.

Desde el nivel arquitectura las entradas provienen del mismo archivo de registro ya que la instrucción está dentro del IA. (campos IS1/IS2 o 0 en el caso de C). Si AMUX es 0 el registro

a seleccionar es el campo A del MIR, si es 1 IS1.

El output es el registro de 6 bits a seleccionar, que alimenta al decodificador correspondiente que los decodifica en 38.

CLE las microinstrucciones operan sobre un ciclo de 2 fases Fionco Positivo: cambian las

secciones maestra de los registros Σ . Fionco negativo: lo almacenado en la maestra baza a los satciano $\Sigma \rightarrow$ add, mult, csl.

Incrementador de direcciones de la memoria de control. Su propósito es incrementar en uno la instrucción. Sus entradas son la instrucción que se está ejecutando que proviene del MUX de direcciones y un bit MSB de la memo PCF. Este último es enviado por la memo PCF para reconocer que terminó la CF, hasta entonces no incrementa. Esta unidad solo funciona cuando el MUX de dirs recibe NEXT. Su output es la nueva instrucción a ejecutar que alimenta al mismo.

El MUX de direcciones de memoria de control. Su responsabilidad es seleccionar la prox microins a ejecutar y pasársela al CSAI (puede ser NEXT, JUMP o DECODE). Inputs: entrada de control de 2 bits de CBL, 11 bits del JMP del MIR (caso JUMP), 11 bits del incrementador (NEXT) y 8 bits del IR (DECODE). Si se toma NEXT, la unidad decide cual es la siguiente instrucción. Si se realiza un DECODE, la instrucción proviene del nivel ISA como fue seleccionado por el usuario, la instrucción del IR se transforma en una microinstrucción para MIR. La decodificación transforma la instrucción de 8 bits en una de 11. Finalmente, en el caso JMP los 11 bits ilegal directo del MIR.

Lógica de saltos de control (CBL) decide si la siguiente instrucción a ejecutar debe ser literalmente la siguiente en la ROM si debe decodificarse del IR o un JMP al JMP Addr del MIR. La unidad es necesaria debido a las distintas instrucciones de bifurcación (be, bneg). Los inputs son: el campo COND de 3 bits del MIR que indica desde donde tomar la prox. instrucción: NEXT de la ROM, DECODE de IR o JUMP de MIR. Otra entrada es el bit 13 del IR que sirve como condición para ejecutar la prox microinstrucción. Por ultimo, tiene como entrada los 4 bits de condición (n,z,c,v) del %PSR. Con todas las entradas, el CBL resuelve cual de los 3 escenarios toma lugar.

PSR Contiene información sobre el estado del procesador incluida la info sobre los resultados de las operaciones aritméticas. Inputs: CLOCK UNIT (los FF de los registros necesitan el CLK), los cuatro flags establecidos por la ALU y una serial de la misma que le indica al registro que actualice los bits. No realiza ningún cálculo y de sus 32 bits los flags n, z, c y v corresponden a los bits 23 a 20. Su output son los códigos de condición actualizados que alimentan al CBL que es la unidad encargada de determinar la siguiente instrucción teniendo en cuenta los códigos de condición.

indica la cantidad de datos serían representados en la linea

que es el numero de bits.

La memoria principal. solo las operaciones de acceso a la memoria permitidas pueden cargar o almacenar un valor en o desde los registros (maquina de almacen de carga). Toda función aritmética o lógica debe operar sobre valores contenidos en registros.

Los inputs son los bits de RD y WR que indican si se está realizando una operación de lectura o escritura. Los buses A y B también sirven como entrada para la memoria ya que contienen los operandos de la instrucción y un acceso posible es que un operando sea una dirección de memo. Los outputs son los datos recuperados de memoria que van al MUX del bus C (que de ahí vuelve al bus C a ser almacenados en el registro destino) y un bit que le indica al MUX de decir que se completo el acceso a memoria y el CS1 puede continuar su ciclo.

Temporización

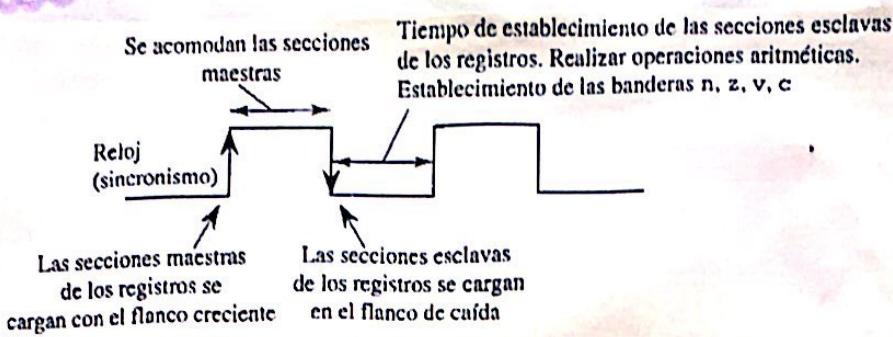


Figura 6.14 • Relaciones de tiempo para el funcionamiento de los registros.

la microarquitectura opera en un ciclo de 2 fases

~~F~~ Cambian 10.5 ~~Y~~ la info del
secciones maes- maestro se
tro de los registros Transfiere al esclavo en el estado bajo
se llevan a cabo las fun-
ciones de ALU, MUX y CBL

Microinstrucciones

Dirección	Sentencias operativas	Comentario
0:	R[ir] \leftarrow AND(R[pc], R[pc]); READ;	/ Leer una instrucción de ARC desde memoria principal
1:	DECODE; / sethi	/ Salto (256 posibilidades) condicionado al código de operación
1152:	R[rd] \leftarrow LSHIFT10(ir); GOTO 2047; / call	/ Copiar el campo imm22 en el registro de destino
1280:	R[15] \leftarrow AND(R[pc], R[pc]); 1281: R[temp0] \leftarrow ADD(R[ir], R[ir]); 1282: R[temp0] \leftarrow ADD(R[temp0], R[temp0]); 1283: R[pc] \leftarrow ADD(R[pc], R[temp0]); GOTO 0; / addcc	/ Guardar R[pc] en %15 / Desplazar el campo disp30 a izquierda / Desplazar nuevamente / Salto a subrutina
1600:	IF R[IR[13]] THEN GOTO 1602; 1601: R[rd] \leftarrow ADDCC(R[rs1], R[rs2]); GOTO 2047;	/ El segundo operando origen está en modo inmediato? / Resolver ADDCC sobre registros origen
1602:	R[temp0] \leftarrow SEXT13(R[ir]); 1603: R[rd] \leftarrow ADDCC(R[rs1], R[temp0]); GOTO 2047; / andcc	/ Obtener el campo simm13, con extensión de signo / Resolver ADDCC sobre operandos origen en registro/simm13
1604:	IF R[IR[13]] THEN GOTO 1605; 1605: R[rd] \leftarrow ANDCC(R[rs1], R[rs2]); GOTO 2047;	/ El segundo operando origen está en modo inmediato? / Resolver ANDCC sobre registros origen
1606:	R[temp0] \leftarrow SIMM13(R[ir]); 1607: R[rd] \leftarrow ANDCC(R[rs1], R[temp0]); GOTO 2047; / orcc	/ Obtener el campo simm13 / Resolver ANDCC sobre operandos origen en registro/simm13
1608:	IF R[IR[13]] THEN GOTO 1610; 1609: R[rd] \leftarrow NORCC(R[rs1], R[rs2]); GOTO 2047;	/ El segundo operando origen está en modo inmediato? / Resolver NORCC sobre registros origen
1610:	R[temp0] \leftarrow SIMM13(R[ir]); 1611: R[rd] \leftarrow ORCC(R[rs1], R[temp0]); GOTO 2047; / orcc	/ Obtener el campo simm13 / Resolver ORCC sobre operandos origen en registro/simm13
1624:	IF R[IR[13]] THEN GOTO 1626; 1625: R[rd] \leftarrow NORCC(R[rs1], R[rs2]); GOTO 2047;	/ El segundo operando origen está en modo inmediato? / Resolver ORNCC sobre registros origen
1626:	R[temp0] \leftarrow SIMM13(R[ir]); 1627: R[rd] \leftarrow NORCC(R[rs1], R[temp0]); GOTO 2047; / srl	/ Obtener el campo simm13 / Resolver NORCC sobre operandos origen en registro/simm13
1688:	IF R[IR[13]] THEN GOTO 1690; 1689: R[rd] \leftarrow SRL(R[rs1], R[rs2]); GOTO 2047;	/ El segundo operando origen está en modo inmediato? / Resolver SRL sobre registros origen
1690:	R[temp0] \leftarrow SIMM13(R[ir]); 1691: R[rd] \leftarrow SRL(R[rs1], R[temp0]); GOTO 2047; / jmp1	/ Obtener el campo simm13 / Resolver SRL sobre operandos origen en registro/simm13
1760:	IF R[IR[13]] THEN GOTO 1762; 1761: R[pc] \leftarrow ADD(R[rs1], R[rs2]); GOTO 0;	/ El segundo operando origen está en modo inmediato? / Resolver ADD sobre operandos origen en registro/simm13

Figura 6.15 • Microprograma parcial de ARC. Las micropalabras se muestran en secuencia lógica (no numérica).

1762: R[temp0] ← SEXTI3(R[ir]);	/ Obtener el campo simm13, con extensión de signo
1763: R[pc] ← ADD(R[rs1], R[temp0]);	/ Resolver ADD sobre operandos origen en registro/simm13
GOTO 0;	
/ id	
1792: R[temp0] ← ADD(R[rs1], R[rs2]);	/ Calcular dirección origen
IF R[IR[13]] THEN GOTO 1794;	
1793: R[rd] ← AND(R[temp0], R[temp0]);	/ Colocar dirección origen sobre el bus A
READ; GOTO 2047;	
1794: R[temp0] ← SEXTI3(R[ir]);	/ Obtener el campo simm13 para la dirección origen
1795: R[temp0] ← ADD(R[rs1], R[temp0]);	/ Calcular dirección de origen
GOTO 1793;	
/ st	
1808: R[temp0] ← ADD(R[rs1], R[rs2]);	/ Calcular dirección de destino
IF R[IR[13]] THEN GOTO 1810;	
1809: R[ir] ← RSHIFT5(R[ir]); GOTO 40;	/ Mover el campo rd hacia la posición del campo r42
40: R[ir] ← RSHIFT5(R[ir]);	/ desplazándolo 25 bits a la derecha.
41: R[ir] ← RSHIFT5(R[ir]);	
42: R[ir] ← RSHIFT5(R[ir]);	
43: R[ir] ← RSHIFT5(R[ir]);	
44: R[0] ← AND(R[temp0], R[re2]);	/ Colocar la dirección de destino sobre el bus A y
WRITE; GOTO 2047;	/ el operando sobre el bus B
1810: R[temp0] ← SEXTI3(R[ir]);	/ Obtener el campo simm13 para calcular la dirección de destino
1811: R[temp0] ← ADD(R[rs1], R[temp0]);	/ Calcular dirección de destino
GOTO 1809;	
/ Instrucciones de salto: be, be, bcs, bvs, bneg	
1098: GOTO 2;	/ Árbol de decodificación para saltos
2: R[temp0] ← LSHIFT10(R[ir]);	/ Extender el signo de los 22 bits menos
3: R[temp0] ← RSHIFT5(R[temp0]);	/ significativos de Rtemp0, desplazando
4: R[temp0] ← RSHIFT5(R[temp0]);	/ primero 10 bits a izquierda, luego 10 bits a derecha
5: R[ir] ← RSHIFT5(R[ir]);	/ La extensión de signo se realiza a través de RSHIFT5
6: R[ir] ← RSHIFT5(R[ir]);	/ Mover el campo COND a IR[13] utilizando tres veces RSHIFT5
7: R[ir] ← RSHIFT5(R[ir]);	/ La extensión de signo no afecta la operación.
8: IP R[IR[13]] THEN GOTO 12;	/ La instrucción es ba?
R[ir] ← ADD(R[ir], R[ir]);	
9: IF R[IR[13]] THEN GOTO 13;	/ La instrucción no es he?
R[ir] ← ADD(R[ir], R[ir]);	
10: IF Z THEN GOTG 12;	/ Ejecutar be
R[ir] ← ADD(R[ir], R[ir]);	
11: GOTO 2047;	/ El salto indicado por bc no se ejecuta
12: R[pc] ← ADD(R[pc], R[temp0]);	/ Ejecutar el salto
GOTO 0;	
13: IF R[IR[13]] THEN GOTO 16;	/ Es bcs?
R[ir] ← ADD(R[ir], R[ir]);	
14: IF C THEN GOTG 12;	/ Ejecutar bcs
15: GOTO 2047;	/ El salto indicado por bcs no se ejecuta
16: IF R[IR[13]] THEN GOTO 19;	/ Es bvs?
17: IF N THEN GOTO 12;	/ Ejecutar bneg
18: GOTO 2047;	/ El salto indicado por bneg no se ejecuta
19: IF V THEN GOTO 12;	/ Ejecutar bvs
20: GOTO 2047;	/ El salto indicado por bvs no se ejecuta
2047: R[pc] ← INCPC(R[pc]); GOTO 0;	/ Incrementar %pc y empezar de nuevo

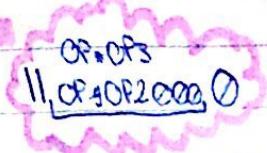
Figura 6.15 • Continuación.

Microprograma

- Es el encargado de interconectar hardware y software.
 - Se lo suele llamar firmware y su propósito es interpretar el conjunto de instrucciones visibles al usuario.
 - La memoria de control posee 2048 palabras y cada sentencia del microprograma está antecedida por un número decimal que indica su dirección en memoria.
 - La primera tarea es cargar la instrucción apuntada por %PC (inizializado al fin del microprograma) al %IH.
 - Despues se sigue con la prox sentencia, a menos que se detecte GOTO o DECODE.
 - Cada linea del microprograma corresponde a una palabra en la memoria de control; se puede ensamblar linea a linea en un único paso.

Como escribir cualquier microinstrucción (menos jmp, sech, call, ld, st, branch)

Decode N



N: if $R[13]$ then goto N+2;

N+1: $R[Rd] \leftarrow \text{INST}(R[rs1], R[rs2]);$

GOTO 2047

N+2: $R[Tempo] \leftarrow \underline{\quad} (R[ir]);$

ADDCC → SEXT13

resto → SiMM13

N+3: $R[Rd] \leftarrow \text{INST}(R[rs1], R[Tempo]);$

GOTO 2047;

Traps e interrupciones.

• Traps: procedimiento automático de llamada generado por el hardware como consecuencia de una condición excepcional que se produce en la ejecución del programa (ej: instrucción ilegal).

• Interrupción: situación que se produce luego de algún evento circuital excepcional (por ej: accionamiento de una tecla del teclado).

Nanoprogramación.

Se puede ahorrar espacio de memoria de microprograma colocando una copia de cada palabra de microcódigo en un elemento de nanalmacenamiento, usando la memoria de microprograma como índice a la memoria de nano código.

En lugar de acceder a solo a microcódigo, ahora se debe acceder al microcódigo y luego al nanocódigo. La máquina funcionará más lentamente pero ocupando menos espacio.