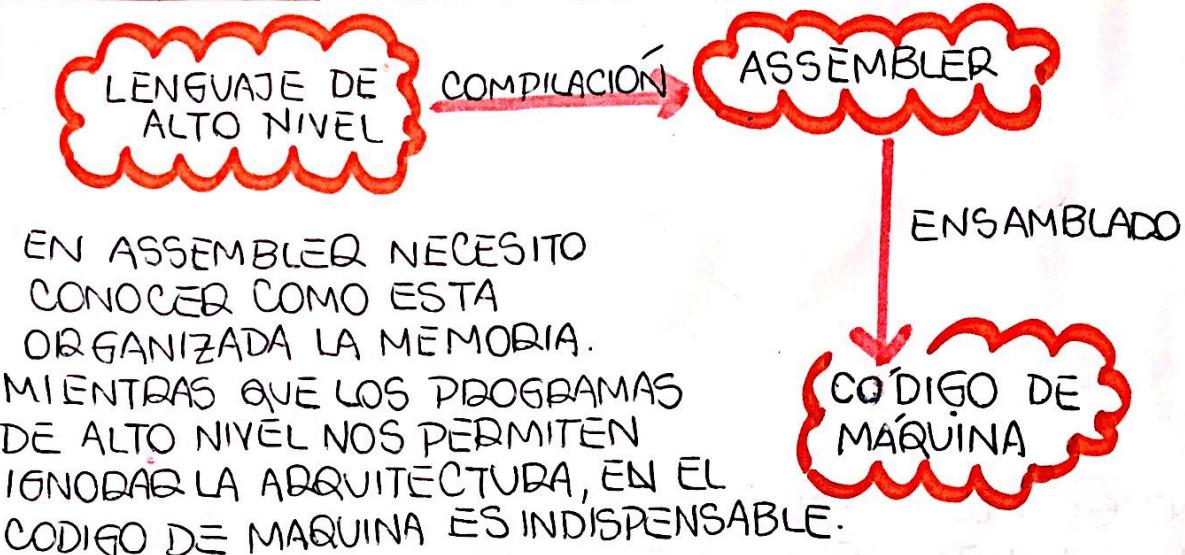


# ESTRUCTURA DEL COMPUTADOR

CAP. 4

ARQUITECTURA  
DE PROGRAMACIÓN



DESPUES DE ESTOS 3 PASOS, SE ALMACENA EN DISCO.

EL SIST OPERATIVO LO PASA DEL DISCO A LA MEMORIA PRINCIPAL DURANTE LA EJECUCIÓN CADA INSTRUCCIÓN SE CARGA EN LA CPU DESDE LA MEMORIA (UNA POR VEZ) CON CUALQUIER OTRO DATO NECESARIO PARA EJECUTARLA. LA SALIDA DEL PROGRAMA SE COLOCA EN UN DISPOSITIVO (UNIDAD DE DISCO, PANTALLA DE VIDEO). TODO ESTO ES REGULADO POR LA UNIDAD DE CONTROL. TODA ESTA COMUNICACIÓN ES ATRAVÉS DE BUSES.

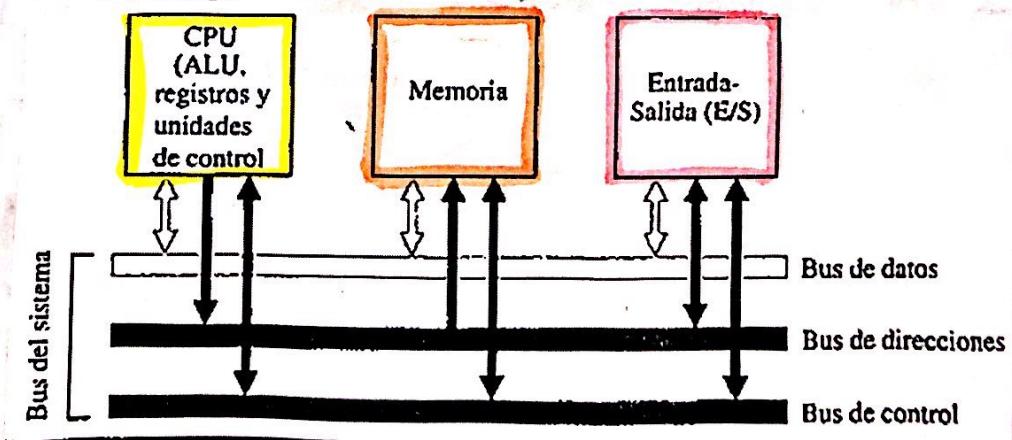
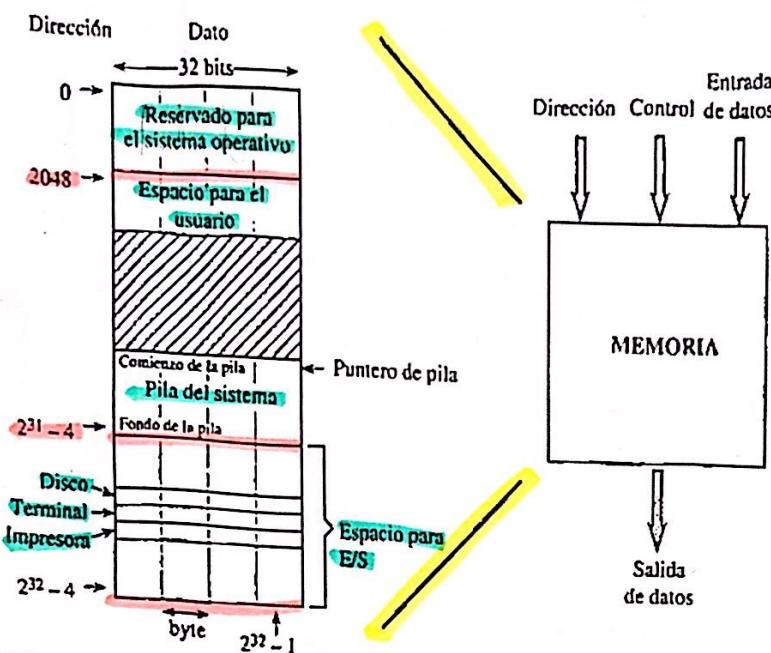


Figura 4.1 • El modelo de un sistema de computación con estructura de bus.

# MEMORIA RAM



- CONJUNTO DE REGISTROS DIRECCIONADOS QUE CADA UNO ALMACENA 1 BYTE (8 BITS)

- LA MEMORIA ES DISEÑABLE POR BYTES

Figura 4.4 • Un mapa de memoria para un ejemplo de arquitectura. (No dibujado en escala.)

CADA BLOQUE DE LA TABLA ES IDENTIFICADO POR SU DIRECCIÓN. LA DIRECCIÓN TIENE 32 BITS, ENTONCES HAY  $2^{32} - 1$  (EMPIEZA EN CERO).

CADA BLOQUE GUARDA CADA 4 BYTES (32B).

LOS PROCESADORES EN GENERAL PUEDEN ACCEDER SIMULTÁNEAMENTE A 1, 2, 3 O + BYTES.

$$X = 000F4F9DH$$

QUIERO ALMACENAR  $\downarrow$   
X EN MEMORIA.

EXPRESA  
HEXADECIMAL

9D
4F
0F
00

EL QUE SE ELIJA DEPENDE DEL PROCESADOR.

00
0F
4F
9D

**BIG  
ENDIAN**

BYTE MENOS SIGNIFICATIVO

**LITTLE  
INDIAN**

BYTE MENOS SIGNIFICATIVO EN LA DIRECCIÓN MÁS BAJA.

• EL TAMAÑO DEL ESPACIO DIRECCIONABLE ES ESPECÍFICO DEL PROCESADOR. EL MAPA ES ESPECÍFICO DEL TIPO DE COMPUTADORA (NO DEL CPU).

DOS SISTEMAS BASADOS EN EL MISMO PROCESADOR NO NECESARIAMENTE TIENEN EL MISMO MAPA.

# 2

# CPU

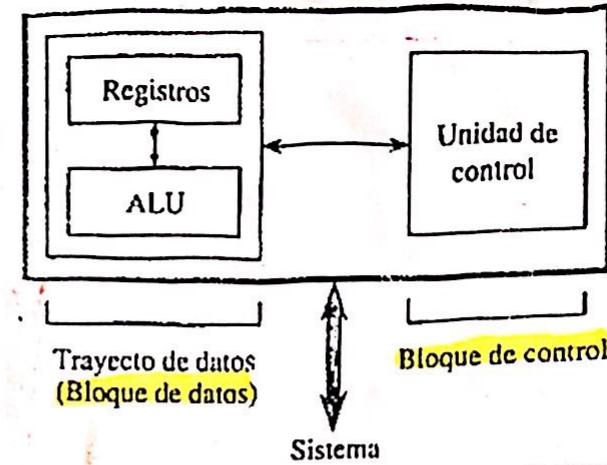


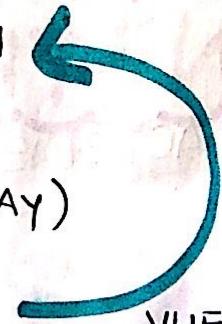
Figura 4.5 • La CPU vista en alto nivel.

## UNIDAD DE CONTROL

ES RESPONSABLE DE LA EJECUCIÓN DE LAS INSTRUCCIONES DEL PROGRAMA, LAS QUE ESTAN EN LA MEMORIA RAM.

LOS PASOS QUE LLEVA A CABO DURANTE LA EJECUCIÓN DE UN PROGRAMA SON:

1. BUSCAR LA PRÓXIMA INSTRUCCIÓN
2. DECODIFICARLA
3. BUSQUEDA DE OPERANDOS (SI HAY)
4. EJECUTARLA Y ALMACENAR RESULTADO



VUELVE  
AL PASO 1.

INTERFAZ  
(2 REGISTROS)



CONTIENE LA DIRECCIÓN DE LA INSTRUCCIÓN EN EJECUCIÓN



LA INSTRUCCIÓN ES ALMACENADA AQUÍ

## SECCIÓN DE DATOS

ESTA FORMADA POR ALU Y REGISTROS  
EL CONJUNTO DE REGISTROS PUEDE VERSE COMO UNA PEQUEÑA MEMORIA, RÁPIDA.

ESTOS SE UTILIZAN PARA EL ALMACENAMIENTO DURANTE LAS OPERACIONES

LA ALU IMPLEMENTA UNA CANTIDAD DE OPERACIONES BINARIAS Y UNITARIAS: (OR AND NOR)

# COMPUTADORA ARC SU ARQUITECTURA

- PROCESADOR SPARC
- ES UNA COMPUTADORA CON UN CONJUNTO REDUCIDO DE INSTRUCCIONES **RISC.**
- **MEMORIA:**

CONTIENE PALABRAS DE 32 BITS DIRECCIONABLES POR BYTES. UTILIZA SU ALMACENAMIENTO **BIG ENDIAN**. ANTES.

## INSTRUCCIONES ARC

CARACTERÍSTICAS DEL CPU:

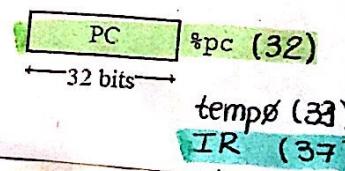
Registros ARC - Murdocca

Register 00	\$r0 [= 0]
Register 01	\$r1
Register 02	\$r2
Register 03	\$r3
Register 04	\$r4
Register 05	\$r5
Register 06	\$r6
Register 07	\$r7
Register 08	\$r8
Register 09	\$r9
Register 10	\$r10
PSR	\$psr

Register 11	\$r11
Register 12	\$r12
Register 13	\$r13
Register 14	\$r14 [%sp]
Register 15	\$r15 [link]
Register 16	\$r16
Register 17	\$r17
Register 18	\$r18
Register 19	\$r19
Register 20	\$r20
Register 21	\$r21

Register 22	\$r22
Register 23	\$r23
Register 24	\$r24
Register 25	\$r25
Register 26	\$r26
Register 27	\$r27
Register 28	\$r28
Register 29	\$r29
Register 30	\$r30
Register 31	\$r31

- 32 REGISTROS DE USO GRAL. DE 32 BITS
- TIENE UN PC Y IR.



REGISTRO DE ESTADO DEL PROCESADOR

E, N, C, V (LOS FLAGS DE CONDICIÓN)

AQC ES UNA MAQUINA DE CARGA Y DESCARGA.  
LAS UNICAS INSTRUCCIONES PERMITIDAS PARA EL ACCESO  
A MEMORIA SON:

- CARGAR UN VALOR DE MEMORIA EN REGISTRO
- ALMACENAR EN MEMORIA EL CONTENIDO DE ALGUN REGISTRO.

Formato del set de instrucciones de ARC - Murdocca

Mnemonic	Meaning
ld	Load a register from memory
st	Store a register into memory
sethi	Load the 22 most significant bits of a register
andcc	Bitwise logical AND
orcc	Bitwise logical OR
orncc	Bitwise logical NOR
srl	Shift right (logical)
addcc	Add
call	Call subroutine
jmpl	Jump and link (return from subroutine call)
be	Branch if equal
bneg	Branch if negative
bcs	Branch on carry
bvs	Branch on overflow
ba	Branch always

} MEMORIA

} LÓGICAS

} ARITMETICA

} CONTROL

- LOS SUFIJOS 'CC' ESPECIFICAN QUE LUEGO DE REALIZAR LA OPERACIÓN DEBEN ACTUALIZARSE LOS CODIGOS DE CONDICIÓN DE PSR.

## DESPLAZAMIENTO

**SRL** DESPLAZA UN REGISTRO HACIA LA DERECHA Y CARGA CEROS EN LOS BITS MAS SIGNIFICATIVOS

**SRA** DESPLAZA UN REGISTRO HACIA LA DERECHA Y ALMACENA UNA COPIA DEL BIT MAS SIGNIFICATIVO DEL CONTENIDO ORIGINAL EN LOS BITS VACIOS.  
(EXTENSIÓN DE SIGNO)

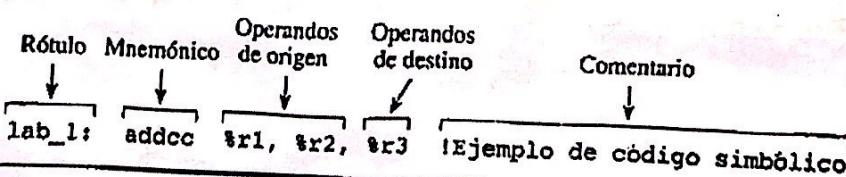


Figura 4.8 • Formato de una sentencia en el lenguaje simbólico SPARC (también ARC).

DISTINGUE MAYÚSCULA DE MINÚSCULA

# FORMATOS DE INSTRUCCIÓN

IDENTIFICA  
CUAL DE  
LAS 5 ES

EL FORMATO DE INSTRUCCIÓN DEFINE LA MANERA EN QUE EL ENSAMBLADOR DISTRIBUYE LOS CAMPOS DE UNA INSTRUCCIÓN Y LA FORMA QUE LO INTERPRETA LA UNIDAD DE CONTROL

**DISP30** UN DESPLAZAMIENTO DE 30 BITS QUE SE UTILIZAN PARA DETERMINAR LA DIRECCION DE LA RUTINA INVOCADA.

IMM22

DISP22

RD

IDENTIFICA UN REGISTRO ORIGEN EN CASO DE ST O UNO DE DESTINO PARA LAS INSTRUCCIONES RESTANTES.

RS1

IDENTIFICA EL PRIMER REGISTRO OBIVEN:

RS2

IDENTIFICA EL SEGUNDO REGISTRO ORIGEN.

SIMM13

ES UN VALOR DE 13 BITS, SE EXTIENDE CON SIGNO  
PARA EL SEGUNDO REGISTRO ORIGEN

# DIRECTIVAS

Directiva	Forma de uso	Función o significado
.equ	.equ X #10	Asignar a X el valor (10) <sub>16</sub>
.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.

TAMBIÉN EXISTEN DIRECTIVAS QUÉ NO SON CÓDIGOS DE OPERACIÓN SINO INSTRUCCIONES DIRIGIDAS AL ENSAMBLADOR PARA QUE REALICE CIERTAS ACCIONES EN EL MOMENTO DEL ENSAMBLE.

## MODOS DE DIRECCIONAMIENTO

Modo de direccionamiento	Sintaxis	Significado
Inmediato	#K	K
Directo	K	M[K]
Indirecto	(K)	M[M[K]]
Indirecto a través de registro	(Rn)	M[Rn]
Indexado a través de registro	(Rm + Rn)	M[Rm + Rn]
Basado en registro	(Rm + X)	M[Rm + X]
Indexado basado en registro	(RM + Rn + X)	M[RM + Rn + X]

Tabla 4.1 • Modos de direccionamiento.

# SUBRUTINAS

UNA SUBRUTINA (FUNCIÓN O PROCEDIMIENTO) ES UNA SECUENCIA DE INSTRUCCIONES A LA QUE SE INVOCÁ COMO SI FUERA UNA ÚNICA INSTRUCCIÓN.

CUANDO UN PROGRAMA LLAMA A UNA SUBRUTINA, SE TRANSFIERE EL CONTROL DEL PROGRAMA A LA SUBRUTINA. ESTA EJECUTA LA SECUENCIA DE INSTRUCCIONES Y LUEGO VUELVE A LA POSICIÓN SIGUIENTE A LA QUE GENERÓ EL LLAMADO.

## CONVENCIONES DE LLAMADA

①

UNA DE ELLAS COLOCA LOS ARGUMENTOS EN REGISTROS.

②

ZONA DE TRANSFERENCIA DE DATOS.  
LA DIRECCIÓN DE LA ZONA DE T. SE ENTREGA A LA RUTINA INVOCADA EN UN REGISTRO PREDETERMINADO.

③

USO DE LA PILA.

LA IDEA GENERAL ES QUE LA RUTINA INVOCANTE COLOCA TODOS SUS ARGUMENTOS EN UNA PILA DEL TIPO "LIFO". LA RUTINA INVOCADA EXTRAÉ DE LA PILA LOS ARGUMENTOS TRANSFERIDOS, A LA VEZ QUE COLOCA EN LA MISMA QUALQUIER VALOR QUE DEBA RETORNAR. A SU VEZ EL PROGRAMA DESCARTA ESOS VALORES Y CONTINUA CON LA EJECUCIÓN.

**CAP. 5****LOS LENGUAJES Y  
LA MÁQUINA****COMPILACIÓN**

— ES EL PROCESO DE TRADUCCIÓN DE UN PROGRAMA ESCRITO EN LENGUAJE DE ALTO NIVEL A LENGUAJE ENSAMBLADOR.

EL PROCESO DE PASAR DE ENSAMBLADO A MAQUINA TIENE UNA RELACION UNO A UNO.

**PASOS DE COMPILACIÓN :**

- ANÁLISIS LEXICOGRÁFICO, RECONOCER DENTRO DEL TEXTO DEL PROGRAMA LOS SIMBOLOS BASICOS DEL LENGUAJE.
- ANALISIS SINTACTICO, ANALIZAR LOS SIMBOLOS PARA RECONOCER LA ESTRUCTURA DE PROGRAMACIÓN
- ANALISIS DE NOMBRES, ASOCIAR LOS NOMBRES CON VARIABLES PARTICULARES Y LUEGO CON POSICIONES DE MEMORIA.
- ANALISIS DE TIPO, DETERMINAR EL TIPO DE TODOS LOS DATOS REQUERIDOS
- ASIGNACIÓN DE ACCIONES Y GENERACIÓN DE CÓDIGO, ASOCIAR LAS SENTENCIAS DE PROGRAMA CON LA SECUENCIA APROPIADA DEL ENSAMBLADOR.
- EXISTEN ACCIONES ADICIONALES QUE DEBEN SER DESVELTAS POR EL COMPILEADOR, COMO LA ASIGNACIÓN DE VARIABLES A REGISTROS, SU CONTROL DE USO Y OPTIMIZACIÓN.

ANALISIS  
SEMANTICO

**ESPECIFICACIÓN DE ASIGNACIÓN DEL COMPILADOR**

SE DEBE INCLUIR EN LA ESTRUCTURA DEL COMPILADOR LA INFO. ACERCA DE LA ARQ DE PROGRAMACIÓN DEL PROCESADOR.

TAMBIÉN HAY QUE TENER EN CUENTA LAS CARACTERÍSTICAS Y LIMITACIONES DE LA MAQUINA.

## VARIABLES EN MEMORIA

### VARIABLES GLOBALES

SU DIRECCIÓN ES CONOCIDA EN EL MOMENTO DE COMPILEACIÓN.

### VARIABLES LOCALES

SE LES ASIGNA UNA POSICIÓN DE MEMORIA EN EL MOMENTO DE TRADUCCIÓN, TIEMPO DE ENSAMBLADO.

**PILA** → LAS VARIABLES QUE SE ALMACENAN EN LA PILA COBRAN VALOR CUANDO SE CREA LA PILA Y SE INVOCAN LA FUNCIÓN.

ES HABITUAL COPIAR EL CONTENIDO DE %SP A %FP (PUNTERO BASE), EL QUE SE UTILIZA PARA EL ACCESO A LAS VARIABLES DE LA PILA DURANTE LA VIDA DE LA FUNCIÓN. EL USO DE FP IMPLICA QUE EL COMPILADOR PUEDE DETERMINAR UN DESPLAZAMIENTO CONSTANTE ENTRE EL Y LO ALMACENADO EN LA PILA.

PARA ACCEDER A LAS VARIABLES DE LA PILA SE USA DIRECCIONAMIENTO BASE ( $LD[ \%FP - 12], \%R1 \rangle$ )

## DE COMPILADOR A ENSAMBLADOR

### ① MOVIMIENTO DE DATOS

### ESTRUCTURAS

EL COMPILADOR ACOMODA LA ESTRUCTURA EN MEMORIA CONSECUTIVA, PUDIENDO ACCEDER POR BASE.

### ARREGLOS

PARA CALCULAR LA DIRECCIÓN DE MEMORIA EN EL MOMENTO DE EJECUCIÓN:

$$= \text{BASE} + (\text{INDICE} - \text{COMIENZO}) \cdot \text{TAMAÑO}$$

## ② INSTRUCCIONES ARITMETICAS

SIEMPRE ES POSIBLE QUE EL COMPILADOR ENCUENTRE ALGUNA INSTRUCCIÓN ARITMÉTICA QUE DEQUIERE MAYOR CANTIDAD DE REGISTROS DE LOS QUE DISPONE. EN ESTE CASO ALMACENA TEMPORALMENTE ALGUNAS VARIABLES EN LA PILA.

## ③ CONTROL DE SECUENCIA

IMPLEMENTACIÓN DE BIFURCACIONES CONDICIONALES MÁS COMUNES.

### GO TO

SE IMPLEMENTA CON UNA INSTRUCCIÓN DE SALTO INCONDICIONAL:

### IF - ELSE

BA ROTULO  
(%R1 == %R2) SE USA  
BNE Y OVER.

### WHILE

BA TEST  
TRUE: ADD R3, 1, R3  
TEST: SUBCC R1, R2, R0  
BE TRUE

LO MISMO  
CON FOR  
Y DO-W

## ENSAMBLADO

ES LA TRADUCCIÓN DEL PROGRAMA EN SAMBLADO A AL LENGUAJE DE MAQUINA. ES LINEAL Y SIMPLE.

TENEMOS UN ENSAMBLADOR DE "DOS PASADAS", ESTO QUIERE DECIR QUE DECODDE DOS VECES EL TEXTO.

### PRIMER PASADA

SE DEDICA A DETERMINAR LAS DIRECCIONES DE TODOS LOS DATOS E INSTRUCCIONES DEL PROGRAMA Y A SELECCIONAR QUÉ INSTRUCCIÓN DEL LENG. DE MÁQUINA DEBE GENERARSE PARA CADA UNA DEL LENG. SIMBÓLICO. (AÚN NO LO GENERA)

ESTO SE HACE MEDIANTE EL **CONTADOR DE POSICIÓN**  
ESTE CONTADOR LLEVA EL CONTROL DE LA DIRECCIÓN DE  
LA INSTRUCCIÓN O EL DATO. SE INICIALIZA EN CERO Y SE  
INCREMENTA EN PASOS = AL TAMAÑO DE CADA INSTRUCCIÓN  
• QDG HACE QUE SE UBIQUE DONDE INDICA (.QDG 2048)  
POR LO QUE LA INST. O DATO SE ASIGNA A ESA DIREC<sup>↑</sup>

EN LA PRIMERA PASADA, TMB REALIZA CUALQUIER  
OPERACIÓN ARITMÉTICA E INSERTA LAS DEFINICIONES  
DE TODOS LOS PÓTULOS Y CTES EN UNA.

### TABLA DE SÍMBOLOS

UN PÓTULO O NOMBRE  
SÍMBOlico, QUE SE REFIERE  
A UN VALOR UTILIZADO EN  
EL ENSAMBLADO.

## SEGUNDA PASADA

SI AL FINALIZAR LA PRIMERA PASADA HAY  
PÓTULOS SIN DEFINIR SE PRODUCE UN ERROR.  
SI NO SUCEDE ESTO, LA SEGUNDA PASADA  
GENERA EL CODIGO DE MAQUINA CON TODO LO  
HECHO EN LA PASADA ANTERIOR.

## TAREAS FINALES

EL ENSAMBLADOR DEBE AGREGAR INFO  
ADICIONAL PARA EL USO DE ENLACE Y CARGA:

- ✿ NOMBRE Y TAMAÑO DE MODULO
- ✿ DIRECCIÓN DEL SÍMBOLO DE COMIENZO
- ✿ INFORMACIÓN ACERCA DE SÍMBOLOS GLOBALES Y EXTERNOS
- ✿ INFORMACIÓN DE REUBICACIÓN.

# ENLACE Y CARGA

EL ENLACE ES UN PROGRAMA QUE COMBINA PROGRAMAS EN SAMBLADOS POR SEPARADO (MODULOS OBJETO) EN UN ÚNICO PROGRAMA (MODULO DE CARGA). ESTE RESUELVE TODAS LAS REFERENCIAS GLOBALES Y EXTERNAS Y REUBICA LAS DIRECCIONES DE LOS DIFERENTES MÓDULOS. EL MODULO DE CARGA PUEDE SER CARGADO EN MEMORIA POR EL CARGADOR, PUEDE NECESITAR MODIFICAR DIREC. SI SE CARGA EN UNA DIREC. ≠ A LA DE ORIGEN USADA POR EL ENLACE.



BIBLIOTECA DE ENLACE DINAMICO QUE POSPONE EL ENLACE DE ALGUNOS COMPONENTES HASTA QUE SEAN REQUERIDOS EFECTIVAMENTE



EL PROGRAMA DE ENLACE DEBE:

① RESOLVER REFERENCIAS DE DIRECCIONES EXTERNAS

→ PARA RESOLVER ESTO EL PROGRAMA DEBE DISTINGUIR LOS NOMBRES LOCALES DE LOS GLOBALES. ESTO SE LOGRA MEDIANTE EL USO DE .GLOBAL .EXTERN

SE UTILIZA EN EL MÓDULO  
EN EL QUE SE DEFINE UN  
SÍMBOLO

→ SE UTILIZA EN CADA  
MÓDULO QUE HAGA DEF  
A AQUEL

SOLO LOS SÍMBOLOS  
DE DIRECCIONES  
PUEDES SERLO.

☺ DEUBICAR CADA MÓDULO PARA LA COMBINACIÓN MÁS APROPIADA.  
↑ (QUÉ VAN A SER ENLACADOS)

↳ PUEDE SUCEDER QUE DOS PROGRAMAS COMIENZEN CON ·ORG 2048, ESTO NO ES POSIBLE

PARA RESOLVER ESTO, EL ENSAMBLADOR DEFINE COMO **DEUBICABLES** A SIMBOLOS QUE ADMITAN QUÉ SU DIRECCIÓN SEA MODIFICADA.

TODO ES DEUBICADO CON LA MISMA INCODIMENTACIÓN.  
(LAS DIRECC. DEUBICABLES SE MODIFICAN EN EL MISMO VALOR QUE EL ORIGEN)

EL ENSAMBLADOR DETERMINA QUE MÓDULOS SON DEUBICABLES O NO.

CTES DEFINIDAS POR ·EQU  
O LOS CONTENIDOS DE VARIABLES.

- ☺ ESPECIFICAR EL SIMBOLO DE COMIENZO DEL MÓDULO DE CARGA.
- ☺ ESPECIFICAR CONTENIDO DE SEGMENTOS.

## CARGA

ES UN PROGRAMA QUE UBICA EL MÓDULO DE CARGA EN LA MEMORIA PRINCIPAL. ESTE DEBE CARGAR LOS DIVERSOS SEGMENTOS DE MEMORIA CON LOS VALORES APROPIADOS E INICIAR CERCIOS REGISTROS (COMO %SP Y %PC)

EL CARGADOR MODIFICA LAS DIRECCIONES DEUBICABLES QUE ENCUENTRA DENTRO DE UN MÓDULO DE CARGA DADO PARA PERMITIR LA COEXISTENCIA EN MEMORIA DE VARIOS PROGRAMAS EN FORMA SIMULTÁNEA.

## MACROS

[ES COMO UNA FUNCIÓN EN C  
• MACRO NOMBRE ARG1, ARG2, ...  
; :  
• END MACRO (PUEDE SER RECURSIVA)]

SE ACCDE EN TIEMPO DE ENSAMBLADO CONVIERTIENDOLA EN SU CODIGO EQUIVALENTE. EL PROCESO DE TIRAS LADAS LA MACRO A SU CODIGO → EXPANSIÓN.

## CAP 6

TRAYECTO DE DATOS  
Y CONTROL

## MICROARQUITECTURA

LA MICROARQUITECTURA ES EL CPU, ESTA CONSTITUIDA POR LA UNIDAD DE CONTROL, REGISTROS Y LA ALU. SE OCUPA DE TODO LO QUE VIMOS EN EL CAP 4.

## TRAYECTO DE DATOS



PUEDE REALIZAR UNA DE **16 OPERACIONES** SOBRE LOS BUSES A Y B Y EL RESULTADO DE 32 B SE COLOCA EN EL BUS C, A MENOS QUE QUEDA BLOQUEADA POR EL MUX COMO CONSECUENCIA DE COLOCAR EN EL UNA PALABRA PROVENIENTE DE LA RAM.

SON DE LA ALU, NO SON LAS ISA

**SR1** DESPLAZA EL CONTENIDO DE A HACIA LA DERECHA EN BITS ESPECIFICADOS EN B. SE INTRODUCEN 0 EN LOS MENOS SIGNIFICATIVOS

**LSHIFT2 / LSHIFT10**

DESPLAZA A LA IZQ EL CONTENIDO DE A EN 2 O 10 POSICIONES. COMPLETA CON 0 LOS MENOS SIGNIFICATIVOS.

**SIMM13** RECUPERA LOS 13B MENOS SIGNIFICATIVOS DE A Y LOS OTROS 19 LES PONE CERO.

**SEXT13** EXTENSION DE SIGNO DE LOS 13B MENOS SIGNIFICATIVOS

**INC** +1 AL VALOR EN A. SIMM13

**INCPC** +4 AL VALOR EN A. A 32B EN LOS 5B LIBRES EL SIGNO. AUMENTAR EL CONTADOR.

$F_3\ F_2\ F_1\ F_0$	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

**RSHIFT5** DESPLAZA EN 5 LUGARES HACIA

EXTENSION DE DERECHA DE A, COPIANDO

A 32B EN LOS 5B LIBRES EL SIGNO.

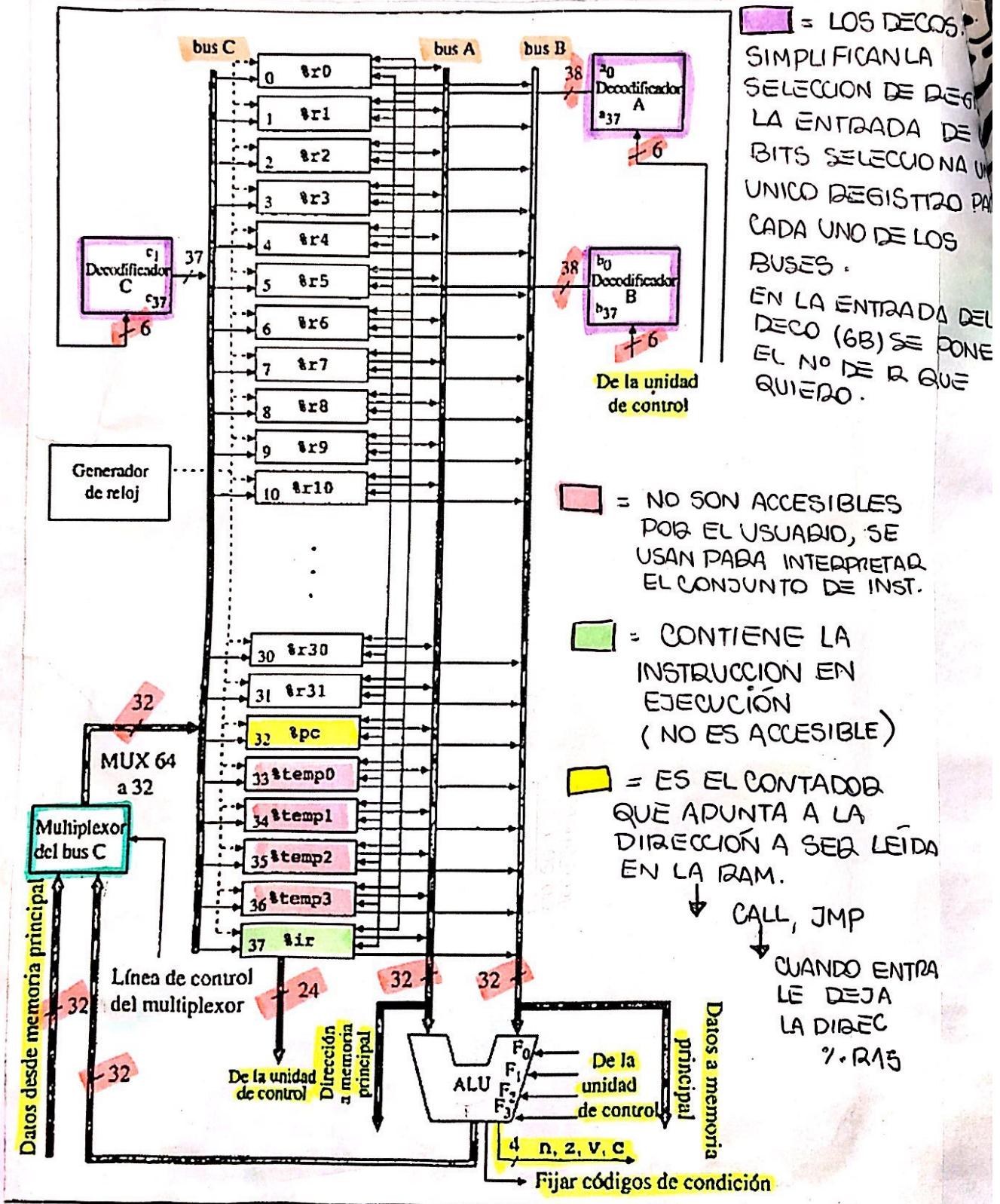


Figura 6.3 • Trayecto de datos en ARC.

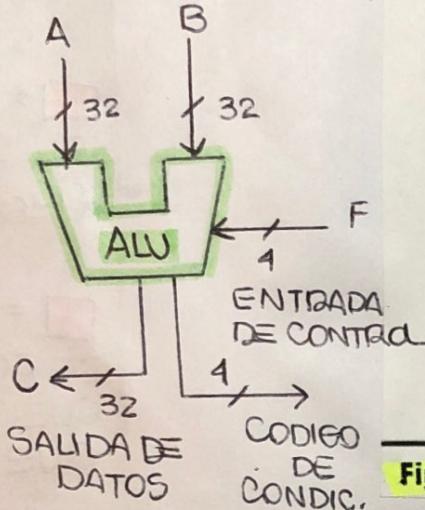
LA ALU GENERA LOS CÓDIGOS DE CONDICIÓN. EN EL CASO DE TENER FLAGs SE GENERA UNA SEÑAL SCC QUE LE INDICA A A %PSR QUE DEBE ACTUALIZAR SUS CÓDIGOS DE CONDICIÓN.

# IMPLEMENTACIÓN DE LA ALU

PUEDE IMPLEMENTARSE DE DISTINTAS FORMAS

## TABLA DE BUSQUEDA

DOS ENTRADAS DE DATOS (A Y B)



$F_3$	$F_2$	$F_1$	$F_0$	Arrastre de entrada	$a_i$	$b_i$	$z_i$	Arrastre de salida
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	1	1	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	1	1	0
0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	1	1	0
0	0	0	1	0	1	0	1	0
0	0	0	1	0	1	1	1	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	1	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	1	1	0
0	0	0	1</					

SE INDICA EN LA ENTRADA DE DESPLAZAMIENTO A DERECHA  
EL PROCESO SE REPITE HASTA OBSERVAR EL BIT SA<sub>4</sub> EN EL  
NIVEL SUPERIOR. DONDE NO HAY NADA SE COLOCAN CEDOS  
CON ESTO PUEDO GENERAR CUALQUIER DESPLAZAMIENTO

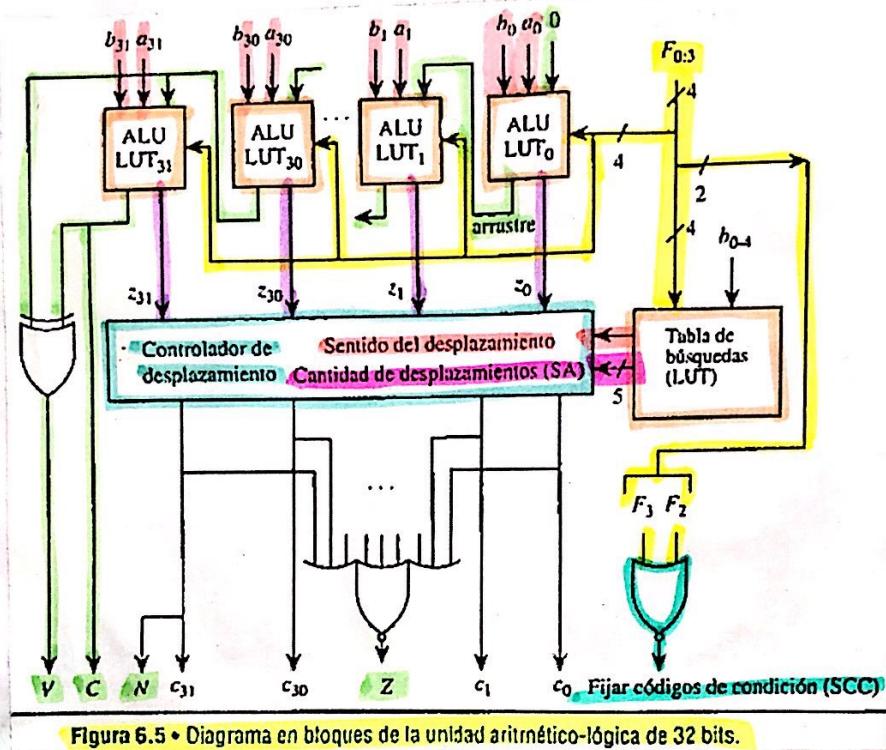
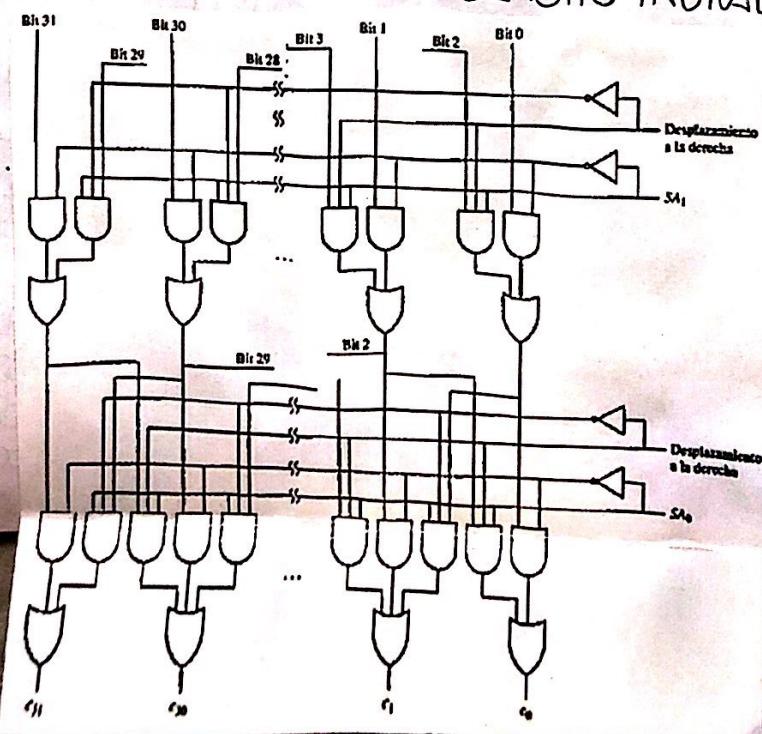


Figura 6.5 • Diagrama en bloques de la unidad aritmético-lógica de 32 bits.

- = LOS PRIMEROS BITS DE CADA BUS, YA QUE ES BIT A BIT
- = ESTOS 4 BITS INDICAN QUE OPERACIÓN (DEL 0 AL 15)
- = CABO Y (POB SI HACES UNA SUMA POR EJEMPLO)
- = DESPLAZA LA CANTIDAD DE BITS INDICADOS



- = SABER SI LA OP TERMINA EN CERO.

Figura 6.6 • Esquema circuital del control de desplazamiento.

# REGISTROS

3

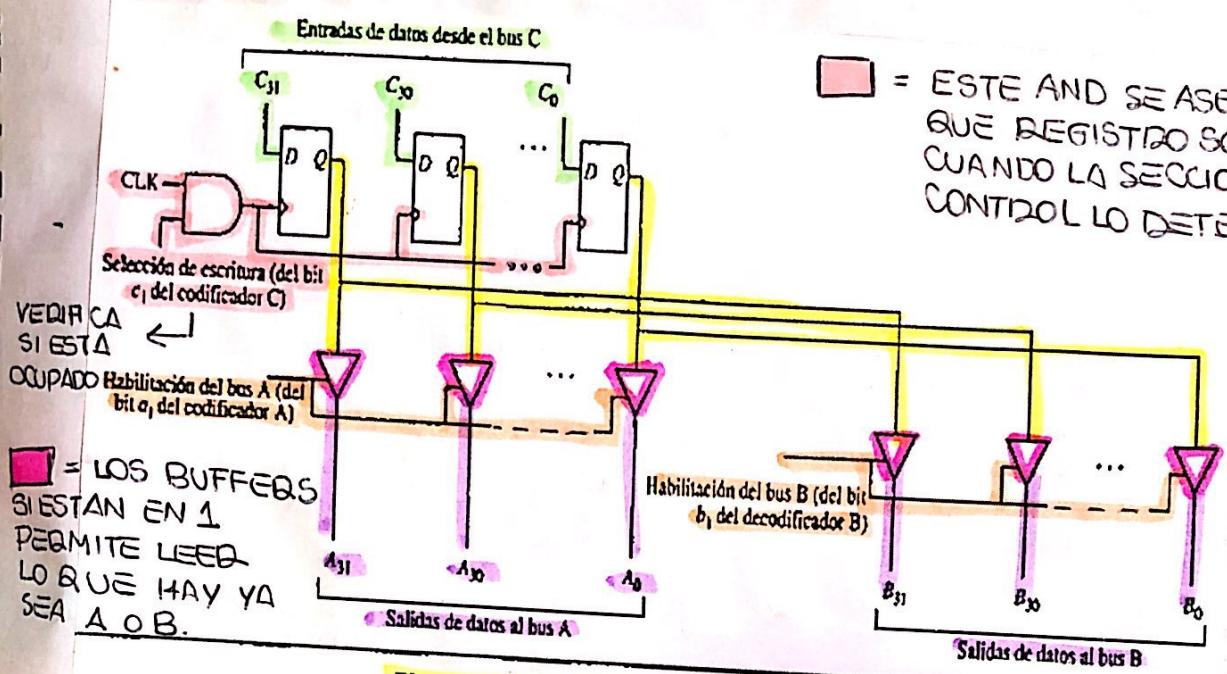


Figura 6.8 • Diseño del registro 32 bits.

- TODOS SON IMPLEMENTADOS CON FLIP FLOPS D, ACTIVADOS POR FLANCO NEGATIVO.
- TODOS ADOPTAN UN FORMATO SIMILAR.
- TODOS TIENEN 32 BITS DE ANCHO.
- EL REGISTRO %R0 SIEMPRE CONTIENE 0 Y NO PUEDE SER MODIFICADO. ESTE NO TIENE ENTRADA DESDE EL BUS C NI DESDE EL DECODIFICADOR, ENTONCES NO REQUIERE FLIPFLOPS.
- EL REGISTRO %R1 TIENE SALIDAS ADICIONALES QUE CORRESPONDEN A RD, BS1, BS2, OP, OP2, OP3 Y EL BIT13. LA UNIDAD DE CONTROL UTILIZA ESTAS SALIDAS EN LA INTERPRETACION DE LAS INSTRUCCIONES.
- EL CONTADOR %PC SOLO PUEDE CONTENER VALORES QUE SEAN MULTIPLOS DE 4, POR LO QUE LOS 2 BITS MENOS SIGNIFICATIVOS PUEDEN SER 0 SIEMPRE.

# SECCIÓN DE CONTROL

LA FIGURA ILUSTRA EL TRAYECTO DE DATOS, LA UNIDAD DE CONTROL Y LAS CONEXIONES ENTRE ELLAS.

## MEMORIA DE LECTURA (ROM)

TIENE 2048 PALABRAS DE 41 BITS. ESTA CONTIENE LINEAS QUE DEBEN CONTROLARSE PARA IMPLEMENTAR CADA INSTRUCCIÓN. SE CONSIDERA MEMORIA DE CONTROL.

CADA PALABRA DE 41B ES UNA **MICROINSTRUCCIÓN**

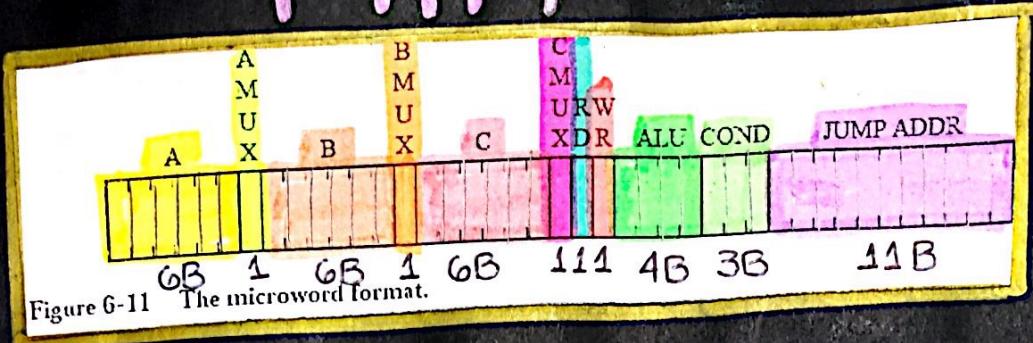
MIR

ES UN DÉGISTRO QUE CONTROLA LA EJECUCIÓN DE LAS MICROINSTRUCCIONES, JUNTO CON EL % PSR (R DE ESTADO) Y UN MECANISMO QUE PERMITE DETERMINAR CUAL ES LA PRÓXIMA MICROINSTRUCCIÓN FORMADO POR CBL (UNIDAD DE SALTOS DE CONTROL) Y EL MUX DE DIRECCIONES DE MEMORIA DE CONTROL.

INICIALMENTE SE COLOCA LA MICROINSTRUCCIÓN DE LA DIRECCIÓN O DE LA ROM EN EL MIR PARA SER EJECUTADA.

LUEGO SE SELECCIONA LA SIGUIENTE MICROINSTRUCCIÓN DE ALGUNA DE LAS ENTRADAS NEXT DECODE O JUMP SOBRE LA BASE DE LO QUE DICE EL CAMPO COD DEL MIR.

# MiR



## = CAMPO A

DETERMINA CUÁL ES EL REGISTRO CUYO CONTENIDO DEBE COLOCARSE SOBRE EL BUS A. (PARÁMETRO DE LA OPERACIÓN)

## = CAMPO AMUX

SELECCIONA SI EL DECODIF. A OBTIENE SU ENTRADA DESDE EL CAMPO A ( $AMUX=0$ ) O DESDE EL CAMPO RS1 DE %IR ( $AMUX=1$ )

## = CAMPO C

DETERMINA EN QUÉ REGISTRO SE ALMACENARÁ EL DATO TRANSFERIDO A TRAVÉS DEL BUS C.

## = CAMPO CMUX

ELIGE SI LA ENTRADA DEL DECODIFICADOR C SE OBTIENE DESDE EL CAMPO C ( $CMUX=0$ ) O DESDE EL RD DEL %IR ( $CMUX=1$ )

## = CAMPO ALU

DETERMINA CUÁL DE LAS 16 OPERACIONES A-L SE VA A EJECUTAR, NO HAY FORMA DE APAGARLA POR LO TANTO SI SE DESEA LEER SE EJECUTA UNA OPERACIÓN QUE NO MODIFIQUE NADA.

## = CAMPO B

DETERMINA CUÁL DE LOS REGISTROS DEBE COLOCAR SU CONTENIDO EN EL BUS B.

## = CAMPO BMUX

DETERMINA SI EL DECODIF. B OBTIENE SUS ENTRADAS DESDE EL CAMPO B ( $BMUX=0$ ) O DESDE EL CAMPO RS2 DEL %IR ( $BMUX=1$ )

## = CAMPO RD

DETERMINA SI SE TIENE QUE LEER ( $RD=1$ ) O NO ( $RD=0$ ) EN MEMORIA. TMB CONTROLA EL MUXC DE 64 A 32 B DEL BUS C Y DETERMINA SI EL BUS C SE CARGA DESDE LA MEMORIA ( $RD=1$ ) O DESDE LA ALU ( $RD=0$ )

## = WR

DETERMINA SI SE DEBE ESCRIBIR ( $WR=1$ ) O NO ( $WR=0$ ) EN MEMORIA. RD Y WR NO PUEDEN SER 1 A LA VEZ PERO SI 0.

## = CAMPO JUMP ADDR

SON 11 BITS DE DIRRECIONAMIENTO PARA PODER ACCEDER A CUALQUIERA DE LAS POSICIONES DE LA MEMORIA DE CONTROL

## DABA AMBAS OPERACIONES

LA DIREC DE MEMORIA SE TOMA DEL BUS A, EL DATO QUE SE INGRESA EN MEMORIA BUS B, Y LA SALIDA DESDE LA MEMORIA SE COLOCA EN BUS C

DEL OTRO LADO →

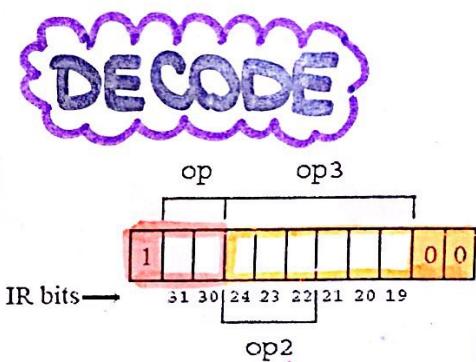
█ = CAMPO COND

HACE QUE EL MICROCONTROLADOR DESCATE LA MICROINSTRUCCIÓN SIGUIENTE. YA SEA DESDE

- LA POSICIÓN SIGUIENTE EN ROM.
- POSICIÓN INDICADA EN EL JUMP ADDR.
- LO ALMACENADO EN %IR.

EL REGISTRO %IR CONTIENE LA INSTRUCCIÓN QUE ESTÁ SIENDO EJECUTADA. LOS CAMPOS OP, OP2 Y OP3 SON LOS DE ESA INSTRUCCIÓN.

$C_2 \ C_1 \ C_0$	Operation
0 0 0	Use NEXT ADDR
0 0 1	Use JUMP ADDR if $n = 1$
0 1 0	Use JUMP ADDR if $z = 1$
0 1 1	Use JUMP ADDR if $v = 1$
1 0 0	Use JUMP ADDR if $c = 1$
1 0 1	Use JUMP ADDR if $IR[13] = 1$
1 1 0	Use JUMP ADDR
1 1 1	DECODE



█ = 000

NO HAY SALTO. SE UTILIZA LA ENTRADA NEXT DEL MUX DE DIRECCIONES DE LA ROM.

ESTA ENTRADA VA AL CSAI Y INCRAEMENTA EN 1 LA SALIDA DEL MUX.



█ = 111

DURANTE LA DECODIFICACIÓN SE ADOPTA ESTE VALOR. EN ESTE CASO LA MICROI NO SE TOMA NI DESDE LA ENTRADA NEXT DEL MUX NI DE JUMP SINO DESDE LA COMBINACIÓN DE 11B

AGREGADO DE UN 1 A LA IZQ DE LOS BITS 30 Y 31 DE %IR + AGREGAR CEDOS A LA DERECHA DE LOS BITS 19-24 DE %IR. ↓ DECODE.

█ = 001, 010, 011, 100, 101

HAY SALTO A LA POSICIÓN DE LA ROM INDICADA EN EL JUMP ADDR DE ACUERDO CON EL VALOR DE LOS FLAGS O DEL BIT 13 DEL %IR

█ = 110

SALTO INCONDICIONAL. (GO TO)

# MICROARQUITECTURA

= MUXS, DECIDEN DE DÓNDE VA A PROVENIR SU SALIDA SI DEL CAMPO O DEL Y, IR.

= CONJUNTO DE REGISTROS (0-31), PC, (TEMP 0-3), IR.

= ES EL MUX DEL BUS, DECIDE SI TOMADA LA INFO PE LA ALU O DE LA RAM.

= ES EL CLOCK, LAS MICROINSTRUCIONES OPERAN SOBRE UN CICLO DE 2 FASES. VAMBIEN LAS SE CUIONES MASTERS F POSITIVO = DE LOS REGISTROS, F NEG = LO ALMACENADO EN LO MAESTRO BAJA A LO ESTANDAR → ALU, MUX, CBL

- DEL MIR → MIA PERMITE ESCOGERLA EN LA RAM SI ES LO EXECUTADO
- DEL MIR → MUX SELECT DETERMINA QUE ENTRADA PASA POR LOS MUXS (ABC)
- MIA → AMUX 6B BS1 → AMUX 5B + 0 SON LAS ENTRADAS.
- MIA → CMUX 6B BD → CMUX 0+5B
- MIA → BMUX 6B BS2 → BMUX 0+5B
- CSAI → CS MUX → ENTRADA CAMPOS OPS → MUX - 8B
- DEL MIR → DAM MIR → CBUS MUX PARA PODER LEEB SE DIRIGE A LADAM Y CONTROLA LA SALIDA DEL MUX PORQ EN EL CASO DE SER ELEGIDA LA SALIDA DECODE, SE USAN ESOLOS CAMPOS.
- DEL MIA → ALU - 4B LA MICROINSTRUCCION DESTINADA A RESOLVER LA ALU.

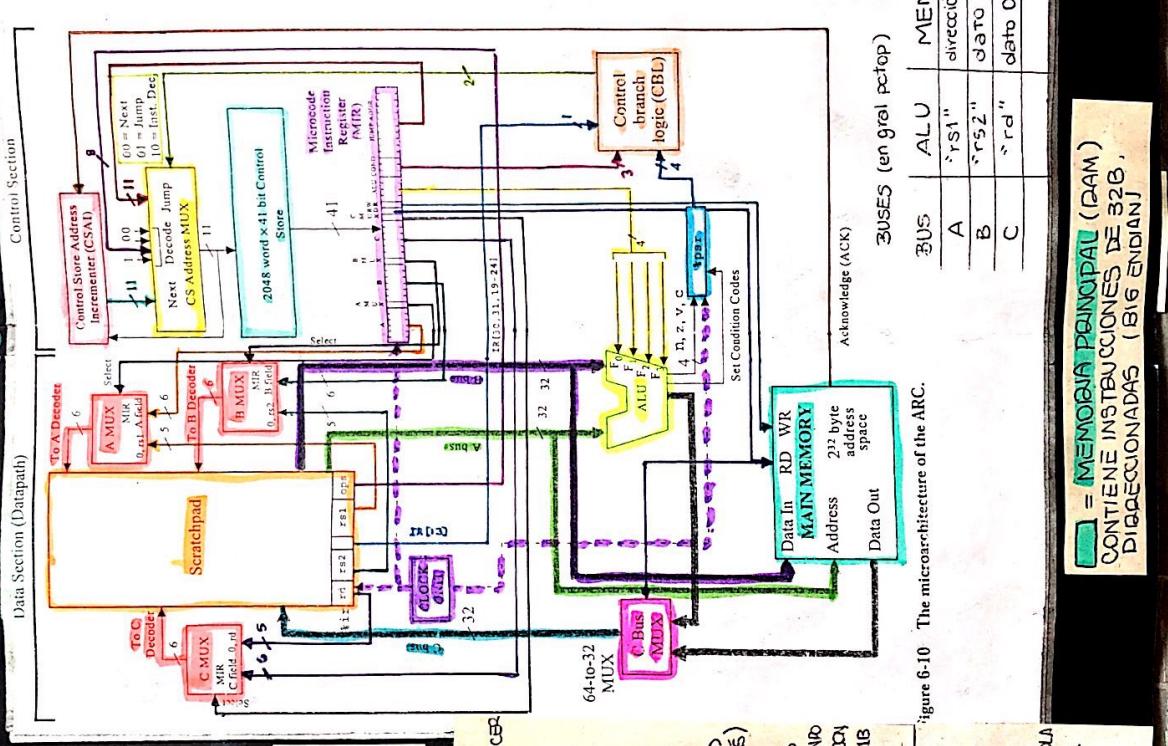


figure 6-10 The microarchitecture of the ARC.

- = ES EL CSAI, QUE INCORPORA EN 1 LA DIRECCIÓN DE MEMORIA CUANDO EL CAMPO COND INDICA COO.
- = ES EL MUX DE LA RAM, DECIDE QUE DIRECCIÓN DE MEMORIA LE ENVIA A LA RAM. PUEDE PROVENIR DE NEXT, DECODE O JUMP.
- = ES LA MEMORIA DE CONTROL (ROM) QUE CONTIENE 2048 MICROINSTRUCCIONES DE 41 B
- = ES EL MIR, CONTROLA LA EJECUCIÓN DE LAS MICROINSTRUCCIONES JUNTO CON EL %PSR, CBL Y EL MUX DE CONTROL
- = ES EL CBL, SE ENCARGA DE HACER LOS SALTOS CONDICIONALES OBTENIDOS EN EL CAMPO COND
- = %PSR, CONTIENE LOS DATOS DE LOS FLAGS DE CONDICIÓN
- = ES LA ALU, DA LA SALIDA DE LOS BUSES A Y B Y SE COLOCA EN C.

= MEMORIA PRINCIPAL (RAM) CONTIENE INSTRUCCIONES DE 32B, DIRECCIONADAS (BIG ENDIAN)

# TEMPORIZACIÓN

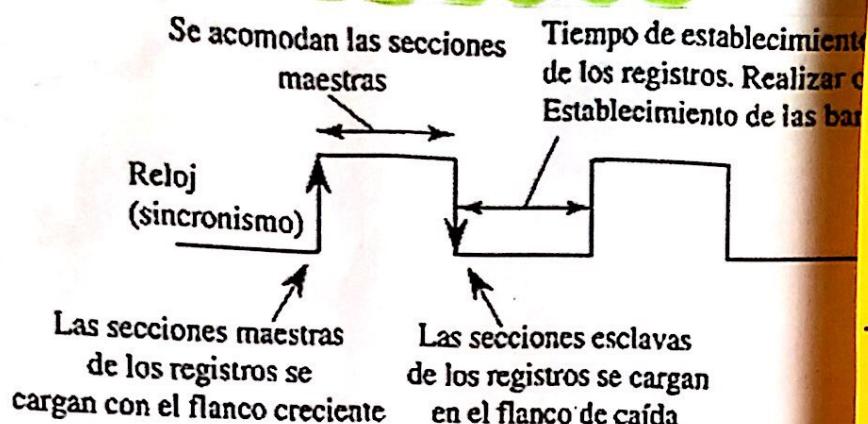


Figura 6.14 • Relaciones de tiempo para el funcionamiento de

LA MICROARQUITECTURA OPERA EN UN CICLO DE DOS FASES:

- ↑ CAMBIAN LAS SECCIONES MAESTRAS DE LOS REGISTROS.
- ↓ LA INFO EN LA PARTE MAESTRA SE TRANFIERE A LA PARTE ESCLAVA EN EL ESTADO BAJO SE LLEVA A CABO LAS FUNCIONES DE ALU, MUX Y CBL

## MICROINSTRUCCIONES

### Dirección Sentencias operativas

```

D: R(ir) ← AND(R(pc), R(pc)); READ;
1: DECODE;
    / sethi
1152: R(rd) ← LSHIFT10(ir); GOTO 2047;
    / call
1280: R(15) ← AND(R(pc), R(pc));
1281: R(temp0) ← ADD(R(ir), R(ir));
1282: R(temp0) ← ADD(R(temp0), R(temp0));
1283: R(pc) ← ADD(R(pc), R(temp0));
    GOTO 0;
    / addcc
1600: IF R(IR[13]) THEN GOTO 1602;
1601: R(rd) ← ADDCC(R[r01], R[r02]);
    GOTO 2047;
1602: R(temp0) ← SEXT13(R(ir));
1603: R(rd) ← ADDCC(R[r01], R[temp0]);
    GOTO 2047;
    / andcc
1604: IF R(IR[13]) THEN GOTO 1606;
1605: R(rd) ← ANDCC(R[r01], R[r02]);
    GOTO 2047;
1606: R(temp0) ← SIMM13(R(ir));
1607: R(rd) ← ANDCC(R[r01], R[temp0]);
    GOTO 2047;
    / orcc
1608: IF R(IR[13]) THEN GOTO 1610;
1609: R(rd) ← ORCC(R[r01], R[r02]);
    GOTO 2047;
1610: R(temp0) ← SIMM13(R(ir));
1611: R(rd) ← ORCC(R[r01], R[temp0]);
    GOTO 2047;
    / orncc
1624: IF R(IR[13]) THEN GOTO 1626;
1625: R(rdi) ← NORCC(R[r01], R[r02]);

```

### Comentario

/ Leer una instrucción de ARC desde memoria principal	
/ Salto (256 posibilidades) condicionado al código de operación	
1152: / Copiar el campo imm22 en el registro de destino	
1280: / Guardar R_pc en R_15	
1281: / Desplazar el campo disp30 a izquierda	
1282: / Desplazar nuevamente	
1283: / Salto a rutina	
1600: / El segundo operando origen está en modo inmediato?	
1601: / Resolver ADDCC sobre registros origen	
1602: / Obtener el campo simm13, con extensión de signo	
1603: / Resolver ADDCC sobre operandos origen en registro/simm13	
1604: / El segundo operando origen está en modo inmediato?	
1605: / Resolver ANDCC sobre registros origen	
1606: / Obtener el campo simm13	
1607: / Resolver ANDCC sobre operandos origen en registro/simm13	
1608: / El segundo operando origen está en modo inmediato?	
1609: / Resolver NORCC sobre registros origen	
1610: / Obtener el campo simm13	
1611: / Resolver ORCC sobre operandos origen en registro/simm13	
1624: / El segundo operando origen está en modo inmediato?	
1625: / Resolver ORNCC sobre registros origen	

```

1762: R[temp0] ← SEXT13(R[ir]);           / Obtener el campo simm13, con extensión de signo
1763: R[pc] ← ADD(R[rs1], R[temp0]);      / Resolver ADI sobre operandos origen en registros/simm13
    GOTO 0;
    / ld
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] ← SEXT13(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 rs2
    40: R[ir] ← RSHIFT5(R[ir]);
    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[rs2]);        / Colocar la dirección de destino sobre el bus A y
    WRITE; GOTO 2047;                         / el operando sobre el bus B
1810: R[temp0] ← SEXT13(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: ba, be, bcs, bvs, bneg
1088: 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 %temp0, 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 RSHIFTS
6: R[ir] ← RSHIFT5(R[ir]);                  / Mover el campo COND a IR[13] utilizando tres veces RSHIFTS
7: R[ir] ← RSHIFT5(R[ir]);                  / La extensión de signo no afecta la operación
8: IF R[IR[13]] THEN GOTO 12;                / ¿La instrucción es ba?
    R[ir] ← ADD(R[ir], R[ir]);
9: IF R[IR[13]] TBEN GOTO 13;              / ¿La instrucción no es be?
    R[ir] ← ADD(R[ir], R[ir]);
10: IF Z TBEN GOTO 12;                     / Ejecutar be
    R[ir] ← ADD(R[ir], R[ir]);
11: GOTO 2047;                            / El salto indicado por be no se ejecuta
12: R[pc] ← ADD(R[pc], R[temp0]);          / Ejecutar el salto
    GOTO 0;
13: IF R[IR[13]] TREN GOTO 16;             / ¿Es bcs?
    R[ir] ← ADD(R[ir], R[ir]);
14: IF C THEN GOTO 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

```

= ESTAS 3 LINEAS SIEMPRE SE EJECUTAN EN TODAS LAS MICRO INSTRUCCIONES