C

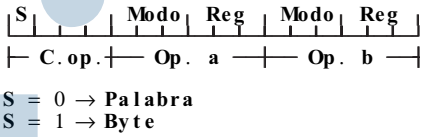
JUEGO DE INSTRUCCIONES DE LA SERIE DE COMPUTADORES PDP–11

C.1.- Resumen de los modos de direccionamiento

Sobre los registros de uso general			Sobre el contador de programa	
Código	Nombre	Notación	Nombre	Notación
0	Directo por registro	Rn		
1	Indirecto por registro	(Rn) ó @Rn		
2	Autoincremental	(Rn)+	Inmediato	#Constante
3	Autoincremental indirecto	@(Rn)+	Absoluto	@#Dirección
4	Autodecremental	-(Rn)		
5	Autodecremental indirecto	@-(Rn)		
6	Indexado	X(Rn)	Relativo	Dirección
7	Indexado indirecto	@X(Rn)	Relativo indirecto	@Dirección

C.2.- Instrucciones de dos operandos

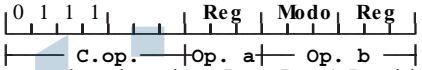
Nemónico	C.op.	Operación
MOV[B] a,b	S1	a → b
CMP[B] a,b	S2	a-b → tmp
BIT[B] a,b	S3	a∧b → tmp
BIC[B] a,b	S4	$\bar{a}$∧b → b
BIS[B] a,b	S5	a∨b → b
ADD a,b	06	a+b → b
SUB a,b	16	b-a → b



S = 0 → **Palabra**
S = 1 → **Byte**

C.3.- Instrucciones de dos operandos con un operando en registro

Nemónico	C. op.	Operación
MUL b,a	070	(1) a*b → a
DIV b,a	071	(2) a/b → a
ASH b,a	072	a<<b → a
ASHC b,a	073	(3) a<<b → a
XOR a,b	074	a⊕b → b
SOB a,E	077	(4) if (--a) PC–2*d→PC



Notas:

(1) En MUL el destino tiene 32 bits y es el par de registros Reg y Reg+1. Reg debe ser par.

(2) En DIV el dividendo es el número de 32 bits dado por los registros Reg y Reg+1. Reg debe ser par.

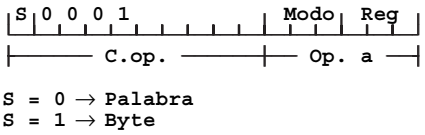
(3) En ASHC (desplaz. aritmético combinado) el operando es el par de registros Reg y Reg+1. Reg debe ser par.

(4) En SOB (resta 1 y bifurca si no es 0) los últimos 6 bits codifican un número positivo (d) que es la diferencia, en palabras, entre el PC (ya incrementado) y la etiqueta E.

C.4.- Instrucciones de un operando

Nemónico	C.op	Operación
CLR[B] a	S050	0 → a
COM[B] a	S051	$\overline{a}$ → a
INC[B] a	S052	a++
DEC[B] a	S053	a--
NEG[B] a	S054	$\overline{a}$+1 → a
ADC[B] a	S055	a+C → a
SBC[B] a	S056	a-C → a

Nemónico	C.op	Operación
TST[B] a	S057	a → tmp
ROR[B] a	S060	Rota a la dcha. (1)
ROL[B] a	S061	Rota a la izda. (1)
ASR[B] a	S062	a>>1 → a (2)
ASL[B] a	S063	a<<1 → a (2)
SXT a	0067	N → $a_{15:0}$ (3)
SWAB a	0003	$a_{15:8}$ ↔ $a_{7:0}$ (4)



Notas:
(1) Todas las rotaciones se realizan a través del bit C (*carry*). Esto significa que el valor del bit C pasa al bit entrante y el bit saliente pasa a C.
(2) Todos los desplazamientos dejan el bit saliente en C.
(3) SXT (signo extendido) copia el bit N en todos los bits de a.
(4) La instrucción SWAB (*swab bytes*, intercambio de bytes) se sale del formato ya que el bit 11 es 0. Esta instrucción intercambia los dos bytes del operando a.

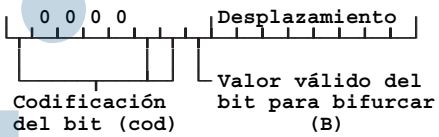
C.5.- Instrucciones de salto y subrutina

Nemónico	Codificación	Operación
JMP a	0001MR (2)	&a → PC
JSR R,b	004RMR' (3)	&b → tmp; R → -(SP); PC → R; tmp → PC;
RTS R	00020R	R → PC; (SP)+ → R;
RTI	000002	(SP)+ → PC; (SP)+ → PSW;
RTT	000002	(SP)+ → PC; (SP)+ → PSW;

Notas:
(1) La codificación de estas instrucciones se refiere a la palabra de instrucción completa y está en octal.
(2) En la instrucción JMP la *dirección* de destino a está especificada en la forma general por M y R (M≠0).
(3) En la instrucción JSR la *dirección* de destino b está especificada en la forma general por M y R'(M≠0).

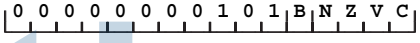
C.6.- Instrucciones de bifurcación

Cod.	Bit analizado	Nem. B=0	Nem. B=1
000	Ninguno	-	BR
001	Z	BNE	BEQ
100	N	BPL	BMI
110	V	BVC	BVS
111	C	BCC	BCS
010	N⊕V	BGE	BLT
011	Z+(N⊕V)	BGT	BLE
101	C+Z	BHI	BLOS



Notas:
(1) Las instrucciones BCC y BCS también se reconocen por los nemónicos BHIS y BLO respectivamente.
(2) Las instrucciones BHIS (*branch if higher or same*), BLO (*branch if lower or same*), BHI (*branch if higher*) Y BLOS (*branch if lower or same*) se usan para realizar comparaciones de números sin signo mientras que BGE (*branch if greater or equal*), BLT (*branch if less*), BGT (*branch if greater*) y BLE (*branch if less or equal*) se usan para comparar números en complemento a 2.
(3) Las instrucciones de bifurcación actúan así: if (bit analizado==B) PC + 2*desplazamiento → PC.
(4) La sintaxis de estas instrucciones es: C.op. a donde a es una etiqueta que representa la dirección de la instucción de destino, es decir, el nuevo valor del contador de programa.

C.7.- Instrucciones de manipulación de los bits de condición



Estas instrucciones tienen los nemónicos CLx (*clear x*, poner a 0 x) o SEx (*set x*, poner a 1 x) siendo x uno de los cuatro bits de condición (N, Z V o C). En CLx el bit B del formato vale 0 y en SEx vale 1. En el formato de la figura se pone a 1 el bit que representa al de condición afectado. También existen los nemónicos CCC (*clear condition codes*, poner a 0 todos los bis de condición) y SCC (*set condition codes*, poner a 1 todos los bits de condición) que afectan a los 4 bits simultáneamente.

C.8.- Instrucciones sin operandos

Nemónico	Codificación	Nombre
HALT	000000	Parada del procesador
WAIT	000001	Esperar a interrupción
NOP	000240	No operación

Notas:
(1) La codificación de estas instrucciones se refiere a la palabra de instrucción completa y está en octal.

D

JUEGO DE INSTRUCCIONES DE LOS COMPUTADORES VAX

D.1.- Resumen de los modos de direccionamiento

Sobre los registros de uso general			Sobre el contador de programa	
Cód.	Nombre	Notación	Nombre	Notación
0-3	Literal	#Constante		
4	Indexado	Indice[Rn]		
5	Directo por registro	Rn		
6	Indirecto por registro	(Rn) ó @Rn		
7	Autodecremental	-(Rn)		
8	Autoincremental	(Rn)+	Inmediato	#Constante
9	Autoincremental indirecto	@(Rn)+	Absoluto	@#Dirección
A,C,E	Desplazamiento (B, W o L)	X(Rn)	Relativo	Dirección
B,D,F	Desplazamiento (B, W o L) indirecto	@X(Rn)	Relativo indirecto	@Dirección

D.2.- Juego de instrucciones en orden alfabético

Nemónico	C.op.	Operandos	Descripción	N	Z	V	C
ACBB	9D	lim.b, sum.b, índice.b, desp.w	Suma, compara y salta (byte) (1)	*	*	*	-
ACBD	6F	lim.d, sum.d, índice.d, desp.w	Suma, compara y salta (D_float) (1)	*	*	*	-
ACBF	4F	lim.f, sum.f, índice.f, desp.w	Suma, compara y salta (F_float) (1)	*	*	*	-
ACBG	4FFD	lim.g, sum.g, índice.g, desp.w	Suma, compara y salta (G_float) (1)	*	*	*	-
ACBH	6FFD	lim.h, sum.h, índice.h, desp.w	Suma, compara y salta (H_float) (1)	*	*	*	-
ACBL	F1	lim.l, sum.l, índice.l, desp.w	Suma, compara y salta (doble palabra) (1)	*	*	*	-
ACBW	3D	lim.w, sum.w, índice.w, desp.w	Suma, compara y salta (palabra) (1)	*	*	*	-
ADAWI	58	sum.w, suma.w	Suma *aligned word interlocked* (2)	*	*	*	*
ADDB2	80	sum.b,suma.b	sum+suma → suma	*	*	*	*
ADDB3	81	sum1.b,sum2.b,suma.b	sum1+sum2 → suma	*	*	*	*
ADDD2	60	sum.d, suma.d	sum+suma → suma	*	*	*	0
ADDD3	61	sum1.d, sum2.d, suma.d	sum1+sum2 → suma	*	*	*	0
ADDF2	40	sum.f, suma.f	sum+suma → suma	*	*	*	0
ADDF3	41	sum1.f, sum2.f, suma.f	sum1+sum2 → suma	*	*	*	0
ADDG2	40FD	sum.g, suma.g	sum+suma → suma	*	*	*	0
ADDG3	41FD	sum1.g, sum2.g, suma.g	sum1+sum2 → suma	*	*	*	0
ADDH2	60FD	sum.h, suma.h	sum+suma → suma	*	*	*	0

Nemónico	C.op.	Operandos	Descripción	N	Z	V	C
ADDH3	61FD	sum1.h, sum2.h, suma.h	sum1+sum2 → suma	*	*	*	0
ADDL2	C0	sum.l,suma.l	sum+suma → suma	*	*	*	*
ADDL3	C1	sum1.l,sum2.l,suma.l	sum1+sum2 → suma	*	*	*	*
ADDP4	20	ndig_sum.w,sum_dir.b, ndig_suma.w,suma_dir.b	Suma BCD, 4 operandos sum+suma → suma (3)	*	*	*	0
ADDP6	21	ndig_sum1.w, sum1_dir.b, ndig_sum2.w, sum2_dir.b, ndig_suma.w, suma_dir.b	Suma BCD, 6 operandos sum1+sum2 → suma (4)	*	*	*	0
ADDW2	A0	sum.w,suma.w	sum+suma → suma	*	*	*	*
ADDW3	A1	sum1.w, sum2.w, suma.w	sum1+sum2 → suma	*	*	*	*
ADWC	D8	sum.l,suma.l	sum+suma+C → suma	*	*	*	*
AOBLEQ	F3	lim.l, índ.l, desp.b	índ++; bifurca si índ<=lim	*	*	*	-
AOBLSS	F2	lim.l, índ.l, desp.b	índ++; bifurca si índ<lim	*	*	*	-
ASHL	78	numlug.b, fnt.l, dst.l	Desplazamiento aritmét. (5)	*	*	*	0
ASHP	F8	numlug.b, ndig_fnt.w, fnt_dir.b, redon.b, ndig_dst.w, dst_dir.b	Desplazamiento aritmético y redondeo de BCD (3) (5)	*	*	*	0
ASHQ	79	numlug.b, fnt.q, dst.q	Desplazamiento aritmético	*	*	*	0
BBC	E1	pos.l, bas.b, desp.b	Analiza el bit dado por pos y base, bifurca si es 0	-	-	-	-
BBCC	E5	pos.l, base.b, desp.b	Analiza el bit dado por pos y base, bifurca si es 0. Pone el bit a 0.	-	-	-	-
BBCCI	E7	pos.l, base.b, desp.b	Analiza el bit dado por pos y base, bifurca si es 0. Pone ese bit a 0 y lo bloquea. (2)	-	-	-	-
BBCS	E3	pos.l, base.b, desp.b	Analiza el bit dado por pos y base, bifurca si es 0. Pone ese bit a 1.	-	-	-	-
BBS	E0	pos.l, base.b, desp.b	Analiza el bit dado por pos y base, bifurca si es 1	-	-	-	-
BBSC	E4	pos.l, base.b, desp.b	Analiza el bit dado por pos y base, bifurca si es 1. Pone ese bit a 0.	-	-	-	-
BBSS	E2	pos.l, base.b, desp.b	Analiza el bit dado por pos y base, bifurca si es 1. Pone ese bit a 1.	-	-	-	-
BBSSI	E6	pos.l, base.b, desp.b	Analiza el bit dado por pos y base, bifurca si es 1. Pone ese bit a 1 y lo bloquea. (2)	-	-	-	-
BCC	1E	desp.b	Salta si C=0	-	-	-	-
BCS	1F	desp.b	Salta si C=1	-	-	-	-
BEQL	13	desp.b	Salta si igual (Z=1) (con signo)	-	-	-	-
BEQLU	13	desp.b	Salta si igual (Z=1) (sin signo)	-	-	-	-
BGEQ	18	desp.b	Salta si mayor o igual (N=0) (con signo)	-	-	-	-
BGEQU	1E	desp.b	Salta si mayor o igual (C=0) (sin signo)	-	-	-	-
BGTR	14	desp.b	Salta si mayor (N∨Z=0) (con signo)	-	-	-	-
BGTRU	1A	desp.b	Salta si mayor (C∨Z=0) (sin signo)	-	-	-	-
BICB2	8A	masc.b, dst.b	¬masc∧dst → dst	*	*	0	-
BICB3	8B	masc.b, fnt.b, dst.b	¬masc∧fnt → dst	*	*	0	-
BICL2	CA	masc.l, dst.l	¬masc∧dst → dst	*	*	0	-
BICL3	CB	masc.l, fnt.l, dst.l	¬masc∧fnt → dst	*	*	0	-
BICPSW	B9	masc.w	¬masc∧PSW → PSW	*	*	*	*
BICW2	AA	masc.w, dst.w	¬masc∧dst → dst	*	*	0	-
BICW3	AB	masc.w, fnt.w, dst.w	¬masc∧fnt → dst	*	*	0	-
BISB2	88	masc.b, dst.b	masc∨dst → dst	*	*	0	-
BISB3	89	masc.b, fnt.b, dst.b	masc∨ftn → dst	*	*	0	-
BISL2	C8	masc.l, dst.l	masc∨dst → dst	*	*	0	-
BISL3	C9	masc.l, fnt.l, dst.l	masc∨fnt → dst	*	*	0	-
BISPSW	B8	masc.w	masc∨PSW → PSW	*	*	*	*
BISW2	A8	masc.w, dst.w	masc∨dst → dst	*	*	0	-
BISW3	A9	masc.w, fnt.w, dst.w	masc∨fnt → dst	*	*	0	-
BITB	93	masc.b, fnt.b	masc∧fnt → tmp. Varían los *flags*	*	*	0	-
BITL	D3	masc.l, fnt.l	masc∧fnt → tmp. Varían los *flags*	*	*	0	-

Nemónico	C.op.	Operandos	Descripción	N	Z	V	C
BITW	B3	masc.w, fnt.w	masc∧fnt → tmp. Varían los *flags*	*	*	0	-
BLBC	E9	fnt.l, desp.b	Salta si bit fnt(0) es 0	-	-	-	-
BLBS	E8	fnt.l, desp.b	Salta si bit fnt(0) es 1	-	-	-	-
BLEQ	15	desp.b	Salta si menor o igual (N∨Z=1) (con signo)	-	-	-	-
BLEQU	1B	desp.b	Salta si menor o igual (C∨Z)=1 (sin signo)	-	-	-	-
BLSS	19	desp.b	Salta si menor (N=1) (con signo)	-	-	-	-
BLSSU	1F	desp.b	Salta si menor (C=1) (sin signo)	-	-	-	-
BNEQ	12	desp.b	Salta si no igual (Z=0) (con signo)	-	-	-	-
BNEQU	12	desp.b	Salta si no igual (Z=0) (sin signo)	-	-	-	-
BPT	03		Punto de ruptura (2)	0	0	0	0
BRB	11	desp.b	Salto incondicional (desplazamiento byte)	-	-	-	-
BRW	31	desp.w	Salto incondicional (desplazamiento palabra)	-	-	-	-
BSBB	10	desp.b	Salta a subrutina (desplazamiento byte)	-	-	-	-
BSBW	30	desp.w	Salta a subrutina (desplazamiento palabra)	-	-	-	-
BUGL	FDFF	mensaje.l	Aviso (*Bugcheck*)	0	0	0	0
BUGL	FEFF	mensaje.w	Aviso (*Bugcheck*)	0	0	0	0
BVC	1C	desp.b	Salta si V=0	-	-	-	-
BVS	1D	desp.b	Salta si V=1	-	-	-	-
CALLG	FA	arglist.b, dst.b	Llamada a procedimiento con una lista general de argumentos (6)	0	0	0	0
CALLS	FB	numarg.l, dst.b	Llamada a procedimiento con lista de argumentos en la pila (6)	0	0	0	0
CASEB	8F	sel.b, base.b, lim.b, desp[o].w,.., desp[lim].w	sel-base → tmp; si tmp≤lim: PC+desp[tmp] → PC si no: PC+2*(lim+1) → PC	*	*	0	*
CASEL	CF	sel.l, base.l, lim.l, desp[o].w,.., desp[lim].w	sel-base → tmp; si tmp≤lim: PC+desp[tmp] → PC si no: PC+2*(lim+1) → PC	*	*	0	*
CASEW	AF	sel.w, base.w, lim.w, desp[o].w,.., desp[lim].w	sel-base → tmp;si tmp≤lim: PC+desp[tmp] → PC si no: PC+2*(lim+1) → PC	*	*	0	*
CHME	BD	tipo.w	Cambia a modo *executive*(2)	0	0	0	0
CHMK	BC	tipo.w	Cambia a modo *kernel* (2)	0	0	0	0
CHMS	BE	tipo.w	Cambia a supervisor (2)	0	0	0	0
CHMU	BF	tipo.w	Cambia a modo usuario (2)	0	0	0	0
CLRB	94	dst.b	0 → dst	0	1	0	-
CLRD	7C	dst.d	0 → dst	0	1	0	-
CLRF	D4	dst.f	0 → dst	0	1	0	-
CLRG	7C	dst.g	0 → dst	0	1	0	-
CLRH	7CFD	dst.h	0 → dst	0	1	0	-
CLRL	D4	dst.l	0 → dst	0	1	0	-
CLRO	7CFD	dst.o	0 → dst	0	1	0	-
CLRQ	7C	dst.q	0 → dst	0	1	0	-
CLRW	B4	dst.w	0 → dst	0	1	0	-
CMPB	91	fnt1.b, fnt2.b	fnt1-fnt2 → tmp. Varían los *flags*	*	*	0	*
CMPC3	29	lon.w, fnt1_dir.b, fnt2_dir.b	Compara caracteres, 3 operandos (3)	*	*	0	*
CMPC5	2D	fnt1_lon.w, fnt1_dir.b, rll.b, fnt2_lon.w, fnt2_dir.b	Compara caracteres, 5 operandos (3)	*	*	0	*
CMPD	71	fnt1.d, fnt2.d	Compara D_float	*	*	0	0
CMPF	51	fnt1.f, fnt2.f	Compara F_float	*	*	0	0
CMPG	51FD	fnt1.g, fnt2.g	Compara G_float	*	*	0	0
CMPH	71FD	fnt1.h, fnt2.h	Compara H_float	*	*	0	0
CMPL	D1	fnt1.l, fnt2.l	fnt1-fnt2 → tmp. Varían los *flags*	*	*	0	*
CMPP3	35	ndig_fnt.w, fnt1_dir.b, fnt2_dir.b	Compara 2 operandos BCD de igual longitud (3)	*	*	0	0
CMPP4	37	ndig_fnt1.w, fnt1_dir.b, ndig_fnt2.w, fnt2_dir.b	Compara 2 operandos BCD (distinta longitud) (3)	*	*	0	0

Nemónico	C.op.	Operandos	Descripción	N	Z	V	C
CMPV	EC	pos.l, tam.b, base.b, fnt.l	Compara campos de bit (con extensión de signo)	*	*	0	*
CMPW	B1	fnt1.w, fnt2.w	fnt1-fnt2 → tmp. Varían los *flags*	*	*	0	*
CMPZV	ED	pos.l, tam.b, base.b, fnt.l	Compara campos de bit(con extensión de 0)	*	*	0	*
CRC	0B	tbl.b, inicrc.l, cad_lon.w, stream.b	Calcula el código cíclico redundante (7)	*	*	0	0
CVTBD	6C	fnt.b, dst.d	Convierte byte a D_float	*	*	*	0
CVTBF	4C	fnt.b, dst.f	Convierte byte a F_float	*	*	*	0
CVTBG	4CFD	fnt.b, dst.g	Convierte byte a G_float	*	*	*	0
CVTBH	6CFD	fnt.b, dst.h	Convierte byte a H_float	*	*	*	0
CVTBL	98	fnt.b, dst.l	Convierte byte a doble pal.	*	*	*	0
CVTBW	99	fnt.b, dst.w	Convierte byte a palabra	*	*	*	0
CVTDB	68	fnt.d, dst.b	Convierte D_float a byte	*	*	*	0
CVTDF	76	fnt.d, dst.f	Convierte D_float a F_float	*	*	*	0
CVTDH	32FD	fnt.d, dst.h	Convierte D_float a H_float	*	*	*	0
CVTDL	6A	fnt.d, dst.l	Convierte D_float a doble palabra	*	*	*	0
CVTDW	69	fnt.d, dst.w	Convierte D_float a palabra	*	*	*	0
CVTFB	48	fnt.f, dst.b	Convierte F_float a byte	*	*	*	0
CVTFD	56	fnt.f, dst.d	Convierte F_float a D_float	*	*	*	0
CVTFG	99FD	fnt.f, dst.g	Convierte F_float a G_float	*	*	*	0
CVTFH	98FD	fnt.f, dst.h	Convierte F_float a H_float	*	*	*	0
CVTFL	4A	fnt.f, dst.l	Convierte F_float a doble palabra	*	*	*	0
CVTFW	49	fnt.f, dst.w	Convierte F_float a palabra	*	*	*	0
CVTGB	48FD	fnt.g, dst.b	Convierte G_float a byte	*	*	*	0
CVTGF	33FD	fnt.g, dst.f	Convierte G_float a F_float	*	*	*	0
CVTGH	56FD	fnt.g, dst.h	Convierte G_float a H_float	*	*	*	0
CVTGL	4AFD	fnt.g, dst.l	Convierte G_float a doble palabra	*	*	*	0
CVTGW	49FD	fnt.g, dst.w	Convierte G_float a palabra	*	*	*	0
CVTHB	68FD	fnt.h, dst.b	Convierte H_float a byte	*	*	*	0
CVTHD	F7FD	fnt.h, dst.d	Convierte H_float a D_float	*	*	*	0
CVTHF	F6FD	fnt.h, dst.f	Convierte H_float a F_float	*	*	*	0
CVTHG	76FD	fnt.h, dst.g	Convierte H_float a G_float	*	*	*	0
CVTHL	6AFD	fnt.h, dst.l	Convierte H_float a doble palabra	*	*	*	0
CVTHW	69FD	fnt.h, dst.w	Convierte H_float a palabra	*	*	*	0
CVTLB	F6	fnt.l, dst.b	Convierte doble pal. a byte	*	*	*	0
CVTLD	6E	fnt.l, dst.d	Convierte doble palabra a D_float	*	*	*	0
CVTLF	4E	fnt.l, dst.f	Convierte doble palabra a F_float	*	*	*	0
CVTLG	4EFD	fnt.l, dst.g	Convierte doble palabra a G_float	*	*	*	0
CVTLH	6EFD	fnt.l, dst.h	Convierte doble palabra a D_float	*	*	*	0
CVTLP	F9	fnt.l, ndig_dst.w, dst_dir.b	Convierte doble palabra a BCD (3)	*	*	*	0
CVTLW	F7	fnt.l, dst.w	Convierte doble palabra a palabra	*	*	*	0
CVTPL	36	ndig_fnt.w, fnt_dir.b, dst.l	Convierte BCD a doble palabra (3)	*	*	*	0
CVTPS	08	ndig_fnt.w, fnt_dir.b, ndig_dst.w, dst_dir.b	Convierte BCD a una cadena numérica (*Leading Separate Numeric*) (3)	*	*	*	0
CVTPT	24	ndig_fnt.w, fnt_dir.b, tbldir.b, ndig_dst.w, dst_dir.b	Convierte BCD a una cadena numérica (Trailing Numeric) (3)	*	*	*	0
CVTRDL	6B	fnt.d, dst.l	Redondea D_float a doble palabra	*	*	*	0
CVTRFL	4B	fnt.f, dst.l	Redondea F_float a doble palabra	*	*	*	0
CVTRGL	4BFD	fnt.g, dst.l	Redondea G_float a doble palabra	*	*	*	0
CVTRHL	6BFD	fnt.h, dst.l	Redondea H_float a doble palabra	*	*	*	0
CVTSP	09	ndig_fnt.w, fnt_dir.b, ndig_dst.w, dst_dir.b	Convierte cadena numérica (*Leading Separate Numeric*) a BCD (3)	*	*	*	0
CVTTP	26	ndig_fnt.w, fnt_dir.b, tbldir.b, ndig_dst.w, dst_dir.b	Convierte cadena numérica (*Trailing Numeric*) a BCD (3)	*	*	*	0

Nemónico	C.op.	Operandos	Descripción	N	Z	V	C
CVTWB	33	fnt.w, dst.b	Convierte palabra a byte	*	*	*	0
CVTWD	6D	fnt.w, dst.d	Convierte palabra a D_float	*	*	*	0
CVTWF	4D	fnt.w, dst.f	Convierte palabra a F_float	*	*	*	0
CVTWG	4DFD	fnt.w, dst.g	Convierte palabra a G_float	*	*	*	0
CVTWH	6DFD	fnt.w, dst.h	Convierte palabra a H_float	*	*	*	0
CVTWL	32	fnt.w, dst.l	Convierte palabra a doble palabra	*	*	*	0
DECB	97	dst.b	dst-1 → dst	*	*	*	*
DECL	D7	dst.l	dst-1 → dst	*	*	*	*
DECW	B7	dst.w	dst-1 → dst	*	*	*	*
DIVB2	86	divr.b, cnt.b	cnt/divr → cnt	*	*	*	0
DIVB3	87	divr.b, divd.b, cnt.b	divd/divr → cnt	*	*	*	0
DIVD2	66	divr.d, cnt.d	cnt/divr → cnt	*	*	0	0
DIVD3	67	divr.d, divd.d, cnt.d	divd/divr → cnt	*	*	0	0
DIVF2	46	divr.f, cnt.f	cnt/divr → cnt	*	*	0	0
DIVF3	47	divr.f, divd.f, cnt.f	divd/divr → cnt	*	*	0	0
DIVG2	46FD	divr.g, cnt.g	cnt/divr → cnt	*	*	0	0
DIVG3	47FD	divr.g, divd.g, cnt.g	divd/divr → cnt	*	*	0	0
DIVH2	66FD	divr.h, cnt.h	cnt/divr → cnt	*	*	0	0
DIVH3	67FD	divr.h, divd.h, cnt.h	divd/divr → cnt	*	*	0	0
DIVL2	C6	divr.l, cnt.l	cnt/divr → cnt	*	*	*	0
DIVL3	C7	divr.l, divd.l, cnt.l	divd/divr → cnt	*	*	*	0
DIVP	27	ndig_divr.w, divr_dir.b, ndig_divid.w,divid_dir.b, ndig_cnt.w,cnt_dir.b	Divide BCD: divd/divr → cnt (4)	*	*	*	*
DIVW2	C6	divr.w, cnt.w	cnt/divr → cnt	*	*	*	0
DIVW3	A7	divr.w, divd.w, cnt.w	divd/divr → cnt	*	*	*	0
EDITPC	38	ndig_fnt.w, fnt_dir.b, patrón.b, dst_dir.b	Edición de BCD a cadena de caracteres	*	*	*	*
EDIV	7B	divr.l, divd.q, cnt.l, res.l	divd/divr → cnt divd mod divr → res	*	*	*	0
EMODD	74	mulr.d, mulrx.b, muld.d, int.l, frac.d	int(mulr:mulrx*muld)→int frac(mulr:mulrx*muld)→frac	*	*	*	0
EMODF	54	mulr.f, mulrx.b, muld.f, int.l, fract.f	int(mulr:mulrx*muld)→int frac(mulr:mulrx*muld)→frac	*	*	*	0
EMODG	54FD	mulr.g, mulrx.w, muld.g, int.l, fract.g	int(mulr:mulrx*muld)→int frac(mulr:mulrx*muld)→frac	*	*	*	0
EMODH	74FD	mulr.h, mulrx.w, muld.h, int.l, fract.h	int(mulr:mulrx*muld)→int frac(mulr:mulrx*muld)→frac	*	*	*	0
EMUL	7A	mulr.l, muld.l, sum.l, prod.q	multr*multd+sum → prod	*	*	0	0
EXTV	EE	pos.l, tam.b, base.b, dst.l	Campo dado por posición, tamaño y base → dst (con extensión de signo)	*	*	0	-
EXTZV	EF	pos.l, tam.b, base.b, dst.l	Campo dado por posición, tamaño y base → dst (con extensión de 0)	*	*	0	-
FFC	EB	posini.l, tam.b, base.b, encpos.l	0 → Z si se encuentra un 0 a partir de posini, tam y base. Si lo encuentra, da su posición en encpos.	0	*	0	0
FFS	EA	posini.l, tam.b, base.b, encpos.l	0 → Z si se encuentra un 1 a partir de posini, tam y base. Si lo encuentra, da su posición en encpos.	0	*	0	0
HALT	00		En modo *kernel*, se detiene el procesador y no varían los códigos de condición. En otro modo, todo a 0. (2)	0 -	0 -	0 -	0 -
INCB	96	dst.b	dst+1 → dst	*	*	*	*
INCL	D6	dst.l	dst+1 → dst	*	*	*	*
INCW	B6	dst.w	dst+1 → dst	*	*	*	*
INDEX	0A	subín.l, bajo.l, alto.l, tam.l, ínin.l, ínout.l	(ínin+subín)*tam → ínout. Si subín ∉ [bajo,alto], se provoca un desvío.	*	*	0	0
INSQHI	5C	entrada.b, cabec.q	Inserta la entrada en la cola detrás de la cabecera. Flags varían según éxito (2)	0 0	* 0	0 0	0 1
INSQTI	5D	entrada.b, cabec.q	Inserta entrada en la cola antes de la cabecera. Flags varían según éxito (2)	0 0	* 0	0 0	0 1
INSQUE	0E	entrada.b, pred.b	Inserta entrada en la cola, detrás del operando pred.	*	*	0	*
INSV	F0	fnt.l, pos.l, tam.b, base.b	Bits fnt(tam-1:0) → campo dado por pos, tam y base	-	-	-	-

Nemónico	C.op.	Operandos	Descripción	N	Z	V	C
JMP	17	dst.b	Salto incondicional: Dirección (dst) → PC	-	-	-	-
JSB	16	dst.b	Salto a subrutina. El PC se guarda en la pila y después dirección (dst) → PC	-	-	-	-
LDPCTX	06		Carga el contexto de proceso (2)	-	-	-	-
LOCC	3A	car.b, lon.w, dir.b	El carácter se compara con los bytes especificados por dir y lon (8)	0	*	0	0
MATCHC	39	pat_lon.w, pat_dir.b, fnt_lon.w, fnt_dir.b	Localiza la cadena pat en la cadena fnt (3)	0	*	0	0
MCOMB	92	fnt.b, dst.b	¬fnt → dst	*	*	0	-
MCOML		fnt.l, dst.l	¬fnt → dst	*	*	0	-
MCOMW		fnt.w, dst.w	¬fnt → dst	*	*	0	-
MFPR	DB	proc_reg.l, dst.l	Copia el contenido del registro dado a dst. (2)	*	*	0	-
MNEGB	8E	fnt.b, dst.b	-fnt → dst	*	*	*	*
MNEGD	72	fnt.d, dst.d	-fnt → dst	*	*	*	*
MNEGF	52	fnt.f, dst.f	-fnt → dst	*	*	*	*
MNEGG	52FD	fnt.g, dst.g	-fnt → dst	*	*	*	*
MNEGH	72FD	fnt.h, dst.h	-fnt → dst	*	*	*	*
MNEGL	CD	fnt.l, dst.l	-fnt → dst	*	*	*	*
MNEGW	AE	fnt.w, dst.w	-fnt → dst	*	*	*	*
MOVAB	9E	fnt.b, dst.b	Dirección(fnt) → dst	*	*	0	-
MOVAD	7E	fnt.d, dst.d	Dirección(fnt) → dst	*	*	0	-
MOVAF	DE	fnt.f, dst.f	Dirección(fnt) → dst	*	*	0	-
MOVAG	7E	fnt.g, dst.g	Dirección(fnt) → dst	*	*	0	-
MOVAH	7EFD	fnt.h, dst.h	Dirección(fnt) → dst	*	*	0	-
MOVAL	DE	fnt.l, dst.l	Dirección(fnt) → dst	*	*	0	-
MOVAO	7EFD	fnt.o, dst.o	Dirección(fnt) → dst	*	*	0	-
MOVAQ	7E	fnt.q, dst.q	Dirección(fnt) → dst	*	*	0	-
MOVAW	3E	fnt.w, dst.w	Dirección(fnt) → dst	*	*	0	-
MOVB	90	fnt.b, dst.b	fnt → dst	*	*	0	-
MOVC3	28	lon.w, fnt_dir.b, dst_dir.b	Copia lon bytes a partir de fnt_dir en dst_dir. Flags varían según éxito (4)	0 *	1 *	0 0	0 *
MOVC5	2C	fnt_lon.w, fnt_dir.b, rell.b, dst_lon.w, dst_dir.b	Copia la cadena fnt_dir a partir de dst_dir. Si las longitudes son distintas rellena con rell. Flags varían según éxito (4)	0 *	1 *	0 0	0 *
MOVD	70	fnt.d, dst.d	fnt → dst	*	*	0	-
MOVF	50	fnt.f, dst.f	fnt → dst	*	*	0	-
MOVG	50FD	fnt.g, dst.g	fnt → dst	*	*	0	-
MOVH	70FD	fnt.h, dst.h	fnt → dst	*	*	0	-
MOVL	D0	fnt.l, dst.l	fnt → dst	*	*	0	-
MOVO	7DFD	fnt.o, dst.o	fnt → dst	*	*	0	-
MOVP	34	ndig.w, fnt_dir.b, dst_dir.b	Copia cadenas BCD igual longitud (lon) (3)	*	*	0	-
MOVPSL	DC	dst.l	PSL → dst	-	-	-	-
MOVQ	7D	fnt.q, dst.q	fnt → dst	*	*	0	-
MOVTC	2E	fnt_lon.w, fnt_dir.b, fill.b, tbldir.b, dst_lon.w, dst_dir.b	Copia la cadena fnt a dst. Cada byte es índice de una tabla de direcciones. Rellena con fill si la cadena dst es mas larga que fnt (4)	*	*	0	*
MOVTUC	2F	fnt_lon.w, fnt_dir.b, fill.b, tbldir.b, dst_lon.w, dst_dir.b	Copia la cadena fnt a dst hasta encontrar est o finalizar. Cada byte es índice de una tabla de direcciones (4)	*	*	*	*
MOVW	B0	fnt.w, dst.w	fnt → dst	*	*	0	-
MOVZBL	9A	fnt.b, dst.l	Bits fnt(7:0) → dst 0 → fnt(31:8)	0	*	0	-
MOVZBW	9B	fnt.b, dst.w	Bits fnt(7:0) → dst 0 → fnt(15:8)	0	*	0	-
MOVZWL	3C	fnt.w, dst.l	Bits fnt(15:0) → dst 0 → fnt(31:16)	0	*	0	-
MTPTR	DA	fnt.l, regproc.l	Carga fnt en un registro del procesador. Flags varían según éxito (2)	* -	* -	0 -	- -

Nemónico	C.op.	Operandos	Descripción	N	Z	V	C
MULB2	84	mulr.b, prod.b	Parte baja (mulr*prod) → prod	*	*	*	0
MULB3	85	mulr.b, muld.b, prod.b	Parte baja (mulr*muld) → prod	*	*	*	0
MULD2	64	mulr.d, prod.d	mulr*prod → prod	*	*	0	0
MULD3	65	mulr.d, muld.d, prod.d	mulr*muld → prod	*	*	0	0
MULF2	44	mulr.f, prod.f	mulr*prod → prod	*	*	0	0
MULF3	45	mulr.f, muld.f, prod.f	mulr*muld → prod	*	*	0	0
MULG2	44FD	mulr.g, prod.g	mulr*prod → prod	*	*	0	0
MULG3	45FD	mulr.g, muld.g, prod.g	mulr*muld → prod	*	*	0	0
MULH2	64FD	mulr.h, prod.h	mulr*prod → prod	*	*	0	0
MULH3	65FD	mulr.h, muld.h, prod.h	mulr*muld → prod	*	*	0	0
MULL2	C4	mulr.l, prod.l	Parte baja(mulr*prod) → prod	*	*	*	0
MULL3	C5	mulr.l, muld.l, prod.l	Parte baja(mulr*muld) → prod	*	*	*	0
MULP	25	mulr_lon.w, mulr_dir.b, muld_lon.w, muld_dir.b, prod_lon.w, pro_dir.b	Multiplicación en BCD mulr*muld → prod. (4)	*	*	*	0
MULW2	A4	mulr.w, prod.w	Parte baja(mulr*prod) → prod	*	*	*	0
MULW3	A5	mulr.w, muld.w, prod.w	Parte baja(mulr*muld) → prod	*	*	*	0
NOP	01		No operación	-	-	-	-
POLYD	75	arg.d, grado.w, tbldir.b	Operando Tabla de direcciones apunta a una tabla de coeficientes polinóminos. Método Horner. (8)	*	*	0	0
POLYF	55	arg.f, grado.w, tbldir.b	Operando Tabla de direcciones apunta a una tabla de coeficientes polinóminos. Método Horner. (7)	*	*	0	0
POLYG	55FD	arg.g, grado.w, tbldir.b	Operando Tabla de direcciones apunta a una tabla de coeficientes polinóminos. Método Horner. (8)	*	*	0	0
POLYH	75FD	arg.h, grado.w, tbldir.b	Operando Tabla de direcciones apunta a una tabla de coeficientes polinóminos. Método Horner. (3)	*	*	0	0
POPR	BA	masc.w	Los registros seleccionados por los bits a 1 de masc se recuperan de la pila.	-	-	-	-
PROBER	0C	modo.b, lon.w, base.b	Chequea la lectura del primer y último byte de la dirección dada por base y lon. (2)	0	*	0	-
PROBEW	0D	modo.b, lon.w, base.b	Chequea la lectura del primer y último byte de la dirección dada por base y lon. (2)	0	*	0	-
PUSHAB	9F	fnt.b	Apila la dirección de fnt	*	*	0	-
PUSHAD	7F	fnt.d	Apila la dirección de fnt	*	*	0	-
PUSHAF	DF	fnt.f	Apila la dirección de fnt	*	*	0	-
PUSHAG	7F	fnt.g	Apila la dirección de fnt	*	*	0	-
PUSHAH	7FFD	fnt.h	Apila la dirección de fnt	*	*	0	-
PUSHAL	DF	fnt.l	Apila la dirección de fnt	*	*	0	-
PUSHAO	7FFD	fnt.o	Apila la dirección de fnt	*	*	0	-
PUSHAQ	7F	fnt.q	Apila la dirección de fnt	*	*	0	-
PUSHAW	3F	fnt.w	Apila la dirección de fnt	*	*	0	-
PUSHL	DD	fnt.l	Apila la doble palabra fnt	*	*	0	-
PUSHR	BB	masc.w	Apila los registros seleccionados por los bits a 1 de masc.	-	-	-	-
REI	02		Retorno de interrupción o excepción. (2)	*	*	*	*
REMQHI	5E	cabecera.q, dir.l	Borra la entrada cabecera de la cola y carga su dirección en dir. Flags varían según éxito (2)	0 0	* 0	* 1	0 1
REMQTI	5F	cabecera.q, dir.l	Borra la entrada final de la cola y carga su dirección en dir. Flags varían según éxito (2)	0 0	* 0	* 1	0 1
REMQUE	0F	entrada.b, dir.l	Borra la entrada de la cola y carga su dirección en dir	*	*	*	*
RET	04		Retorno de procedimento	*	*	*	*
ROTL	9C	numlug.b, fnt.l, dst.l	Rotación de fnt en dst (5)	*	*	0	-
RSB	05		Retorno de subrutina. Cima de pila → PC	-	-	-	-
SBWC	D9	sub.l, dif.l	dif-sub-C → dif	*	*	*	*

Nemónico	C.op.	Operandos	Descripción	N	Z	V	C
SBWC	D9			*	*	*	*
SCANC	2A	lon.w, dir.b, tlbdir.b, masc.b	Cada byte de la cadena dada por lon y dir se usa como índice en la tabla, cuyo contenido efectúa un AND con masc. (3)	0	*	0	0
SKPC	3B	car.b, lon.w, dir.b	Compara el carácter con los bytes de la cadena lon y dir. Si igual, 0 → Z. (8)	0	*	0	0
SOBGEQ	F4	índ.l, desp.b	índ--; bifurca si índ≥0	*	*	*	-
SOBGTR	F5	índ.l, desp.b	índ--; bifurca si índ>0				
SPANC	2B	lon.w, dir.b, tbladdr.b, masc.b	Los bytes de la cadena dada por lon y dir indexan la tabla de direcciones, cuyo contenido efectúa AND con masc. (3)	0	*	0	0
SUBB2	82	sus.b, dif.b	dif-sus → dif	*	*	*	*
SUBB3	83	sus.b, min.b, dif.b	min-sus → dif	*	*	*	*
SUBD2	62	sus.d, dif.d	dif-sus → dif	*	*	0	0
SUBD3	63	sus.d, min.d, dif.d	min-sus → dif	*	*	0	0
SUBF2	42	sus.f, dif.f	dif-sus → dif	*	*	0	0
SUBF3	43	sus.f, min.f, dif.f	min-sus → dif	*	*	0	0
SUBG2	42FD	sus.g dif.g	dif-sus → dif	*	*	0	0
SUBG3	43FD	sus.g min.g dif.g	min-sus → dif	*	*	0	0
SUBH2	62FD	sus.h, dif.h	dif-sus → dif	*	*	0	0
SUBH3	63FD	sus.h, min.h, dif.h	min-sus → dif	*	*	0	0
SUBL2	C2	sus.l, dif.l	dif-sus → dif	*	*	*	*
SUBL3	C3	sus.l, min.l, dif.l	min-sus → dif	*	*	*	*
SUBP4	22	sus_lon.w, sus_dir.b, dif_lon.w, dif_dir.b	Resta dos cadenas (BCD). dif-sus → dif (3)	*	*	*	0
SUBP6	23	sus_lon.w, sus_dir.b, min_lon.w, min_dir.b, dif_lon.w, dif_dir.b	Resta dos cadenas (BCD). min-sus → dif (3)	*	*	*	0
SUBW2	A2	sus.w, dif.w	dif-sus → dif	*	*	*	*
SUBW3	A3	sus.w, min.w, dif.w	min-sus → dif	*	*	*	*
SVPCTX	07		Salva el contexto de proceso. (2)	-	-	-	-
TSTB	95	fnt.b	fnt → tmp. Varían los *flags*	*	*	0	0
TSTD	73	fnt.d	fnt → tmp. Varían los *flags*	*	*	0	0
TSTF	53	fnt.f	fnt → tmp. Varían los *flags*	*	*	0	0
TSTG	53FD	fnt.g	fnt → tmp. Varían los *flags*	*	*	0	0
TSTH	73FD	fnt.h	fnt → tmp. Varían los *flags*	*	*	0	0
TSTL	D5	fnt.l	fnt → tmp. Varían los *flags*	*	*	0	0
TSTW	B5	fnt.w	fnt → tmp. Varían los *flags*	*	*	0	0
XFC	FC		Da extensiones definidas por el usuario al juego de instrucciones del VAX-11	0	0	0	0
XORB2	8C	masc.b, dst.b	masc⊕dst → dst	*	*	0	0
XORB3	8D	masc.b, fnt.b, dst.b	masc⊕fnt → dst	*	*	0	-
XORL2	CC	masc.l, dst.l	masc⊕dst → dst	*	*	0	-
XORL3	CD	masc.l, fnt.l, dst.l	masc⊕fnt → dst	*	*	0	-
XORW2	AC	masc.w, dst.w	masc⊕dst → dst	*	*	0	-
XORW3	AF	masc.w, fnt.w, dst.w	masc⊕fnt → dst	*	*	0	-

Claves para el manejo de esta tabla

El número que figura al final de los nemónicos de muchas instrucciones, indica el numero de operandos. La letra que figura inmediatamente antes del número, o al final, en los nemónicos en que no se indica el número de operandos, indica el tipo de operando sobre el que se realiza la operación, según el siguiente código:

B → Byte
D → Punto flotante tipo D
F → Punto flotante tipo F
G → Punto flotante tipo G
H → Punto flotante tipo H
L → Doble palabra
O → Óctuple palabra
P → BCD
Q → Cuádruple palabra
S → Cadena tipo S
T → Cadena tipo T
W → Palabra

Este código también se emplea, en minúscula y precedido de un punto, para indicar el tipo de cada uno de los operandos.

Significado de las abreviaturas de los *flags*:
* *Flag* cambia
\- *Flag* no cambia
0 0 → *flag*
1 1 → *flag*

Desplazamientos:
Los operandos de la forma desp.b o desp.w son desplazamientos de 8 y 16 bits respectivamente y no tienen especificación previa en la forma modo-registro

Notas:
(1) La operación realizada por estas instrucciones es:
sum + índice → índice; índice se compara con lim. Si sum≥0 e índice<=lim ó si sum<0 e indice≥lim, se produce bifurcación, es decir, PC + despl → PC.
(2) Instrucción que sólo puede ejecutarse en los modos del sistema.
(3) Los registros R0 a R3 varían al usar esta instrucción.
(4) Los registros R0 a R5 varían al usar esta instrucción.
(5) En desplazamientos y rotaciones, el número de lugares que se desplaza o rota se interpreta de la siguiente forma:
numlug > 0: izquierda
numlug < 0: derecha
(6) Se requieren las siguientes condiciones:
* R0 y R1 deben estar siempre disponibles para que la función devuelva sus valores y no deben ser nunca guardados en la mascara de entrada.
* Los registros R2 a R11 que sean modificados en procedimiento deben ser guardados en la mascara de entrada.
(7) El registro R0 varía al usar esta instrucción.
(8) Los registros R0 y R1 varían al usar esta instrucción.

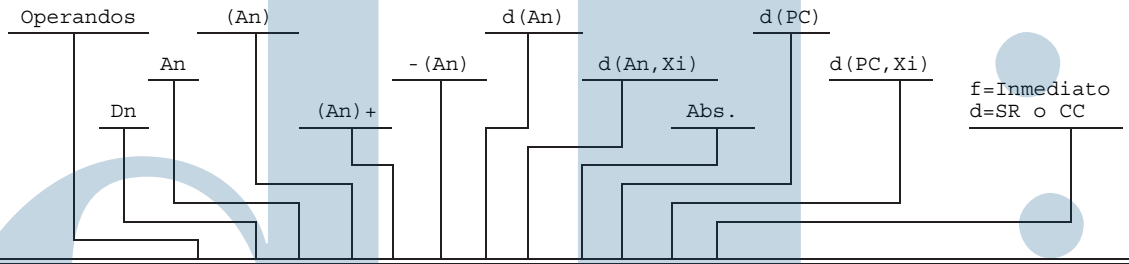
D.3.- Códigos de operacion en orden numérico

00	HALT	3C	MOVZWL	64FD	MULH2	86	DIVB2	C4	MULL2
01	NOP	3D	ACBW	65	MULD3	87	DIVB3	C5	MULL3
02	REI	3E	MOVAW	65FD	MULH3	88	BISB2	C6	DIVL2
03	BPT	3F	PUSHAW	66	DIVD2	89	BISB3	C7	DIVL3
04	RET	40	ADDF2	66FD	DIVH2	8A	BICB	C8	BISL2
05	RSB	40FD	ADDG2	67	DIVD3	8B	BICB3	C9	BISL3
06	LDPCTX	41	ADDF3	67FD	DIVH3	8C	XORB2	CA	BICL2
07	SVPCTX	41FD	ADDG3	68	CVTDB	8D	XORB3	CB	BICL3
08	CVTPS	42	SUBF2	68FD	CVTHB	8E	MNEGB	CC	XORL2
09	CVTSP	42FD	SUBG2	69	CVTDW	8F	CASEB	CD	XORL3
0A	INDEX	43	SUBF3	69FD	CVTHW	90	MOVB	CE	MNEGL
0B	CRC	43FD	SUBG3	6A	CVTDL	91	CMPB	CF	CASEL
0C	PROBER	44	MULF2	6AFD	CVTHL	92	MCOMB	D0	MOVL
0D	PROBEW	44FD	MULG2	6B	CVTRDL	93	BITB	D1	CMPL
0E	INSQUE	45	MULF3	6BFD	CVTRHL	94	CLRB	D2	MCOML
0F	REMQUE	45FD	MULG3	6C	CVTBD	95	TSTB	D3	BITL
10	BSBB	46	DIVF2	6CFD	CVTBH	96	INCB	D4	CLRF
11	BRB	46FD	DIVG2	6D	CVTWD	97	DECB	D4	CLRL
12	BNEQ	47	DIVF3	6DFD	CVTWH	98	CVTBL	D5	TSTL
12	BNEQU	47FD	DIVG3	6E	CVTLD	98FD	CVTFH	D6	INCL
13	BEQL	48	CVTFB	6EFD	CVTLH	99	CVTBW	D7	DECL
13	BEQLU	48FD	CVTGB	6F	ACBD	99FD	CVTFG	D8	ADWC
14	BGTR	49	CVTFW	6FFD	ACBH	9A	MOVZBL	D9	SBWC
15	BLEQ	49FD	CVTGW	70	MOVD	9B	MOVZBW	DA	MTPR
16	JSB	4A	CVTFL	70FD	MOVH	9C	ROTL	DB	MFPR
17	JMP	4AFD	CVTGL	71	CMPD	9D	ACBB	DC	MOVPSL
18	BGEQ	4B	CVTRFL	71FD	CMPH	9E	MOVAB	DD	PUSHL
19	BLSS	4BFD	CVTRGL	72	MNEGD	9F	PUSHAB	DE	MOVAF
1A	BGTRU	4C	CVTBF	72FD	MNEGH	A0	ADDW2	DE	MOVAL
1B	BLEQU	4CFD	CVTBG	73	TSTD	A1	ADDW3	DF	PUSHAF
1C	BVC	4D	CVTWF	73FD	TSTH	A2	SUBW2	DF	PUSHAL
1D	BVS	4DFD	CVTWG	74	EMODD	A3	SUBW3	E0	BBS
1E	BCC	4E	CVTLF	74FD	EMODH	A4	MULW2	E1	BBC
1E	BGEQU	4EFD	CVTLG	75	POLYD	A5	MULW3	E2	BBSS
1F	BCS	4F	ACBF	75FD	POLYH	A6	DIVW2	E3	BBCS
1F	BLSSU	4FFD	ACBG	76	CVTDF	A7	DIVW3	E4	BBSC
22	SUBP4	50	MOVF	76FD	CVTHG	A8	BISW2	E5	BBCC
23	SUBP6	50FD	MOVG	78	ASHL	A9	BISW3	E6	BBSSI
24	CVTPT	51	CMPF	79	ASHQ	AA	BICW2	E7	BBCCI
25	MULP	51FD	CMPG	7A	EMUL	AB	BICW3	E8	BLBS
26	CVTTP	52	MNEGF	7B	EDIV	AC	XORW2	E9	BLBC
27	DIVP	52FD	MNEGG	7C	CLRD	AD	XORW3	EA	FFS
28	MOVC3	53	TSTF	7C	CLRG	AE	MNEGW	EB	FFC
29	CMPC3	53FD	TSTG	7C	CLRQ	AF	CASEW	EC	CMPV
2A	SCANC	54	EMODF	7CFD	CLRH	B0	MOVW	ED	CMPZV
2B	SPANC	54FD	EMODG	7CFD	CLRO	B1	CMPW	EE	EXTV
2C	MOVC5	55	POLYF	7D	MOVQ	B2	MCOMW	EF	EXTZV
2D	CMPC5	55FD	POLYG	7DFD	MOVO	B3	BITW	F0	INSV
2E	MOVTC	56	CVTFD	7E	MOVAD	B4	CLRW	F1	ACBL
2F	MOVTUC	56FD	CVTGH	7E	MOVAG	B5	TSTW	F2	AOBLSS
30	BSBW	58	ADAWI	7E	MOVAQ	B6	INCW	F3	AOBLEQ
31	BRW	5C	INSQHI	7EFD	MOVAH	B7	DECW	F4	SOBGEQ
32	CVTWL	5D	INSQTI	7EFD	MOVAO	B8	BISPSW	F5	SOBGTR
32FD	CVTDH	5E	REMQHI	7F	PUSHAD	B9	BICPSW	F6	CVTLB
33	CVTWB	5F	REMQTI	7F	PUSHAG	BA	POPR	F6FD	CVTHF
33FD	CVTGF	60	ADDD2	7F	PUSHAQ	BB	PUSHR	F7	CVTLW
34	MOVP	60FD	ADDH2	7FFD	PUSHAH	BC	CHMK	F7FD	CVTHD
35	CMPP3	61	ADDD3	7FFD	PUSHAO	BD	CHME	F8	ASHP
36	CVTPL	61FD	ADDH3	80	ADDB2	BE	CHMS	F9	CVTLP
37	CMPP4	62	SUBD2	81	ADDB3	BF	CHMU	FA	CALLG
38	EDITPC	62FD	SUBH2	82	SUBB2	C0	ADDL2	FB	CALLS
39	MATCHC	63	SUBD3	83	SUBB3	C1	ADDL3	FC	XFC
3A	LOCC	63FD	SUBH3	84	MULB2	C2	SUBL2	FDFF	BUGL
3B	SKPC	64	MULD2	85	MULB3	C3	SUBL3	FEFF	BUGW

E

JUEGO DE INSTRUCCIONES DEL MICROPROCESADOR MC68000

E.1.- Juego de instrucciones en orden alfabético



Nemón.	Tipo		D	A	()	+	-	d	d	ab	dP	dP	I	Palabra de instrucción	Descripción	Cambios en *flags*				
																H	N	Z	V	C
ABCD	B	f= d=	* *											1100RRR100000rrr	Suma BCD f+d+X → d	*	?	*	?	*
		f= d=					* *							1100RRR100001rrr						
ADD	B,W,L	f= d=	*		*	*	*	*	*	*				1101DDD1SSEEEEEE	f+d → d	*	*	*	*	*
		f= d=	* *	*	*	*	*	*	*	*	*	*	*	1101DDD0SSeeeeee						
ADDA	W	f= d=	*	* *	*	*	*	*	*	*	*	*	*	1101AAA011eeeeee	f+An → An	-	-	-	-	-
	L													1101AAA111eeeeee						
ADDI	B,W,L	f= d=	*		*	*	*	*	*	*			*	00000110SSEEEEEE	f+d → d	*	*	*	*	*
ADDQ	B,W,L	d=	*	*	*	*	*	*	*	*				0101QQQ0SSEEEEEE	Q+d → d	*	*	*	*	*
ADDX	B,W,L	f= d=	* *											1101RRR1SS000rrr	f+d+X → d	*	*	*	*	*
		f= d=					* *							1101RRR1SS01rrr						
AND	B,W,L	f= d=	*		*	*	*	*	*	*				1100DDD1SSEEEEEE	f∧d → d	-	*	*	0	0
		f= d=	* *		*	*	*	*	*	*	*	*	*	1100DDD0SSeeeeee						
ANDI	B,W,L	f= d=	* *		*	*	*	*	*	*			* *	00000010SSEEEEEE	f∧d → d	-	*	*	0	0
ASL	B,W,L	c= d=	* *											1110rrr1SS100DDD	d<<c → d	*	*	*	*	*
		c=Q d=	*											1110QQQ1SS000DDD						
	W	c=1 d=			*	*	*	*	*	*				1110000111EEEEEE						
ASR	B,W,L	c= d=	* *											1110rrr0SS100DDD	d>>c → d	*	*	*	*	*
		c=Q d=	*											1110QQQ0SS000DDD						
	W	c=1 d=			*	*	*	*	*	*				1110000011EEEEEE						

Nemón.	Tipo		D	A	()	+	-	d	d	ab	dP	dP	I	Palabra de instrucción	Descripción	Cambios en *flags*				
																H	N	Z	V	C
BCHG	B	n= d=	*		*	*	*	*	*	*				0000rrr101EEEEEE	$\overline{d_n} \rightarrow Z$ $\overline{d_n} \rightarrow d_n$	-	-	*	-	-
		n= d=			*	*	*	*	*	*			*	0000100001EEEEEE						
	L	n= d=	* *											0000rrr101EEEEEE						
		n= d=	*										*	0000100001EEEEEE						
BCLR	B	n= d=	*		*	*	*	*	*	*				0000rrr110EEEEEE	$\overline{d_n} \rightarrow Z$ $0 \rightarrow d_n$	-	-	*	-	-
		n= d=			*	*	*	*	*	*			*	0000100010EEEEEE						
	L	n= d=	* *											0000rrr110EEEEEE						
		n= d=	*										*	0000100010EEEEEE						
BSET	B	n= d=	*		*	*	*	*	*	*				0000rrr111EEEEEE	$\overline{d_n} \rightarrow Z$ $1 \rightarrow d_n$	-	-	*	-	-
		n= d=			*	*	*	*	*	*			*	0000100011EEEEEE						
	L	n= d=	* *											0000rrr111EEEEEE						
		n= d=	*										*	0000100011EEEEEE						
BTST	B	n= d=	*		*	*	*	*	*	*				0000rrr100EEEEEE	$\overline{d_n} \rightarrow Z$	-	-	*	-	-
		n= d=			*	*	*	*	*	*			*	0000100000EEEEEE						
	L	n= d=	* *											0000rrr100EEEEEE						
		n= d=	*										*	0000100000EEEEEE						
CHK	W	f= d=	* *		*	*	*	*	*	*	*	*	*	0100DDD110eeeeee	si Dn∉[0,s] desvío	-	*	?	?	?
CLR	B,W,L	d=	*		*	*	*	*	*	*				01000010SSEEEEEE	0 → d	-	0	1	0	0
CMP	B,W,L	f= d=	* *	*	*	*	*	*	*	*	*	*	*	1011DDD0SSeeeeee	d-f → tmp	-	*	*	*	*
CMPA	W	f= d=	*	* *	*	*	*	*	*	*	*	*	*	1011AAA011eeeeee	An-f → tmp	-	*	*	*	*
	L													1011AAA111eeeeee						
CMPI	B,W,L	f= d=	*		*	*	*	*	*	*			*	00001100SSEEEEEE	d-f → tmp	-	*	*	*	*
CMPM	B,W,L	f= d=				* *								1011RRR1SS001rrr	d-f → tmp	-	*	*	*	*
DIVS	W	f= d=	* *		*	*	*	*	*	*	*	*	*	1000DDD111eeeeee	Dn/f → Dn Dn 32 bits	-	*	*	*	0
DIVU	W	f= d=	* *		*	*	*	*	*	*	*	*	*	1000DDD011eeeeee	Dn/f → Dn Dn 32 bits	-	*	*	*	0
EOR	B,W,L	f= d=	* *		*	*	*	*	*	*				1011rrr1SSEEEEEE	f⊕d → d	-	*	*	0	0
EORI	B,W,L	f= d=	* *		*	*	*	*	*	*			* *	00001010SSEEEEEE	f⊕d → d	-	*	*	0	0
EXG	L	d1= d2=	* *											1100DDD101000DDD	d1 ↔ d2	-	-	-	-	-
		d1= d2=	*	* *										1100DDD110001AAA 1100AAA101001AAA						
EXT	W	d=	*											0100100010000DDD	$d_7 \rightarrow d$	-	*	*	0	0
	L	d=	*											0100100011000DDD	$d_{15} \rightarrow d$	-	*	*	0	0
JMP		d=			*			*	*	*	*	*		0100111011EEEEEE	d → PC	-	-	-	-	-
JSR		d=			*			*	*	*	*	*		0100111010EEEEEE	PC → -(SP) d → PC	-	-	-	-	-
LEA	L	f= d=		*	*			*	*	*	*	*		0100AAA111EEEEEE	&f → An	-	-	-	-	-
LINK		f= ds=			*								*	0100111010EEEEEE	An → -(SP) SP → An SP+ds → SP					

Nemón.	Tipo		D	A	()	+	-	d	d	ab	dP	dP	I	Palabra de instrucción	Descripción	Cambios en *flags*				
																H	N	Z	V	C
LSL	B,W,L	c= d=	* *											1110rrr1SS101DDD	d<<c → d	*	*	*	0	*
		c=Q d=	*											1110QQQ1SS001DDD						
	W	c=1 d=			*	*	*	*	*	*				1110001111EEEEEE						
LSR	B,W,L	c= d=	* *											1110rrr0SS101DDD	d>>c → d	*	*	*	0	*
		c=Q d=	*											1110QQQ0SS001DDD						
	W	c=1 d=			*	*	*	*	*	*				1110001011EEEEEE						
MOVE	B,W,L	f= d=	* *	*	* *	* *	* *	* *	* *	* *	*	*	*	00XXRRRMMMeeeeee	f → d	-	*	*	0	0
	W	f= d=CC	*		*	*	*	*	*	*	*	*	*	0100010011eeeeee		*	*	*	*	*
		f= d=SR	*		*	*	*	*	*	*	*	*	*	0100011011eeeeee						
		f=SR d=	*		*	*	*	*	*	*				0100000011EEEEEE		-	-	-	-	-
	L	f=SP d=		*										0100111001101AAA		-	-	-	-	-
		f= d=SP		*										0100111001100AAA						
MOVEA	W,L	f= d=	*	* *	*	*	*	*	*	*	*	*	*	00XXAAA001eeeeee	f → An	-	-	-	-	-
MOVEM	W	f=Rn d=			*		*	*	*	*				0100100010EEEEEE	Rn → d (1)	-	-	-	-	-
		f= d=Rn		*	*		*	*	*	*	*	*		0100110010eeeeee	f → Rn (1)					
	L	f=Rn d=			*			*	*	*				0100100011EEEEEE	Rn → d (1)					
		f= d=Rn			*	*		*	*	*	*	*		0100110011eeeeee	d → Rn (1)					
MOVEP	W	f= d=	*					*						0000DDD110001AAA	Dn → d (por bytes)	-	-	-	-	-
		f= d=	*					*						0000DDD100001AAA	s → Dn (por bytes)					
	L	f= d=	*					*						0000DDD111001AAA	Dn → d (por bytes)					
		f= d=	*					*						0000DDD101001AAA	s → Dn (por bytes)					
MOVEQ	L	f=Q d=	*											0111DDD0QQQQQQQQ	Q → Dn	-	*	*	0	0
MULS	W	f= d=	* *		*	*	*	*	*	*	*	*	*	1100DDD111eeeeee	Dn*f → Dn	-	*	*	0	0
MULU	W	f= d=	* *		*	*	*	*	*	*	*	*	*	1100DDD011eeeeee	Dn*f → Dn	-	*	*	0	0
NBCD	B	d=	*		*	*	*	*	*	*				0100100000EEEEEE	-d-X → d	*	*	*	*	*
NEG	B,W,L	d=	*		*	*	*	*	*	*				01000100SSEEEEEE	-d → d	*	*	*	*	*
NEGX	B,W,L	d=	*		*	*	*	*	*	*				01000000SSEEEEEE	-d-X → d	*	*	*	*	*
NOP														0100111001110001	No operación	-	-	-	-	-
NOT	B,W,L	d=	*		*	*	*	*	*	*				01000110SSEEEEEE	$\overline{d}$ → d	-	*	*	0	0
OR	B,W,L	f= d=	* 		* 	* 	* 	* 	* 	* 				1000DDD1SSEEEEEE	f∧d → d	-	*	*	0	0
		f= d=	* *		* 	* 	* 	* 	* 	* 	* 	* 	* 	1000DDD0SSeeeeee						
ORI	B,W,L	f= d=	*		*	*	*	*	*	*			* *	00000000SSEEEEEE	f∧d → d	-	*	*	0	0
PEA	L	f=			*			*	*	*	*	*		0100100001eeeeee	&f → -(SP)	-	-	-	-	-
RESET														0100111001110000	Activa línea de RESET	-	-	-	-	-
ROL	B,W,L	c= d=	* *											1110rrr1SS111DDD	Rota d a la izquierda c lugares	-	*	*	0	*
		c=Q d=	*											1110QQQ1SS011DDD						
	W	c=1 d=			*	*	*	*	*	*				1110011111EEEEEE						

Nemón.	Tipo		D	A	()	+	-	d	d	ab	dP	dP	I	Palabra de instrucción	Descripción	Cambios en *flags*				
																H	N	Z	V	C
ROR	B,W,L	c= d=	* *											1110rrr0SS111DDD	Rota d a la derecha c lugares	-	*	*	0	*
		c=Q d=	* 											1110QQQ0SS011DDD						
	W	c=1 d=			*	*	*	*	*	*				1110011011EEEEEE						
ROXL	B,W,L	c= d=	* *											1110rrr1SS110DDD	Rota d a la izquierda sobre el bit X c lugares	*	*	*	0	*
		c=Q d=	* 											1110QQQ1SS010DDD						
	W	c=1 d=			*	*	*	*	*	*				1110010111EEEEEE						
ROXR	B,W,L	c= d=	* *											1110rrr0SS110DDD	Rota d a la derecha sobre el bit X c lugares	*	*	*	0	*
		c=Q d=	* 											1110QQQ0SS010DDD						
	W	c=1 d=			*	*	*	*	*	*				1110010011EEEEEE						
RTE														0100111001110011	Retorno de excepción (SP)+ → SR (SP)+ → PC	*	*	*	*	*
RTR														0100111001110111	Retorno y restauración (SP)+ → CC (SP)+ → PC	*	*	*	*	*
RTS														0100111001110101	Retorno de subrutina (SP)+ → PC	-	-	-	-	-
SBCD	B	f= d=	* *											1000RRR100000rrr	Resta BCD X-f-d → d	*	?	*	?	*
		f= d=					* *							1000RRR100001rrr						
Scc	B	d=	*		*	*	*	*	*	*				0101CCCC11EEEEEE	Si cc (2) FF → d si no 0 → d	-	-	-	-	-
STOP		f=											*	0100111001110010	f → SR esperar int.	*	*	*	*	*
SUB	B,W,L	f= d=	*		*	*	*	*	*	*				1001DDD1SSEEEEEE	d-f → d	*	*	*	*	*
		f= d=	* *	*	*	*	*	*	*	*	*	*	*	1001DDD0SSeeeeee						
SUBA	W	f= d=	*	* *	*	*	*	*	*	*	*	*	*	1001AAA011eeeeee	An-f → An	-	-	-	-	-
	L													1001AAA111eeeeee						
SUBI	B,W,L	f= d=	*		*	*	*	*	*	*			*	00000100SSEEEEEE	d-f → d	*	*	*	*	*
SUBQ	B,W,L	d=	*	*	*	*	*	*	*	*				0101QQQ1SSEEEEEE	d-Q → d	*	*	*	*	*
SUBX	B,W,L	f= d=	* *											1101RRR1SS000rrr	d-f-X → d	*	*	*	*	*
		f= d=					* *							1101RRR1SS01rrr						
SWAP	W	d=	*											0100100001000DDD	$d_{31:16} \leftrightarrow d_{15:0}$	-	*	*	0	0
TAS	B	d=	*		*	*	*	*	*	*				0100101011EEEEEE	d → tmp 1 → d_7	-	*	*	0	0
TRAP														010011100100VVVV	PC → -(SP) SR → -(SP) (V) → PC	-	-	-	-	-
TRAPV														0100111001110110	Si V==1: PC → -(SP) SR → -(SP) (trapv) → PC	-	-	-	-	-
TST	B,W,L	d=	*		*	*	*	*	*	*				01001010SSEEEEEE	d → tmp	-	*	*	0	0
UNLK		d=		*										0100111001011AAA	An → SP (SP)+ → An	-	-	-	-	-

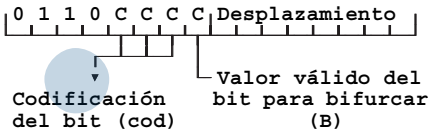
Notas: (1) Esta instrucción transfiere los registros especificados en una palabra adicional, que se interpreta como una máscara, desde/hacia el operando.
(2) El significado de cc está explicado en el párrafo E.2.

Explicación de los símbolos utilizados en la tabla anterior:

SS: Codificación del tipo: 00, byte; 01, palabra; 10, doble palabra.
XX: Codificación del tipo para MOVE: 01, byte; 11, palabra; 10, doble palabra.
MMM: Codificación de modo (ver apartado E.3).
RRR: Codificación de registro (datos o direcciones).
DDD: Codificación de un registro de datos.
AAA: Codificación de un registro de direcciones.
Rn: Registro del procesador (datos o direcciones).
eeeeee: Especificación de operando fuente: MMMRRR (ver apartado E.3).
EEEEEE: Especificación de operando destino: MMMRRR (ver apartado E.3).
QQQ o QQQQQQQQ: Operando inmediato rápido de 3 u 8 bits.
CCCC: Código de condición (ver apartado E.2).
VVVV: Número de vector.
SR: Registro de estado.
CC: Registro de códigos de condición (*flags*).

E.2.- Instrucciones de bifurcación

Cod	Bit analizado	cc B=0	cc B=1
000	0	T	F
011	Z	NE	EQ
101	N	PL	MI
100	V	VC	VS
010	C	CC	CS
110	N⊕V	GE	LT
111	Z+(N⊕V)	GT	LE
001	C+Z	HI	LS

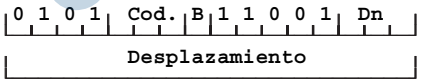


Notas:

(1) Si el desplazamiento es 0 significa que se codifica en los 16 bits de la palabra siguiente.
(2) El nemónico de estas instrucciones es Bcc donde cc figura en la tabla para cada caso.
(3) La condición T (*true*) se reserva para la instrucción BRA (*branch allways*, bifurcación incondicional) y la F (*false*) para BSR (bifurcación a subrutina).
(4) Las instrucciones de bifurcación operan así: if (bit analizado==B) PC + desplazamiento → PC

La instrucción DBcc utiliza la misma codificación para las condiciones pero con un formato distinto (ver figura de la derecha). Esta instrucción sirve para efectuar bucles y opera de la siguiente forma:

```
if (bit analizado != B)
  if(--Dn != -1)
    PC +d → PC;
```



E.3.- Resumen de los modos de direccionamiento

Código	Nombre	Notación
0	Directo por registro de datos	Dn
1	Directo por registro de direcciones	An
2	Indirecto por registro de direcciones	(An)
3	Postincremental	(An)+
4	Predecremental	-(An)
5	Indexado simple	d_{16}(An)
6	Indexado completo	d_8(An,X_i)
7.0	Absoluto corto	Dirección$_{16}$
7.1	Absoluto largo	Dirección$_8$
7.2	Relativo simple	d_{16}(PC) o Etiqueta
7.3	Relativo completo	d_8(PC,X_i)
7.4	Inmediato	#Constante