CAPITOLUL 3

INSTRUCŢIUNI ALE UNITĂŢII CENTRALE DE PRELUCRARE CPU12

3.1. INTRODUCERE

În acest capitol se prezintă operaţiile realizate prin execuţia unor instrucţiuni ale unităţii centrale de prelucrare CPU12. Prezentarea cuprinde următoarele grupuri de instrucţiuni:

- instrucţiuni de transfer de date;
- instrucţiuni aritmetice;
- instrucţiuni logice;
- instrucţiuni de deplasare şi rotire.

3.2. INSTRUCŢIUNI DE TRANSFER DE DATE

Instrucţiuni de transfer între registrele UCP şi memorie

Într-o instrucţiune *LD_* (*load instruction*) se realizează transferul unui cuvânt de 8 sau 16 biţi dintr-o locaţie a memoriei într-un registru al UCP, fără modificarea conţinutului sursei, tabelul 3.1.

Într-o instrucţiune *ST_* (*store instruction*) se realizează transferul conţinutului unui registru al UCP de 8 sau 16 biţi într-o locaţie a memoriei, fără modificarea conţinutului sursei, tabelul 10.

Instrucţiunile *LD_* şi *ST_* poziţionează biţii indicator de zero Z şi indicator de semn N din registrul de condiţii CCR.

Instrucţiuni de încărcare a registrelor UCP cu adrese efective

Într-o instrucţiune *LEA_* (*load effective address instruction*) se realizează încărcarea unui registru SP, X sau Y al UCP cu adresa efectivă generată prin tehnica de adresare indexată, tabelul 3.2. Instrucţiunile *LEA_* pot fi considerate şi ca instrucţiuni specializate de adunare şi scădere.

Instrucţiunile *LEA_* nu modifică biţii registrului de condiţii CCR.

Tabelul 3.1

Mnemonic	Function	Operation
Load Instructions		
LDAA	Load A	(M)$\Longrightarrow$A
LDAB	Load B	(M)$\Longrightarrow$B
LDD	Load D	(M : M + 1) $\Longrightarrow$ (A:B)
LDS	Load SP	(M : M + 1) $\Longrightarrow$ SP_H:SP_L
LDX	Load index register X	(M : M + 1) $\Longrightarrow$ X_H:X_L
LDY	Load index register Y	(M : M + 1) $\Longrightarrow$ Y_H:Y_L
LEAS	Load effective address into SP	Effective address $\Longrightarrow$ SP
LEAX	Load effective address into X	Effective address $\Longrightarrow$ X
LEAY	Load effective address into Y	Effective address $\Longrightarrow$ Y
Store Instructions		
STAA	Store A	(A)$\Longrightarrow$M
STAB	Store B	(B)$\Longrightarrow$M
STD	Store D	(A) $\Longrightarrow$ M, (B) $\Longrightarrow$ M + 1
STS	Store SP	(SP_H:SP_L) $\Longrightarrow$ M : M + 1
STX	Store X	(X_H:X_L) $\Longrightarrow$ M : M + 1
STY	Store Y	(Y_H:Y_L) $\Longrightarrow$ M : M + 1

Tabelul 3.2

Mnemonic	Function	Operation
LEAS	Load result of indexed addressing mode Effective Address Calculation into Stack Pointer	r ± constant $\Longrightarrow$ SP or (r) + (accumulator) $\Longrightarrow$ SP r = X, Y, SP, or PC
LEAX	Load result of indexed addressing mode Effective Address Calculation into X Index Register	r ± constant $\Longrightarrow$X or (r) + (accumulator) $\Longrightarrow$X r = X, Y, SP, or PC
LEAY	Load result of indexed addressing mode Effective Address Calculation into Y Index Register	r ± constant $\Longrightarrow$Y or (r) + (accumulator) $\Longrightarrow$ Y r = X, Y, SP, or PC

Instrucţiuni de transfer şi schimb între registrele UCP

Într-o instrucţiune de transfer între registrele UCP (*transfer instruction*), conţinutul unui registru se transferă în altul, fără modificarea conţinutului sursei, tabelul 3.3. Instrucţiunea de transfer *TFR* cuprinde toate variantele de transfer între registrele UCP. Celelalte instrucţiuni de transfer indicate în tabelul 3.3 servesc pentru compatibilitate cu familia de microcontrolere M68HC11.

Într-o instrucţiune de schimb între registrele UCP (*exchange instruction*), conţinuturile a două registre se transferă între ele, tabelul 3.3.

În instrucţiunea de transfer *SEX* (*Sign Extend into 16-Bit Register*) operandul sursă este de 8 biţi (A, B sau CCR) şi destinaţia este de 16 biţi (D, X, Y sau SP), tabelul 3.3. Astfel, transferul se realizează prin extinderea bitului de semn pentru menţinerea valorii şi semnului operandului reprezentat în cod complementul lui doi.

Tabelul 3.3

Mnemonic	Function	Operation
Transfer Instructions		
TAB	Transfer A to B	(A) $\Longrightarrow$ B
TAP	Transfer A to CCR	(A) $\Longrightarrow$ CCR
TBA	Transfer B to A	(B) $\Longrightarrow$ A
TFR	Transfer register to register	(A, B, CCR, D, X, Y, or SP) $\Longrightarrow$ A, B, CCR, D, X, Y, or SP
TPA	Transfer CCR to A	(CCR) $\Longrightarrow$ A
TSX	Transfer SP to X	(SP) $\Longrightarrow$ X
TSY	Transfer SP to Y	(SP) $\Longrightarrow$ Y
TXS	Transfer X to SP	(X) $\Longrightarrow$ SP
TYS	Transfer Y to SP	(Y) $\Longrightarrow$ SP
Exchange Instructions		
EXG	Exchange register to register	(A, B, CCR, D, X, Y, or SP) $\Longleftrightarrow$ (A, B, CCR, D, X, Y, or SP)
XGDX	Exchange D with X	(D) $\Longleftrightarrow$ (X)
XGDY	Exchange D with Y	(D) $\Longleftrightarrow$ (Y)
Sign Extension Instruction		
SEX	Sign extend 8-Bit operand	Sign-extended (A, B, or CCR) $\Longrightarrow$ D, X, Y, or SP

Instrucţiuni de transfer în memorie

Într-o instrucţiune din această categorie se realizează transferul unui cuvânt de 8 sau 16 biţi dintr-o locaţie a memoriei în alta, (*move instruction*), tabelul 3.4.

Tabelul 3.4

Mnemonic	Function	Operation
MOVB	Move byte (8-bit)	$(M_1) \Longrightarrow M_2$
MOVW	Move word (16-bit)	$(M : M + 1_1) \Longrightarrow M : M + 1_2$

Instrucţiuni cu registrul indicator de stivă

Aceste instrucţiuni sunt indicate în tabelul 3.5.

Instrucţiuni de transfer între registrele UCP şi memoria stivă

O instrucţiune *PSH_* realizează transferul conţinutului unui registru al UCP în memoria stivă, tabelul 3.5.

O instrucţiune *PUL_* realizează încărcarea unui registru al UCP din memoria stivă, tabelul 3.5.

Tabelul 3.5

Mnemonic	Function	Operation
Stack Pointer Instructions		
CPS	Compare SP to memory	(SP) – (M : M + 1)
DES	Decrement SP	(SP) – 1 $\Longrightarrow$ SP
INS	Increment SP	(SP) + 1 $\Longrightarrow$ SP
LDS	Load SP	(M : M + 1) $\Longrightarrow$ SP
LEAS	Load effective address into SP	Effective address $\Longrightarrow$ SP
STS	Store SP	(SP) $\Longrightarrow$ M : M + 1
TSX	Transfer SP to X	(SP) $\Longrightarrow$ X
TSY	Transfer SP to Y	(SP) $\Longrightarrow$ Y
TXS	Transfer X to SP	(X) $\Longrightarrow$ SP
TYS	Transfer Y to SP	(Y) $\Longrightarrow$ SP
Stack Operation Instructions		
PSHA	Push A	(SP) – 1 $\Longrightarrow$ SP; (A) $\Longrightarrow$ $M_{(SP)}$
PSHB	Push B	(SP) – 1 $\Longrightarrow$ SP; (B) $\Longrightarrow$ $M_{(SP)}$
PSHC	Push CCR	(SP) – 1 $\Longrightarrow$ SP; (CCR) $\Longrightarrow$ $M_{(SP)}$
PSHCW	Push CCR_H:CCR	(SP) – 2 $\Longrightarrow$ SP; (CCR_H:CCR) $\Longrightarrow$ $M_{(SP)}$: $M_{(SP+1)}$
PSHD	Push D	(SP) – 2 $\Longrightarrow$ SP; (A : B) $\Longrightarrow$ $M_{(SP)}$: $M_{(SP+1)}$
PSHX	Push X	(SP) – 2 $\Longrightarrow$ SP; (X) $\Longrightarrow$ $M_{(SP)}$: $M_{(SP+1)}$
PSHY	Push Y	(SP) – 2 $\Longrightarrow$ SP; (Y) $\Longrightarrow$ $M_{(SP)}$: $M_{(SP+1)}$
PULA	Pull A	($M_{(SP)}$) $\Longrightarrow$ A; (SP) + 1 $\Longrightarrow$ SP
PULB	Pull B	($M_{(SP)}$) $\Longrightarrow$ B; (SP) + 1 $\Longrightarrow$ SP
PULC	Pull CCR	($M_{(SP)}$) $\Longrightarrow$ CCR; (SP) + 1 $\Longrightarrow$ SP
PULCW	Pull CCR_H:CCR	($M_{(SP)}$: $M_{(SP+1)}$) $\Longrightarrow$ CCR_H:CCR; (SP) + 2 $\Longrightarrow$ SP
PULD	Pull D	($M_{(SP)}$: $M_{(SP+1)}$) $\Longrightarrow$ A : B; (SP) + 2 $\Longrightarrow$ SP
PULX	Pull X	($M_{(SP)}$: $M_{(SP+1)}$) $\Longrightarrow$ X; (SP) + 2 $\Longrightarrow$ SP
PULY	Pull Y	($M_{(SP)}$: $M_{(SP+1)}$) $\Longrightarrow$ Y; (SP) + 2 $\Longrightarrow$ SP

Aplicaţie

Se consideră: (SP)=4000*h*; (D)=1234*h*; (X)=ABCD*h*;

a. Să se indice conţinuturile memoriei stivă şi a registrului indicator de stivă după execuţia instrucţiunilor PSHD şi PSHX.

b. Să se indice conţinuturile memoriei stivă şi a registrelor SP, B şi X după execuţia instrucţiunilor PULB şi PULX.

a.

(SP)	3FFC*h*	AB*h*
	3FFD*h*	CD*h*
	3FFE*h*	12*h*
	3FFF*h*	34*h*

b.

(SP)	3FFF*h*	34*h*

(B)=AB*h*;
(X)=CD12*h*.

3.3. INSTRUCŢIUNI ARITMETICE

Instrucţiuni de adunare şi scădere

Instrucţiunile de adunare şi scădere sunt cu operanzi de 8 şi/sau 16 biţi, tabelul 3.6.

Într-o instrucţiune de adunare un operand sursă şi destinaţia sunt date de un registru A, B, X, Y sau D. Al doilea operand sursă este dat de registrul B sau de o locaţie a memoriei de 8 sau 16 biţi.

Într-o instrucţiune de scădere un operand sursă (descăzutul) şi destinaţia sunt date de un registru A, B sau D. Al doilea operand sursă (scăzătorul) este dat de registrul B sau de o locaţie a memoriei de 8 sau 16 biţi.

Instrucţiunile care utilizează adunarea/scăderea bitului indicator de transport C sunt utilizate pentru calcule de precizie cu operanzi reprezentaţi prin mai mult de 16 biţi.

Tabelul 3.6

Mnemonic	Function	Operation
Addition Instructions		
ABA	Add B to A	(A) + (B) $\Longrightarrow$ A
ABX	Add B to X	(B) + (X) $\Longrightarrow$ X
ABY	Add B to Y	(B) + (Y) $\Longrightarrow$ Y
ADCA	Add with carry to A	(A) + (M) + C $\Longrightarrow$ A
ADCB	Add with carry to B	(B) + (M) + C $\Longrightarrow$ B
ADDA	Add without carry to A	(A) + (M) $\Longrightarrow$ A
ADDB	Add without carry to B	(B) + (M) $\Longrightarrow$ B
ADDD	Add to D	(A:B) + (M : M + 1) $\Longrightarrow$ A : B
Subtraction Instructions		
SBA	Subtract B from A	(A) – (B) $\Longrightarrow$ A
SBCA	Subtract with borrow from A	(A) – (M) – C $\Longrightarrow$ A
SBCB	Subtract with borrow from B	(B) – (M) – C $\Longrightarrow$ B
SUBA	Subtract memory from A	(A) – (M) $\Longrightarrow$ A
SUBB	Subtract memory from B	(B) – (M) $\Longrightarrow$ B
SUBD	Subtract memory from D (A:B)	(D) – (M : M + 1) $\Longrightarrow$ D

Aplicaţie

Să se scrie un program care să calculeze suma a doi octeţi care se găsesc în memorie la adresele 1000*h* şi 1001*h*. Suma se încarcă în memorie la adresa 1002*h*.

```
LDAA    $1000
ADDA    $1001
STAA    $1002
```

Aplicaţie

Să se scrie un program care să calculeze suma a două cuvinte de câte 16 biţi care se găsesc în memorie la adresele 1000*h* şi 1002*h*. Suma se încarcă în memorie la adresa 1004*h*.

Un cuvânt de 16 biţi corespunzător unei adrese efective de memorie *a* este conţinut în două locaţii succesive ale memoriei cu adresele *a* şi *a*+1, în ordinea octetul mai semnificativ şi octetul mai puţin semnificativ.

```
LDD     $1000
ADDD    $1002
STD     $1004
```

Instrucţiuni de decrementare şi incrementare

Operandul sursă care este decrementat/incrementat cu o unitate este conţinutul unui registru A, B, SP, X, Y sau conţinutul unei locaţii de 8 biţi a memoriei, tabelul 3.7.

Tabelul 3.7

Mnemonic	Function	Operation
Decrement Instructions		
DEC	Decrement memory	(M) – $01 ⟹ M
DECA	Decrement A	(A) – $01 ⟹ A
DECB	Decrement B	(B) – $01 ⟹ B
DES	Decrement SP	(SP) – $0001 ⟹ SP
DEX	Decrement X	(X) – $0001 ⟹ X
DEY	Decrement Y	(Y) – $0001 ⟹ Y
Increment Instructions		
INC	Increment memory	(M) + $01 ⟹ M
INCA	Increment A	(A) + $01 ⟹ A
INCB	Increment B	(B) + $01 ⟹ B
INS	Increment SP	(SP) + $0001 ⟹ SP
INX	Increment X	(X) + $0001 ⟹ X
INY	Increment Y	(Y) + $0001 ⟹ Y

Instrucţiuni de comparare şi testare

Într-o instrucţiune de comparare (*compare instruction*) se compară doi operanzi de 8 sau 16 biţi, daţi de registre ale UCP şi de memorie, tabelul 3.8. Comparaţia se realizează printr-o operaţie de scădere fără memorarea rezultatului, dar care poziţionează biţii registrului de condiţii CCR.

Într-o instrucţiune de testare (*test instruction*) se compară cu zero un operand de 8 biţi, dat de un registru acumulator sau de memorie, tabelul 3.8. Comparaţia se realizează printr-o operaţie de scădere care poziţionează biţii registrului de condiţii CCR.

Poziţionarea biţilor de stare ai registrului de condiţii ca urmare a execuţiei unei instrucţiuni de comparare sau testare este utilă înainte de o instrucţiune de (salt) transfer condiţionat al controlului (*Branch Instruction*).

Tabelul 3.8

Mnemonic	Function	Operation
Compare Instructions		
CBA	Compare A to B	(A) – (B)
CMPA	Compare A to memory	(A) – (M)
CMPB	Compare B to memory	(B) – (M)
CPD	Compare D to memory (16-bit)	(A : B) – (M : M + 1)
CPS	Compare SP to memory (16-bit)	(SP) – (M : M + 1)
CPX	Compare X to memory (16-bit)	(X) – (M : M + 1)
CPY	Compare Y to memory (16-bit)	(Y) – (M : M + 1)
Test Instructions		
TST	Test memory for zero or minus	(M) – $00
TSTA	Test A for zero or minus	(A) – $00
TSTB	Test B for zero or minus	(B) – $00

Instrucţiuni de anulare, complementare şi negare

Într-o instrucţiune de negare (*negate instruction*) operandul sursă şi destinaţie de 8 biţi este conţinutul unei locaţii a memoriei sau al unui registru acumulator, iar operaţia este de negare corespunzătoare codului complementul lui doi, tabelul 3.9.

Tabelul 3.9

Mnemonic	Function	Operation
CLC	Clear C bit in CCR	$0 \Longrightarrow C$
CLI	Clear I bit in CCR	$0 \Longrightarrow I$
CLR	Clear memory	$\$00 \Longrightarrow M$
CLRA	Clear A	$\$00 \Longrightarrow A$
CLRB	Clear B	$\$00 \Longrightarrow B$
CLV	Clear V bit in CCR	$0 \Longrightarrow V$
COM	One's complement memory	$\$FF - (M) \Longrightarrow M$ or $\overline{(M)} \Longrightarrow M$
COMA	One's complement A	$\$FF - (A) \Longrightarrow A$ or $\overline{(A)} \Longrightarrow A$
COMB	One's complement B	$\$FF - (B) \Longrightarrow B$ or $\overline{(B)} \Longrightarrow B$
NEG	Two's complement memory	$\$00 - (M) \Longrightarrow M$ or $\overline{(M)} + 1 \Longrightarrow M$
NEGA	Two's complement A	$\$00 - (A) \Longrightarrow A$ or $\overline{(A)} + 1 \Longrightarrow A$
NEGB	Two's complement B	$\$00 - (B) \Longrightarrow B$ or $\overline{(B)} + 1 \Longrightarrow B$

Instrucţiuni de multiplicare şi divizare

Instrucţiunile de multiplicare şi divizare utilizează adresarea inerentă pentru operanzii sursă şi destinaţie, tabelul 3.10.

Multiplicarea se realizează între doi operanzi de câte 8 biţi cu produs de 16 biţi în instrucţiunea *MUL* sau între doi operanzi de câte 16 biţi cu produs de 32 biţi în instrucţiunile *EMUL* şi *EMULS*. În instrucţiunile *MUL* şi *EMUL* operanzii sunt consideraţi cu valori fără semn în cod binar natural, iar în instrucţiunea *EMULS* operanzii sunt consideraţi cu valori cu semn în cod complementul lui doi.

Tabelul 3.10

Mnemonic	Function	Operation
Multiplication Instructions		
EMUL	16 by 16 multiply (unsigned)	(D) × (Y) $\Longrightarrow$ Y : D
EMULS	16 by 16 multiply (signed)	(D) × (Y) $\Longrightarrow$ Y : D
MUL	8 by 8 multiply (unsigned)	(A) × (B) $\Longrightarrow$ A : B
Division Instructions		
EDIV	32 by 16 divide (unsigned)	(Y : D) ÷ (X) $\Longrightarrow$ Y Remainder $\Longrightarrow$ D
EDIVS	32 by 16 divide (signed)	(Y : D) ÷ (X) $\Longrightarrow$ Y Remainder $\Longrightarrow$ D
FDIV	16 by 16 fractional divide	(D) ÷ (X) $\Longrightarrow$ X Remainder $\Longrightarrow$ D
IDIV	16 by 16 integer divide (unsigned)	(D) ÷ (X) $\Longrightarrow$ X Remainder $\Longrightarrow$ D
IDIVS	16 by 16 integer divide (signed)	(D) ÷ (X) $\Longrightarrow$ X Remainder $\Longrightarrow$ D

Aplicaţie

Să se calculeze rezultatele execuţiilor instrucţiunilor *EMUL* şi *EMULS* dacă: (D)=FFFF*h* şi (Y)=0010*h*.

Rezultatul execuţiei instrucţiunii *EMUL*: (Y : D)=000F FFF0*h*.

Rezultatul execuţiei instrucţiunii *EMULS*: (Y : D)=FFFF FFF0*h*.

În cazul divizării deîmpărţitul este de 16 biţi în instrucţiunile *IDIV*, *IDIVS* şi FDIV şi de 32 de biţi în instrucţiunile *EDIV* şi *EDIVS*. În toate instrucţiunile de divizare împărţitorul, câtul şi restul sunt de câte 16 biţi. În instrucţiunile *IDIV*, *EDIV* şi *FDIV* operanzii sunt consideraţi cu valori fără semn în cod binar natural, iar în instrucţiunile *IDIVS* şi *EDIVS* operanzii sunt consideraţi cu valori cu semn în cod complementul lui doi. În cazul instrucţiunii *FDIV* (*fractional divide*) valoarea deîmpărţitului trebuie să fie mai mică decât cea a împărţitorului, iar câtul şi restul se obţin în cod binar natural fracţionar. Execuţia instrucţiunii *FDIV* poate fi echivalată cu

înmulţirea deîmpărţitului cu 2^{16} urmată de o împărţire cu operanzi întregi în care deîmpărţitul este de 32 de biţi şi împărţitorul este de 16 biţi.

Aplicaţie

Să se calculeze rezultatele execuţiilor instrucţiunilor *IDIV* şi *IDIVS* dacă: (D)=FFEF*h* şi (X)=0010*h*.

Rezultatul execuţiei instrucţiunii *IDIV*: cât (X) =0FFE*h* şi rest (D)=000F*h*.

Rezultatul execuţiei instrucţiunii *IDIVS*: cât (X)=FFFF*h* şi rest (D)=FFFF*h*.

3.4. INSTRUCŢIUNI LOGICE

În instrucţiunile logice, tabelul 3.11, se efectuează operaţii logice ŞI, SAU şi SAU-Exclusiv cu operanzi octeţi. Un operand sursă şi destinaţia este un registru A, B sau CCR, iar al doilea operand sursă este o locaţie a memoriei.

Tabelul 3.11

Mnemonic	Function	Operation
ANDA	AND A with memory	(A) • (M) $\Longrightarrow$ A
ANDB	AND B with memory	(B) • (M) $\Longrightarrow$ B
ANDCC	AND CCR with memory (clear CCR bits)	(CCR) • (M) $\Longrightarrow$ CCR
EORA	Exclusive OR A with memory	(A) $\oplus$ (M) $\Longrightarrow$ A
EORB	Exclusive OR B with memory	(B) $\oplus$ (M) $\Longrightarrow$ B
ORAA	OR A with memory	(A) + (M) $\Longrightarrow$ A
ORAB	OR B with memory	(B) + (M) $\Longrightarrow$ B
ORCC	OR CCR with memory (set CCR bits)	(CCR) + (M) $\Longrightarrow$ CCR

Aplicaţie

Să se calculeze rezultatele execuţiilor instrucţiunilor *ANDA*, *EORA* şi *ORAA* dacă: (A)=1C*h* şi (M)=2F*h*.

Rezultatul execuţiei instrucţiunii *ANDA*: (A)=0C*h*.

Rezultatul execuţiei instrucţiunii *EORA*: (A)=33*h*.

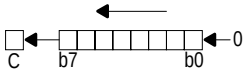
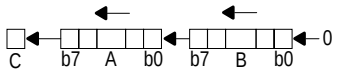
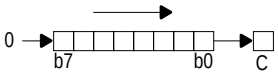
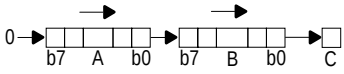
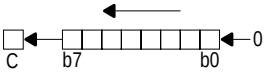
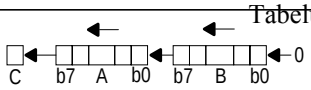
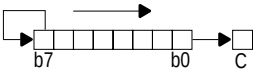
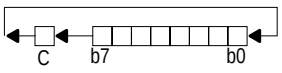
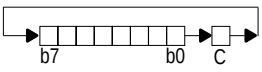
Rezultatul execuţiei instrucţiunii *ORAA*: (A)=3F*h*.

3.5. INSTRUCŢIUNI DE DEPLASARE ŞI ROTIRE

Operaţiile de deplasare şi rotire constau, în principal, în deplasarea spre stânga sau dreapta cu un bit a conţinutului unui registru A, B, D sau a unei locaţii a memoriei. În instrucţiunile de deplasare şi rotire, tabelul 3.12, intervine ca operand bitul indicator de transport C. Se precizează că deplasarea logică spre stânga este identică cu deplasarea aritmetică spre stânga. La deplasarea aritmetică spre dreapta bitul cel mai

semnificativ al operandului supus deplasării nu se modifică pentru a menţine semnul acestuia.

Tabelul 3.12

Mnemonic	Function	Operation
	Logical Shifts	
LSL LSLA LSLB	Logic shift left memory Logic shift left A Logic shift left B	
LSLD	Logic shift left D	
LSR LSRA LSRB	Logic shift right memory Logic shift right A Logic shift right B	
LSRD	Logic shift right D	
	Arithmetic Shifts	
ASL ASLA ASLB	Arithmetic shift left memory Arithmetic shift left A Arithmetic shift left B	
ASLD	Arithmetic shift left D	
ASR ASRA ASRB	Arithmetic shift right memory Arithmetic shift right A Arithmetic shift right B	
	Rotates	
ROL ROLA ROLB	Rotate left memory through carry Rotate left A through carry Rotate left B through carry	
ROR RORA RORB	Rotate right memory through carry Rotate right A through carry Rotate right B through carry	

Tabelul 3.11

Aplicaţie

Să se scrie un program care să realizeze deplasarea spre stânga cu un bit a cuvântului de 16 biţi din memorie corespunzător adresei 1000*h*.

```
LSL   $1001          LDD   $1000
ROL   $1000          LSLD
                     STD   $1000
```