CAPITOLUL 2

UNITATEA CENTRALĂ DE PRELUCRARE CPU12

2.1. INTRODUCERE

Unitatea centrală de prelucrare CPU12 este componentă a unui microcontroler din familia HCS12X. Principalele componente ale microcontrolerului MC9S12XDP512 din familia HCS12X, fig. 2.1, sunt:

- unitate centrală de prelucrare CPU12;
- generator pentru tact şi inițializare;
- stabilizator de tensiune;
- memorie RAM, 32 Kocteți;
- memorie EEPROM, 4 Kocteți;
- memorie flash EEPROM, 512 Kocteți;
- logică de control întreruperi, EMIM;
- interfață cu un sistem gazdă pentru dezvoltarea aplicațiilor, BDM;
- şase interfețe de comunicație serială asincronă, SCI;
- trei interfețe de comunicație serială sincronă SPI;
- temporizator cu funcții de numărare, comparare şi captură, ECT;
- temporizator pentru întreruperi periodice, PIT;
- modulator de impulsuri în durată, PWM;
- două convertoare analog numerice de 10 biți, ATD;
- două porturi seriale I^2C, IIC;
- cinci porturi seriale de comunicație conform protocolului CAN 2.0A/B;
- procesor pentru prelucrare şi transfer de date între perifericele şi memoriile interne ale microcontrolerului, XGATE;
- interfață cu magistrale externe, EBI;
- porturi paralele.

Unitatea centrală de prelucrare (UCP) CPU12 conține:

- unitate aritmetică şi logică de 16 biți;
- registre interne;
- memorie de instrucțiuni;
- logică de decodificare, generare adrese şi control.

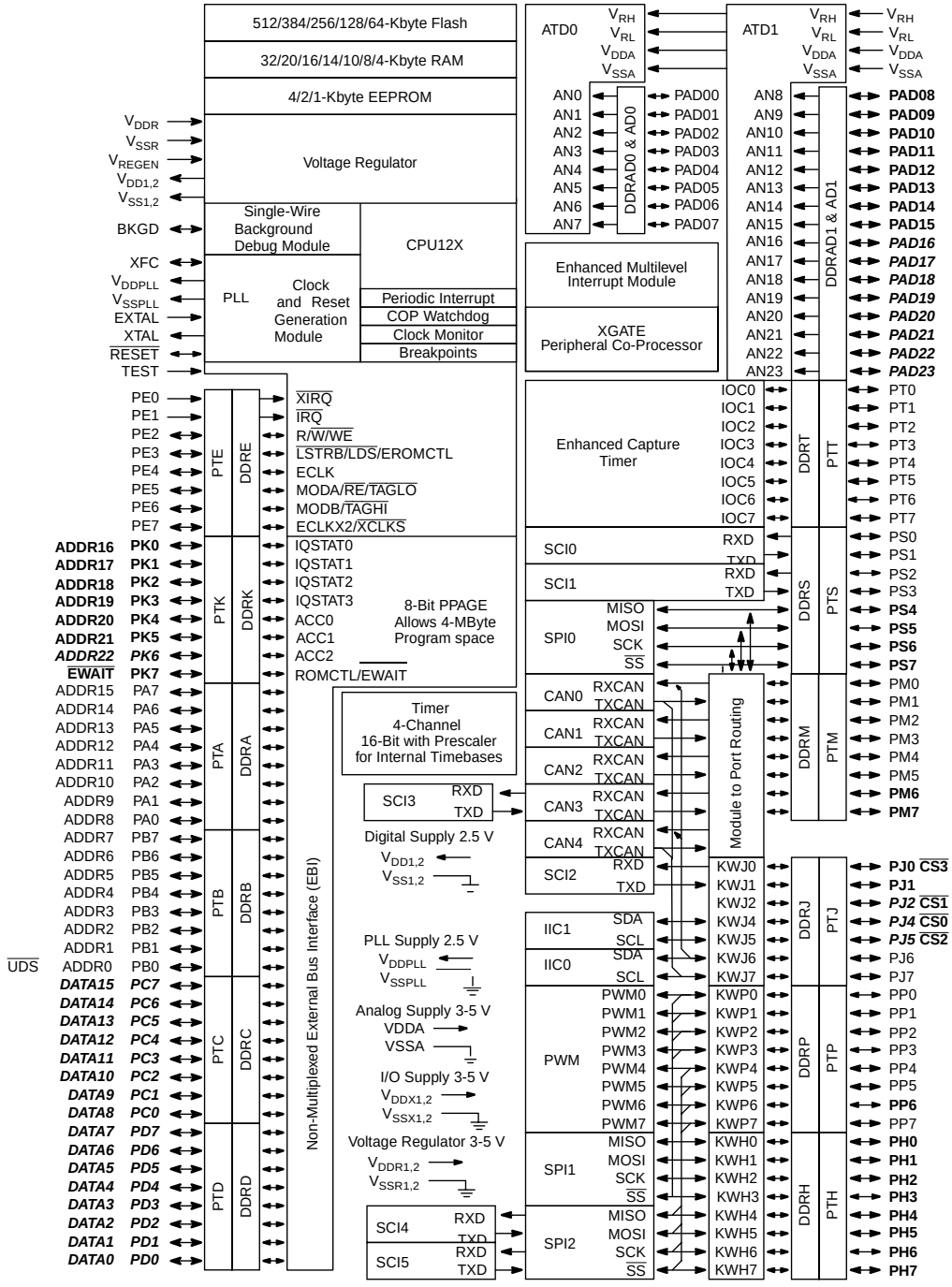


Fig. 2.1. Structura microcontrolerului MC9S12XDP512 din familia HCS12X.

2.2. REGISTRELE UNITĂȚII CENTRALE DE PRELUCRARE CPU12

Registrele unității centrale de prelucrare CPU12 sunt indicate în fig. 2.2.

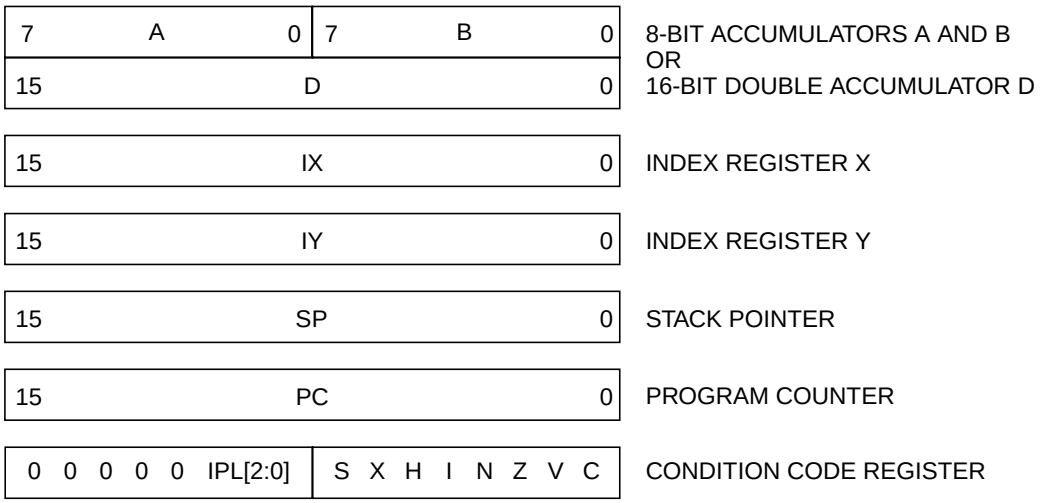


Fig. 2.2. Registrele unității centrale de prelucrare CPU12.

Registrele acumulator A şi **B** de câte 8 biți sunt utilizate pentru operanzi sursă şi destinație în multe instrucțiuni. În unele instrucțiuni se utilizează registrul acumulator **D** de 16 biți, format din registrele A şi B (A : B).

Registrele index X şi **Y** de câte 16 biți sunt utilizate pentru adresarea indexată a operanzilor.

Registrul indicator de stivă SP de 16 biți este utilizat pentru adresarea memoriei stivă.

Registrul numărător de adrese PC de 16 biți este utilizat pentru adresarea programului.

Registrul de condiții $CCR_W = CCR_H : CCR_L$/CCR (*condition code register*) de 16 biți conține biți indicatori de stare şi biți de control.

Biții nivel de prioritate întrerupere IPL (*Interrupt Priority Level*)

Biții IPL se utilizează pentru controlul execuției subrutinelor de întrerupere în varianta *nesting of interrupts* (întreruperea execuției unei subrutine de întrerupere pentru execuția unei alte subrutine de întrerupere corespunzătoare unei surse de nivel de prioritate mai ridicat). Biții IPL conțin nivelul de prioritate alocat sursei în curs de deservire prin execuția subrutinei de întrerupere corespunzătoare.

Bitul de mascare întreruperi X (*X Masc Bit*)

Întreruperile externe prin pinul **/XIRQ** al microcontrolerului sunt destinate unor evenimente majore privind funcționarea necorespunzătoare a sistemului, de exemplu probleme de alimentare a unor componente din sistem. Aceste întreruperi se validează numai după alimentarea completă şi inițializarea sistemului (funcționarea stabilă). Astfel, la inițializarea procesorului, bitul X este (activat) poziționat la nivel logic 1 ceea ce invalidează întreruperile /XIRQ. După funcționarea stabilă a sistemului, bitul X este pus la nivel logic 0 cu o instrucțiune corespunzătoare, de exemplu ***ANDCC*** #$BF, prin care se validează întreruperile /XIRQ. Bitul X nu poate fi poziționat la nivel logic 1 prin program. Rezultă că pe durata funcționării stabile a sistemului întreruperile /XIRQ sunt nemascabile.

Bitul de mascare întreruperi I (*I Masc Bit*)

Bitul I are rolul de rolul de invalidare/validare a întreruperilor mascabile, corespunzător stărilor activă/inactivă (nivel logic1/nivel logic 0). Prin inițializarea procesorului, bitul I este poziționat la nivel logic 1. În starea de invalidare (mascare), cererile de întrerupere sunt memorate şi în asteptare (procesorul nu acceptă intreruperi mascabile). Validarea întreruperilor mascabile se obține prin poziționarea bitului I la nivel logic 0, de exemplu cu instrucțiunea ***ANDCC*** #$EF. La acceptarea unei cereri de întrerupere mascabilă, se salvează în memoria stivă registrele UCP (inclusiv registrul CCR) şi apoi bitul I este activat înainte de execuția primei instrucțiuni din subrutina de întrerupere pentru prevenirea suprapunerii întreruperilor. Prin instrucțiunea ***RTI*** de revenire dintr-o subrutină de întrerupere, plasată la sfârşitul acesteia se restabilesc din memoria stivă conținuturile registrelor UCP şi implicit se validează întreruperile mascabile.

Bitul de control S (*S Control Bit*)

La nivel logic 0, bitul S validează instrucțiunea ***STOP*** prin execuția căreia se salvează registrele UCP în memoria stivă, se blochează generatorul (generatoarele) de tact şi procesorul trece în stare de aşteptare minimizând consumul de la sursa de alimentare. Activarea unei linii /RESET, /XIRQ sau /IRQ determină ieşirea procesorului din starea de aşteptare. În cazul unei cereri de întrerupere nemascabilă /XIRQ, procesorul trece din stare de aşteptare la execuția subrutinei de întrerupere dacă bitul X este la nivel logic 0 sau la execuția instrucțiunii următoare instrucțiunii ***STOP*** dacă bitul X este la nivel logic 1.

Bitul indicator de transport/împrumut C (*C Status Bit*)

Bitul C se poziționează la nivel logic 1 în urma execuției unei instrucțiuni de adunare care a produs transport din poziția bitului cel mai semnificativ al rezultatului şi în urma execuției unei instrucțiuni de scădere care a necesitat împrumut în poziția bitului cel mai semnificativ al descăzutului.

Bitul indicator de transport H (*H Status Bit*)

Bitul H se poziționează la nivel logic 1 în urma execuției unei instrucțiuni de adunare care a produs transport din poziția bitului 3 al acumulatorului A. Indicatorul H se utilizează pentru aplicații în aritmetică cu operanzi în format zecimal codat binar,

BCD, precizând un transport între cele două grupe de câte 4 biți ale unui octet reprezentând două ranguri zecimale. Rezultatul adunării din acumulatorul A se corectează conform formatului BCD prin execuția instrucțiunii ***DAA*** care utilizează informația dată de bitul H.

Bitul indicator de depăşire V (*V Status Bit*)

Bitul V se poziționează la nivel logic 1 în urma execuției unei operații al cărei rezultat depăşeşte domeniul de valori corespunzător reprezentării în cod complementul lui doi.

Bitul indicator de zero Z (*Z Status Bit*)

Bitul Z se poziționează la nivel logic 1 în urma execuției unei instrucțiuni al cărei rezultat este zero. De exemplu, o instrucțiune de comparare a doi operanzi implică o operație de scădere al cărei rezultat se concretizează prin poziționarea unor biți ai registrului de condiții CCR, inclusiv a bitului Z.

Bitul indicator de semn N (*N Status Bit*)

Bitul N se poziționează în urma execuției unei instrucțiuni, la nivelul logic corespunzător bitului de semnificație maximă al rezultatului. Considerând operanzii în cod complementul lui doi, rezultă că indicatorul N precizează semnul rezultatului unei operații aritmetice sau logice.

Aplicație

Se consideră operațiile de adunare a câte doi operanzi reprezentați prin octeți:

a. 50*h* + 2D*h*
b. 69*h* + 2D*h*
c. 2D*h* + F6*h*
d. 88*h* + F6*h*
e. 96*h* + 6A*h*

Se cere:

- să se indice rezultatele în hexazecimal ale operațiilor de adunare cu o unitate aritmetică de 8 biți;
- să se indice valorile zecimale ale operanzilor şi rezultatelor adunărilor în cod binar natural CBN şi în cod complementul lui doi CCD;
- să se indice şi să se justifice modurile în care se poziționează, prin operațiile de adunare, biții indicatori de transport C, de depăşire V, de semn N şi de zero Z.

a. 50*h* + 2D*h* = 7D*h*

80 + 45 = 125	;în CBN şi CCD
C = 0	;nu este depăşire în CBN
V = 0	;nu este depăşire în CCD
N = 0	;rezultatul este cu valoare pozitivă în CCD
Z = 0	;rezultatul nu este zero

b. 69*h* + 2D*h* = 96*h*

105 + 45 = 150	;în CBN
105 + 45 = -106	;în CCD
C = 0	;nu este depăşire în CBN
V = 1	;este depăşire în CCD
N = 1	;rezultatul este cu valoare negativă în CCD
Z = 0	;rezultatul nu este zero

c. 2D*h* + F6*h* = 23*h*

45 + 246 = 35	;în CBN
45 + (-10) = 35	;în CCD
C = 1	;este depăşire în CBN
V = 0	;nu este depăşire în CCD
N = 0	;rezultatul este cu valoare pozitivă în CCD
Z = 0	;rezultatul nu este zero

d. 88*h* + F6*h* = 7E*h*

136 + 246 = 126	;în CBN
(-120) + (-10) = 126	;în CCD
C = 1	;este depăşire în CBN
V = 1	;este depăşire în CCD
N = 0	;rezultatul este cu valoare pozitivă în CCD
Z = 0	;rezultatul nu este zero

e. 96*h* + 6A*h* = 0*h*

150 + 106 = 0	;în CBN
(-106) + 106 = 0	;în CCD
C = 1	;este depăşire în CBN
V = 0	;nu este depăşire în CCD
N = 0	;rezultatul este cu valoare zero (pozitivă) în CCD
Z = 1	;rezultatul este zero

2.3. MEMORIA DE INSTRUCȚIUNI A UNITĂȚII CENTRALE DE PRELUCRARE CPU12

Unitatea centrală de prelucrare CPU12 conține o memorie de instructiuni de tip FIFO cu 3 locații de câte 16 biți (*instruction queue*) denumite etaje şi numerotate cu 1, 2 şi 3, fig 2.3. Etajul 1 este intrarea şi etajul 3 este ieşirea. Într-un ciclu de acces la memorie pentru extragere coduri instrucțiuni, UCP adresează memoria cu registrul PC şi extrage un cuvânt de 16 biți aliniat la o adresă pară (doi octeți de la adrese

succesive, prima adresă fiind pară) care se introduce în etajul 1 al memoriei de instrucțiuni. UCP poate accesa orice octet din memoria de instrucțiuni. Acest acces se realizează după decodificarea octetului cod operație al unei instrucțiuni.

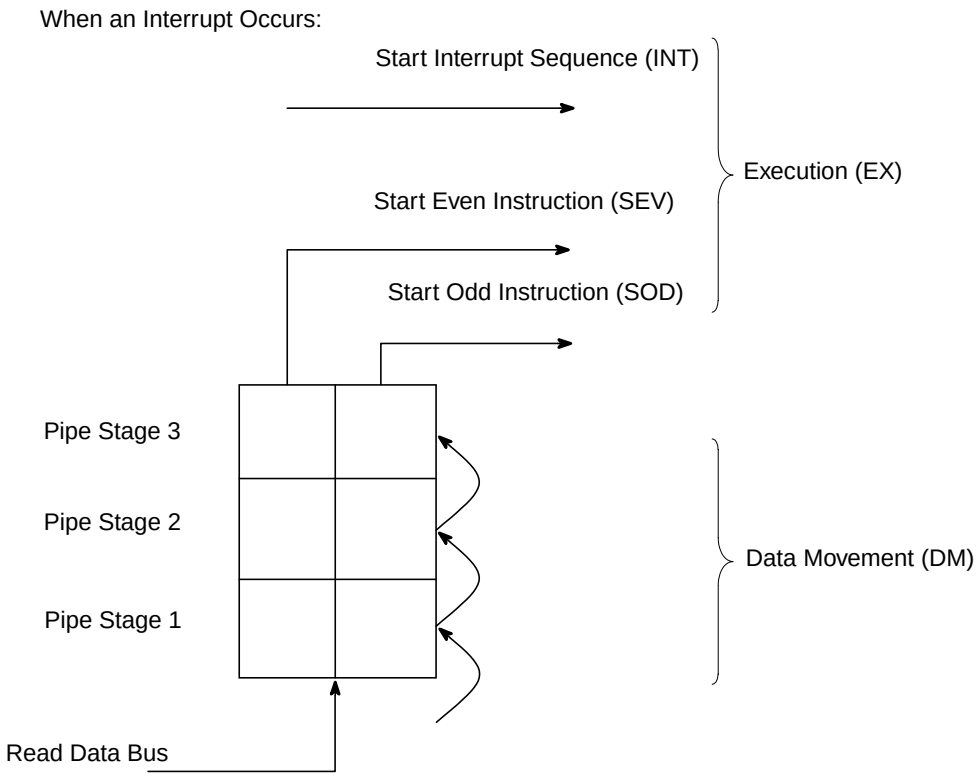


Fig. 2.3. Memoria de instrucțiuni a unității centrale de prelucrare CPU12.

Pe durata execuției unei instrucțiuni, UCP efectuează cicluri de acces la memorie pentru extragere coduri instrucțiuni prin care se actualizează conținutul memoriei de instrucțiuni cu coduri mașină ale instrucțiunilor ce urmează a fi executate. Numărul acestor cicluri este egal cu numărul de etaje eliberate în memoria de instrucțiuni prin execuția instrucțiunii corespunzătoare. Rezultă ca acest număr este funcție de numărul de octeți din codul mașină al instrucțiunii executate şi de plasarea în memorie a acesteia (primul octet din codul mașină este la o adresă pară sau impară) (*even/odd instruction*).

2.4. TEHNICILE DE ADRESARE A OPERANZILOR

Activitatea UCP constă în execuţia unor secvenţe de instrucţiuni. Execuţia instrucţiunilor de către UCP se realizează pe baza codurilor maşină ale instrucţiunilor şi a operanzilor corespunzători care se obţin din registrele UCP şi din memorie. Pentru obţinerea operanzilor din memorie UCP execută cicluri de acces la memorie.

Adresarea memoriei pentru extragerea de coduri instrucţiuni se realizează prin utilizarea informaţiei din registrul PC care conţine adresa primului octet al instrucţiunii care urmează a fi executată după cea in curs de execuţie (adresa următoarei instrucţiuni). Operanzii din registrele UCP se obţin prin tehnica de adresare inerentă. Operanzii din locaţiile memoriei se obţin prin tehnicile de adresare imediată, relativă, directă, indexată şi indexată-indirectă. Aceste tehnici de adresare sunt prezentate sintetic în tabelul 2.1.

Tabelul 2.1

Addressing Mode	Source Format	Abbreviation	Description
Inherent	**INST** (no externally supplied operands)	INH	Operands (if any) are in CPU12 registers
Immediate	**INST** *#opr8i* or **INST** *#opr16i*	IMM	Operand is included in instruction stream 8- or 16-bit size implied by context
Direct	**INST** *opr8a*	DIR	Operand is the lower 8 bits of an address in the range $0000–$00FF
Extended	**INST** *opr16a*	EXT	Operand is a 16-bit address
Relative	**INST** *rel8* or **INST** *rel16*	REL	An 8-bit or 16-bit relative offset from the current pc is supplied in the instruction
Indexed (5-bit offset)	**INST** *oprx5*,*xysp*	IDX	5-bit signed constant offset from X, Y, SP, or PC
Indexed (pre-decrement)	**INST** *oprx3*,**–***xys*	IDX	Auto pre-decrement x, y, or sp by 1 ~ 8
Addressing Mode	**Source Format**	**Abbreviation**	**Description**
Indexed (pre-increment)	**INST** *oprx3*,**+***xys*	IDX	Auto pre-increment x, y, or sp by 1 ~ 8
Indexed (post-decrement)	**INST** *oprx3*,*xys***–**	IDX	Auto post-decrement x, y, or sp by 1 ~ 8
Indexed (post-increment)	**INST** *oprx3*,*xys***+**	IDX	Auto post-increment x, y, or sp by 1 ~ 8
Indexed (accumulator offset)	**INST** *abd*,*xysp*	IDX	Indexed with 8-bit (A or B) or 16-bit (D) accumulator offset from X, Y, SP, or PC
Indexed (9-bit offset)	**INST** *oprx9*,*xysp*	IDX1	9-bit signed constant offset from X, Y, SP, or PC (lower 8 bits of offset in one extension byte)
Indexed (16-bit offset)	**INST** *oprx16*,*xysp*	IDX2	16-bit constant offset from X, Y, SP, or PC (16-bit offset in two extension bytes)
Indexed-Indirect (16-bit offset)	**INST [***oprx16*,*xysp***]**	[IDX2]	Pointer to operand is found at... 16-bit constant offset from X, Y, SP, or PC (16-bit offset in two extension bytes)
Indexed-Indirect (D accumulator offset)	**INST [D,***xysp***]**	[D,IDX]	Pointer to operand is found at... X, Y, SP, or PC plus the value in D

2.4.1. ADRESAREA INERENTĂ

Tehnica de adresare inerentă (*inherent*) **INH** este utilizată într-o instrucțiune pentru operanzi corespunzători registrelor UCP. Această tehnică de adresare este atribuită şi instrucțiunilor care nu necesită operanzi.

Exemple:
Instrucțiunea ***INX*** (*Increment Index Register X*), cod maşină 08
- (X) + $0001 → X;
- registrul index X este sursă şi destinație prin adresare inerentă.

Instrucțiunea ***SBA*** (*Subtract Accumulators*), cod maşină 18 16
- (A) – (B) → A;
- registrul acumulator A este sursă şi destinație prin adresare inerentă;
- registrul acumulator B este sursă şi se obține prin adresare inerentă.

Instrucțiunea ***NOP*** (*Null Operation*), cod maşină A7
- instrucțiunea introduce o întârziere de un ciclu;
- instrucțiunea nu necesită operanzi.

2.4.2. ADRESAREA IMEDIATĂ

Tehnica de adresare imediată (*immediate*) **IMM** este utilizată într-o instrucțiune pentru accesul unui operand sursă de 8 sau 16 biți care este conținut în codul maşină al instrucțiunii corespunzătoare, deci în memorie. Prin adresarea memoriei de program cu registrul PC în cicluri de extragere coduri instrucțiuni, codul maşină corespunzător instrucțiunii în curs de execuție se încarcă în memoria de instrucțiuni (*instruction queue*) de unde UCP accesează operandul corespunzător adresării immediate. Utilizarea adresării imediate este indicată prin simbolul # la scrierea instrucțiunii în limbaj de asamblare.

Exemple:
Instrucțiunea ***LDAA*** #$55 (*Load Accumulator A*), cod maşină 86 55
- $55 → A;
- operandul sursă obținut prin adresare imediată este octetul $55, al doilea din codul maşina al instrucțiunii;
- registrul acumulator A este destinația prin adresare inerentă.

Instrucțiunea ***LDX*** #$1234 (*Load Index Register X*), cod maşină CE 12 34
- $1234 → X;
- operandul sursă obținut prin adresare imediată este cuvântul de 16 biți $1234, octeții doi şi trei din codul maşina al instrucțiunii;

- registrul index X este destinația prin adresare inerentă.

Instrucțiunea ***LDY*** #$67 (*Load Index Register Y*), cod maşină CD 00 67
- $0067 → Y;
- operandul sursă obținut prin adresare imediată este cuvântul de 16 biți $0067, octeții doi şi trei din codul maşina al instrucțiunii;
- registrul index Y este destinația prin adresare inerentă.

2.4.3. ADRESAREA RELATIVĂ

Tehnica de adresare relativă (*relative*) **REL** este utilizată într-o instrucțiune de salt relativ (*branch instruction*) pentru accesul unui operand sursă de 8 sau 16 biți care este conținut în codul maşină al instrucțiunii corespunzătoare şi care reprezintă un deplasament relativ la registrul numărător de adrese PC. Deplasamentul este cu semn în cod complementul lui doi.

Exemple:
Instrucțiunea ***BRA*** *+15 (*Branch Always*), cod maşină 20 0D
- (PC) + $0002 + 13 → PC;
- operandul sursă obținut prin adresare relativă este octetul cu valoarea 13, al doilea din codul maşina al instrucțiunii;
- registrul PC este sursă şi destinație prin adresare inerentă.

Instrucțiunea ***LBRA*** *+$1234 (*Long Branch Always*), cod maşină 18 20 12 30
- (PC) + $0004 + $1230 → PC;
- operandul sursă obținut prin adresare relativă este cuvântul de 16 biți $1230 dat prin doi octeți în codul maşina al instrucțiunii;
- registrul PC este sursă şi destinație prin adresare inerentă.

2.4.4. ADRESAREA DIRECTĂ

Tehnica de adresare directă (*direct*) este utilizată într-o instrucțiune pentru accesul unui operand sursă sau destinație corespunzător unei locații de memorie a cărei adresă este dată în codul maşină al instrucțiunii. Adresa din codul maşină este dată prin 8 sau 16 biți, corespunzător variantelor de adresare directă:

- adresare directă scurtă (*direct*) **DIR**;
- adresare directă extinsă (*extended*) **EXT**.

Adresarea directă scurtă

În varianta de adresare directă scurtă (*direct*) **DIR,** codul maşină al instrucțiunii conține un octet care reprezintă cei 8 biți mai puțin semnificativi ai adresei efective. Cei 8 biți mai semnificativi ai adresei efective sunt dați de registrul DIRECT. În

comparație cu varianta de adresare directă extinsă, această variantă prezintă avantaje privind încărcarea memoriei de program şi timpul de execuție.

Exemple:
Instrucțiunea ***LDAA*** $55 (*Load Accumulator A*), cod maşină 96 55
- ((DIRECT) : $55) → A;
- operandul sursă obținut prin adresare directă scurtă este octetul din locația de memorie cu adresa efectivă $xy55, unde octetul xy este conținutul registrului DIRECT şi octetul 55 este al doilea din codul maşina al instrucțiunii;
- registrul acumulator A este destinația prin adresare inerentă.

Instrucțiunea ***LDX*** $20 (*Load Index Register X*), cod maşină DE 20
- ((DIRECT) : $20) → X_H, ((DIRECT) : $20 + 1) → X_L;
- operandul sursă obținut prin adresare directă scurtă este cuvântul de 16 biți din locațiile de memorie cu adresele efective $xy20 şi $xy21, octetul mai semnificativ şi respectiv octetul mai puțin semnificativ. Octetul xy este conținutul registrului DIRECT şi octetul 20 este al doilea din codul maşina al instrucțiunii;
- registrul index X este destinația prin adresare inerentă.

Adresarea directă extinsă

În varianta de adresare directă extinsă (*extended*) **EXT**, codul maşină al instrucțiunii conține doi octeți care reprezintă cei 16 biți ai adresei efective.

Exemplu:
Instrucțiunea ***LDAA*** $F03B (*Load Accumulator A*), cod maşină B6 F0 3B
- ($F03B) → A
- operandul sursă obținut prin adresare directă extinsă este octetul din locația de memorie cu adresa efectivă $F03B dată prin octeții doi şi trei din codul maşina al instrucțiunii;
- registrul acumulator A este destinația prin adresare inerentă.

2.4.5. ADRESAREA INDEXATĂ

Tehnica de adresare indexată (*indexed*) este utilizată într-o instrucțiune pentru accesul unui operand sursă sau destinație corespunzător unei locații de memorie a cărei adresă efectivă se obține prin adunarea la o adresă index conținută de unul dintre registrele X, Y, SP şi PC ale UCP a unui deplasament. Există variante de adresare indexată la care conținutul registrului utilizat pentru adresa index nu se modifică şi variante la care conținutul registrului se modifică în cadrul execuției instrucțiunii corespunzătoare prin incrementare sau decrementare pentru accesul secvențelor de date. Codul maşină al unei instrucțiuni care utilizează adresarea indexată conține, după octetul cod operație, un al doilea octet (*postbyte*) care precizează registrul utilizat

pentru adresa index, modul de modificare a conținutului acestui registru şi deplasamentul, conform tabelului 2.2.

Tabelul 2.2

Postbyte Code (xb)	**Source Code Syntax**	**Comments** **rr; 00 = X, 01 = Y, 10 = SP, 11 = PC**
rr0nnnnn	,r n,r –n,r	**5-bit constant offset** n = –16 to +15 r can specify X, Y, SP, or PC
111rr0zs	n,r –n,r	**Constant offset** (9- or 16-bit signed) z- 0 = 9-bit with sign in LSB of postbyte(s)–256 ≤ n ≤ 255 1 = 16-bit –32,768 ≤ n ≤ 65,535 if z = s = 1, 16-bit offset indexed-indirect (see below) r can specify X, Y, SP, or PC
111rr011	[n,r]	**16-bit offset indexed-indirect** rr can specify X, Y, SP, or PC –32,768 ≤ n ≤ 65,535
rr1pnnnn	n,–r n,+r n,r– n,r+	**Auto predecrement**, **preincrement**, **postdecrement**, or **postincrement**; p = pre-(0) or post-(1), n = –8 to –1, +1 to +8 r can specify X, Y, or SP (PC not a valid choice) +8 = 0111 … +1 = 0000 –1 = 1111 … –8 = 1000
111rr1aa	A,r B,r D,r	**Accumulator offset** (unsigned 8-bit or 16-bit) aa-00 = A 01 = B 10 = D (16-bit) 11 = see accumulator D offset indexed-indirect r can specify X, Y, SP, or PC
111rr111	[D,r]	**Accumulator D offset indexed-indirect** r can specify X, Y, SP, or PC

În funcție de modul de obținere a deplasamentului, variantele de adresare indexată fără modificarea conținutului registrului utilizat pentru adresa index sunt:

- adresare indexată cu deplasament de 5 biți (*indexed* 5-*bit offset*) **IDX**;
- adresare indexată cu deplasament de 9 biți (*indexed* 9-*bit offset*) **IDX1**;
- adresare indexată cu deplasament de 16 biți (*indexed* 16-*bit offset*) **IDX2**;
- adresare indexată cu deplasament dat de un registru acumulator (*indexed accumulator offset*) **IDX.**

În funcție de modul de modul de modificare a conținutului registrului utilizat pentru adresa index, variantele de adresare indexată sunt:

- adresare indexată cu predecrementare (*indexed pre-decrement*) **IDX**;
- adresare indexată cu preincrementare (*indexed pre-increment*) **IDX**;
- adresare indexată cu postdecrementare (*indexed post-decrement*) **IDX**;
- adresare indexată cu postincrementare (*indexed post-increment*) **IDX**;

Adresarea indexată cu deplasament de 5 biți

Adresarea indexată cu deplasament de 5 biți (*indexed 5-bit offset*) **IDX** utilizează pentru adresa index unul dintre registrele X, Y, SP sau PC şi un deplasament cu semn de 5 biți în cod complementul lui doi. Registrul utilizat şi valoarea deplasamentului sunt indicate în al doilea octet din codul maşină al instrucțiunii, conform tabelului 2.2.

Exemple:
Instrucțiunea ***LDAA*** 0,X (*Load Accumulator A*), cod maşină A6 00
- ((X)) → A;
- operandul sursă obținut prin adresare indexată cu deplasament de 5 biți este octetul din locația de memorie cu adresa efectivă egală cu adresa index din registrul X deoarece deplasamentul este 0;
- registrul acumulator A este destinația prin adresare inerentă.

Instrucțiunea ***STAB*** -8,Y (*Strore Accumulator B*), cod maşină 6B 58
- (B) → ((Y) + (-8)) ;
- operandul sursă este conținut în registrul acumulator B şi se obține prin adresare inerentă;
- destinația obținută prin adresare indexată cu deplasament de 5 biți este locația de memorie cu adresa efectivă calculată prin adunarea la adresa index din registrul Y a deplasamentului cu valoarea -8.

Adresarea indexată cu deplasament de 9 biți

Adresarea indexată cu deplasament de 9 biți (*indexed 9-bit offset*) **IDX1** utilizează pentru adresa index unul dintre registrele X, Y, SP sau PC şi un deplasament cu semn de 9 biți în cod complementul lui doi. Registrul utilizat şi bitul de semn al deplasamentului sunt indicate în al doilea octet din codul maşină al instrucțiunii, conform tabelului 2.2. Cei 8 biți mai puțin semnificativi ai deplasamentului sunt în al treilea octet din codul maşină al instrucțiunii.

Exemple:
Instrucțiunea ***LDAA*** $FF,X (*Load Accumulator A*), cod maşină A6 E0 FF
- ((X) + $00FF) → A;
- operandul sursă obținut prin adresare indexată cu deplasament de 9 biți este octetul din locația de memorie cu adresa efectivă calculată prin adunarea la adresa index din registrul X a deplasamentului cu valoarea 255;
- registrul acumulator A este destinația prin adresare inerentă.

Instrucțiunea ***STAB*** -20,Y (*Strore Accumulator B*), cod maşină 6B E9 EC
- (B) → ((Y) + (-20));

- operandul sursă este conținut în registrul acumulator B şi se obține prin adresare inerentă;
- destinația obținută prin adresare indexată cu deplasament de 9 biți este locația de memorie cu adresa efectivă calculată prin adunarea la adresa index din registrul Y a deplasamentului cu valoarea -20.

Adresarea indexată cu deplasament de 16 biți

Adresarea indexată cu deplasament de 16 biți (*indexed* 16-*bit offset*) **IDX2** utilizează pentru adresa index unul dintre registrele X, Y, SP sau PC indicat în al doilea octet din codul maşină al instrucțiunii, conform tabelului 2.2. Deplasamentul de 16 biți este dat prin doi octeți suplimentari în codul maşină al instrucțiunii şi poate fi considerat cu semn sau fără semn deoarece este dat prin acelaşi număr de biți ca şi adresa efectivă.

Exemple:
Instrucțiunea ***LDAA*** $102,X (*Load Accumulator A*), cod maşină A6 E2 01 02
- ((X) + $0102) → A;
- operandul sursă obținut prin adresare indexată cu deplasament de 16 biți este octetul din locația de memorie cu adresa efectivă calculată prin adunarea la adresa index din registrul X a deplasamentului cu valoarea 258;
- registrul acumulator A este destinația prin adresare inerentă.

Instrucțiunea ***STAB*** 256,Y (*Strore Accumulator B*), cod maşină 6B EA 01 00
- (B) → ((Y) + 256);
- operandul sursă este conținut în registrul acumulator B şi se obține prin adresare inerentă;
- destinația obținută prin adresare indexată cu deplasament de 16 biți este locația de memorie cu adresa efectivă calculată prin adunarea la adresa index din registrul Y a deplasamentului cu valoarea 256.

Adresarea indexată cu deplasament dat de un registru acumulator

Adresarea indexată cu deplasament dat de un registru acumulator (*indexed accumulator offset*) **IDX** utilizează pentru adresa index unul dintre registrele X, Y, SP sau PC şi un deplasament fără semn dat de un registru acumulator A, B sau D. Registrele utilizate pentru adresa index şi pentru deplasament sunt indicate în al doilea octet din codul maşină al instrucțiunii, conform tabelului 2.2.

Exemple:
Instrucțiunea ***LDAA*** B,X (*Load Accumulator A*), cod maşină A6 E5
- ((X) + (B)) → A;
- operandul sursă obținut prin adresarea indexată cu deplasament dat de un registru acumulator este octetul din locația de memorie cu adresa efectivă calculată prin

adunarea la adresa index din registrul X a deplasamentului conținut in registrul acumulator B;
- registrul acumulator A este destinația prin adresare inerentă.

Instrucțiunea ***STAB*** A,Y (*Strore Accumulator B*), cod maşină 6B EC
- (B) → ((Y) + (A));
- operandul sursă este conținut în registrul acumulator B şi se obține prin adresare inerentă;
- destinația obținută prin adresarea indexată cu deplasament dat de un registru acumulator este locația de memorie cu adresa efectivă calculată prin adunarea la adresa index din registrul Y a deplasamentului conținut în registrul acumulator A.

Adresarea indexată cu pre/post decrementare/incrementare

Variantele de adresare indexată pre/post decrementare/incrementare (*pre/post decrement/increment indexed adressing*) **IDX** utilizează pentru adresa index unul dintre registrele X, Y sau SP al cărui conținut se modifică în cadrul execuției instrucțiunii corespunzătoare prin incrementare sau decrementare pentru accesul secvențelor de date. Într-o instrucțiune care utilizează adresarea indexată cu predecrementare sau preincrementare, se modifică corespunzător conținutul registrului utilizat pentru adresa index care, după modificare, conține adresa efectivă utilizată pentru accesul locației de memorie. Într-o instrucțiune care utilizează adresarea indexată cu postdecrementare sau postincrementare, modificarea conținutului registrului utilizat pentru adresa index se face după generarea adresei efective. Registrul utilizat pentru adresa index, modul de modificare pre/post decrementare/incrementare a conținutului registrului utilizat pentru adresa index şi valoarea întreagă de la 1 la 8 cu care se face decrementarea/incrementarea sunt indicate în al doilea octet din codul maşină al instrucțiunii, conform tabelului 2.2.

Exemple:
Instrucțiunea ***LDAA*** 4,-X (*Load Accumulator A*), cod maşină A6 2C
- (X) – 4 → X; ((X)) → A;
- operandul sursă se obține prin adresare indexată cu predecrementare;
- conținutul registrului index X este decrementat cu 4;
- operandul sursă este octetul din locația de memorie cu adresa efectivă conținută în registrul index X după decrementare;
- registrul acumulator A este destinația prin adresare inerentă.

Instrucțiunea ***STAB*** 4,Y+ (*Strore Accumulator B*), cod maşină 6B 73
- (B) → ((Y)); (Y) + 4 → Y;
- operandul sursă este conținut în registrul acumulator B şi se obține prin adresare inerentă;
- destinația se obține prin adresare indexată cu postincrementare;

- destinația este locația de memorie cu adresa efectivă conținută de registrul index Y;
- conținutul registrului index Y este incrementat cu 4.

2.4.6. ADRESAREA INDEXATĂ-INDIRECTĂ

Tehnica de adresare indexată-indirectă (*indexed-indirect*) este utilizată într-o instrucțiune pentru accesul unui operand sursă sau destinație corespunzător unei locații de memorie a cărei adresă efectivă numită adresă indicator (*pointer*) se obține din memorie (două locații succesive) prin adresare indexată. Adresa index este dată de unul dintre registrele X, Y, SP sau PC ale UCP. În funcție de modul de obținere a deplasamentului, variantele de adresare indexată-indirectă sunt:

- adresare indexată-indirectă cu deplasament de 16 biți (*indexed-indirect* 16-*bit offset*) **[IDX2]**;
- adresare indexată-indirectă cu deplasament dat de registrul acumulator D (*indexed-indirect accumulator D offset*) **[D,IDX]**.

Adresarea indexată-indirectă cu deplasament de 16 biți

Adresarea indexată-indirectă cu deplasament de 16 biți (*indexed-indirect* 16-*bit offset*) **[IDX2]** utilizează pentru adresa index unul dintre registrele X, Y, SP sau PC indicat în al doilea octet din codul mașină al instrucțiunii, conform tabelului 2.2. Deplasamentul de 16 biți este dat prin doi octeți suplimentari în codul mașină al instrucțiunii şi poate fi considerat cu semn sau fără semn deoarece este dat prin același număr de biți ca şi adresa efectivă.

Exemple:
Instrucțiunea ***LDAA*** [10,X] (*Load Accumulator A*), cod mașină A6 E3 00 0A
- (((X) + 10)) → A;
- operandul sursă obținut prin adresare indexată-indirectă cu deplasament de 16 biți este octetul din locația de memorie de la adresa indicator conținută în memorie la adresa calculată prin adunarea la adresa index din registrul X a deplasamentului cu valoarea 10;
- registrul acumulator A este destinația prin adresare inerentă.

Instrucțiunea ***STAB*** [256,Y] (*Strore Accumulator B*), cod mașină 6B EB 01 00
- (B) → (((Y) + 256));
- operandul sursă este conținut în registrul acumulator B şi se obține prin adresare inerentă;
- destinația obținută prin adresare indexată-indirectă cu deplasament de 16 biți este locația de memorie cu adresa indicator conținută în memorie la adresa calculată prin adunarea la adresa index din registrul Y a deplasamentului cu valoarea 256.

Adresarea indexată-indirectă cu deplasament dat de registrul acumulator D

Adresarea indexată-indirectă cu deplasament dat de registrul acumulator D (*indexed-indirect accumulator D offset*) **[D,IDX]** utilizează pentru adresa index unul dintre registrele X, Y, SP sau PC indicat în al doilea octet din codul maşină al instrucțiunii, conform tabelului 2.2. Deplasamentul este dat de registrul acumulator D.

Exemple:
Instrucțiunea ***LDAA*** [D,X] (*Load Accumulator A*), cod maşină A6 E7
- (((X) + (D))) → A;
- operandul sursă obținut prin adresare indexată-indirectă cu deplasament dat de registrul acumulator D este octetul din locația de memorie de la adresa indicator conținută în memorie la adresa calculată prin adunarea la adresa index din registrul X a deplasamentului conținut in registrul acumulator D;
- registrul acumulator A este destinația prin adresare inerentă.

Instrucțiunea ***STAB*** [D,Y] (*Strore Accumulator B*), cod maşină 6B EF
- (B) → (((Y) + (D)));
- operandul sursă este conținut în registrul acumulator B şi se obține prin adresare inerentă;
- destinația obținută prin adresare indexată-indirectă cu deplasament dat de registrul acumulator D este locația de memorie cu adresa indicator conținută în memorie la adresa calculată prin adunarea la adresa index din registrul Y a deplasamentului conținut in registrul acumulator D.

Instrucțiunea ***JMP*** [D,PC] (*Jump*), cod maşină 05 FF
- (((PC) + 2 + (D))) → PC;
- operandul sursă obținut prin adresare indexată-indirectă cu deplasament dat de registrul acumulator D este cuvântul de 16 biți din memorie de la adresa indicator conținută în memorie la adresa calculată prin adunarea la adresa index din registrul PC a deplasamentului conținut in registrul acumulator D;
- registrul numărător de adrese PC este destinația prin adresare inerentă.

2.5. TRANSFERUL CONTROLULUI

Execuția în ordine succesivă (normală) a instrucțiunilor se realizează pe baza extragerilor codurilor acestora prin adresarea memoriei cu registrul numărător de adrese PC şi incrementarea succesivă a conținutului acestuia. Transferul controlului (*change in execution flow*) constă în comutarea execuției normale a instrucțiunilor dintr-un spațiu de adresare în altul al memoriei prin încărcarea registrului PC cu adresa de transfer.

Transferul controlului se realizează prin:

- salturi;
- apeluri de subrutine;
- reveniri din subrutine;
- întreruperi;
- inițializări.

Transferul controlului printr-un salt, apel de subrutină sau revenire din subrutină se produce ca urmare a execuției unei instrucțiuni corespunzătoare din programul în curs de execuție şi reprezintă un eveniment care face parte din contextul acestuia.

Transferul controlului printr-o întrerupere sau inițializare se numeşte excepție şi se produce, în general, ca urmare a unui eveniment extern contextului programului în curs de execuție.

2.5.1. SALTURI

Un salt se poate obține prin execuția unei instrucțiuni corespunzătoare care realizează încărcarea registrului PC cu adresa de salt. Transferul controlului într-o instrucțiune de salt poate fi în variantele absolut sau relativ şi necondiționat sau condiționat.

Instrucțiunea de salt *JMP* (*Jump Instruction*)

Execuția instrucțiunii de salt absolut necondiționat ***JMP*** constă în încărcarea registrului PC cu adresa de salt care coincide cu adresa efectivă obținută prin una dintre tehnicile de adresare directă extinsă, indexată sau indexată-indirectă.

Instrucțiuni de salt relativ scurt (*Short Branch Instructions*)

Saltul relativ se realizează prin adunarea la conținutul registrului numărător de adrese PC a unui deplasament reprezentat printr-un octet cu valori în cod complementul lui doi. Deplasamentul este dat prin al doilea octet din codul maşină al instrucțiunii de salt relativ scurt, conform tehnicii de adresare relativă. Saltul se realizează în intervalul de adrese deplasate cu -128 ÷ 127 locații față de adresa care urmează după cea corespunzătoare deplasamentului.

Instrucțiunile de salt relativ scurt în variantă condiționată utilizează pentru condițiile de salt biții de stare din registrul de condiții CCR, conform tabelului 2.3. Dacă instrucțiunea de salt este precedată de o instrucțiune de scădere sau comparare a doi operanzi **R** şi **M**, condițiile de salt reprezintă relații de tipul >, ≥, <, ≤ între cei doi operanzi considerați fără semn sau cu semn, conform tabelului 2.3.

Tabelul 2.3

Mnemonic	**Function**		**Equation or Operation**
Unary Branches			
BRA	Branch always		1 = 1
BRN	Branch never		1 = 0
Simple Branches			
BCC	Branch if carry clear		C = 0
BCS	Branch if carry set		C = 1
BEQ	Branch if equal		Z = 1
BMI	Branch if minus		N = 1
BNE	Branch if not equal		Z = 0
BPL	Branch if plus		N = 0
BVC	Branch if overflow clear		V = 0
BVS	Branch if overflow set		V = 1
Unsigned Branches			
		Relation	
BHI	Branch if higher	R > M	C + Z = 0
BHS	Branch if higher or same	R ≥ M	C = 0
BLO	Branch if lower	R < M	C = 1
BLS	Branch if lower or same	R ≤ M	C + Z = 1
Signed Branches			
BGE	Branch if greater than or equal	R ≥ M	N ⊕ V = 0
BGT	Branch if greater than	R > M	Z + (N ⊕ V) = 0
BLE	Branch if less than or equal	R ≤ M	Z + (N ⊕ V) = 1
BLT	Branch if less than	R < M	N ⊕ V = 1

Instrucțiuni de salt relativ lung (*Long Branch Instructions*)

Saltul relativ se realizează prin adunarea la conținutul registrului numărător de adrese PC a unui deplasament reprezentat printr-un cuvânt de 16 biți cu valori în cod complementul lui doi. Deplasamentul este dat prin doi octeți din codul maşină al

instucțiunii de salt relativ lung, conform tehnicii de adresare relativă. Saltul se realizează în intervalul de adrese deplasate cu -32768 ÷ 32767 locații față de adresa care urmează după cea corespunzătoare deplasamentului. Rezultă că saltul se poate face la orice locație corespunzătoare unei memorii de 64 Kocteți.

Instrucțiunile de salt relativ lung utilizează aceleaşi condiții de salt ca şi instrucțiunile de salt relativ scurt, conform tabelului 2.4.

Tabelul 2.4

Mnemonic	Function	Equation or Operation
Unary Branches		
LBRA	Long branch always	1 = 1
LBRN	Long branch never	1 = 0
Simple Branches		
LBCC	Long branch if carry clear	C = 0
LBCS	Long branch if carry set	C = 1
LBEQ	Long branch if equal	Z = 1
LBMI	Long branch if minus	N = 1
LBNE	Long branch if not equal	Z = 0
LBPL	Long branch if plus	N = 0
LBVC	Long branch if overflow clear	V = 0
LBVS	Long branch if overflow set	V = 1
Unsigned Branches		
LBHI	Long branch if higher	C + Z = 0
LBHS	Long branch if higher or same	C = 0
LBLO	Long branch if lower	Z = 1
LBLS	Long branch if lower or same	C + Z = 1
Signed Branches		
LBGE	Long branch if greater than or equal	N ⊕ V = 0
LBGT	Long branch if greater than	Z + (N ⊕ V) = 0
LBLE	Long branch if less than or equal	Z + (N ⊕ V) = 1
LBLT	Long branch if less than	N ⊕ V = 1

Instrucțiuni de salt condiționat de biți din memorie (*Bit Condition Branch Instructions*)

Aceste instrucțiuni, tabelul 2.5, sunt de salt relativ scurt în care condițiile de salt sunt date de nivelurile logice ale unor biți dintr-o locație de memorie obținută prin una dintre tehnicile de adresare directă scurtă, directă extinsă sau indexată. Codul

maşină al unei instrucțiuni de salt condiționat de biți din memorie conține un octet mască *mm* care selectează prin niveluri logice 1 biții octetului din memorie care sunt testați pentru condiția de salt. Condiția de salt se realizează dacă toți biții selectați sunt la nivel logic 0, instrucțiunea ***BRCLR*** sau la nivel logic 1, instrucțiunea ***BRSET***. De asemenea, codul maşină al unei instrucțiuni de salt condiționat de biți din memorie conține deplasamentul în poziția ultimului octet.

Tabelul 2.5

Mnemonic	Function	Equation or Operation
BRCLR	Branch if selected bits clear	(M) • (mm) = 0
BRSET	Branch if selected bits set	$(\overline{M})$ • (mm) = 0

Instrucțiuni de salt pentru bucle (*Loop Primitive Branch Instructions*)

Aceste instrucțiuni, tabelul 2.6, sunt de salt relativ scurt (deplasament pe 9 biți în cod complementul lui doi) în care condiția de salt este dată de conținutul zero sau diferit de zero al unui registru al unității centrale de prelucrare A, B, D, X, Y sau SP. Conținutul registrului testat pentru condiția de salt poate fi modificat înainte de testare prin decrementare sau incrementare cu o unitate ceea ce permite contorizarea execuțiilor unei bucle (unui grup de instrucțiuni). Al doilea octet din codul maşină al unei instrucțiuni de salt pentru bucle conține biți pentru selecția registrului testat şi bitul de semn al deplasamentului. Ceilalți opt biți ai deplasamentului sunt dați în al treilea octet din codul maşină al instrucțiunii.

Tabelul 2.6

Mnemonic	Function	Equation or Operation
DBEQ	Decrement counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	(counter) – 1 ⟹ counter If (counter) = 0, then branch; else continue to next instruction
DBNE	Decrement counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	(counter) – 1 ⟹ counter If (counter) not = 0, then branch; else continue to next instruction
IBEQ	Increment counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	(counter) + 1 ⟹ counter If (counter) = 0, then branch; else continue to next instruction
IBNE	Increment counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	(counter) + 1 ⟹ counter If (counter) not = 0, then branch; else continue to next instruction
TBEQ	Test counter and branch if = 0 (counter = A, B, D, X,Y, or SP)	If (counter) = 0, then branch; else continue to next instruction
TBNE	Test counter and branch if ≠ 0 (counter = A, B, D, X,Y, or SP)	If (counter) not = 0, then branch; else continue to next instruction

2.5.2. APELURI DE SUBRUTINE

Apelul unei subrutine necesită salvarea adresei de revenire din registrul numărător de adrese PC în memoria stivă şi apoi încărcarea registrului PC cu adresa de început a subrutinei.

Instrucţiunea de apel de subrutină *BSR* (*Branch to Subroutine*)

Instrucţiunea de apel de subrutină ***BSR***, tabelul 2.7, realizează transferul relativ al controlului prin adunarea la conţinutul registrului numărător de adrese PC a unui deplasament reprezentat printr-un octet cu valori în cod complementul lui doi. Deplasamentul este dat prin al doilea octet din codul maşină al instucţiunii de apel de subrutină, conform tehnicii de adresare relativă. Adresa de început a subrutinei este plasată în intervalul de adrese deplasate cu -128 ÷ 127 locaţii faţă de adresa care urmează după cea corespunzătoare deplasamentului.

Tabelul 2.7

Mnemonic	Function	Operation
BSR	Branch to subroutine	$SP - 2 \Longrightarrow SP$ $RTN_H : RTN_L \Longrightarrow M_{(SP)} : M_{(SP+1)}$ Subroutine address $\Longrightarrow$ PC
CALL	Call subroutine in Expanded Memory	$SP - 2 \Longrightarrow SP$ $RTN_H : RTN_L \Longrightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Longrightarrow SP$ $(PPAGE) \Longrightarrow M_{(SP)}$ Page $\Longrightarrow$ PPAGE Subroutine address $\Longrightarrow$ PC
JMP	Jump	Address $\Longrightarrow$ PC
JSR	Jump to subroutine	$SP - 2 \Longrightarrow SP$ $RTN_H : RTN_L \Longrightarrow M_{(SP)} : M_{(SP+1)}$ Subroutine address $\Longrightarrow$ PC
RTC	Return from call	$M_{(SP)} \Longrightarrow$ PPAGE $SP + 1 \Longrightarrow SP$ $M_{(SP)} : M_{(SP+1)} \Longrightarrow PC_H : PC_L$ $SP + 2 \Longrightarrow SP$
RTS	Return from subroutine	$M_{(SP)} : M_{(SP+1)} \Longrightarrow PC_H : PC_L$ $SP + 2 \Longrightarrow SP$

Instrucţiunea de apel de subrutină *JSR* (*Jump to Subroutine*)

Instrucţiunea de apel de subrutină ***JSR***, tabelul 2.7, realizează transferul absolut al controlului prin încărcarea registrului PC cu adresa de început a subrutinei care coincide cu adresa efectivă obţinută prin una dintre tehnicile de adresare directă scurtă, directă extinsă, indexată sau indexată-indirectă.

Instrucțiunea de apel de subrutină *CALL* (*Call Subroutine in Expanded Memory*)

Instrucțiunea ***CALL***, tabelul 7, realizează apelul unei subrutine a cărei adresă de început este plasată în memoria de program extinsă adresabilă cu registrul PC şi cu registrul de 8 biți **PPAGE** (*Program Page Index Register*). Rezultă că salvarea adresei de revenire şi încărcarea adresei de început a subrutinei se referă la registrul PC precum şi la registrul PPAGE. Apelul subrutinei de tipul transfer absolut al controlului se realizează prin încărcarea registrului PC cu adresa efectivă obținută prin una dintre tehnicile de adresare directă extinsă, indexată sau indexată-indirectă şi încărcarea registrului PPAGE cu adresa paginii de memorie. În cazul utilizării tehnicilor de adresare directă extinsă şi indexată, adresa paginii este conținută într-un octet din codul maşină al instrucțiunii. În cazul utilizării tehnicii de adresare indexată-indirectă, adresa paginii este conținută în memorie după adresa efectivă care se încarcă în registrul PC.

2.5.3. REVENIRI DIN SUBRUTINE

Revenirea dintr-o subrutină necesită extragerea adresei de revenire din memoria stivă şi încărcarea acesteia în registrul numărător de adrese PC.

Instrucțiunea de revenire din subrutină *RTS* (*Return from Subroutine*)

Instrucțiunea de revenire din subrutină ***RTS***, tabelul 7, se utilizează pentru revenire dintr-o subrutină apelată cu o instrucțiune ***BSR*** sau ***JSR***.

Instrucțiunea de revenire din subrutină în memoria extinsă ***RTC*** (*Return from Call*)

Instrucțiunea de revenire din subrutină în memoria extinsă ***RTS***, tabelul 2.7, se utilizează pentru revenire dintr-o subrutină apelată cu o instrucțiune ***CALL***.

2.5.4. EXCEPȚII

Excepțiile sunt evenimente externe contextului programului în curs de execuție care sunt precizate prin inițializări şi cereri de întrerupere. Tratarea unei excepții se realizează prin transferul controlului. Astfel, fiecărei excepții îi corespunde un vector excepție de 16 biți care reprezintă adresa de început a subrutinei de tratare a excepției. Aceşti vectori sunt conținuți în memorie la adresele indicate în tabelul 2.8, unde **IVBR** (*Interrupt Vector Base Register*) este registrul bază vectori întrerupere al microcontrolerului. Rezultă că tratarea unei excepții se realizează prin apelul şi execuția subrutinei a cărei adresă de început este dată de vectorul excepție corespunzător.

Tabelul 2.8

Adresele vectorilor excepție [*h*]	Excepție
FFFE	Inițializări sistem
FFFC	Inițializare monitor tact
FFFA	Inițializare temporizator *watchdog*
(IVBR) : F8	Întrerupere prin instrucțiunea *TRAP*
(IVBR) : F6	Întrerupere prin instrucțiunea *SWI*
(IVBR) : F4	Întrerupere nemascabilă. Pinul /XIRQ
(IVBR) : F2	Întrerupere mascabilă. Pinul /IRQ
(IVBR) : (F0 ÷ 12)	Întreruperi mascabile. Perifericele interne
(IVBR) : 10	Întrerupere mascabilă spurious

Inițializări

Inițializările corespunzătoare unității centrale de prelucrare CPU12 sunt:
- inţializări sistem;
- inițializare monitor tact;
- inițializare execuție incorectă aplicație.

Inițializările sistem se generează la conectarea tensiunii de alimentare (*power-on reset*), prin pinul /RESET, la scăderea tensiunii de alimentare (*low voltage reset*) şi la generarea unei adrese incorecte (*illegal address reset*).

Inițializarea monitor tact (*clock monitor reset*) se generează la scăderea frecvenței de tact sub o anumită valoare.

Inițializarea execuție incorectă aplicație (*Computer Operating Properly watchdog reset*) se generează de un circuit temporizator (*watchdog timer*) la anularea conținutului numărătorului. Programul aplicație trebuie să încarce periodic numărătorul pentru evitarea anulării conținutului acestuia, deci pentru evitarea inițializării. Rezultă că o astfel de inițializare se generează în cazul execuției incorecte a aplicației.

Întreruperi

Tratarea unei întreruperi începe după execuția completă a unei instrucțiuni dintr-un program şi se realizează prin apelul şi execuția subrutinei a cărei adresă de început este dată de vectorul întrerupere (excepție) corespunzător, tabelul 2.8. Subrutina de întrerupere se termină cu o instrucțiune de revenire din întrerupere ***RTI*** (*Return from Interrupt*) care determină reluarea execuției programului întrerupt cu instrucțiunea următoare celei după a cărei execuție a început tratarea întreruperii. Pentru reluarea corectă a execuției programului întrerupt este necesară salvarea contextului acestuia, înainte de execuția subrutinei de întrerupere şi respectiv restaurarea contextului acestuia, inainte de reluarea execuției programului întrerupt. Pentru unitatea centrală de prelucrare CPU12, contextul programului este dat de conținuturile registrelor UCP, inclusiv a registrului numărător de adrese PC care conține adresa de revenire. Salvarea şi restaurarea registrelor UCP se face în/din memoria stivă. Astfel, tratarea unei întreruperi începe cu generarea de către logica de

control întreruperi a microcontrolerului a adresei vectorului întrerupere. De la această adresă UCP extrage vectorul întrerupere şi continuă cu operaţii de salvare a registrelor UCP în memoria stivă. Aceste operaţii se întrepătrund cu extrageri de cuvinte coduri instrucţiuni din subrutina de întrerupere pentru completarea memoriei de instrucţiuni. În tabelul 2.9 se prezintă conţinutul memoriei stivă după salvarea registrelor UCP, unde (SP) este conţinutul registrului indicator de stivă după salvarea registrelor UCP şi RTN este adresa de revenire. Restaurarea registrelor UCP se realizează prin execuţia instrucţiunii de revenire din întrerupere ***RTI***.

Tabelul 2.9

Adresă memorie stivă	**Registrele UCP**
SP	CCR_H : CCR_L
SP + 2	Y
SP + 4	X
SP + 6	B : A
SP + 8	PC (RTN_H : RTN_L)

Întreruperile corespunzătoare unităţii centrale de prelucrare CPU12 sunt:
- întrerupere prin instrucţiunea *TRAP*;
- întrerupere prin instrucţiunea *SWI*;
- întrerupere nemascabilă prin pinul /XIRQ;
- întrerupere mascabilă prin pinul /IRQ;
- întreruperi mascabile de la perifericele interne.

2.6. CICLURI DE ACCES LA MEMORIE

Execuţia unei instrucţiuni necesită cicluri de acces ale UCP la memorie pentru extragere coduri instrucţiuni şi, în general, pentru citire şi scriere date. Un ciclu de acces la memorie este notat cu o literă care simbolizează informaţia transferată între UCP şi memorie. O literă mare este asociată pentru transferul unui cuvânt de 16 biţi şi o literă mică este asociată pentru transferul unui cuvânt de 8 biţi.

În documentaţia de prezentare a instrucţiunilor unui microcontroler din familia HCS12X se indică, în coloana ***access detail***, succesiunile de cicluri de acces la memorie corespunzătoare execuţiei instrucţiunilor. La viteza maximă de lucru care se poate obţine, de exemplu, utilizând pentru date memoria internă RAM, durata unui ciclu de acces la memorie este egală cu durata unei perioade a semnalului cu frecvenţa ciclurilor f_{BUS}. Rezultă posibilitatea de a calcula durata de execuţie a unei instrucţiuni. În cele ce urmează se prezintă unele tipuri de cicluri de acces la memorie.

P -ciclu de extragere coduri instrucţiuni (*program word fetch*)

Într-un ciclu de extragere coduri instrucţiuni UCP citeşte din memorie un cuvânt de 16 biţi aliniat la o adresă pară (doi octeţi de la adrese succesive, prima adresă fiind pară) care se introduce în etajul 1 al memoriei de instrucţiuni.

f -ciclu inactiv (*free cycle*)

Într-un ciclu inactiv UCP nu accesează memoria.

O -ciclu opţional de extragere coduri instrucţiuni (*optional program word fetch*)

Un ciclu opţional de extragere coduri instrucţiuni este un ciclu **P,** dacă instrucţiunea este plasată în memorie începând de la o adresă impară şi are un număr impar de octeţi în codul maşină şi este un ciclu **f,** în toate celelalte cazuri.

Exemple

Execuţia instrucţiunii *DEX* (*Decrement Index Register X*) care are codul maşină de un octet se realizează într-un ciclu opţional **O**. Dacă această instrucţiune este plasată în memorie începând de la o adresă impară, prin execuţia acesteia se eliberează un etaj din memoria de instrucţiuni şi ciclul opţional este un ciclu de extragere coduri instrucţiuni **P.** Dacă această instrucţiune este plasată în memorie începând de la o adresă pară, prin execuţia acesteia nu se eliberează nici un etaj din memoria de instrucţiuni şi ciclul opţional este un ciclu inactiv **f.**

Execuţia instrucţiunii *LDAA* #opr8i (*Load Accumulator A*) care are codul maşină din doi octeţi se realizează într-un ciclu de extragere coduri instrucţiuni **P**, deoarece se eliberează un singur etaj din memoria de instrucţiuni în ambele variante de plasare în memorie a instrucţiunii.

Instrucţiunile unui microcontroler din familia HCS12X conţin codul operaţiei în primul octet din codul maşină (*page*1 *instruction opcodes*) sau în al doilea octet (*page*2 *instruction opcodes*). Toate instrucţiunile cu codul operaţiei în al doilea octet au primul octet din codul maşină 18*h* (*prebyte*). Din punctul de vedere al accesului

UCP la memorie, acest octet este considerat ca o instrucțiune specială cu un octet în codul maşină.

Exemple

Execuția instrucțiunii *TAB* (*Transfer from Accumulator A to Accumulator B*) care are codul maşină din doi octeți cu primul octet 18*h* se realizează în două cicluri opționale **O,** deoarece, din punctul de vedere al accesului UCP la memorie, se consideră execuția a două instrucțiuni succesive de câte un octet în codul maşină. Dacă instrucțiunea *TAB* este plasată în memorie începând de la o adresă impară, primul ciclu opțional este un ciclu de extragere coduri instrucțiuni **P** şi al doilea ciclu opțional este un ciclu inactiv **f**. Dacă instrucțiunea *TAB* este plasată în memorie începând de la o adresă pară, primul ciclu opțional este un este un ciclu inactiv **f** şi al doilea ciclu opțional ciclu de extragere coduri instrucțiuni **P**.

r -ciclu de citire dată de 8 biți (8-*bit data read*)
R -ciclu de citire dată de 16 biți (16-*bit data read*)
w -ciclu de scriere dată de 8 biți (8-*bit data write*)
W -ciclu de scriere dată de 16 biți (16-*bit data write*)

Exemple

Execuția instrucțiunii *LDAA* opr16a (*Load Accumulator A*) care are codul maşină din trei octeți se realizează în trei cicluri: **rPO**.

Execuția instrucțiunii *LDD* opr16a (*Load Double Accumulator*) care are codul maşină din trei octeți se realizează în trei cicluri: **RPO**.

Execuția instrucțiunii *STAA* opr16a (*Store Accumulator A*) care are codul maşină din trei octeți se realizează în trei cicluri: **PwO**.

Execuția instrucțiunii *STD* opr16a (*Store Double Accumulator*) care are codul maşină din trei octeți se realizează în trei cicluri: **PWO**.

s -ciclu de scriere dată de 8 biți în stivă (*stack* 8-*bit data*)
S -ciclu de scriere dată de 16 biți în stivă(*stack* 16-*bit data*)
u -ciclu de citire dată de 8 biți din stivă(*unstack* 8-*bit data*)
U -ciclu de citire dată de 16 biți din stivă(*unstack* 16-*bit data*)

Exemple

Execuția instrucțiunii *PSHB* (*Push B onto Stack*) care are codul maşină dintr-un octet se realizează în două cicluri: **Os**.

Execuția instrucțiunii *PSHD* (*Push Double Accumulator onto Stack*) care are codul maşină dintr-un octet se realizează în două cicluri: **OS**.

Execuția instrucțiunii *PULB* (*Pull B from Stack*) care are codul maşină dintr-un octet se realizează în trei cicluri: **ufO**.

Execuția instrucțiunii *PULD* (*Pull Double Accumulator from Stack*) care are codul maşină dintr-un octet se realizează în trei cicluri: **UfO**.

2.7. MEMORIA

Unitatea centrală de prelucrare CPU12 utilizează tehnica paginării cu posibilitatea adresării unei memorii globale de 8 Mocteți corespunzătoare unei magistrale de adresare de 23 de biți. Pentru adresarea paginilor memoriei, microcontrolerul conține registre index de pagină. Conținuturile acestor registre se pot stabili prin programare pentru configurarea unei memorii locale de 64 Kocteți corespunzătoare unei magistrale de adresare de 16 de biți. Harta memoriilor locale şi globale ale unui microcontroler din familia HCS12X este prezentată în figura 2.4. Această figură indică intervalele de adresare corespunzătoare memoriilor fereastră (*window*) care se selectează prin paginare şi registrele index de pagină corespunzătoare.

Registrul index de pagină RAM, de 8 biți, **RPAGE** (*RAM Page Index Register*) selectează în intervalul de adresare 1000*h*÷1FFF*h* de 4 Kocteți din memoria locală a uneia dintre cele 256 de pagini de câte 4 Kocteți din intervalul de adresare 0÷0FFFFF*h* corespunzător memoriei globale. Pagina 0 a memoriei RAM (RPAGE=0) include zona de memorie cu registrele microcontrolerului. Paginile 254 şi 255 ale memoriei RAM (RPAGE=FE*h* şi RPAGE=FF*h*) coincid cu cele două pagini din intervalul de adresare 2000*h*÷3FFF*h* ale memoriei locale (8K RAM). Prin inițializare registrul RPAGE se încarcă cu octetul FD*h* pentru selecția paginii 253 din memoria globală. Generarea adresei globale din adresa paginii RPAGE şi adresa locală dată de UCP se realizează prin concatenare, conform figurii 2.5.

Registrul index de pagină EEPROM, de 8 biți, **EPAGE** (*EEPROM Page Index Register*) selectează în intervalul de adresare 0800*h*÷0BFF*h* de 1 Koctet din memoria locală a uneia dintre cele 256 de pagini de câte 1 Koctet din intervalul de adresare 100000*h*÷13FFFF*h* corespunzător memoriei globale. Pagina 255 a memoriei EEPROM (EPAGE=FF*h*) coincide cu pagina din intervalul de adresare 0C00*h*÷0FFF*h* a memoriei locale (1K EEPROM). Prin inițializare registrul EEPAGE se încarcă cu octetul FE*h* pentru selecția paginii 254 din memoria globală. Generarea adresei globale din adresa paginii EEPAGE şi adresa locală dată de UCP se realizează prin concatenare, conform figurii 2.6.

Registrul index de pagină FLASH, de 8 biți, **PPAGE** (*Program Page Index Register*) selectează în intervalul de adresare 8000*h*÷BFFF*h* de 16 Kocteți din memoria locală a uneia dintre cele 256 de pagini de câte 16 Kocteți din intervalul de adresare 400000÷7FFFFF*h* corespunzător memoriei globale. Paginile 253 şi 255 ale memoriei FLASH (PPAGE=FD*h* şi PPAGE=FF*h*) coincid cu cele două pagini din intervalele de adresare 4000*h*÷7FFF*h* respectiv C000*h*÷FFFF*h* ale memoriei locale (*unpaged* 16K FLASH). Prin inițializare registrul PPAGE se încarcă cu octetul FE*h* pentru selecția paginii 254 din memoria globală. Generarea adresei globale din adresa paginii PPAGE şi adresa locală dată de UCP se realizează prin concatenare, conform figurii 2.7.

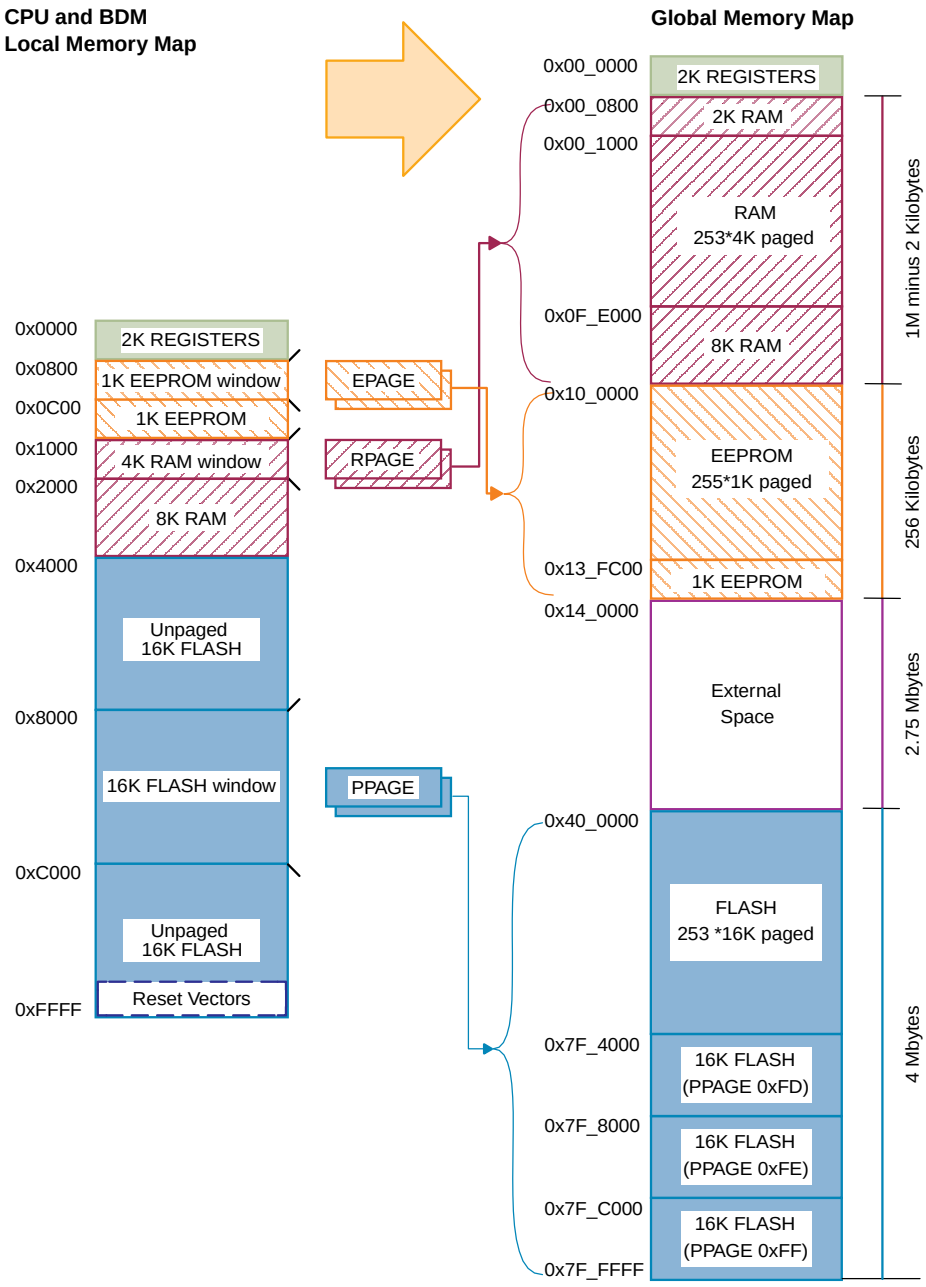


Fig. 2.4. Harta memoriilor locale şi globale ale unui microcontroler HCS12X.

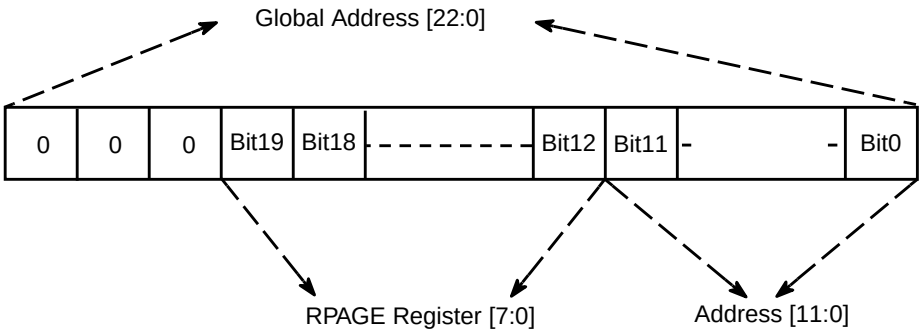


Fig. 2.5. Generarea adresei globale pentru memoria RAM.

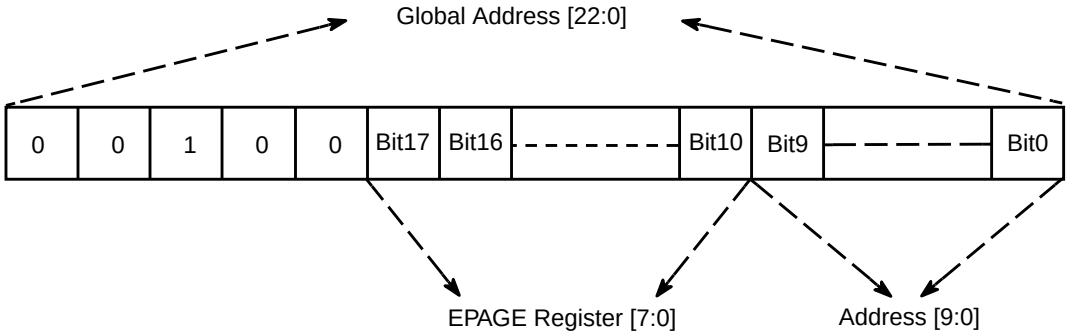


Fig. 2.6. Generarea adresei globale pentru memoria EEPROM.

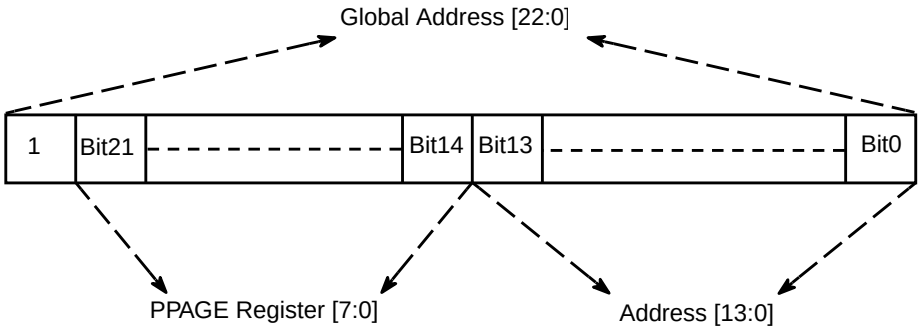


Fig. 2.7. Generarea adresei globale pentru memoria FLASH.

Memoria globală a microcontrolerului MC9S12XDP512, figura 2.8, cuprinde:

- memorie RAM, 32 Kocteți, în intervalul de adresare RAM_LOW=0F8000*h* ÷0FFFFF, 8 pagini de câte 4 Kocteți, RPAGE= F8*h*÷FF*h*;

- memorie EEPROM, 4 Kocteți, în intervalul de adresare EEPROM_LOW= 13F000*h*÷13FFFF, 4 pagini de câte 1 Koctet, EPAGE=FC*h*÷FF*h*;

- memorie flash EEPROM, 512 Kocteți, în intervalul de adresare FLASH_LOW=780000*h*÷7FFFFF, 32 pagini de câte 16 Kocteți, PPAGE=E0*h*÷FF*h*.

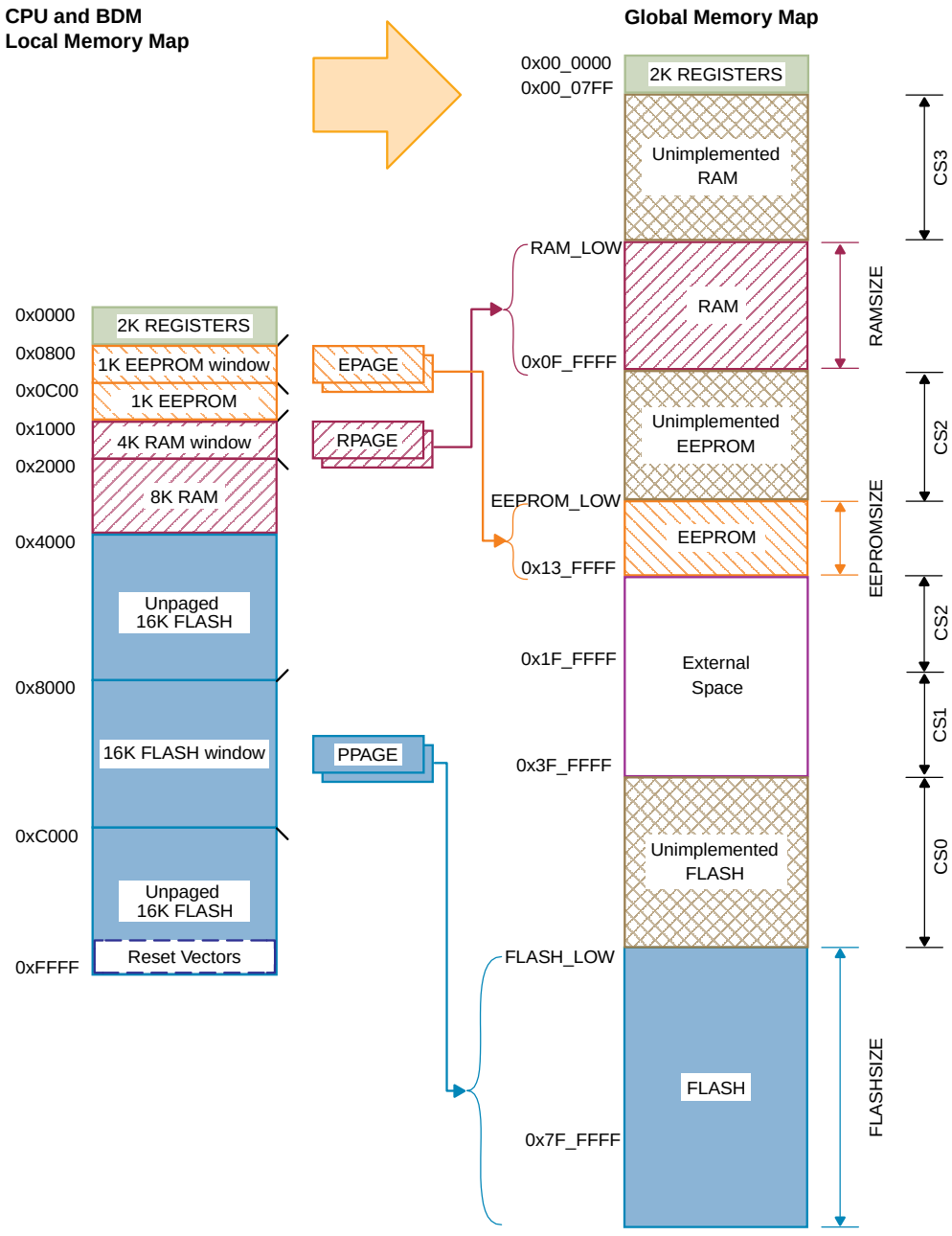
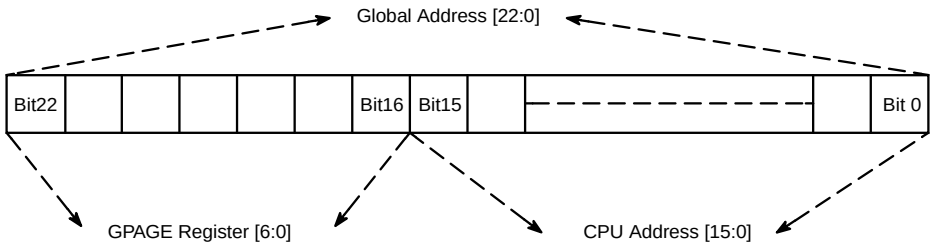


Fig. 2.8. Harta memoriilor locale şi globale ale microcontrolerului MC9S12XDP512.

În general, pentru accesul la memorie, UCP generează adrese efective (locale) de 16 biți utilizate pentru accesul memoriei locale configurată cu registrele index de pagină RPAGE, EPAGE şi PPAGE. Microcontrolerele HCS12X permit accesul la memoria globală prin tehnica de adresare numită globală care utilizează registrul index de pagină memorie globală, de 8 biți, **GPAGE** (*Global Page Index Register*). Această tehnică de adresare este utilizată în instrucțiuni specifice de transfer între registrele UCP şi memorie. Aceste instrucțiuni au mnemonice începând cu litera G. Generarea adresei globale din adresa paginii GPAGE şi adresa locală dată de UCP se realizează prin concatenare, conform figurii 2.9.



.

Fig. 2.9. Generarea adresei globale cu registrul GPAGE.

Aplicație

1. Să se calculeze intervalele de adresare globală corespunzătoare paginilor 253 din memoria RAM, EEPROM şi FLASH. Să se indice conținuturile registrului GPAGE pentru accesul în aceste intervale cu instrucțiunile specifice care tehnica de adresare globală.

- RAM: 0FD000*h* ÷ 0FDFFF*h*; GPAGE = 0F*h*;
- EEPROM: 13F400*h* ÷ 13F7FF*h*; GPAGE = 13*h*;
- RAM: 7F4000*h* ÷ 7F7FFF*h*; GPAGE = 7F*h*.