**CAP. 4**

**CALEA DE CONTROL**

|  |
| --- |
| **4.1 FUNCŢIA UNITĂŢII DE CONTROL**  Divizarea procesorului în două părţi distincte: *Calea de Date* şi *Calea de Control* (Unitatea de Control)nu este o opţiune numai sub aspect didactic,este o opţiune şi din punct de vedere al proiectării. La un procesor cu set de instrucţiuni, din punct de vedere al organizării şi funcţionării, apar două acţiuni distincte, care trebuie privite separat: secvenţierea instrucţiunii şi execuţia instrucţiunii.   * *Secvenţierea instrucţiunilor* este acţiunea realizată de Unitatea de Control (UC), acțiunea prin care se   selectează instrucţiunea următoare ce trebuie executată, sau altfel spus, cum se trece la instrucţiunea următoare și aceasta poate fi: 1. fie în ordinea de creştere a adreselor în sensul de creştere al numerelor naturale (regulă implicită, deci nu se mai specifică în instrucţiune);fie 2. după o altă regulă, care nu păstrează succesiunea în accesarea adreselor de creştere în sensul numerelor naturale şi care trebuie specificată în instrucţiune în modul următor:  1. PC← PC+k, k fiind pasul de aliniere a instrucţiunilor în memorie, (ADRESA)modulo k = 0;  2. PC←Adresă: – fixă | – salt necondiţionat  – calculată => | – salt condiţionat  – introdusă(vector) | – Call/Return  | – Excepţii (hard, soft)  Secvenţialiatea supervizată de unitatea de control, pentru realizarea traseului prin memorie (zona text) prin extragerea instrucţiunilor (subciclul Fetch), se contiună apoi şi prin secvenţierea etapelor în executarea înstrucţiunilor (subciclul Execuţie) în calea de date.   * *Execuţia* este acţiunea efectuată în calea de date sub controlul căii de control pentru realizarea funcţiei/operaţiei specificate în codul instrucţiunii. Deoarece calea de date, cu un repertoriu limitat de componente (unităţi funcţionale (**UF**), bloc de registre, magistrale, buffere, shiftere) trebuie să execute întreg setul de instrucţiuni, în acest scop calea de control pentru execuţia fiecărui instrucţiuni realizează:   1. selectarea componentele necesare din calea de date;  2. configurarea traseele dintre aceste componente selectate;  3. succesiunea transferurilor pe traseele dintre componente selectate încât să realizeze în hardware   execuţia instrucţiunii.  Aceste selectări de componente, configurări de trasee şi secvenţierea transferurilor în calea de date se realizează prin semnalele generate de unitatea de control (în urma decodificării OPCODE şi pe baza semnalelor primite din calea de date).  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **EXEMPLUL 4.1** Pentru o cale de date în care sunt trei registre (R1, R2, R3), o unitate funcţională, UF, şi un registru acumulator, A, cu transfer pe o singură magistrală (Figura a) să se stabilească configuraţia traseelor şi punctele de aplicare a semnalelor de control.  Traseele între elementele componente şi punctele de control corespunzătoare pentru efectuarea transferurilor pe aceste trasee sunt desenate în figura b. Traseele sau ales ţinând cont că fiecare registru trebuie să comunice (”să aibă ieşire”) cu unitatea funcţională pe singura magistrală (evident, dacă ar fi două sau trei magistrale transferurile s-ar putea face mai uşor, în paralel ). La unitatea funcţională trebuie să poată fi adusă informaţia din fiecare element, iar rezultatul să poată fi înscris în oricare element. Registrul acumulator trebuie să înscrie în UF şi să fie înscris din UF, precum şi de pe magistrală. Rezultă traseele şi punctele de control din figura b. În fiecare punct de control, figura c, semnalul Data de transmis este aplicat pe o poartă AND împreună cu semnalul de control (care închide sau deschide poarta), semnal care trebuie generat de unitatea de control. Deoarce la UF se aplică informaţie de la mai multe surse, selectarea uneia dintre aceste surse se realizează prin multiplexor. S-au mai introdus două semnale de control şi pentru programarea operaţiilor executabile pe UF (deci în acest caz UF cu cele două semnale se pot patru operaţii). |

|  |
| --- |
| F510E996 |
| Semnalele de control generate de calea de control pentru coordonarea acţiunilor din calea de date se supun tripletului de acțiuni : ”unde” , ”când” şi ”cum”.  Prin acţiune de ”**cum**” semnalele de control trebuie să configureze/selecteze cu elementele din calea de date (magistrale, registre, unităţi funcţionale) un traseu încât să se poată modela hardware starea respectivă în execuţia instrucţiunii (operaţia).  Prin acţiune de ”unde” semnalul de control trebuie să realizeze comanda într-un anumit punct din calea de date (locul).  Prin acţiune de ” când ” (în ce moment) semnalul de control trebuie să devină activ într-un anumit moment şi să fie activ pe o anumită durată (secvenţialitatea).  Semnalele de control, în general, sunt sincrone cu semnal de ceas şi pot fi:  1. semnale de strob, care au o durată scurtă (impuls) servind realizarea unei acţiuni, de exemplu o   înscriere într-un registru;  2. semnale active pe palier, care au durata palierului de unu sau mai multe perioade de ceas şi servesc în   general pentru configurare de trasee, de exemplu cât este deschisă o poartă de pe un traseu.  Sinteza semnalelor de control are la bază o diagrama de stări (este o organigramă de tip ASM - Algorithmic State Machine, diagramă FSM - Finite State Machine), care descrie grafic etapele pentru intrepretarea unei  instrucţiuni (Fetch + Execuţie), un model generic/didactic fiind reprezentat în figura următoare. În această diagramă de stări se disting cele două subcicluri de Fetch şi Execuţie, fiecare dintre subcicluri fiind constituit din UNA sau mai multe etape/faze; subciclul de Fetch având o singură etapă iar cel de Execuţie patru etape (accesare operanzi, realizarea operaţiei, acces la memorie (pentru înscriere sau citire date) şi înscrierea rezultatului produs de instrucţiune). Fiecare etapă, la rândul său, conţine una sau mai multe stări, uzual o stare este pe durata unui tact de ceas, deci o etapă se consumă pe durata unei sau a mai multor perioade de ceas (unele stări pot consuma mai multe perioade de ceas, în acest caz se rămâne mai multe tacte în starea respectiv) |

|  |
| --- |
| **C:\Users\Owner\Documents\Diagramă ASM.tif** |
| În fiecare stare i ( = 1,2,3,....,k) a diagramei de stări, pe durata unui tact, se realizează o anumită funcţie, Fi, iar pentru *interpretarea instrucţiunii* pe durata ciclului instrucțiune (Fetch + Execuţie) se realizează succesiunea de funcţii,  F1, F2, F2, ...., Fk  corespunzătore tuturor celor k stări din diagrama (minimum, pe un număr de k tacte de ceas), necesare pentru interpretarea instrucţiunii respective. O funcţie Fi , la rândul său (pe un tact), poate fi compusă dintr-o serie de **microoperaţii** realizate în paralel ; exemplu de microoperații: citirea unui registru, înscrirea unui registru, transferul unui cuvânt pe magistrală, realizarea unei operaţii pe UF etc. Fiecare microoperaţie, notată cu f într-un **limbaj de descriere RTL** (Register Transfer Language) se exprimă simbolic în felul următor:  Z ← f (X1, X2, ..., Xn)  în care: – Z, X1, X2, ...,Xn sunt registre sau valori/(cuvinte)care se incrementează, adună, decodifică etc.  – ← indică sensul de transfer.  În fiecare stare i (deci pe durata a unui tact de ceas), a unei etape din diagrama de control, se analizează care sunt microperaţiile necesare a se efectua în calea de date, în consecință se deduce care este setul de semnale de control {**cij**} necesar a fi generat de către calea de control pentru efectuarea acestor microoperații; *într-o stare i pot fi realizate în paralel mai multe microoperaţii* f1, f2, f3, …,fj,... . Folosind descrierea RTL se poate exprima funcţia Fi, corespunzătoare stării i prin conjuncţia de j funcţii/microoperaţii cu primitiva logică (IF.... THEN....)  **IF** {cij} **THEN** Fi Fi = f1 ( X1, X2, X3, .... Xn) f2 ( X1, X2, X3, .... Xn) f3( )⋂.......⋂ |

|  |  |
| --- | --- |
| De exemplu, în diagrama de stări prezentată în figura anterioară pentru instrucţiunea add $c, $a, $b ; ($c←$a+$b), din etapa Accesare operanzi, realizată în acest caz pe o singură stare (o perioadă de ceas), este necesar a se efectua următoarele două microperaţii f1 şi f2 în paralel:  1. f1 ={citire $a, Ms1← $a}, citirea primului operand din registrul A şi transferul pe magistrala Ms1;  2. f2 ={citire $b, Ms2 ← $b}, citirea celui de al doilea operand din registrul B şi depunerea pe magistrala Ms2.  Deci pentru această etapă i, care consumă doar un singur tact, unitatea de control trebuie să genereze setul de semnale de control {, } astfel ca să se realizeze în paralel cele două microoperaţii care sunt necesare funcţiei accesare operanzi  F = f1 f2 = {citire $a, Ms1← $a}{citire $b, Ms2 ← $b}  Unitatea de control poate fi implemnetată în două variante:   1. Unitate de control cablată; 2. Unitate de control microprogramată.   **4.2. UNITATEA DE CONTROL CABLATĂ**  Unitatea de control cablată este, în fond, o structură de maşină cu algoritm de stare, ASM, a cărui funcţionare realizează în hard (emulează) diagrama de stări a procesorului. Oricare ASM se reduce la o structurare de circuit secvenţial, sub forma Huffman, cum este prezentat în figura următoare. În această structurare a ASM-ului s-a introdus text explicativ adaptat încât aceasta să poată fi privită ca o unitate de control care generează seturi de semnalele de control {cij} pentru o cale de date. | |
| **C599D03** | |
| O structură de unitate de control care generează semnal  pentru calea de date, realizată ca un ASM cu partea de  circuit combinaţional implementat pe o matrice PLA, este reprezentată în figura următoare | **C:\Users\Owner\Documents\Săgeată 1.tif** |
| **C:\Users\Owner\Documents\calea de control.tif** | |
| Calculul logic al semnalelor de control aplicate în calea de date, {}, al semnalului de încărcarea a instrucţiunii în registrului de instrucţiuni, IR şi al semnalului pentru încărcarea codurilor de condiţii în registrul CC, este realizat pe circuitul combinațional implementat pe două niveluri logice (SI–SAU), care poate fi un circuit PLA; de asemenea, se calculează şi starea următoare pentru ASM. “Cablarea” unităţii de control se face prin programarea matricei PLA atât pe nivelul de SI cât şi pe nivelul de SAU. | |

|  |
| --- |
| **EXEMPLUL 4.2** Pentru o arhitectură pe bază de acumulator, din figura următoare, pentru care s-a definit setul de instrucţiuni, ISA, care se pot executa (în număr de 8 instrucţiuni), s-au figurat semnalele de control ‒ din calea de date, cu explicarea microoperaţiilor corespunzătoare realizate, să se implementeze o unitate de control cablată. (La arhitectura pe bază de acumulator, sau arhitectură memorie–la–registru, totdeauna un operand este în registrul acumulator iar celălalt operand, M[x], în memorie la adresa x, rezultatul se obține tot în registrul acumulator). |
| **Calea de date exemolul1** |
| **D920FE42** |

|  |
| --- |
| **–** Pentru cele opt instrucţiuni alese, care formează setul de instrucţiuni al procesorului, ISA, şi pentru elementele componente din calea de date, se analizează toate transferurile necesare interpretării acestor opt instrucţiuni, se stabilesc traseele necesare cu punctele de control rezultând semnalele de control necesare , în număr de 13, care sunt ‒  **–** Se construieşte diagrama de stări pentru intrepretarea setului de instrucţiuni, care cuprinde un subciclu fetch (comun tuturor celor 8 instrucţiuni) realizat pe 3 stări, iar în subciclul de execuție se consumă maxium 3 stări, deci ar fi necesare 6 tacte de ceas pentru ciclul instrucţiune. Dar, deoarece în subciclul fetch există totdeauna un acces la memorie, iar în cel de execuție pentru unele din instrucţiuni cum ar fi : LOAD, ADD, AND există totdeauna încă un acces la memorie, pentru aceste accesări la memorie se consideră că se consumă două perioade de ceas, rezultă că intrepretarea instrucţiunilor cele mai ”lente” necesită în total 8 tacte , (3+1) pentru fetch , plus (3+1) pentru execuţie.  **–**  După această analiză a diagramei de stări pentru intrepretarea instrucţiunilor (ciclicitate în opt tacte de ceas) se alege pentru circuitul unităţii de control o implementare pe baza unui generator de faze modulo 8; opt faze realizate ciclic, , reprezentat în figură anterioară. Pentru subciclul fetch: pe  MAR←PC, pe  şi  Read Memory, pe  IR← MDR şi PC←PC+1 și decodificare; iar fazele ,,şi sunt consumate pentru microoperaţiile din subciclul de execuţie. Circuitul logic combinaţional care calculează setul de semnale de control, c0 ‒ c12 , care se aplică în calea de date, are ca intrări:   1. cele opt faze de la generatorul de faze, ; 2. cele opt semnalele Im, m= 0,…7, obţinute de la decodificaturul codului operaţiei (OPCODE) al instrucţiunii; 3. fanionul de condiţie, acumulator zero, AC = 0.   Ecuaţia logică pentru un semnal de control (j=1,...12) este o sumă logică a produselor dintre semnalul de fază, , și fiecare dintre semnalele de la decodificator Im ( m = 0,…7), de forma    (Semnalele de fază , care secvențializează subciclul de fetch, nu condiționează pe nici una dintre cele 8  ieșiri de la decodificator, ( ), deci , de fapt ieșirile de la decodificator () sunt active în  subciclul de execuție, doar după ce cele patru faze fetch s-au consumat când sunt secvențializate de semnalele  când se extrage instrucţiunea din memorie şi în starea şase şi şapte  când se citeşte operandul din memorie (pentru instrucţiunile LOAD, ADD, AND), s-a considerat că accesul la memorie consumă două perioade de ceas, atunci din relaţia anterioară rezultă  Această expresie logică se poate implementa pe un circuit cu două niveluri logice ca o sumă de produse cum este  prezentat în figura următoare |
| c 3 |
| Caracteristicile unei unități de control cablate sunt:  – viteză de procesare ridicată  – inexistența posibilității de modificare/flexibilitate (eliminare sau completare în ISA) pentru o implementare deja   realizată. |

|  |
| --- |
| **4.3 UNITATEA DE CONTROL MICROPROGRAMATĂ**  Ideea de microprogramare aparţine prof. M.W. Wilkes, de la Cambridge Mathematical Laboratory, care în 1951 a implementat-o, în tehnologia de realizare a unui procesor, ca alternativă la complexitatea unei unităţi de control (UC) cablată. De fapt, prin microprogramare, prof. Wilkes a construit pentru unitatea de control un procesor (de control) care generează semnalele de control, iar aceste semnale de control se aplică în calea de date a procesorului, care execută instrucţiunile programului; apare , în acest fel, o recurenţă “procesor în procesor”, adică un procesor interior – unitatea de control microprogramată, UC, – care comandă procesorul exterior ce procesează instrucțiunile programelor utilizatorului.  Similaritatea procesorului din interiorul procesorului cu procesorul (exterior) care execută instrucțiunile programului şi pe care îl controlează în funcționare, a dus la preluarea termenilor de la procesorul exterior şi adaptarea lor pentru procesorul din interior (de contol) prin adăugarea prefixului *micro*, de exemplu: *micro*operaţie – MO; *micro*instrucţiune – μI ; *micro*cod; *micro*program; *micro*subrutină; *micro*program counter – μPC; registru de *micro*instrucţiuni – μIR etc.  Pentru explicarea tehnicii de microprogramare, în realizarea unităţii de control, se va utiliza reprezentarea din figura următoare. În această figură este prezentată organigrama de stare pentru două instrucţiuni, add şi lw, cu stările atât din subciclul fetch + decodificare cât şi în stările din etapele/fazele subciclului de execuţie (identificare operanzi, execuţie, accesare memorie, înscriere rezultate), iar în fiecare dintre aceste stări sunt figurate mulțimile semnalelor de control {}, necesare a fi generate de unitate de control pentru a se asigura în calea de date intrepretarea instrucţiunilor programului. Setul semnalelor de control,{}, j = 1, 2, 3,...generat într-o anumită stare i, este format dintr-un număr de biţi de control, j, care pot fi strânşi întru-un cuvânt binar, cuvânt care formează o **microinstrucţiune**. |
| **C56096D** |

|  |
| --- |
| Pentru realizarea comenzilor în calea de date, într-o anumită stare, este necesar să se genereze o microinstrucţiune (în exemplul din figura anterioară s-a considerat, pentru uniformizare, că se generează doar cinci biţi de control (j = 1,2,3,4,5) în calea de date pe fiecare tact, deci toate cuvintele microinstrucţiune au lungimea de cinci biti. De exemplu, în cele trei stări ale subciclului de fetch + decodificare pentru oricre instrucțiune de program este necesar ca microrprocesorul de control să genereze următoarele trei seturi succesive de semnale de control={1,0,1,1,0} ,  ={0,1,1,1,0} ,  ={0,1,0,1,0}, adică generarea succesivă a următoarelor trei cuvinte microinstrucţiune 10110, 01110, 01010, (valorile biţilor în aceste microintrucţiuni sunt luate arbitrar), cuvinte microinstrucțiune care pot fi unite într-o *microsubrutină* Fetch :  Fetch: 10110  01110  01010  referită ca microsubrutina de interpretare a subciclului fetch +decodificare. Această microsubrutină fetch + decodificare este comună pentru toate instrucţiunile din programul rulat de procesor (de fapt, subciclul de fetch este standard pentru ISA).  Pentru stările etapelor din subciclul de execuţie, ale instrucţiunii lw, este necesar a se genera de către unitatea de control următoarele patru seturi de semnale de control : {}, {}, {}, {}; adică succesiunea de patru cuvinte microinstrucţiune 10101, 11101, 00011, 10011 care pot fi strânse, de asemenea, într-o microsubrutina de execuţie a instrucţiunii lw referită Execuție:  Execuție: 10101  11101  00011  10011  Aceste două microsubrutine, Fetch şi Execuţie, sunt stocate în memoria procesorului interior, memorie care este referită ca **Memorie de control**. Pentru interpretarea unei instrucţiuni din setul de instrucțiuni (ISA) al procesorului, procesorul de control trebuie să extragă microinstrucţiune după microinstrucţiune din memoria de control (la fel cum procedează procesorul care execută instrucţiunile programului utilizatorului, extrage instrucţiune după instrucţiune din zona text a memoriei sistemului), să depună microinstrucţiunea extrasă într-un microregistru de microinstrucţiuni. Apoi, biţii din microregistru se aplică în punctele din calea de date a procesorului, ca semnale de control, pentru a comanda realizarea **microoperaţiilor** (selectare trasee, transferuri, încărcare registre, citire registre etc) corespunzătoare stări comandate de microinstrucţiunea respectivă. Deci, pentru intrepretarea unei instrucţiunii a programului rulat pe procesor, unitatea de control extrage succesiv din memoria de control microinstrucţiunile microsubrutinei Fetch (care este comună pentru toate instrucţiunile), apoi se extrag succesiv microinstrucţiunile microsubrutinei de Execuţie (pentru fiecare instrucţiune din ISA corespunde o microsubrutină de execuţie specifică), după care se trece la execuţia instrucţiunii următoare a programului prin rularea aceleiaşi microsubrutine Fetch şi a microsubrutinei de Exectuţie specifice instrucţiunii respective; acest proces, de extragere şi execuţie de microinstrucţiuni din memoria de control, se repetă pentru interpretarea fiecarei instrucţiuni din program rulat pe procesor. Pentru interpretarea fiecărei instrucțiuni a programului care se rulează se extrag, din memorie de control, și se execută un număr de microinstrucțiuni egal cu numărul de microinstrucțiuni cuprins în microsubrutina Fetch plus numărul de microinstrucțiuni cuprins în microsubrutina Execuție a instrucțiunii respective. Pentru codificarea (arbitrară a instrucțiunilor add și lw) de mai sus, microprogramul stocat în memoria de control, ar avea structurarea următoare.  **Fetch:** 10110  01110  01010    **Execuție:** sw,......., lw ,................, add ,......, slt , etc.  . 10101 01000 .  . 11101 10001 .  . 00011 00000 .  . 10011 00010 . |

|  |
| --- |
| Se pune întrebarea cum se înlănţuie (se trece) de la microsubrutina Fetch la microsubrutina specifică execuţiei instrucţiunii respective din programul de executat? Această trecere se realizează în momentul când  s-au executat toate microinstrucţiunile microsubrutinei Fetch , s-au realizat deja toate microoperaţiile care au efectuat aducerea instrucţiunii programului din memoria sistemului în registrul de instrucţiuni al procesorului. Câmpul OPCODE al instrucţiunii programului de rulat, prezent acum în registrul de instrucţiuni al procesorului, μIR, *este convertit în adresa primei microinstrucţiuni a microsubrutinei specifice de Execuție din memoria de control,* adresă de început de la care în continuare care vor fi citite microinstrucţiunile subrutinei de Execuție necesare ciclululi de execuţie pentru instrucţiunea respectivă. Ultima microinstrucţiune din microsubrutina Fetch este o microinstrucţiune de salt în memoria de control, adresa țintă de salt din această microintrucțiune este tocmai adresa care s-a obţinut din conversia OPCODE–ului instrucţiunii, deci în felul acesta microsubrutina de Fetch se continuă (”se leagă”) în memoria de control cu microsubrutina de Execuţie. Microsubrutina Execuţie se termină obligatoriu cu o microinstrucţiune de salt necondiţionat (jump) la adresa de început a microsubrutinei Fetch, deci se reia ciclul în memoria de control pentru interpretarea următoarei instrucțiuni din programul de executat al procesorului. Prin această rulare ciclică în memoria de control, a microsubrutinei Fetch urmată de microsubrutina specifică de Execuţie, se interpretează instrucţiune după instrucţiune din programulrulat pe procesor.  Termenul de **interpretare** pentru instrucțiune este potrivit deoarece instrucțiunea programului pe durata ciclului său este procesată prin intrepretarea biților microinstrucțiunilor (care sunt semnale de control) din microsubrutina Fetch urmată de microsubrutina de Execuție.  Microprogramul format din microsubrutina pentru ciclul Fetch împreună cu setul de microsubrutine de execuţie, specifice fiecărei instrucţiuni, stocate în memoriea de control sunt referite prin termenul de **FIRMWARE.** Firmware-ul odată stocat în memoria de control fixează instrucţiunile procesorului, ISA;  eliminarea sau adăugarea unei microsubrutine de execuţie duce la eliminarea respectiv la adăugarea unei instrucţiuni în ISA. Însă, utilizarea unei memorii de **C**ontrol de tip RAM (**C**RAM) face posibilă modificarea uşoară a firmware-ului, deci a setului de instrucţiuni, ISA. Mai mult, prin schimbarea completă a firmware-ului se obţine un nou procesor, un alt ISA, deci se poate ”croi” un alt procesor numai prin elaborarea unui alt firmware. Această flexibilitate nu există la o unitate de control cablată, în schimb deşi procesoarele cablate nu mai pot fi modificate au, în general, avantajul unor viteze de procesare mai ridicată decât cele obţinute pe microprocesoarele microprogramate. Viteza de procesare mai redusă pe procesoarele cu unitate de control  microprogramată rezultă din faptul că o instructiune din ISA este interpretată prin rularea unei succesiuni de microintrucţiuni din memoria de control.   * Formatul microinstrucţiunilor.   **1. Formatul scurt** cuprinde, în general, un număr de 20-30 biţi în cuvântul microinstrucţiune, iar  codificarea este liniară, adică pentru comanda unui punct din calea de date corespunde un bit în cuvântul microintrucţiune, cum este reprezentat în figura următoare. |
| **81823770** |
| O microinstrucţiune, μI, de format scurt prin biţii săi, uzual, realizează una sau doar câteva microoperaţii. Evident, cu μI de format scurt rezultă microprograme lungi (interpretarea unei instrucțiuni din program necesită un număr destul de ridicat de microoperații); maşinile cu μI de format scurt sunt referite cu **microprogramare pe verticală.** |

|  |
| --- |
| **2. Formatul lung** cuprinde, în general, un număr de 300- 400 de biţi în cuvântul microinstrucţiune, care este împărţit în subcâmpuri, pe fiecare subcâmp se realizează o codificare completă. Toate microoperaţiile de executat pentru o anumită componentă a procesorului sunt grupate pe același subcâmp, deci sunt comandate prin coduri stocate în acelaşi subcâmp al microinstrucţiunii; o astfel de grupare de microoperații este posibilă cu condiţia ca microoperaţiile grupate în același subcâmp să nu fie cu execuţie paralală, adică să nu fie executate simultan (datorită codificării complete pe un subcâmp); dacă două microoperaţii sunt cu execuţie simultană (paralelă) atunci acestea se repartizează în subcâmpuri diferite ale microinstrucțiunii. |
| **9D3FE7A9** |
| La o μI de format lung, după ce este extrasă din memoria de control şi depusă în microregistrul de microinstrucţiuni, μIR, fiecare subcâmp al său este decodificat separat, iar ieşirea activată de la decodificatoare ( câte un decodificator, DCD, pentru fiecare subcâmp) se aplică în calea de date ca semnal de control. Avantajele microprogramării cu μI de format lung sunt: viteză mai ridicată de intrepretare (toate subcâmpurile din microinstrucţiune pot genera semnale în paralel), microprograme scurte (o singură uI poate interpreta operaţii complexe); microprogramarea cu instrucţiuni lungi este referită ca **microprogramare pe orizontală.**  Organizarea de principiu pentru o unitate de control microprogramată este reprezentată în figura următoare. |
| **C:\Users\Owner\Documents\microprogramare 455667.tif** |

|  |
| --- |
| Firmware-ul corespunzător interpretării instrucţiunilor din ISA este stocat în memoria de control, CM. Conţinutul registrului microprogram counter, μPC, se aplică la CM ca o adresă, iar din locaţia respectivă se extrage microinstrucţiunea, μI, şi se depune în microregistrul de microinstrucţiuni, μIR. Biţii din μI sunt grupaţi în trei subcâmpuri:  1. Subcâmpul 1 compus din doi biţi, , specifică secvenţialitatea în parcurgerea microprogramului din CM  în modul următor:  – = 00; , următoarea μI va fi extrasă de la adresa următoare din CM, μPC← μPC+1,  prin incrementarea conținutului din μPC.  – = 01; următoarea μI va fi impusă din exteriorul unităţii de control.  – = 10; următoarea μI se obţine din OPCODE-ul instrucţiunii programului care se rulează (La   sfârşitul microsubrutinei Fetch instrucţiunea programului care se rulează este deja adusă din   memoria procesorului şi depusă în registrul de instrucţiuni,IR. Subcâmpul codul operaţiei,   OPCODE, din IR este convertit în adresa de început din CM, adresă unde se află   microsubrutina specifică pentru interpretarea ciclului Execuţie al instrucţiunii,deci în CM se   efectuează un salt necondiţionat).   – = 11; următoarea μI se află în CM la o adresă care este specificată (ca un imediat) în corpul  microinstrucţiunii din μIR, deci în CM se efectuaează un salt necondiţionat. (Un astfel de   salt necondiţionat se efectuează și la sfârşitul fiecărei microsubrutine de execuţie când se   sare la prima microinstrucţiune din microsubrutina Fetch)  Selectarea între aceste patru variante de adresa pentru următoare μI se realizează cu un grup de nxMUX 4:1,  comandat de cuvântul (n este lungimea cuvântului microinstrucțiune).  2. Subcâmpul 2 conţine toți biții de control,{}, care se aplică în calea de date pentru comenziile efectuate   de către μI respectivă.  3. Subcâmpul 3 conţine o adresă de salt (imediat) în program, (când = 11).   * Secvenţialitatea execuţiei micooperaţiilor. Biții din microinstrucţiune comandă efectuarea anumitor   microoperaţii în calea de date, microoperațiile corespunzătoare unei stări. Aplicarea în calea de date a semnalelor (biților) de control conținuți într-o μI pentru execuţia microoperaţiile, pot fi aplicați toți biții peperioada unui semnal de ceas (*execuţie monofazată*) sau succesiv, pe perioada câte unui semnal de ceas se aplică biții decodificați obținuți de la unul sau de la mai multe subcâmpuri (*execuţie polifazată*).  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **EXEMPLUL 4.3.** Pentru calea de date din Exemplul 4.2, pentru care s-a elaborat deja o cale de control cablată, să se elaboraze şi o cale de control microprogramată.Semnale de control rămân aceleaşi  la care se mai adaugă semnalul  prin care se comandă aplicarea la μPC a unei adresei de salt în memoria de control ( μPC←IR(OPCODE)). Această adresă din memoria de control, care corespunde începutului microsubrutinei pentru intrepretarea ciclului de execuţie, este obţinută din conversia OPCODE–ul instrucţiunii din programul de executat.  Unitatea de control microprogramată se conectează de calea de date print-un convertor de cod care translatează codul operatiei (OPCODE) stocat în registrul de instrucțiuni într-o adresă din memoria de control. Citirea microinstrucțiunilor din memoria de control se realizează prin conținutul (ca adresă) registrului de program counter de microinstrucțiuni, μPC. μPC poate fi încărcat cu cuvântul adresa pentru memoria de control, prin intermediul unui bloc selector 6xMUX4:1, de la următoarele patru surse: 1. adresa următoare (+1) din memoria  de control; 2. adresa obținută prin conversia OPCOD – ului instrucțiuni curente care se procesează din programu; 3. o adresă de salt condiționat rezultată din execuția microprogramului din memoria de control; 4. o adresă de salt necondiționat (aplicată din exterior). Selectarea sursei de adresă pentru încărcarea în μPC se realizează printr-un MUX4:1. |

|  |
| --- |
| **C:\Users\Owner\Documents\UC microprogramată.tif** |
| **C:\Users\Owner\Documents\Cale de date semnale.tif** |
| **C:\Users\Owner\Documents\Microprogram.tif** |

|  |
| --- |
| Pentru controlul microperaţiilor din calea de date se alege o codificare liniară în μI, adică pentru fiecare punct de control corespunde un bit în μI, rezultă o microinstrucţiune cu lungimea de 22 biţi.  Formatul microinstrucţiunii este un format scurt şi are doar trei subcâmpuri:  – câmpul pentru secvenţialitate pe doi biţi, (00–adresa μI următoare, 01–adresa μI obţinută din   OPCOD– ul instrucţiunii, 10–salt condiţionat,11salt necondiţionat);  – câmpul pentru adresa de salt (pe 6 biţi, întreg firmware nu depăşete 64 de microinstrucţiuni);  – câmpul pentru biţii de control, în total 14 semnale de control ( – ).  Setul de instrucţiuni cuprinde aceleaşi opt instrucţiuni ca la implementarea cablată:  LOAD Adresă ; AC← M[Adresă]  STORE Adresă ; M[Adresă] ←AC  ADD Adresă ; AC ←AC + M[Adresă]  AND Adresă ; AC ←AC ∩M[Adresă]  JUMP Adresă ; PC← Adresă  JUMPZ Adresă ; if AC =0 then PC← Adresă  COMP ; AC←  RSHIFT ; AC← AC∙  deci firmware-ul va conţine opt microsubrutine de execuţie, câte una pentru intrepretarea subciclului de execuţie al fiecărei instrucţiune, plus microsubrutina de Fetch care este comună pentru toate cele opt instrucţiuni.  Analizând pentru subciclul de fetch şi pentru cele opt subcicluri de execuţie microoperaţiile necesare din calea de date, pentru realizarea acestor subcicluri, se deduc care sunt semnalele de control ce trebuie activate, deci biţii activi din fiecare microinstrucţiune.  Micooperaţiile necesare şi succesiunea acestora comandate de microsubrutina Fetch sunt:  Fetch: MAR←PC ; câmpul = 00, câmpul adresă de salt = 000000, câmpul de control  RED M ; câmpul = 00, câmpul adresă de salt = 000000, câmpul de control =1  PC← PC+1, IR←MDR(OPCODE); câmpul = 00, câmpul adresă de salt = 000000, câmpul de  ;control =1, =1 (aceste două micooperaţii se efectuează  ;în paralel)  Go to μIR ; câmpul = 01(OPCODE este transformat în adresa de început a microsubrutinei   ;de execuţie și se înscrie în μIR), câmpul adresă de salt=000000, câmpul de control  ; =1 (adresa de început a subrutinei este înscrisă în μPC).  Iată că a rezultat microsubrutina Fetch compusă din patru microinstrucţiuni, iar după executarea acesteia, prin ultima microinstrucţiunedin microsubrutină ( =1), se efectuează salt la prima microinstrucţiune din una dintre microsubrutinele de execuţie. Pentru fiecare ciclu de executie al celor opt instrucţiuni din ISA, similar ca şi la subrutina Fetch, se deduc care sunt microoperaţiile necesare şi care este succesiunea lor, apoi biţii din cuvintele microoinstrucţiune apar evident prin identificarea semnalelor de control ce trebuie activate pentru microoperaţiile respective.  Flexibilitate oferită de microprogramarea unităţii de control se poate evidenţia prin următoarea exemplificare. Să presupunem că în ISA s-a omis includerea unei instrucţiuni, pe care să o denumim CLEAR, a cărei funcţie este de a reseta toţi biţii cuvântului din acumulator (în lipsa unui semnal de control de RESETARE ACUMULATOR). Resetarea acumulatorului (înscrierea cu zero) poate fi rezultatul următoarei operaţii logice . Microoperaţiile necesare a se realiza de către microsubrutina de execuţie a instrucţiunii CLEAR, precum şi biţii (semnalele de control) din microinstrucţiunile respective sunt prezentate în figura următoare  BC3E0116 |

|  |
| --- |
| Interfaţa dintre hard (procesor) şi soft (program) este realizată de fapt prin ISA, o interfațare rigidă  ( “ bătută în cuie “ odată ce s-a fixat setul de instrucțiuni și s-a implementat cablat unitatea de control). Microprogramarea prin introducerea a unui nivel intermediar ‒ **Firmware** ‒ produce o interfațare (ISA) mult mai flexibilă între hard (procesor) şi soft (program). Substituind microprogramul din CM , un alt firmware, se obţine o altă maşină sau o îmbunătăţire a celei existente; un alt firmware este o altă mașină! Caracteristicile unei unități de control realizată microprogramat sunt:  – viteză de procesare mai redusă în interpretarea instrucțiunilor (determinată de rularea firmware-ului);  – existența posibilității de modificare –flexibilitate–(ștergere, completare sau chiar înlocuire totată în   ISA) pentru o implementare deja realizată.  **Emularea** unei maşini reprezintă simularea în hard (prin firmware) funcţionarea unei maşini (adică înscrierea în CM a firmware-ului corespunzător).  **Nanoprogramarea** se obţine prin introducerea a încă unui nivel de microprogramare/control, acesta fiind între hard şi micromemoria de control, deci un nou nivel de firmware (care are la rândul său nanomemoria (sa) de control) care comandă un alt nivel de firmware (care are micromemoria de control, CM). De data aceasta recurenţa fiind un procesor (nanoprocesor) care comandă un alt procesor (cel de control, nivelul de firmware, microprogamarea) care la rândul său comandă un alt procesor (microprocesorul care procesează instrucţiunile programului); deci procesor (nanoprocesorul) în procesor (unitatea de control microprogramată) în procesor (microprocesorul care rulează programul). |