Universitatea Tehnică

Cluj-Napoca

Structura Sistemelor de Calcul

Implementarea unui microcontroller 8051

Mureșan Ioana-Diana

Grupa 30237

Îndrumător de proiect: Octavian Beliga

06.01.2020

**Cuprins**

1. Rezumat…………………………………...................3

2. Introducere………………………………..................4

3. Fundamentare teoretică……………………………...6

3.1. Arhitectura microntrollerului 8051………….7

3.2. Setul de instrucțiuni…………………………9

4. Proiectare și implementare…………………………11

4.1. Diagrama bloc……………………...............11

4.2. Descrierea modulelor……………................12

5. Rezultate experimentale……………………………15

6. Concluzii…………………………………………...20

# Rezumat

Acest proiect a presupus împlementarea unui microcontroller 8051, care a fost proiectat de către Intel în anul 1981 și este unul dintre cele mai populare microcontrollere care se folosesc astăzi. Este un microcontroller pe 8 biți, ceea ce înseamnă că operațiile disponibile sunt limitate la 8 biți. Chip-urile 8051 sunt folosite într-o varietate mare de sisteme de control, în robotică și în industria automotive. Obiectivul acestui proiect a fost obținerea unui microcontroller bazat pe această arhitectură care să efectueze anumite operații aritmetice și logice cu numere binare, instrucțiuni de jump și branch și o instrucțiune specială de aprindere a ledurilor. Arhitectura microcontrollerului a fost descrisă în mod structural într-un limbaj de descriere hardware și anume, VHDL, iar implementarea s-a făcut folosind o plăcuță FPGA Basys 3. Rezultatele au fost destul de satisfăcătoare deoarece simularea fiecărei componente a fost corectă, la fel și simularea proiectului final. În urma realizării acestui proiect pot spune că mi-am aprofundat cunoștințele în ceea ce privește limbajul de descriere hardware VHDL si totodată am invățat mai multe detalii despre modul de funcționare al unui microcontroller și importanța lor într-o varietate largă de domenii. Deși consider că am încercat să respect pe cât posibil caracteristicile și funcționalitățile microcontrollerului 8051 sunt de accord că acest proiect poate fi extins prin creșterea numărului de instrucțiuni ce pot fi executate, optimizarea codului și adăugarea altor funcționalități,

# 2. Introducere

Microcontrollerul 8051 a fost proiectat de Intel în anii 1980 și a fost dezvoltat în principal pentru a fi folosit în Sisteme Înglobate. La început a fost creat utilizând tehnologia NMOS, dar aceasta necesită o mare putere de funcționare, astfel că Intel a decis să utilizeze tehnologia CMOS, rezultând ediția de microcontrollere 80C51. Aceste microcontrollere mai moderne au nevoie de o cantitate mai mică de putere pentru a funcționa în comparație cu înaintașii lor.[[1]](#footnote-1)

Microcontrollerul se numea system on a chip pentru că avea 128 bytes RAM, 4K on-chip ROM, 2 temporizatoare, un port serial, 4 porturi de cate 8 biți, toate pe un sigur chip. Când a început să devină popular, Intel a permis și altor producători să facă și să comercializeze diferite modele de 8051, având codul compatibil cu 8051. Acest lucru presupune că un program scris pentru un model (eng. flavor / member) de 8051, va putea fi rulat și pe alte modele indiferent de producător.[[2]](#footnote-2)

Un sistem este un aranjament în care toate unitățile asamblate funcționează împreună după un set de reguli. Spre exemplu, un ceas este un sistem de afișare a timpului. Componentele sale urmează un set de reguli pentru a afișa timpul. Dacă o componentă eșuează, ceasul nu mai funcționează, deci într-un sistem toate componentele depind unele de altele. În cazul unui sistem înglobat (eng. embedded system) acesta poate fi un sistem independent sau poate fi o parte dintr-un sistem mai mare. Un sistem înglobat este un sistem bazat pe un microcontroller sau pe un microprocessor proiectat pentru a îndeplini un task specific. Spre exemplu, o alarmă de incendiu este un sistem înglobat. Cele 3 componente ale unui sistem înglobat sunt: hardware, application software, Real Time Operating system.[[3]](#footnote-3)

Importanța sistemelor înglobate este în continuă creștere considerând multitudinea domeniilor în care pot fi utilizate, cum ar fi: avionică, sisteme de traffic management. Larga lor utilizare ilustrează importanța sistemelor înglobate, mai ales dacă luăm în considerare potențialele efecte ale eșecului lor. Spre exemplu eșecul unui pilot automat sau eșecul unui sistem de frânare poate duce la pierderea unei vieți, eșecul unui sistem de energie electrică poate duce la pierderea vieții sau poate afecta calitatea vieții, eșecul unui sistem de control al producției poate duce la pierderi semnificative de venituri. Dependența noastră de embedded systems necesită dezvoltarea si adoptarea de noi tehnici arhitecturale și de proiectare pentru a răspunde cerințelor de performanță necesare și pentru a atinge fiabilitatea necesară folosind resursele limitate ale acestora în termeni de procesare, memorie și putere. Importanța sistemelor înglobate a dus la apariția unei industrii puternice care le dezvoltă și le utilizează.

În concluzie, sistemele înglobate sunt omniprezente în lumea modernă. Astfel de sisteme sunt folosite într-o varietate largă de aplicații, variind de la dispozitive electronice comune pentru consumatori până la aplicații auto și avionice. O proprietate comună tuturor sistemelor înglobate este interacțiunea lor cu mediul înconjurător. Datorită domeniilor de aplicație în care acest fel de sisteme sunt utilizate, comportamentul lor este adesea constrâns de proprietăți funcționale (relația input-output) și non-funcționale (cum ar fi timpul de execuție si consumul de energie). Acest lucru face ca testarea și validarea acestor sisteme să fie o provocare.[[4]](#footnote-4)

Microcontrollerul 8051 este folosit într-o varietate mare de aplicații cum ar fi: robotica, The Internet of Things, Temperature and Light sensing devices, Process Control Devices.[[5]](#footnote-5)

În acest proiect a fost implementată o variantă mai simplficată a unui microcontroller 8051. Deoarece este un microcontroller pe 8 biți instrucțiunile pot atinge un număr de 256, însă acest număr a fost redus semnificativ, păstrând totuși instrucțiuni din fiecare categorie. În ceea ce privește restul caracteristicilor microcontrollerului s-a încercat respectarea și implementarea lor într-o oarecare măsură. Obiectivul principal este ca acest microcontroller să fie capabil să execute un program scris in memoria ROM, ceea ce înseamnă că acest program trebuie decodificat, iar apoi fiecare instrucțiune să fie executată pe rând pentru a ajunge la rezultatul final.

Pentru atingerea acestui obiectiv proiectul este implementat structural, ceea ce înseamnă că este un sistem sub forma unor componente interconectate. Aceste descrieri permit crearea unor nivele ierarhice multiple, în care un proiect este divizat în unități de proiectare de dimensiuni mai mici. Principalul avantaj al acestei implementări este faptul că fiecare componentă poate fi proiectată și testată individual, înainte de a fi integrată în nivelele superioare ale proiectului. Această testare a nivelelor intermediare este mai simplă decât testarea în cadrul sistemului, și este de obicei mai completă. Aceasta înseamnă că proiectantul poate avea un grad mai ridicat de încredere în componentele utilizate, ceea ce contribuie și la integritatea globală a sistemului.[[6]](#footnote-6)

Am intuit ca proiectul va putea fi împărțit pe mai multe componente, și anume: o componentă care va conține memoria ROM în care se afla programul de executat și care se va ocupa de extragerea fiecărei instrucțiuni, un modul pentru memoria RAM, un modul care va decodifica instrucțiunea și va extrage operanzii necesari, o unitate de control care va seta valorile tuturor semnalelor de control in funcție de opcode-ul operației si o componentă de execuție care se va ocupa de realizarea efectivă a operațiilor aritmetice, logice etc.

În continuare, secțiunea **Fundamentare teoretică** va conține detalii teoretice despre proiect care au fost extrase din literatura existentă și din materiale on-line și care au fost utilizate pentru îndeplinirea obiectivelor proiectului, apoi va fi prezentată secțiunea de **Proiectare și implementare** în care este descrisă fiecare etapă parcursă pentru realizarea proiectului, schema bloc generală a sistemului, descrierea funcției fiecărui modul, detalii de implementare, arhitectura software indicând comunicația între module etc. **Rezultatele experimentale** vor conține rezultatele simulării care vor arăta că sistemul proiectat a fost implementat cu succes și rezultatele obținute sunt valide. În **Concluzie** este prezentat un sumar al raportului proiectului, iar la final va exista o listă a tuturor resurselor consultate pentru realizarea proiectului numită **Bibliografie.**

# Fundamentare teoretică

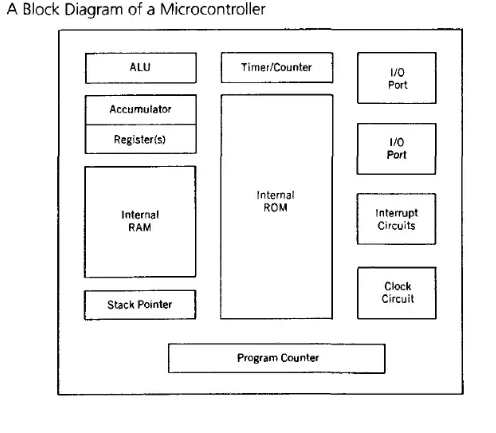


Figura 1. Diagrama bloc a unui microcontroller [[7]](#footnote-7)

Design-ul unui microcontroller include toate componentele ce se găsesc într-un microprocessor CPU: ALU, PC, SP (Stack Pointer) și registers. Pe lângă acestea are și componentele necesare pentru a forma un calculator complet: ROM, RAM, parallel I/O, serial I/O, counters, și clock circuit.

Principalul scop al unui microcontroller este să controleze operațiile folosind un program fix care este stocat în ROM și care nu se schimbă.

Operațiile Microcontrollerului: Program Counter este setat la 0, instrucțiunea de la această adresă este trimisă la Instruction Decoder, care o recunoaște, apoi este excutată și valoarea lui Program Counter este incrementată cu 1, iar procesul este repetat de milioane de ori pe secundă.[[8]](#footnote-8)

## **Arhitectura Microcontrollerului 8051**

Microcontroller-ul 8051 este unul dintre cele mai populare și cele mai utilizate microcontrollere în diferite domenii. A fost dezvoltat de Intel în anul 1980. 8051 este un microcontroller pe 8 biți.

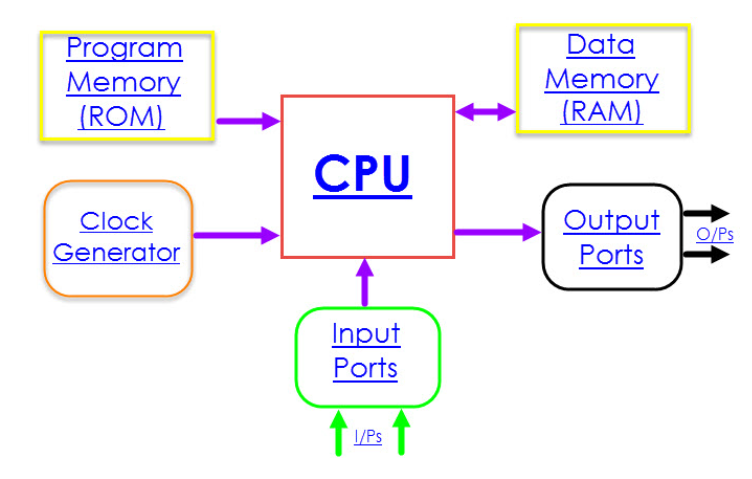


Figura 2. Arhitectura microcontrollerului 8051 [9]

**Caracteristici:**

* **Alu-8 biți:** unitatea aritmetico-logică, care este inima microcontrollerului, efectuează operații aritmetice și pe biți cu numerele binare.
* **Accumulator-8 biți:** este unu registru important asociat cu ALU
* **RAM:** 128 bytes of RAM, 128 bytes SFR si I/O port registers
* **ROM:** 4 KB Program Memory
* **I/O Ports:** 4 porturi I/O pe 8 biți adresabile pe bit si bidirecționale
* **Timers/Counters:** 2 pe 16 biți
* **Serial Port:** supports full duplex UART Communication
* **External Memory[[9]](#footnote-9)**

qComponentele principale

**1.Read only Memory ROM**

Read only Memory este un tip de memorie folosită pentru a salva programul care se execută.

**2.Random Access Memory RAM**

Random Access Memory este un tip de memorie folosită pentru stocarea temporară a datelor și a rezultatelor intermediare create și folosite în timpul operațiilor microcontrollerului.

**3.Electrically Erasable Programmable ROM EEPROM**

EEPROM este un tip special de memorie care nu apare la toate microcontrollerele. Conținutul poate fi schimbat în timpul execuției programului, dar rămâne permanent salvat chiar și după întreruperea curentului.

**4.Special Function Registers SFR**

SFR fac parte din memoria RAM. Au anumite funcții definite de producător și nu pot fi schimbate. Biții lor sunt conectati fizic la circuite particulare din microcontroller. De exemplu, scrierea unui 0 sau 1 pe un SFR care controlează un port I/O face ca pinul portului respectiv să fie configurat ca intrare sau ieșire => fiecare bit al acestui registru controlează funcția unui singur pin.

**5.Program Counter**

Program Counter pointează la adresa de memorie conținând instrucțiunea următoare de executat. Dupa fiecare execuție a unei instrucțiuni valoarea pointerului este incrementată.

**6.Central Processor Unit CPU**

CPU monitorizează și controlează toate procesele. Este format din:

Instruction Decoder

Arithmetical Logical Unit ALU

Accumulator

**7.Input/output ports**

Microcontrollerul poare fi conectat la dispozitive periferice. Fiecare microcontroller are unul sau mai mulți regiștrii conectați la pinii microcontrollerului.

**8.Oscillator[[10]](#footnote-10)**

## **Setul de instrucțiuni**

Scrierea unui program pentru orice microcontroller înseamnă oferirea unor comenzi microcontrollerului într-o anumită ordine în care acestea trebuie executate pentru a realiza un anumit task. Aceste comenzi se numesc setul de instrucțiuni ale microcontrollerului. Deoarece 8051 este un microcontroller pe 8 biți setul de instrucțiuni poate avea până la 256 de instrucțiuni.

Există 49 de mnemonici și care sunt împărțite in 5 grupuri: instrucțiuni de transfer, instrucțiuni aritmetice, instrucțiuni logice, instrucțiuni de manipulare a biților, instrucțiuni de salt. De asemenea există 5 moduri de adresare pentru operanzi: immediate addresssing, register addressing, direct addressing, register-indirect addressing, indexed addressing.

O instrucțiune este formată din opcode, urmat de operanzi de 0 bytes, un byte sau doi bytes. Opcode-ul conține mnemonica care specifică tipul operației si are un byte. Pot fi mai mulți operanzi, iar formatul instrucțiunii este următorul:

MENMONIC DESTINATION OPERAND, SOURCE OPERAND

O instrucțiune simplă este formată doar din opcode, deci are un singur byte. În instrucțiunile de 2 bytes, al doilea byte reprezinta operandul, iar în instructiunile de 3 bytes operandul este reprezentat de al doilea și al treilea byte.[[11]](#footnote-11)

Pentru a reduce complexitatea proiectului am ales să implementez doar câteva instrucțiuni din fiecare categorie.

Instrucțiuni de transfer:

Opcode

MOV A, #DATA A <- DATA 2 BYTES 0x74

MOV A, DIRECT A <- (DIRECT) 2 BYTES 0xE5

MOV DIRECT, #DATA (DIRECT) <- DATA 3 BYTES 0x75

Instrucțiuni aritmetice:

ADD A, #DATA A <- A + DATA 2 bytes 0x24

add a, direct a <- a + (DIRECT) 2 BYTES 0x25

SUB A, #DATA A <- A - DATA 2 bytes 0x94

SUB a, direct a <- a - (DIRECT) 2 BYTES 0x95

Instrucțiuni logice:

ANl a, #data a <- a and data 2 bytes 0x54

anl a, direct a <- a and (direct) 2 bytes 0x55

anl direct, #data (direct)<-(direct) and data 3 bytes 0x53

orl a, #data a <- a or data 2 bytes 0x44

orl a, direct a <- a or (direct) 2 bytes 0x45

orl direct, #data (direct)<-(direct) or data 3 bytes 0x43

Instrucțiuni de manipulare a biților:

CLR C C <- 0 (C = CARRY BIT) 1 BYTE 0xC3

Instrucțiuni de salt:

AJMP page0 ADDR PC <- ADDR 2 BYTES 0x01

CJNE A, #DATA, REL COMPARE DATA WITH A 3 BYTES 0xB4

JUMP TO PC+REL IF NOT EQUAL

# Proiectare și implementare

Proiectul este implementat în limbajul de descriere hardware VHDL, folosind mediul de proiectare Xilinx Vivado IDE care gestionează fișierele sursă incluse în proiect, memorează rezultatele rulării, generează rapoarte ale proiectului și salvează în mod automat starea proiectului. Modulele scrise sunt analizate, simulate, și sintetizate pe Basys 3 FPGA. Simulările au fost realizate pentru fiecare componentă în parte folosind bancuri de test, aceasta fiind cea mai utilizată metodă pentru testarea funcțională a circuitelor componente și a întregului sistem. După verificarea funcționării corecte în urma simulării, descrierea sistemului este translatată într-o listă de conexiuni, care este o descriere compactă a sistemului digital sub formă textuală, în care sunt specificate componentele sistemului, interconexiunile dintre acestea și pinii de intrare/ieșire. Următoarele etape realizează implementarea sistemului proiectat într-un circuit logic programabil: maparea tehnologică, plasarea și rutarea, analiza de timp și configurarea circuitului.

## **Diagrama bloc**

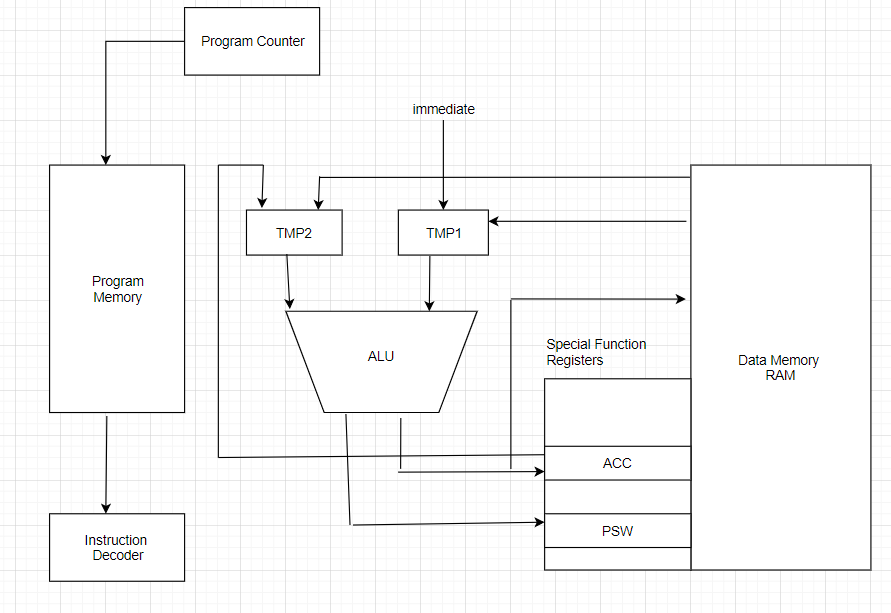
Schema prezentată în Figura 3 de mai jos reprezintă diagrama bloc a proiectului, și cuprinde modulele principale.În continuare se va descrie funcția fiecărui modul.

Figura 3. Diagrama bloc a microcontrollerului 8051

## **Descrierea modulelor**

Proiectul este format din 5 sub-module care compun modulul principal: instruction\_fetch, decode, mem\_ram, unitate\_comanda, execute.

Pentru a putea extrage instructiunile din memoria de program am implementat un modul: instruction fetch. În acest modul se regăsește și semnalul de tip ROM care reprezintă memoria ROM în care se stochează programul de executat. Această memorie este adresabila pe octet, ceea ce inseamna ca vom citi câte un byte din ea. Deoarece microcontrollerul 8051 are instrucțiuni de lungime variabilă, avem novoie de o logică care să citească instrucțiuni de 1 byte, 2 bytes și 3 bytes. În primul rand, avem nevoie de un contor de program, Program Counter care să țină evidența instrucțiunilor ce se execută. Acesta este un registru pe 16 biți care se va folosi pentru adresa instrucțiunii.

Mai departe, vom citi de fiecare dată câte 3 bytes din memorie, dintre care primul byte reprezintă opcode-ul instrucțiunii. În funcție de acest opcode vom decide tipul instrucțiunii și vom asigna unui semnal i valoarea 1, 2 sau 3 corespunzătoare numărului de bytes din care este formată instrucțiunea. Acest semnal va fi folosit la incrementarea contorului de program PC<=PC+i, pentru a sări peste instrucțiunea citită. De asemenea acest semnal va fi folosit și pentru construirea instrucțiunii. Modulul are 3 ieșiri fiecare de câte 8 biți care reprezintă instrucțiunea extrasă din program. Dacă i are valoarea 1 instrucțiunea are un singur byte formată doar din opcode și ceilalți 2 bytes vor avea valoarea 0, daca i are valoarea 2 instrucțiunea are 2 bytes și pe al treilea byte va fi pus 0, iar daca i este 3 instrucțiunea are 3 bytes și cele 3 ieșiri vor corespunde cu cele 3 valori citite din ROM.

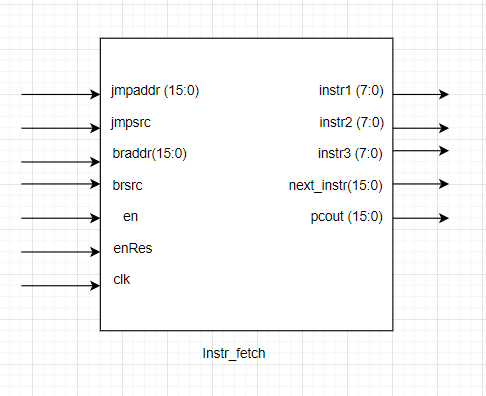
 Modulul este ilustrat în Figura 4 și mai are și intrările: jmpsrc, jmpaddr, brsrc si braddr. Dacă jmpsrc este activă atunci PC va avea valoarea adresei de jump, iar daca brsrc este activă PC va avea valoarea adresei de branch. Există și un semnal de reset pentru a reseta valoarea contorului de program la 0.

Figura 4. Entitatea instr\_fetch

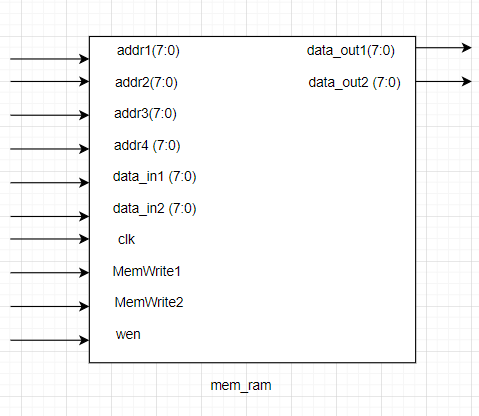
Memoria RAM este descrisă de un modulul din Figura 5 în care avem un semnal de tip array (0 to 255) of std\_logic\_vector (7 downto 0 ), ceea ce reprezintă 256 bytes de memorie RAM, dintre care primii 128 de bytes (00H – 7FH) sunt împărțiți în regiștrii de lucru (32 bytes), 16 regiștrii adresabili pe bit și 80 de regiștrii de uz general, iar următorii 128 de bytes reprezintă regiștrii cu funcții speciale (SFR).[[12]](#footnote-12) Modulul are 2 adrese de citire și 2 adrese de scriere, un semnal de activare a scrierii și 2 semnale diferite pentru cele 2 scrieri și este instanțiat în modulul decode.

Figura 5. Entitatea mem\_ram

Decode este componenta care decodifică instrucțiunea și aduce operanzii necesari din memorie. Aceasta are nevoie de 3 semnale de comandă care aleg sursa celor 2 operanzi, respectiv destinația operației. Pentru instrucțiunile alese destinația poate fi fie acumulatorul A, fie o altă locație de memorie. Primul operand poate fi fie acumulatorul A, fie o altă locație de memorie, iar al doilea operand este fie o constantă fie o locație de memorie. În funcție de aceste semnale au fost implementate procese pentru stabilirea surselor si a destinației. Entitatea modulului este prezentată în Figura 6.

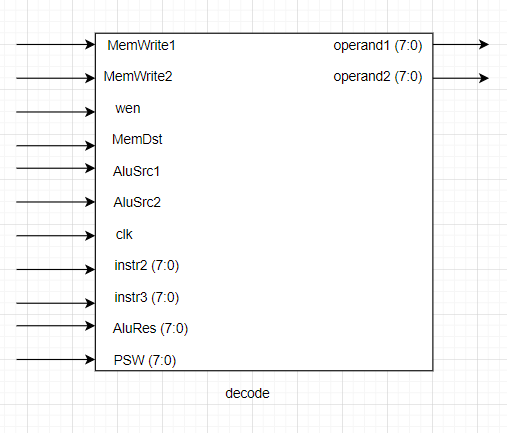


Figura 6. Entitatea decode

Unitatea de comandă a cărei entitate este ilustrată în Figura 7 este responsabilă de generarea tuturor semnalelor de comandă. Aceste semnale se activează în funcție de opcode-ul fiecărei operații: MemWrite1, MemWrite2, AluOp, AluSrc1, AluSrc2, MemDest, JmpSrc, BrSrc, BitOp, InvalidOp. MemDest este activ atunci când rezultatul generat trebuie scris într-o locație de memorie diferită de accumulator, MemWrite1 este activ când se dorește scrierea rezultatului, MemWrite2 este activ, când se dorește scrierea în registrul PSW, AluOp ia valori în funcție de operațiile ce trebuie executate de unitatea de execuție, AluSrc1 este activ când primul operand este o locație de memorie diferită de accumulator, iar AluSrc2 este activ când al doilea operand este o locație de memorie diferită de accumulator.

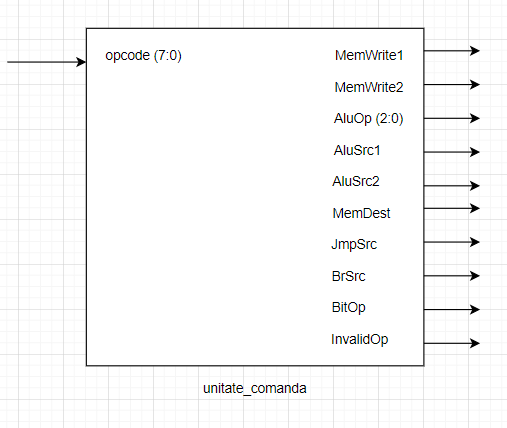
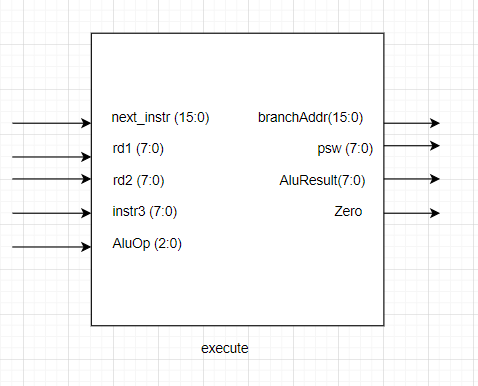


Figura 7. Entitatea unitate\_comanda



Unitatea de execuție realizează operațiile de adunare, scădere, ȘI logic, SAU logic, de transfer, operațiile pe biți și calculează de asemenea adresa de branch ca sumă între adresa următoarei instrucțiuni și valoarea de pe al treilea byte. Semnalul Zero este activat atunci cand rezultatul operației executate este 0, pentru a fi folosit la generarea semnalului de branch. În acest modul se calculează și valorile flag-urilor CY și AC în cazul operațiilor de adunare, scădere și clear carry. Enitatea modulului este prezentată în Figura 8.

Figura 8. Entitatea execute

În modulul principal toate aceste componente sunt interconectate pentru a realiza astfel funcțiile unui microcontroller.

# Rezultate experimentale

Descrierea sistemului în limbajul de descrierea hardware VHDL cu ajutorul mediului de proiectare Xilinx Vivado IDE a fost verificată cu ajutorul bancurilor de test aplicate fiecărei componente în parte.

În Figura 9 de mai jos se poate observa simularea modulului instruction\_fetch. În memoria ROM se afla 3 instrucțiuni:

x"74", x"05", MOV A, 5

x"43", x"02", x"22", ORL (02), 22

x"C3", CLR C

Ieșirile au valori corecte, întrucât prima instrucțiune are 2 bytes și al treilea este setat la x”00”, a doua instrucțiune are 3 bytes și ultima instrucțiune are un singur byte deci următorii 2 sunt setați la x”00”.

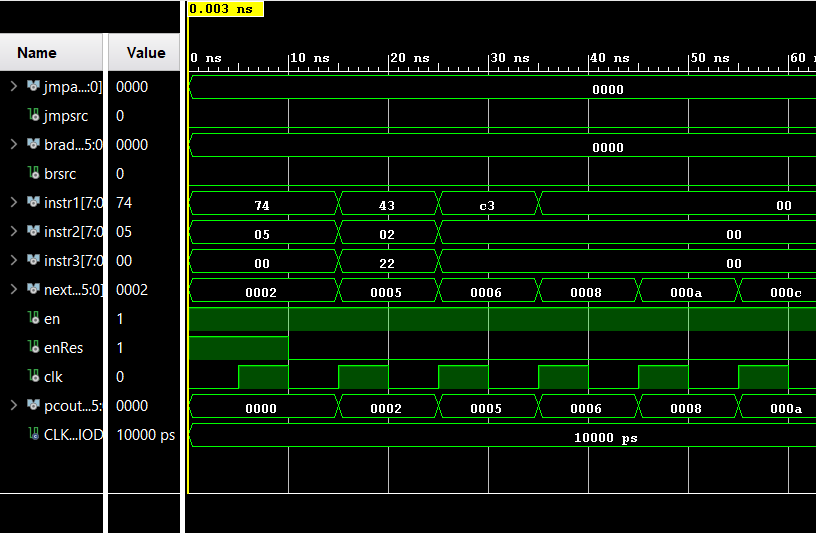


Figura 9. Rezultatul simulării modulului instruction\_fetch

Următoarea simulare ilustrată în Figura 10 verifică funcționarea modulului de decodificare a instrucțiunii și totodată verifică și memoria RAM deoarece aceasta este instanțiată in modulul decode.

Instr1 este 00 deoarece la acest pas nu este relevant opcode-ul. La fel AluRes si PSW au valori irelevante. Inițial semnalele de scriere sunt activate si MemDst=0 ceea ce înseamnă că destinația va fi acumulatorul cu adresa E0. La această adresa va fi scrisă valoarea 05, ceea ce se verifică la urmatorul pas, unde AluSrc1 si AluSrc2 sunt active și se vor citi valorile de la adresele E0 si D0 scrise anterior (D0 este adresa PSW). La următorul pas se verifică și o citire a unei valori care se afla anterior în memorie.

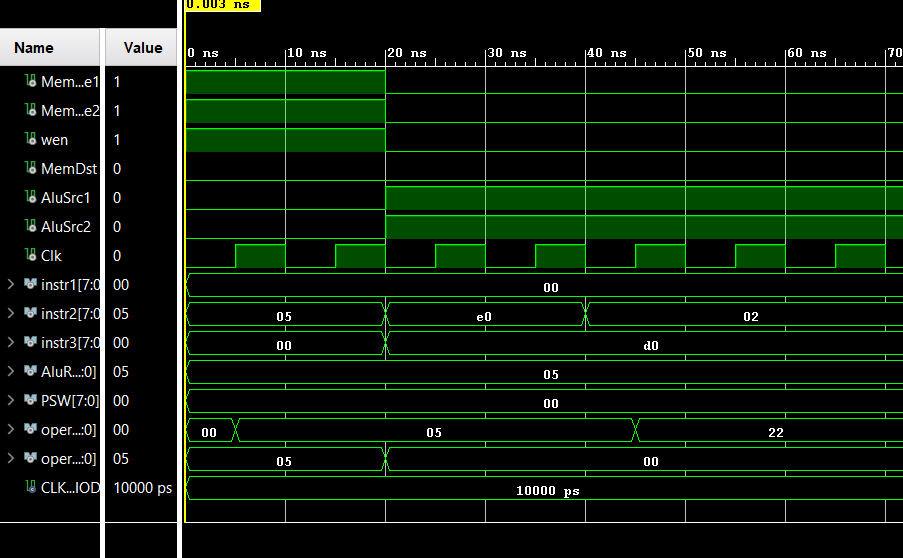
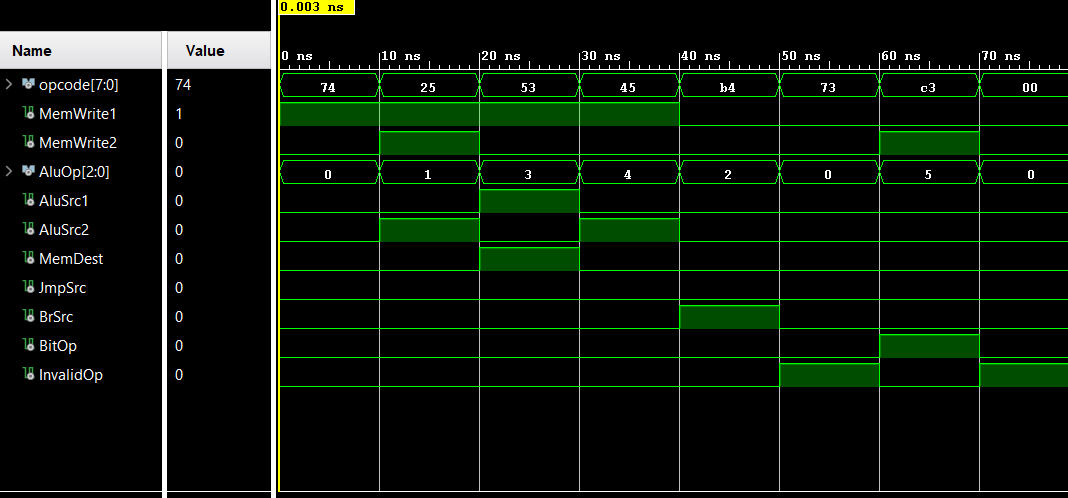


Figura 10. Rezultatul simulării modulului decode

În următoarea imagine din Figura 11 se observă funcționarea corectă a unității de comandă care generează valorile semnalelor de comandă în funcție de opcode-ul fiecărei instrucțiuni.

Figura 11. Rezultatul simulării modulului unității de comandă

În simularea componentei execute ilustrată în Figura 12 se verifică operația de adunare și operația de scădere, fiecare fiind executată cu succes.

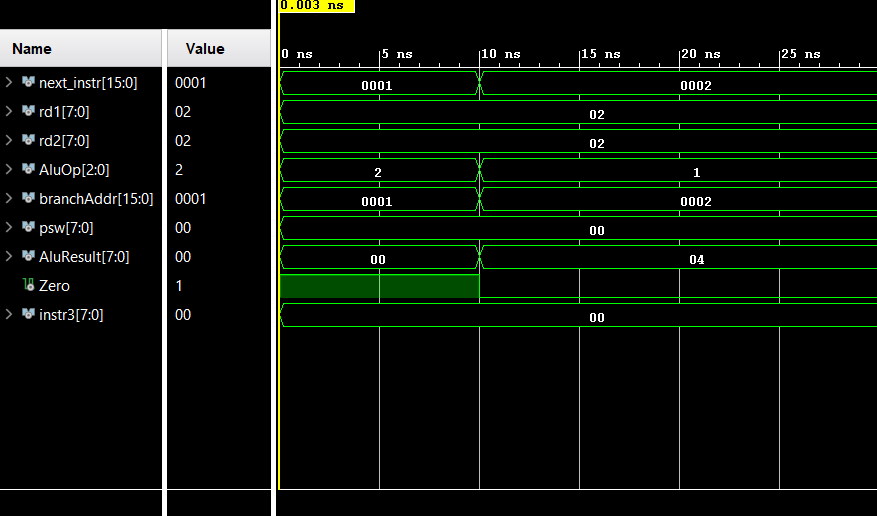


Figura 12. Rezultatul simulării modulului execute

În final se verifică și funcționarea sistemului obținut prin interconectarea tuturor componentelor verificate mai sus, după cum se observă în Figura 13. În memoria ROM este scris un program care acopera aproape toate instrucțiunile implementate.

x"74", x"05", --mov a, 05

x"75", x"02", x"04", --mov (02), 04

x"25", x"02", --add a, (02)

x"94", x"01", --sub a, 01

x"43", x"02",x"07", --orl (02), 07

x"74", x"95", --mov a , 95

x"24", x"70", --add a, 70

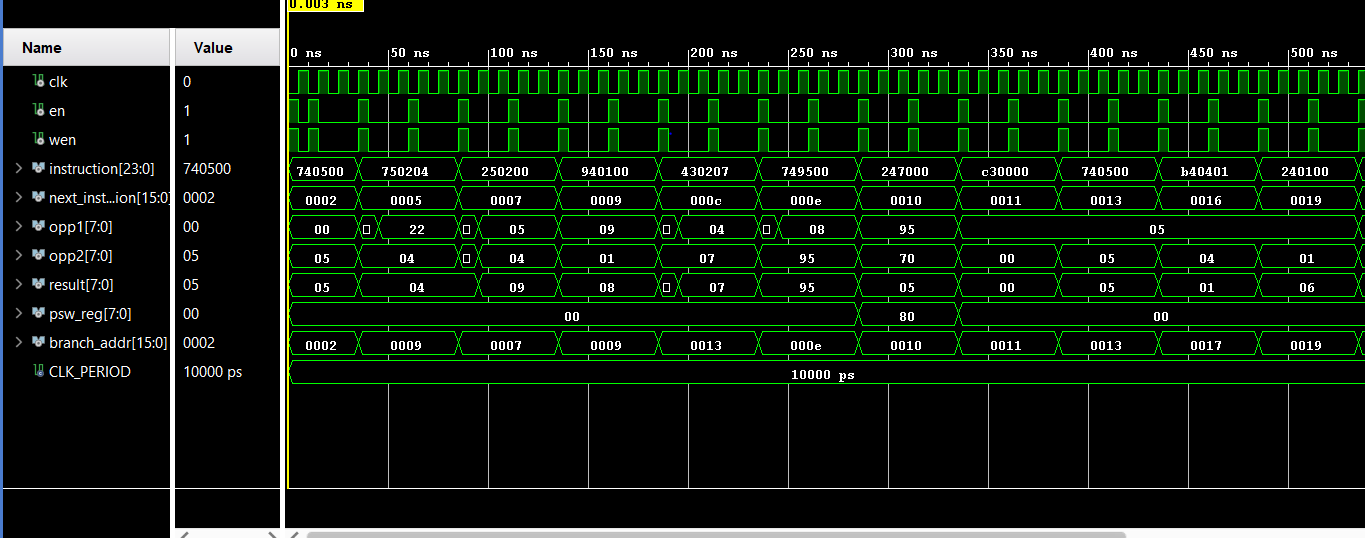
x"C3", --clear carry

x"74", x"05", --mov a, 05

x"B4", x"04",x"01", --cjne a, 05, 01

x"00",

x"24", x"01", --add a, 01

Figura 13. Rezultatul simulării modulului principal

Analiza rapoartelor de implementare: Figura 14 ilustrează sumarul utilizării resurselor, la fel si Tabelul 1.

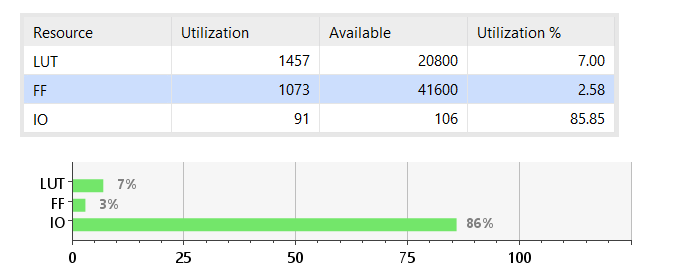


Figura 14. Sumarul generat în urma implementării

Tabel 1. Utilizarea resurselor

|  |  |  |  |
| --- | --- | --- | --- |
| Modul | Slice LUTs | Slice Registers | Slice |
| Decode | 630 | 1048 | 357 |
| Instr\_fetch | 810 | 21 | 319 |
| Execute | 17 | 4 | 13 |

# Concluzii

Acest raport a urmărit proiectarea unui microcontroller 8051, care trebuie să realizeze câteva operații aritmetice, logice, de tranfer, de manipulare a biților și de salt. Implementarea în mod structural a fost realizată prin împărțirea sistemului în module care au fost interconectate într-un modul principal.

Consider că am reușit să respect pe cât posibil caracteristicile acestui microcontroller, deși am redus numărul de instrucțiuni si unele funcționalități. În ceea ce privesc rezultatele, sunt foarte mulțumită de faptul că simulările funcționează corect atât pentru fiecare modul în parte cât și pentru modulul principal.

De asemenea, în urma realizării acestui proiect pot spune că am învățat multe lucruri despre acest microcontroller care este extrem de utilizat și am învățat că microcontrollerele, în general, au o importanță foarte mare în diferite domenii. Cunoștințele mele despre limbajul de descriere hardware VHDL pot spune ca s-au aprofundat odată cu dezvoltarea acestui proiect.

Desigur, acest proiect poate fi extins prin creșterea numărului de instrucțiuni ce pot fi executate, adăugarea unor funcționalități noi și, bineînteles, optimizarea codului.

**Bibliografie**

[1] <https://www.elprocus.com/8051-microcontroller-architecture-and-applications/>

[2] <https://www.tutorialspoint.com/embedded_systems/es_microcontroller.htm>

[3] <https://www.tutorialspoint.com/embedded_systems/es_overview.htm>

[4] <https://www.sciencedirect.com/topics/computer-science/embedded-systems>

[5] <https://engineering.eckovation.com/8051-microcontroller-architecture-block-diagram-explained/>

[6] <http://users.utcluj.ro/~baruch/ssc/labor/Proiectare-Structurala.pdf>

[7] <https://ro.scribd.com/document/422396169/The-8051-Microcontroller-Architecture-Programming-And-Applications-pdf>

[8] <https://pdfs.semanticscholar.org/3eb0/873249c004dfb12057c62cefc8ad9b7d3792.pdf>

[9] <https://www.electronicshub.org/8051-microcontroller-architecture/>

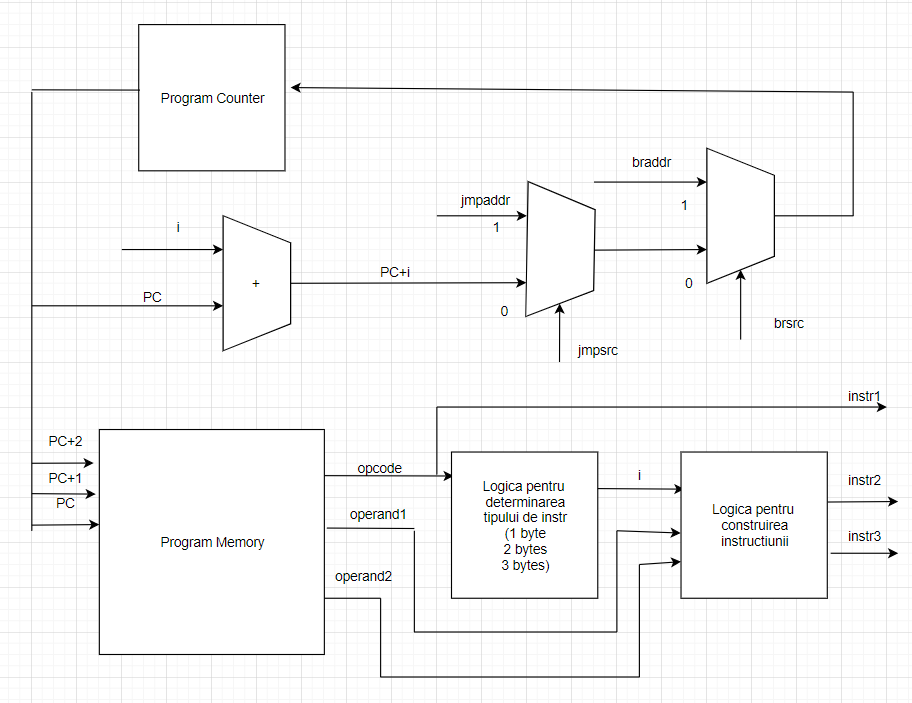
[10] <https://pdfs.semanticscholar.org/3eb0/873249c004dfb12057c62cefc8ad9b7d3792.pdf>

[11] <https://www.electronicshub.org/8051-microcontroller-instruction-set/>

[12] <https://www.electronicshub.org/8051-microcontroller-memory-organization/>

**ANEXE**

Schema în detaliu a modulului de extragere a instrucțiunii este prezentată mai jos:



1. <https://www.elprocus.com/8051-microcontroller-architecture-and-applications/> [↑](#footnote-ref-1)
2. <https://www.tutorialspoint.com/embedded_systems/es_microcontroller.htm> [↑](#footnote-ref-2)
3. <https://www.tutorialspoint.com/embedded_systems/es_overview.htm> [↑](#footnote-ref-3)
4. <https://www.sciencedirect.com/topics/computer-science/embedded-systems> [↑](#footnote-ref-4)
5. <https://engineering.eckovation.com/8051-microcontroller-architecture-block-diagram-explained/> [↑](#footnote-ref-5)
6. <http://users.utcluj.ro/~baruch/ssc/labor/Proiectare-Structurala.pdf> [↑](#footnote-ref-6)
7. <https://ro.scribd.com/document/422396169/The-8051-Microcontroller-Architecture-Programming-And-Applications-pdf> [↑](#footnote-ref-7)
8. <https://pdfs.semanticscholar.org/3eb0/873249c004dfb12057c62cefc8ad9b7d3792.pdf> [↑](#footnote-ref-8)
9. <https://www.electronicshub.org/8051-microcontroller-architecture/> [↑](#footnote-ref-9)
10. <https://pdfs.semanticscholar.org/3eb0/873249c004dfb12057c62cefc8ad9b7d3792.pdf> [↑](#footnote-ref-10)
11. <https://www.electronicshub.org/8051-microcontroller-instruction-set/> [↑](#footnote-ref-11)
12. <https://www.electronicshub.org/8051-microcontroller-memory-organization/> [↑](#footnote-ref-12)