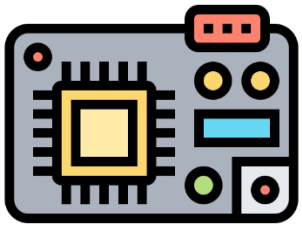


Facultatea: Automatică și Calculatoare

Specializarea: Calculatoare si tehnologia informației

Disciplina: Proiectarea sistemelor numerice

Grupa: 30212

Îndrumător:

Ing. Diana Pop

Realizatori:

Nedelcu Ioan-Andrei

Polman Marian

Cuprins

1. Specificația proiectului

1.1 Cerința

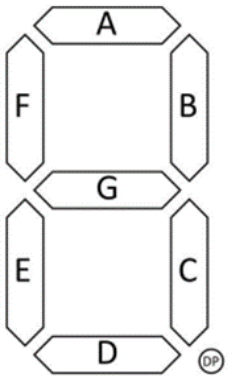
Se urmărește realizarea unui microcontroler pe 8 biți conform documentației XAPP213.pdf și executarea unui program care să folosească toate instrucțiunile implementate.

1.2 Anexa

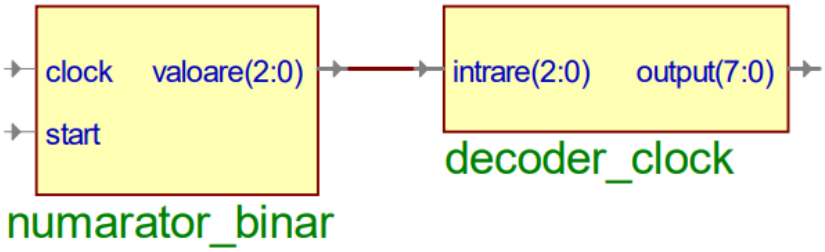
Afișor

Afișor	
Input: • Primele doua cifre • Clock • Ultimele doua cifre	
Output: • Anod 4 • Catod 8	

Componenta Afișor convertește cifrele hexazecimale în catozi și anozi pentru a putea fi afișate pe display. Catozii reprezintă segmentele ce trebuie activate pentru a afișa numărul, iar anozii poziția numărului afișat.



Enable Global

Enable global	
Input: • Clock • Buton Start	
Output: • Magistrala de Enable	

Componenta Enable Global ne permite să activăm componentele secvențial cu ajutorul unui numărător in bucla 0-7; “000” fiind starea de “wait” în care se așteaptă apăsarea butonului start.

Mai apoi stările de la 1-7 sunt decodificate; fiecare stare activând un singur semnal de pe magistrala de enable.

Monostabil

Monostabil	
Input: • Clock • Buton	
Output: • Buton Mono	

Componenta Monostabil, ce are la baza 2 bistabili D se comporta asemănător cu instrucțiunea rising_edge permițând ca în simularea pe placă apăsarea butonului să fie stabilă.

Exemplu Cod

ADR	Assembly	Flowul programului				Codificare
00	ADD S1, FAx	FA->01				"0100000111111010"
01	LOAD S5, 98x	98->02				"0000010110011000"
02	LOAD SC, 1Ax	1A->03				"0000110000011010"
03	AND S1, SC	1A->04				"1100000111000001"
04	LOAD S2,SF	00->05				"1100001011110000"
05	ADD S1, C0x	DA->06	9A->06			"0100000111000000"
06	JMPNC 05x	00->05	00->07			"1001110000000101"
07	ADDCY SC, S5		BE->08			"1100110001010101"
08	XOR S6, E3		E3->09			"0011011011100011"
09	OR S3, S5		98->0A			"1100001101010010"
0A	SUB S4, 44x		BC->0B			"0110010001000100"
0B	SUBCY S6, S4		26->0C			"1100011001000111"
0C	ADD S8, S9		00->0D		EE->0D	"1100100010010100"
0D	JUMPZ 0Fx		00->0F		00->0E	"1001000000001111"
0E	OR S7, F5x				F5->0F	"0010011111110101"
0F	ADD S9,EEx			EE->10	DC->10	"0101100111101110"
10	JUMPNC 0Cx			00->0C	00->11	"1001110000001100"
11	EXIT				00->00	"1111111111111111"

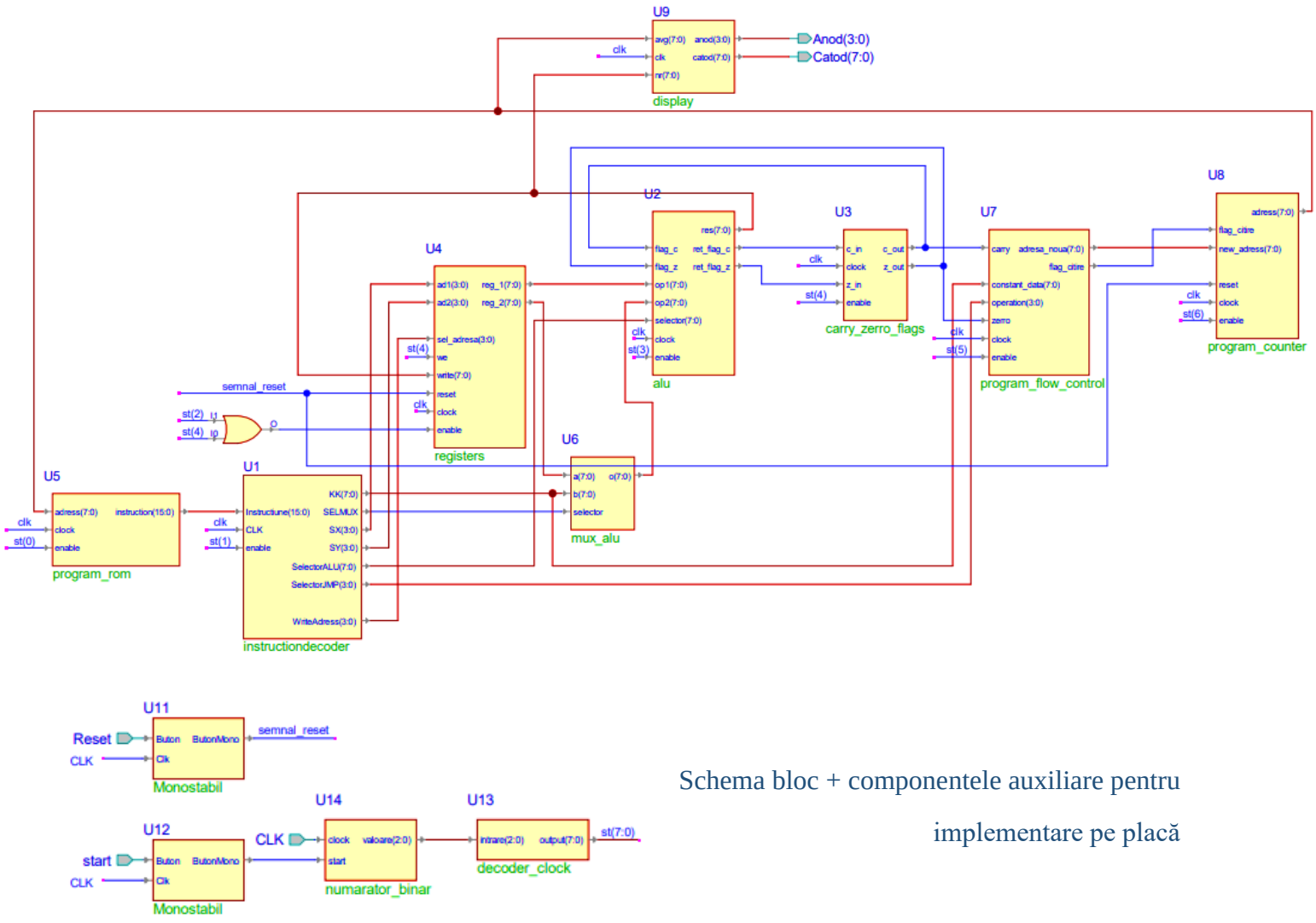
Celulele colorate în albastru semnalează că a avut loc un jump la celula colorată în galben de pe prima coloană din dreapta.

"0100000111111010","0000010110011000","0000110000011010","1100000111000001",
"1100001011110000","0100000111000000","1001110000000101","1100110001010101",
"0011011011100011","1100001101010010","0110010001000100","1100011001000111",
"1100100010010100","1001000000001111","0010011111110101","0101100111101110",
"1001110000001100","1111111111111111",

Copy-Paste Friendly

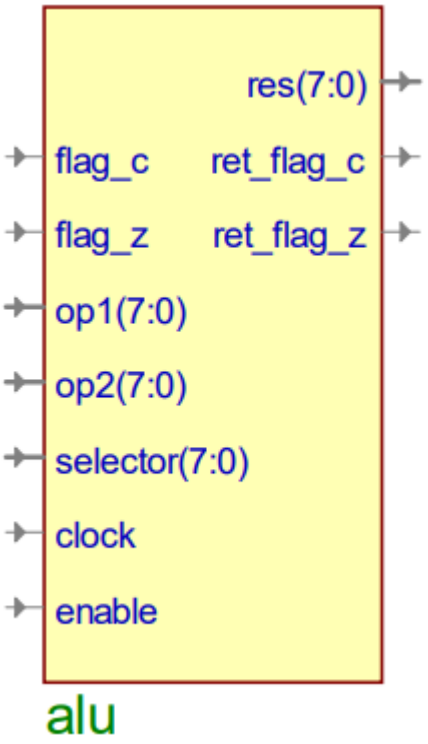
2.Proiectarea

2.1 Schema Bloc



Schema bloc + componentele auxiliare pentru implementare pe placă

2.2 Componentele

ALU	
Input: • Clock • Enable • Carry Flag Precedent • Zero Flag Precedent • Primul Operator • Al doilea Operator • Selector Operație	
Output: • Rezultatul Operației • Carry Flag • Zero Flag	

Componenta ALU este unitatea responsabilă de efectuarea operațiilor simple, de bază. Se execută operația de pe selector între cei doi operanzi de pe input, ținând cont și de flaguri. Unitatea furnizează mai departe rezultatul și dă trigger flagurilor de carry și zero.

Operație	Codificare
Load	0000 0000
And	0000 0001
Or	0000 0010
Xor	0000 0011
ADD	0000 0100
ADDCY	0000 0101
SUB	0000 0110
SUBCY	0000 0111
NULL	1111 1111
INC	0000 1000
DEC	0000 1001

Registrii

Input:

- Clock
- Enable
- Adresa Registrului 1
- Adresa Registrului 2
- Primul Operator (Pentru write)
- Valoarea de scris
- Write Enable
- Reset

Output:

- Valoarea de la Adresa1
- Valoarea de la Adresa2

Componenta Registrii este unitatea în care se încarcă valori constante și rezultatele unor operații. Primește pe magistrala ”write” valoarea ce trebuie înscrisă la adresa Sel_Adresa și trimite mai departe pe cele două magistrale de output valoarea de pe regiștrii adresați pe input.

În momentul în care WE (write enable) este 1, rezultatul operației va fi înscris la adresa primului registru.

Regiștrii	
S0	S8
S1	S9
S2	SA
S3	SB
S4	SC
S5	SD
S6	SE
S7	SF

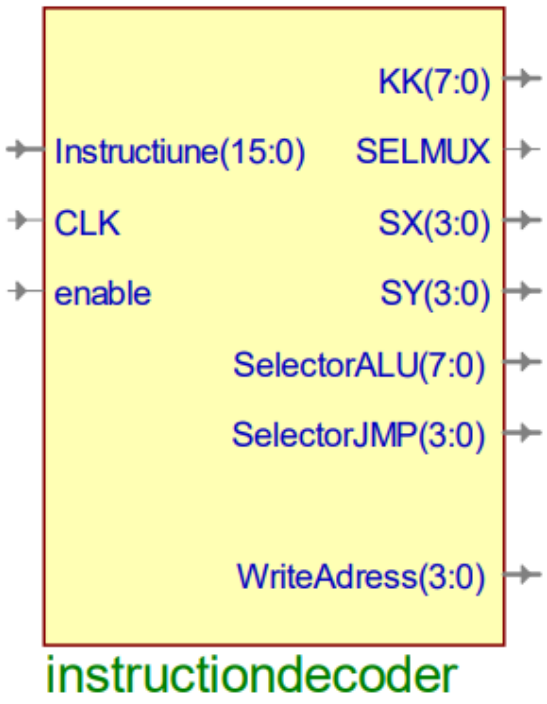
Instruction Decoder

Input:

- Clock
- Enable
- Instrucțiune

Output:

- Valoare Constanta
- Selecția Multiplexor
- Adresa Registru1
- Adresa Registru2
- Selector Operație ALU
- Selector Operație JMP
- Adresa de inscripționat



Componenta Instruction Decoder este unitatea care se ocupă de descifrarea instrucțiunilor înscrise în Rom. Aici se încarcă o Instrucțiune codificată care este segmentată în toate modurile pentru a putea trimite orice informație este necesară spre celelalte componente.

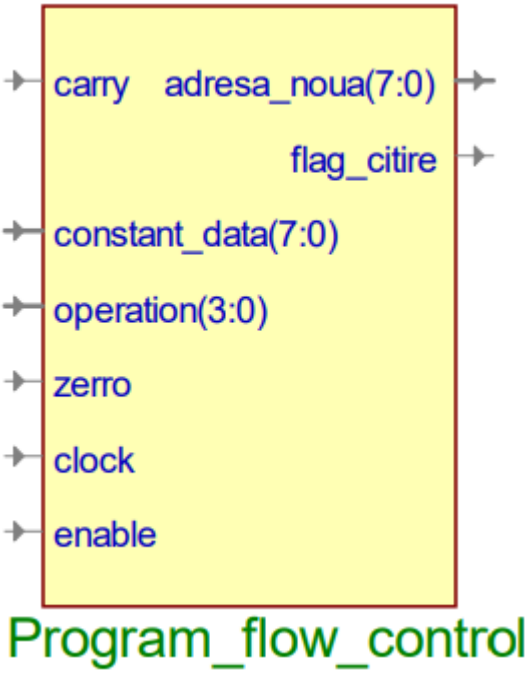
Program Flow Control

Input:

- Clock
- Enable
- Carry Flag
- Zero Flag
- Adresa Constanta
- Operația de Jump

Output:

- Noua Adresa
- Flagul de jump

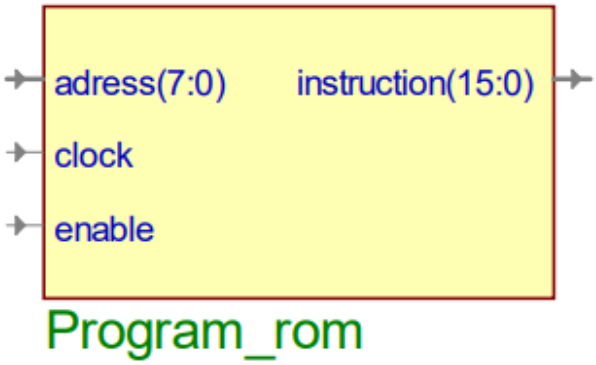


Componenta Program Flow Control este cea responsabila de flowul programului, ea dictează mai departe dacă s-a realizat un jump necondiționat sau condiționat și trimite adresa de jump mai departe spre program counter.

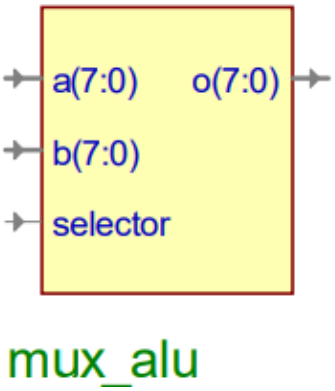
Operație	Codificare
JMP	0000
JMPZ	0001
JMPNZ	0010
JMPC	0011
JMPNC	0100
NULL	1111

Program Counter	
Input: • Clock • Enable • Reset • Flagul de jump • Adresa Constanta	
Output: • Noua Adresa	

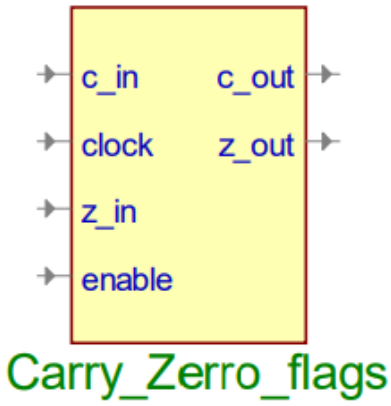
Componenta Program Counter este cea care trimite valoarea adresată de ROM, aceasta incrementează adresa precedenta daca flag_citire nu semnalează un jump și trimite spre ROM adresa la care se efectuează jumpul.

Program Rom	
Input: • Adresa Instrucțiunii • Clock • Enable	
Output: • Instrucțiune	

Componenta Program Rom este memoria în care sunt stocate instrucțiunile codificate, aceasta primește adresa instrucțiunii și o trimite spre Instruction_decoder.

Multiplexor ALU	
Input: • Valoarea A din Registru • Valoarea constanta B • Selector Valoare	
Output: • Output Mux	

Componenta Multiplexor ALU trimite Unității Artimtetico Logice valoarea constantei dacă Instruction Decoder decide că instrucțiunea este una de tip Sx, KK sau trimite valoarea de pe registru Sy daca este o instrucțiune de tip Sx, Sy.

Carry and Zero Flags	
Input: • Carry Flag Actual • Zero Flag Actual • Clock • Enable	
Output: • Carry Flag Precedent • Zero Flag Precedent	

Componenta Carry and Zero Flags este unitatea ce reține valoarea flagurilor de pe operația anterioară și stochează valoarea flagurilor de pe operația actuală. Aceasta facilitează operațiile tip ADDCY, JUMP condiționat, SUBCY.

3. Justificarea soluției alese

Privind funcționalitatea pe placă, am ales să afișăm rezultatul fiecărei instrucțiuni pe primele două cifre și adresa instrucțiunii următoare pe ultimele două cifre pentru a putea observa dacă operațiile aritmetice sau cele de jump s-au executat corect.

Am adăugat în plus doar componentele ce ne ajută să avem un comportament favorabil pe placă.

Privind codul, am optat să folosim în mare parte arhitectură comportamentală pentru a fi mai ușor de înțeles, de corectat și pentru a facilita dezvoltarea ulterioară. Această abordare ne-a permis să implementăm operațiile gradual. Zonele de cod care se pot dezvolta sunt comentate și explicate.

4. Manual de utilizare și întreținere

KK- valoarea constanta în Hexazecimal > **kkkk kkkk** valoarea constantei în binar

SX- registrul cu adresa X > **xxxx** valoarea adresei **X** în binar

SY- registrul cu adresa Y > **yyyy** valoarea adresei **Y** în binar

AA- adresa în Hexazecimal > **aaaa aaaa** valoarea adresei în binar

w- valoare în binar fără importanță

Operație	Codificare
Operații Registru-Constantă	
LOAD SX, KK	0000 xxxx kkkk kkkk
AND SX, KK	0001 xxxx kkkk kkkk
OR SX, KK	0010 xxxx kkkk kkkk
XOR SX, KK	0011 xxxx kkkk kkkk
ADD SX, KK	0100 xxxx kkkk kkkk
ADDCY SX, KK	0101 xxxx kkkk kkkk
SUB SX, KK	0110 xxxx kkkk kkkk
SUBCY SX, KK	0111 xxxx kkkk kkkk
Operații Registru-Registru	
LOAD SX, SY	1100 xxxx yyyy 0000
AMD SX, SY	1100 xxxx yyyy 0001
OR SX, SY	1100 xxxx yyyy 0010
XOR SX, SY	1100 xxxx yyyy 0011
ADD SX, SY	1100 xxxx yyyy 0100
ADDCY SX, SY	1100 xxxx yyyy 0101
SUB SX, SY	1100 xxxx yyyy 0110
SUBCY SX, SY	1100 xxxx yyyy 0111
Operații de JUMP	
JMP AA	1000 wwww aaaa aaaa
JMPZ AA	1001 00ww aaaa aaaa
JMPNZ AA	1001 01ww aaaa aaaa
JMPCY AA	1001 10ww aaaa aaaa
JMPNCY AA	1001 11ww aaaa aaaa
EXIT	1111 1111 1111 1111

Sunt încărcate în ROM instrucțiunile codificate în acest mod, fără spații.

După ce au fost încărcate instrucțiunile în rom, se pornește dispozitivul. Se aplică butonul de Reset.

Pentru a rula fiecare instrucțiune în parte se comuta pe ”on” switchul de start, rezultatul instrucțiunii este afișat pe primele doua cifre ale afișorului iar pe ultimele două este afișată adresa instrucțiunii următoare, pentru a putea urmări dacă va avea sau nu loc un Jump.

În cazul în care rezultatul nu este cel așteptat se recomandă verificare codificării instrucțiunii.

Această abordare permite observarea cu ușurință a greșelilor.

5. Posibilități de dezvoltare ulterioară

ALU este implementat să realizeze și incrementări și decrementări. Instruction Decoder are adresate toate tipurile de shiftări.

În continuare mai pot fi adresate o multitudine de operații (faptul ca avem variabile de tipul don’t care în lista de adresare marchează acest lucru) operațiile adresate pot fi de salt, iar Program_flow_control permite cu ușurință adăugarea acestora, iar operațiile aritmetice pot fi implementate în ALU cu minime cunoștințe în algoritmică.

Operațiile implementate de noi sunt standard, dar se pot implementa și operații mai inedite care să nu existe în limbajul Assembly de bază.

Faptul că programul este sintetizabil și poate fi verificat pe placă este un avantaj pentru dezvoltatorul următor.

6. Bibliografie

-PicoBlaze 8-Bit Microcontroller for Virtex-E and Spartan-II/IIE Devices

-Nexys 3 FPGA Board Reference Manual

-Nexys 3 board turorial (Decoder, ISE 13.2)