

# Facultatea de Automatică și Calculatoare Departamentul Calculatoare si tehnologia informatiei

# **STM8 8-bit Microcontroller**

Nume: Popirtan Vlad-Mihai

Grupa: 30238

Profesor indrumator: Dragos Florin Lisman

## 0) Cuprins

1.	Rezumat	2
2.	Introducere	2
	Fundamentare teoretica.	
	Proiectare si implementare	
	Rezultate experimentale	
	Concluzii	
-	Bibliografie	

## 1)Rezumat

Tema proiectului este un microcontroller pe 8 biti din familia STM8. Acest microcontroller este impartit in trei nivele de pipelining: FETCH, DECODE, EXECUTE, unde fiecare nivel are propriul sau rol in executarea instructiunilor. Microcontrollerul este construit din mai multe blocuri precum: ALU( Arithmetic logic unit), Registre, Stack etc. Fiecare instructiune este codificata intr-un format stabilit la crearea designului microcontrollerui, impreuna cu un set de 43 de instructiuni. Pentru testul acestui microcontroller se va incarca problema gasirii primelor 10 numere din sirul lui Fibonacci.

## 2)Introducere

Familia STM8 este proiectata in jurul arhitecturii de baza a microcontrollerelor pe 8 biti, care impreuna cu blocuri periferice precum ROM (Read only memory), Flash Memory, Numaratoare pe 16 biti, RAM (Random Access Memories) etc. duc la un set mai vast de instructiuni, si la rezolvari mai eficiente din punct de vedere al costului. Aceste microcontrollere sunt folosite intr-o gama larga de domenii precum monitoare video, sisteme automate, componente electrice ale masinilor precum radio si alte produse multimedia sau industriale.

Microcontrollerul a fost construit pe o placuta de dezvoltare Basys 3 din familia Artix-7, cu ajutorul sistemului de dezvoltare Xilinx Vivado in limbajul VHDL, un limbaj de descriere hardware folosit pentru a descrie aplicatii cu circuite integrate si porti logice .

## 3)Fundamentare teoretica

Acest microcontroler este unul pe 8 biti, ceea ce inseamna ca datele prelucrate vor fi de dimensiunea 8, cu instructiuni de 32 de biti. Instructiunile sunt impartite in 8 biti pentru prefix, 8 biti pentru opcode, 16 biti pentru cele 2 adrese din memorie sau un imediat cu care se vor efectua operatiuni de incarcare in memorie sau operatii aritmetice. **Opcode** este un cod unic pentru fiecare instructiune, inclusiv acele instructiuni cu doua tipuri de destinatie si sursa. Majoritatea instructiunilor au o **destinatie** si o **sursa**. Destinatia este locul din memorie in care se vor salva rezultatele operatiilor, iar sursa este locul din memorie din care se preiau date pentru a realiza operatii ce necesita doi termeni.

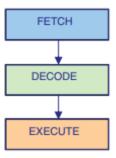
In timpul rularii unui program pot aparea **conflicte de date** atunci cand aceeasi zona din memorie este accesata pentru a realiza:

• Citirea datelor din memorie

• Scrierea datelor in memorie

Din cauza acestor conflicte de date, a aparut arhitectura pipeline, impartita in 3 nivele pentru microcontrollerele din familia STM8

- 1. FETCH: Nivelul FETCH este dedicat preluarii instructiunilor dintr-un bloc de memorie,
- 2. *DECODE:* Nivelul DECODE este nivelul in care se preiau date dintr-o memorie sau din stack. Tot in acest nivel este decodificata instructiunea si trimisa catre blocul execute din nivelul EXECUTE.
- 3. *EXECUTE:* In nivelul EXECUTE sunt identificate semnalele de comanda pentru multiplexoare si semnalele de scriere pentru blocurile de memorii sau registre. Tot in acest nivel se afla blocul ALU, bloc in care se rezolva ecuatiile aritmetice si logice.



#### Modurile de adresare folosite in acest proiect sunt:

- Short direct addressing mode In acest mod de adresare, unul din operanzii instructiunii este o adresa de 8 biti din memorie
- Immediate Addressing mode In acest mod de adresare, unul din operanzi este un operand este o valoarea imediata de 8 biti
- Register adressing mode In acest mod de adresare, unul din registrele de stocare este folosit ca operand.

**Metoda de pipeline** aleasa este execution from Flash Program memory. Prin aceasta metoda este nevoie de 3 cicluri de ceas pentru a fi umplut bufferul de instrucituni, o instructiune fiind scrisa pe 32 de biti, iar bufferul avand 96 de biti. Apare efectul de stall prin conditia ca o noua operatie poate fi transmisa in buffer atunci cand exista un loc pentru aceasta.

Optimized pipeline example - execution from Flash

Instruction	Decod.	Exec.	lath							Су	cle						
instruction	cycles	cycles	lgth	1	2	3	4	5	6	7	8	9	10	11	12	13	14
NEG A	1	1	1		D	Е											
XOR A, \$10	1	1	2	F <sub>1</sub>		D	Е										
LD A, #20	1	1	2	_		D	Е										
SUB A,\$1000	1	1	3		F <sub>2</sub>			D	Ε								
INC A	1	1	1						D	Е							
LD XL, A	1	1	1			_				D	Е						
SRL A	1	1	1			F <sub>3</sub>					D	Е					
SWAP A	1	1	1									D	Е				
SLA \$15	1	1	2				F <sub>1</sub>						D	Е			
CP A,#\$FE	1	1	2				F1							D	Е		

Exemplu preluat din fisa tehnica a microcontrollerului STM8

**Problema Fibonacci:** Sa se gaseasca primele n numeredin sirul lui Fibonacci, unde n = (n-1) + (n-2)

0)mov \$0, 10 - Setam valoarea n din primele n numere fibonacci

1)mov \$1, 2 - Ne setam valoarea cursorului

2)mov \$2, 1 - Setam valoarea initiala a lui a

3)mov \$3, 1 - Setam valoarea initiala a lui b

4)LD A, \$2 - Incarcam valoarea lui a in acumulator

5)add A, \$3 - Adunam a si b

6)mov \$3, \$2 - Mutam in b vechea valoarea a lui a

7)LD \$2, A - Mutam noua valoarea a lui a la zona sa din memorie

8)INC \$1 - Incrementam cursorul

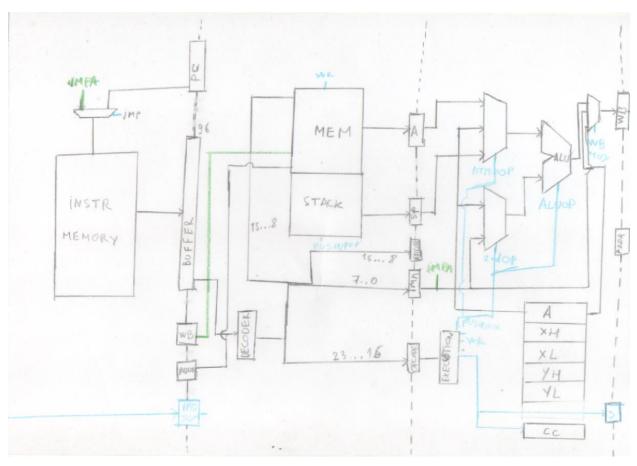
9)LD A, &0 - Mutam valoarea de la adresa 0(n), pentru a o putea compara

10)SUB A,\$1 - Scadem din n, valoarea cursorului pentru a vedea daca programul s-a terminat

11)JEF 4 -Daca valorile nu sunt egale, sarim inapoi la incarcarea lui a in acumulator

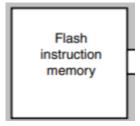
# 4)Proiectare si implementare

Am organizat blocurile in cele 3 nivele de pipeline dupa urmatoarea schema bloc

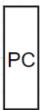


In continuare voi prezenta componentele separat in ordinea nivelelor de pipeline:

## **Fetch:**



Flash instruction memory – Pentru a salva instructionile am ales sa folosesc un flash instruction memory. Acest flash instruction memory va fi activ doar cand Bufferul nu este full si poate prelua instructioni. Acest buffer este structurat pe 96 biti, ceea ce inseamna ca poate pastra 3 instructioni de cate 32 de biti.

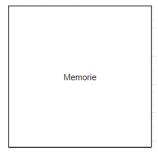


*Program counter* – Aceste registru este un registru pe 24 de biti. In acest registru se salveaza adresa urmatoarei instructiune ce trebuie executata de catre microcontroller.

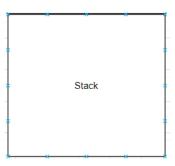
#### Decode/Mem Read



Decode – In acest bloc se imparte instructiunea primita din BUFFER si este decodificata in 4 blocuri de cate 8 biti: prefix, opcode, sursa si destinatie. Prefixul este folosit pentru a distinge doua instructiuni cu acelasi opcode. Opcode-ul reprezinta principalul mod de a diferentia instructiunile intre ele. Sursa si destinatia reprezinta operanzii instructiunilor

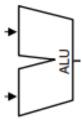


*Memorie* – In blocul de memorie sunt salvate majoritatea datelor folosite de microcontroller.

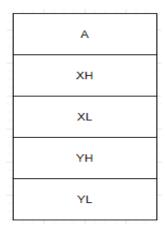


Stack – Stackul functioneaza pe baza unui sistem LIFO (Last in first out). In acest sistem, ultimele date adaugate, sunt cele care vor fi preluate primele. In loc de o adresa la care sunt scrise datele, ele vor fi adaugate la pozitia unui cursos. Acest cursor este incrementat sau decrementat daca are loc o instructiune de incarcat date (PUSH), respectiv preluate din stack(POP).

#### **Execute**



ALU – ALU( sau unitatea aritmetica logica) este un circuit ce are rolul de a executa operatii aritmetice (Adunare, scadere) si operatii logice (Shift left logic, shift right arithmetic etc.). In aceasta componenta are loc si setarea unor biti in blocul condition register.

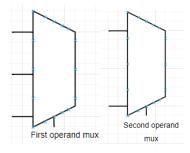


Register Bank – In register bank se afla trei registre de uz general. Acumulatorul este un registru de 8 biti cu rolul de a pastra atat rezultatele operatiilor aritmetice si logice, cat si manipularea datelor. Registrele X si Y sunt registre de 16 biti impartite in cate doua registre de 8 biti. Aceste registre sunt folosite pentru a calcula adrese si pentru a salva temporar date.

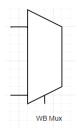
	V	11	н	10	N	Z	С
Ī	-	-	-	-	-	-	-

Condition Code Regtister – Aceste e un registru de 8 biti in care sunt salvate informatii despre ultima instructiune executata. Cele 7 flaguri de conditie sunt V – Overflow, I1 – Intreruperea 1, H – Half Carry, I0 – Intreruperea 0, N – Negative, Z – Zero, C – Carry.

- 1. V Overflow: atunci cand acest bit este setat pe 1, inseamna ca ultima operatie aritmetica a inregistrat o depasire a valorii maxime
- 2. I1 Interrupt mask level 1 Acest flag, impreuna cu I0 descriu daca in program poate aparea o rutina de intrerupere
- 3. H Half carry: Acest bit este setat pe 1 atunci cand are loc un transport la adunarea bitilor 3 si 4 ai operanzilor.
- 4. N Negative: Bitul Negative este setat pe 1 atunci cand rezultatul ultimei operatii este negativ.
- 5. Z Zero: Acest bit este setat pe 1 atunci cand rezultatul ultimei operatii este 0.
- 6. C Carry: Acest bit este setat pe 1 atunci cand are loc un transport.

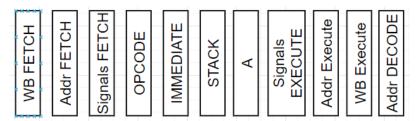


First operand mux/ Second operand mux – Cele doua muxuri controleaza ce linii de date intra in ALU.



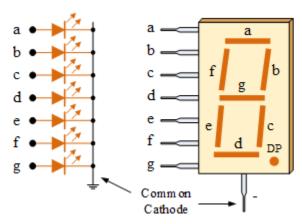
WB Mux – Acest mux controleaza ce linie de date va fi scrisa inapoi in memorie sau in stack.

#### Registre de tranzitie



Aceste registre au rolul de a intarzia datele primite cu un tact de ceas. Aceste registre fac posibila transmiterea datelor intre nivelele de pipeline.

### **Afisare**



Pentru afisare am implementat un display pe 7 segmente. Acesta functioneaza pe baza a 7 leduri pentru fiecare cifra, acestea fiind activate de un semnal de anod. Ca intrari acest modul primeste rezultatul ALU si il decodifica astfel incat sa poata fi afisat pe afisor. Perioada in care se schimba anodul activ este calculata astfel incat sa nu poata fi perceputa de ochiul uman, astfel dand impresia ca toate cifrele sunt active in acelasi timp.

### Setul de instructiuni

Pentru acest microcontroller am ales un set de instructiuni din cele existente pentru microcontrollerul STM8

Instruction group			Description	dst	src	Operation	PREFIX	OPCOD E	ADDR8	ADDR8	CCRegisters affected
Load and Transfer	LD	LD dst,src	Load the destination byte with the source byte	A,mem	mem,A	dst <= src		B6 B7	XX XX		N,Z
	MOV	MOV dst,src	Moves a byte of data from a source address to a destination address.	mem,mem(?)	imm,mem(?)	dst<= src		35 45	IMM XX2	XX1	
	CLR	CLR dst	The destination byte is forced to 00 value			dst <= 00		3F 4F	xx		N,Z
Stack operation	PUSH	PUSH src	Save into the stack the dst byte location. The stack pointer is decremented	A,CC,IMM,ME M		(SP) <= dst		88 8A 4B 3B	XX XX		
	POP	POP dst	Restore from the stack a data byte which will be placed in dst location. The stack pointer is incremented	A,CC,MEM		dst <= (++SP)		84 86 32	XX XX		V,I1,H,I0, N,Z,C(doar pt opcode 86)
Increment/Decrement	INC	INC dst	The destination byte is read, then incremented by one	MEM,A		dst <= dst + 1		3C 4C	XX		V,N,Z

The source byte is subtracted from the contents of the accumulator/SP and the result is stored in the SUB SUB A, SC accumulator/SP and the result is stored in the SUB SUB A, SC accumulator/SP A, SP MEM, IMM A <= A- src    Shift and rotates SUL SUL dist It perform an unsigned multiplication by 2 Mem.Reg    It performs an signed division by 2: The sign bit 7 is not modified    SRA SRA dist modified    Mem.Reg    SRL SRL dist unsigned division by 2 Mem.Reg    SRL SRL dist unsigned division by 2 Mem.Reg    It perform an unsigned division by 2 Mem.Reg    SRL SRL dist unsigned division by 2 Mem.Reg    SRL SRL dist unsigned division by 2 Mem.Reg    The unconditional Jump, simply replaces the content of PC by destination address in same section of memory    PC PC <= IMM    SRD SR the Interrupt mask of the Condition Code (CC) register, which disables    It = 1, 10 Co				T	I						
December								24	\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\		
OCC   OCC dis   Second-morted by one   MULA				The destination byte is read, then					XX		
Compares and Name   N		DEC	DEC dst		MEM.A		dst <= dst - 1	40			V.N.Z
March   Marc				,	,						
Total   Tota								3D	XX		
BCP   BCP Arm   The source for this postbody   A   MEM   MAD   Sec   MEM   M	tests		1 .					4D			
1.0   1.0		TNZ	TNZ ds	(CC) register are updated accordingly	MEM, Reg		Test(dst)				N,Z
1.0   1.0											
The source byte is abdied from the destination byte and the result is lot. It however, N. Z. C Riggs of the destination byte and the result is lot. It however, N. Z. C Riggs of the destination of the contents of the secundator of the secundator of the source byte, is righted byte. It registers are spotted of the contents of the secundator of the secu				The source byte, is ANDed to the			{N, Z} <= A	B5	XX		
Description		BCP	BCP A,src		A	MEM	AND src				N,Z
Comparison							(NL 7, C) -	B1	XX		
Cogleal operations		CP	CP det erc		Δ	MEM					VN7C
AND Acre contents of the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation Accumu		Ci	Ci dat,aic	condition code (cc) register are updated	Α	IVILIVI	rest (dst - src)				V,14,2,C
AND Acre contents of the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation A MEM Acre ACR are BAD XX Interest the accumulation Accumu	Logical operations							B4	VV		
OR   OR   Air   The source byte, is logically Officed with   A   MEM   A <= A OR   Irc   N.	Logical operations		AND	The source byte, is ANDed with the			A <= A AND	В4	\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \		
The source byte, it begrainly CRed with   A		AND	A,src	contents of the accumulator	A	MEM	src				N,Z
No.   Complements the bit position in corrected of the accumulator   A   MEM   A < A AOR size   No.											
OR				The source byte is logically ORed with				BA	XX		
NOTE   NOTE   NOTE   The source byte, is logically XORed with   A   MEM   A < A XOR   BB   XX   N_Z		OR	OR A.src		A	MEM	A <= A OR src				N.Z
No.			1				1				
No.								RR	XX		
Red the destination byte, set the corresponding bit (bit position), and write the result in destination byte.   MEM   #pos   pos								50			
B   Corresponding bit (iii) position), and write   SET   S		XOR	XOR A,src	the contents of the accumulator	А	IMEM	src				N,Z
BR   Coparation   BST	-		+	t	1	_	+ +		_		
BEST				-							
SET   dst.#pos   the result in destination byte   MEM   #pos   pos   dst <= dst   AND   dst <= dst   AND   COMPLEMEN   12   IIMM   MEM	Bit Operation		RSET				det <= det OD	11		IMM	
Read the destination byte, reset the corresponding bit (bit position), and write the result in destination hyte.    RCPL   RCPL		BSET			MEM	#pos					
BRES   dst.#pos   write the result in destination byte.   MEM   #pos   T.pos   12   I.M.M		DOE!	ustympos	the result in destination byte	William	проз					
BRES   ddisplay   with the result in destination byte.   MEM				Read the destination byte, reset the			AND	12		10.40.4	
BCPL st.   Complements the bit position in destination location. Leaves all other bits on changed.   SCPL   St.   September   St.								12		IIVIIVI	
BCPL dst, destination location, Leaves all other bits unchanged.   MEM   Bpos   dst(pos) = 1   13   MMM   MEM   Bpos   dst(pos)   13   MMM   MEM   Bpos   dst(pos)   13   MMM   MEM   Bpos   dst(pos)   14   MEM   MEM   Bpos   dst(pos)   14   MEM		BRES	dst,#pos	write the result in destination byte.	MEM	#pos	T pos				
BCPL dst, destination location, Leaves all other bits unchanged.   MEM   Bpos   dst(pos) = 1   13   MMM   MEM   Bpos   dst(pos)   13   MMM   MEM   Bpos   dst(pos)   13   MMM   MEM   Bpos   dst(pos)   14   MEM   MEM   Bpos   dst(pos)   14   MEM				Complements the hit position in							
Anthematic operations    Anthematic operations   Anthematic operations   The destination byte is read, then each bit is toggled (inverted), and the each bit is toggled (inverted), and the exemulator and the result   ADD   ADD   AJD   AJS			BCPL dst.	-			dst(nos) <= 1	13		IMM	
Arithmetic operations  NEG NEG state of the accumulation and the result is incremented of the source byte is added to the contents of the accumulator and the result is content to contents of the accumulator and the result stored in the accumulator of the accum		BCPL			MEM	#pos					
operations  NEG NEG st bit it toggled (inverted), and the result sold result is incremented on the result of the accumulator and the result of the accumulator and the result of the accumulator and the result of the result is stored in the accumulator/SP and the result is a stored in the accumulator/SP and the result is a stored in the accumulator/SP and the result is a stored in the accumulator/SP and the result is a stored in the accumulator/SP and the result is a stored in the accumulator/SP and the result is a stored in the accumulator/SP and SP an			'	- C		1	" /				
NEG NEG dst result is incremented MEM,A XOR FF) + 1  NEG NEG dst result is incremented MEM,A XOR FF) + 1  NEG NEG dst result is incremented MEM,A XOR FF) + 1  NEG NEG dst result is incremented MEM,A XOR FF) + 1  NEG NEG dst result is incremented MEM,A XOR FF) + 1  NEG NEG STATE AND ASTRICT STATES				-				I	XX		
The source byte is added to the contents of the accumulator and the result of the accumulator of the accu	operations							40			
ADD ADD A, strick is stored in the accumulator A MEM A <= A+ src V, H,N,Z,I  The source byte is subtracted from the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the content of SLL SLL dat unsigned multiplication by 2 Mem.Reg 388 XX		NEG	NEG dst	result is incremented	MEM,A		XOR FF) + 1				V,N,Z,C
ADD ADD A, strick is stored in the accumulator A MEM A <= A+ src V, H,N,Z,I  The source byte is subtracted from the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the contents of the accumulator/SP and the result is stored in the content of SLL SLL dat unsigned multiplication by 2 Mem.Reg 388 XX				The source byte is added to the contents							
The source byte is subtracted from the contents of the accumulator/SP and the result is stored in the SUB SUB A, Src laccumulator/SP and the result is stored in the SUB SUB A, Src laccumulator/SP A, SP MEM, IMM A <= A- src								BB	XX		
Condition Code		ADD	ADD A,sro		Α	MEM	A <= A+ src				V,H,N,Z,C
Shift and rotates  SLL SLL dst   It perform an   Unique division by 2   Mem.Reg   SRA SRA dst   SRL dst   Unique division by 2   Mem.Reg   SRA SRA dst   SRL dst   Unique division by 2   Mem.Reg   SRA SRA dst   SRL dst   SRL dst   Unique division by 2   SRA SRA dst   SRL dst   SRL dst   Unique division by 2   Mem.Reg   SRA SRA dst   SR											
SUB SUB A,src   accumulator/SP   A,SP   MEM,IMM   A <= A- src   V.N.Z.C    Shift and rotates   It perform an								I	1		
Shift and rotates  SLL SLL dst unisgned multiplication by 2 Mem.Reg  It perform an unisgned multiplication by 2 Mem.Reg  It performs an signed division by 2: The sign bit 7 is not modified  SRA SRA dst modified  Mem.Reg  SRL SRL dst unisgned division by 2 Mem.Reg  It perform an unisgned division by 2 Mem.Reg  N,Z,C  Unconditional Jump or Call  JP JP dst address in same section of memory PC  PC =IMM  NOP NOP NOP nothing.  Set the Interrupt mask of the Condition Code (CC) register, which disables in the runter unispned (CO register, which disables in same)  Clear the Interrupt mask of the Condition Code (CC) register, which enable		CLIB	CLID A		A CD	DAEDA IDADA	0 0	52	XX		VN7C
It perform an unsigned multiplication by 2 Mem.Reg  It perform an unsigned multiplication by 2 Mem.Reg  It performs an signed division by 2: The sign bit 7 is not modified  SRA SRA dst modified  SRA SRA dst unsigned division by 2 Mem.Reg  SRI SRL dst unsigned division by 2 Mem.Reg  Unconditional Jump or Call  JP dst address in same section of memory PC  PC PC =IMM  PC PC <=IMM  Condition Code Flag modification  It perform an Mem.Reg  N,Z,C  Mem.Reg  A4 XX  N,Z,C  IMM  PC PC <=IMM  PD PC PC <=IMM  It perform an Mem.Reg  It performs an STA XX AVA AVA AVA AVA AVA AVA AVA AVA AVA	1	ISOR	JOD A,SIC	accumulator/3F	IH,3F	IVILIVI, IIVIIVI	IA <- A-SIC	ı	1	1 1	V,1V,Z,C
It perform an unsigned multiplication by 2 Mem.Reg  It perform an unsigned multiplication by 2 Mem.Reg  It performs an signed division by 2: The sign bit 7 is not modified  SRA SRA dst modified  SRA SRA dst unsigned division by 2 Mem.Reg  SRI SRL dst unsigned division by 2 Mem.Reg  Unconditional Jump or Call  JP dst address in same section of memory PC  PC PC =IMM  PC PC <=IMM  Condition Code Flag modification  It perform an Mem.Reg  N,Z,C  Mem.Reg  A4 XX  N,Z,C  IMM  PC PC <=IMM  PD PC PC <=IMM  It perform an Mem.Reg  It performs an STA XX AVA AVA AVA AVA AVA AVA AVA AVA AVA			1				1				
SLL SLL dst unsigned multiplication by 2 Mem.Reg	Shift and rotates							38	XX		
It performs an signed division by 2: The sign bit 7 is not modified Mem,Reg 37 XX 47 N,Z,C    It performs an signed division by 2: The sign bit 7 is not modified Mem,Reg 34 XX 44 N,Z,C    It perform an		CII	CII I	1 7	M D			48			1170
SRA SRA dst division by 2: The sign bit 7 is not modified Mem,Reg 47 A7		SLL	SLL ast	unsigned multiplication by 2	iviem.Keg	+	+ +		+		N,Z,C
SRA SRA dst division by 2: The sign bit 7 is not modified Mem,Reg 47 N,Z,C    SRL SRL SRL dst   It perform an unsigned division by 2   Mem.Reg 44				It performs an signed				37	xx		
SRA SRA dst modified Mem,Reg 34 XX 1											
It perform an unsigned division by 2   Mem.Reg   44   Mem.Reg		SRA	SRA dst	modified	Mem,Reg						N,Z,C
It perform an unsigned division by 2   Mem.Reg   44   Mem.Reg											
SRL SRL dst unsigned division by 2 Mem.Reg  Unconditional Jump or Call  JP JP dst address in same section of memory PC  NOP NOP NOP NOP nothing.  Set the Interrupt mask of the Condition Flag modification SIM = 1 interrupts  Condition Code (CC) register, which lil = 1, 10 enable  Condition Code (CC) register, which enable				It perform on					XX		
Unconditional Jump or Call  JP JP dst address in same section of memory PC PC <=IMM  This is a single byte instruction that does nothing.  Set the Interrupt mask of the Condition Flag modification SIM = 1 interrupts  Condition Code (CC) register, which lil = 1, 10 code (CC) register, which lil = 1, 10 enable  Condition Code (CC) register, which lil = 1, 10 enable  Condition Code (CC) register, which lil = 1, 10 enable  The unconditional jump, simply replaces the content of PC by destination address in same section of memory PC  PC <=IMM  9D  IIMM  19D  III = 1, 10 Code (CC) register, which lil = 1, 10 enable		SRI	SRI det	1 .	Mem Reg			44			N 7 C
Jump or Call  JP JP dst the content of PC by destination address in same section of memory PC PC <i (cc)="" a="" byte="" code="" condition="" disables="" does="" flag="" i0="" i1="1," ii0="" imm="" instruction="" is="" look="" modification="" nothing.="" of="" register,="" single="" td="" that="" the="" the<="" this="" which=""><td></td><td>ONE</td><td>JIL USL</td><td>ansigned division by 2</td><td>wemneg</td><td>+</td><td>+ +</td><td></td><td>+</td><td></td><td>۱۷,۷,۵</td></i>		ONE	JIL USL	ansigned division by 2	wemneg	+	+ +		+		۱۷,۷,۵
Jump or Call  JP JP dst the content of PC by destination address in same section of memory PC PC <i (cc)="" a="" byte="" code="" condition="" disables="" disables<="" does="" flag="" i0="" i1="1," ii="1," iii="1," iio="" imm="" instruction="" is="" modification="" nothing.="" register,="" single="" td="" that="" this="" which=""><td></td><td></td><td></td><td>The unconditional jump, simply replaces</td><td></td><td></td><td></td><td></td><td></td><td>18.45.4</td><td></td></i>				The unconditional jump, simply replaces						18.45.4	
This is a single byte instruction that does  NOP NOP NOP nothing.  Set the Interrupt mask of the Condition Flag modification SIM = 1 interrupts Clear the Interrupt mask of the Condition Code (CC) register, which lil = 1, 10 enable    11 = 1, 10   Code (CC) register, which lil = 1, 10	Jump or Call									IIVIIVI	
NOP NOP nothing.  Condition Code Flag modification   SIM   = 1   1   1   1   1   1   1   1   1		JP	JP dst	address in same section of memory	PC		PC <=IMM				
NOP NOP nothing.  Condition Code Flag modification   SIM   = 1   1   1   1   1   1   1   1   1											
NOP NOP nothing.  Condition Code Flag modification SIM = 1 interrupt mask of the Condition SIM = 1 interrupt mask of the Condition Code (CC) register, which disables II, 10 Code (CC) register, which disables II, 10 Code (CC) register, which light mask of the Condition Code (CC) register, which light mask of the Code (CC) register, which light mas				This is a single byte instruction that does				9D			
Condition Code Flag modification SIM = 1 interrupts SIM = 1 interrupt mask of the Condition SIM = 1 interrupts Clear the Interrupt mask of the Condition interrupts SIM = 1 interrupts SIM = 1 interrupt mask of the Condition Code (CC) register, which logister, which logister, which enable		NOP	NOP								
Flag modification   11 = 1,  0   Code (CC) register, which disables   98											
1 = 1,   0   Cobe (CC) register, which disables								QP			
Clear the Interrupt mask of the Condition Code (CC) register, which I1 = 1, I0 enable	Flag modification							30			
Condition Code (CC) register, which		SIM	= 1								11, 10
11=1, 0  enable   9A											
			11 = 1.10					9A			
RIM = 0 interrupts   I1, I0		RIM									11, 10

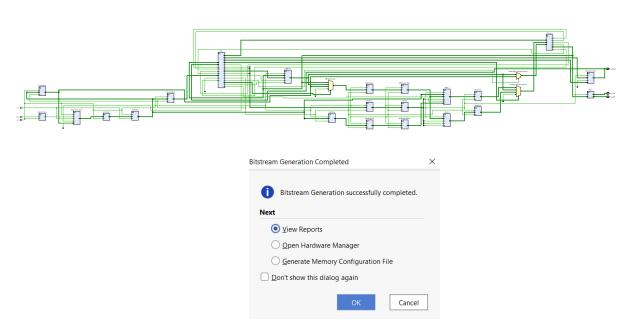
SCF	C=1	Set the carry flag of the Condition Code (CC) register. It may be used as user controlled flag.			99		c
RCF	C=0	Clear the carry flag of the Condition Code (CC) register.			98		С
CCF	CC.C <- CC.C	Complements the Carry flag of the Condition Code (CC) register.			8C		С

Set de instructiuni valabil si la adresa:

https://1drv.ms/x/s!AjvNfVGnHqsEhRV1vmdH6Idd4WoK?e=JozZmf

## 5)Rezultate experimentale

Proiectul nu a fost testat pe placa de dezvoltare deoarece aceasta nu a fost identificata de catre calculator.



## 6)Concluzii

#### Rezumat

In acest proiect s-a implementat o reproducere a unui microcontroller din familia STM8, cu o optimizare pipeline de tip Flash Prgorgram.

### **Dificultati intampinate**

Cel mai complicat proces a fost sa inteleg organizarea componentelor pe nivelele de pipeline, astfel incat sa nu apara conflicte de date.

#### **Dezvoltari ulterioare**

- Adaugarea unui Flash Instruction Memory functional si conceptul de stall pentru o rulare mai eficienta a programelor.
- Introducerea intregului set de instructiuni

## 7)Bibliografie

https://www.st.com/resource/en/programming\_manual/cd00161709-stm8-cpu-programming\_manual-stmicroelectronics.pdf - fisa tehnica a microcontrollerului STM8

Moodle.cs.utcluj.ro/ - Informatii de baza pentru construirea unui microcontroller de n biti <a href="https://en.wikipedia.org/wiki/STM8">https://en.wikipedia.org/wiki/STM8</a> - Setul de instructiuni al microcontrollerului din familia STM8