UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

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

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 instrucitune 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.

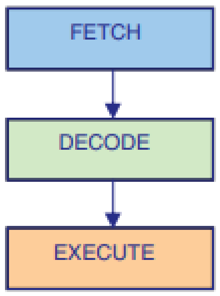
In timpul rularii unui program pot aparea **conflicte de date** atunci cand aceeasi zona din memorie este accesata pentru a realiza:

- 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

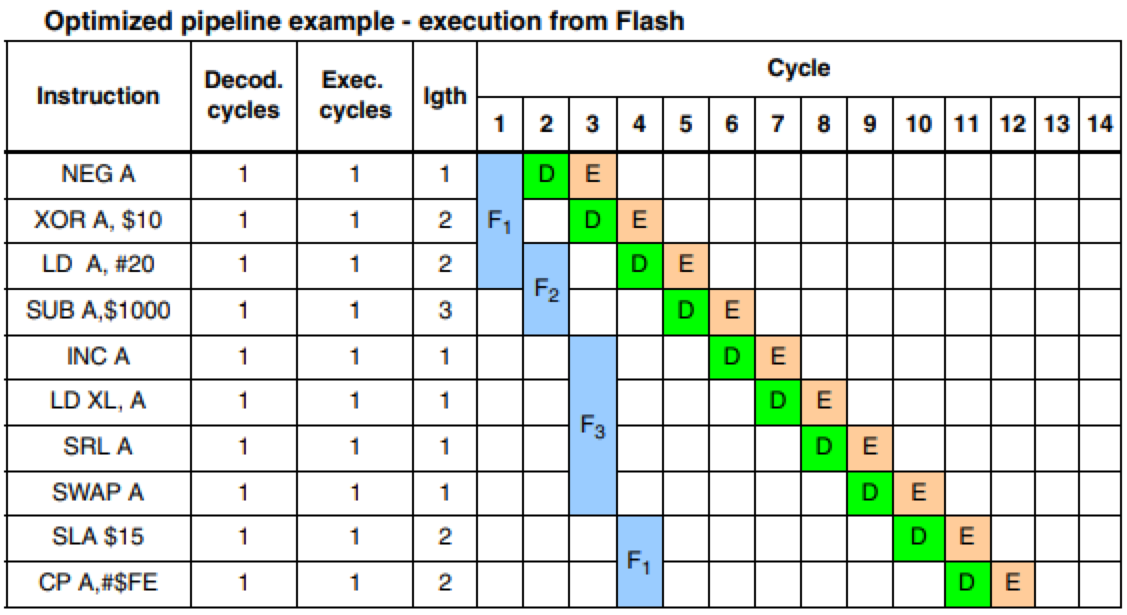
1. *FETCH:* Nivelul FETCH este dedicat preluarii instructiunilor dintr-un bloc de memorie,
2. *DECODE:* Nivelul DECODE este nivelul in care se preiau date dintr-o memorie sau din stack. Tot in acest nivel este decodificata instructiunea si trimisa catre blocul execute din nivelul EXECUTE.
3. *EXECUTE:* 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. cycles	Exec. cycles	lgth	Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13	14
NEG A	1	1	1	F_1	D	E											
XOR A, $10	1	1	2	F_1		D	E										
LD A, #20	1	1	2	F_1	F_2		D	E									
SUB A,$1000	1	1	3		F_2			D	E								
INC A	1	1	1			F_3			D	E							
LD XL, A	1	1	1			F_3				D	E						
SRL A	1	1	1			F_3					D	E					
SWAP A	1	1	1			F_3						D	E				
SLA $15	1	1	2				F_1						D	E			
CP A,#$FE	1	1	2				F_1							D	E		

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

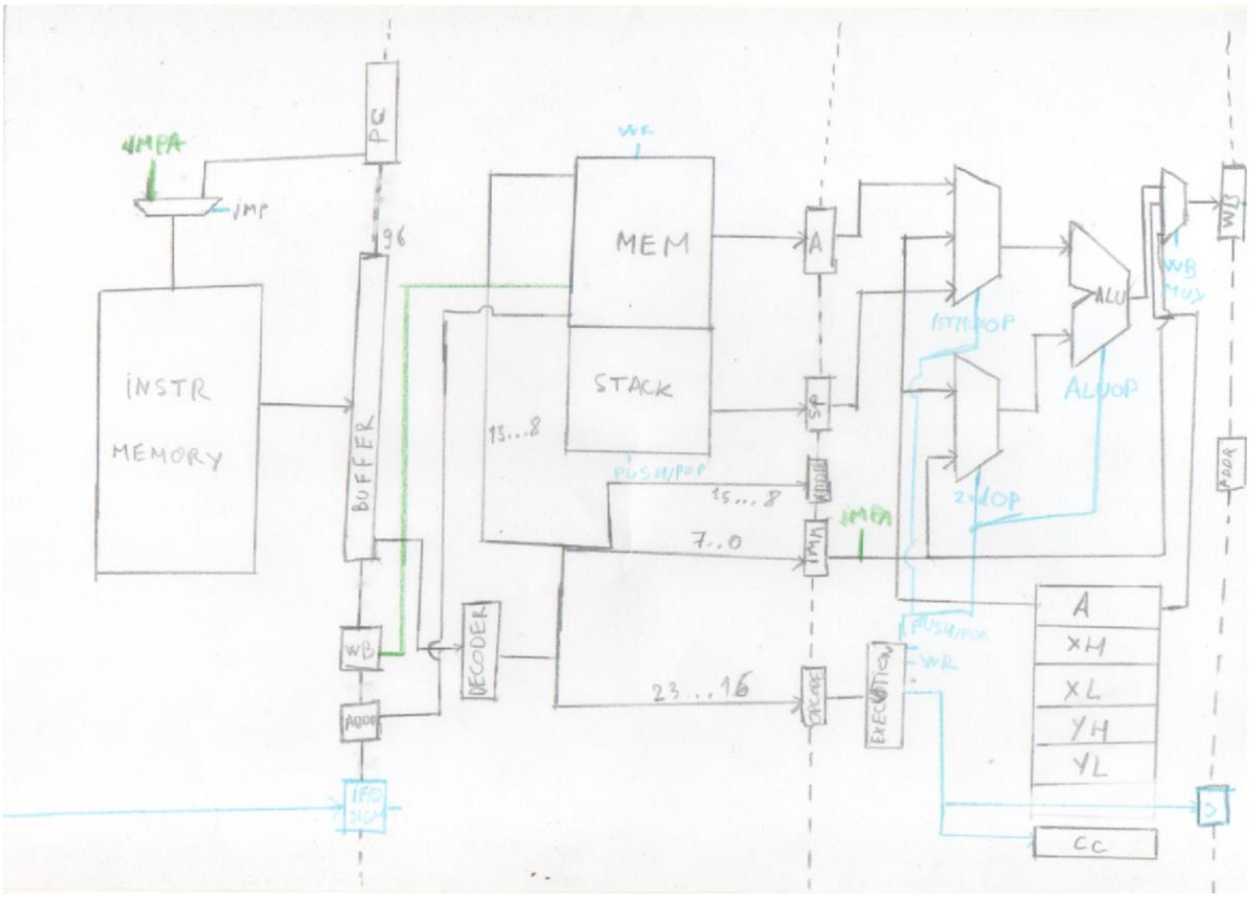
9)LD A, &0 - Mutam valoarea de la adresa 0(n), pentru a o putea compara

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

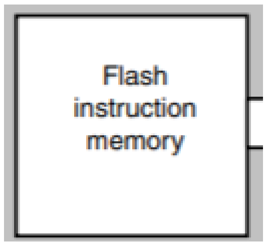
4)Proiectare si implementare

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

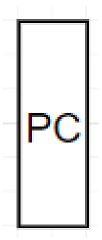


In continuare voi prezenta componentele separat in ordinea nivelelor de pipeline:

Fetch:

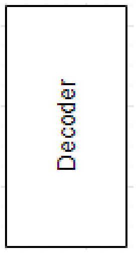


Flash instruction memory – Pentru a salva instructiunile am ales sa folosesc un flash instruction memory. Acest flash instruction memory va fi activ doar cand Bufferul nu este full si poate prelua instructiuni. Acest buffer este structurat pe 96 biti, ceea ce inseamna ca poate pastra 3 instructiuni de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

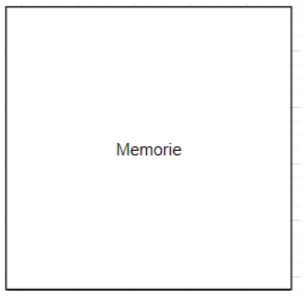


Program counter – Aceste registru este un registru pe 24 de biti. In acest registru se salveaza adresa urmatoarei instructiune ce trebuie executata de catre microcontroller.

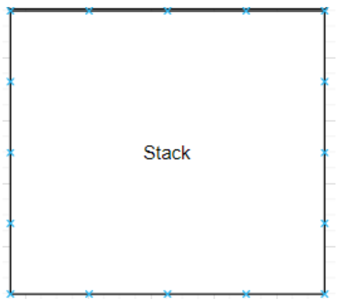
<u>Decode/Mem Read</u>



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

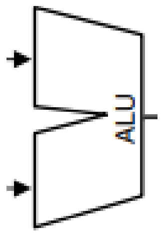


Memorie – In blocul de memorie sunt salvate majoritatea datelor folosite de microcontroller.

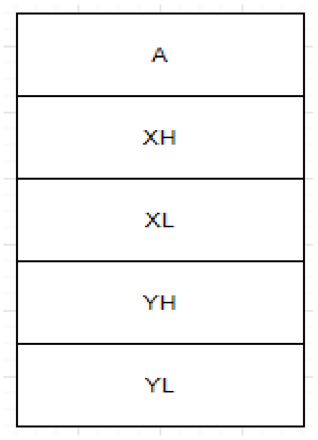


Stack – Stackul functioneaza pe baza unui sistem LIFO (Last in first out). In acest sistem, ultimele date adaugate, sunt cele care vor fi preluate primele. In loc de o adresa la care sunt scrise datele, ele vor fi adaugate la pozitia unui cursos. Acest cursor este incrementat sau decrementat daca are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

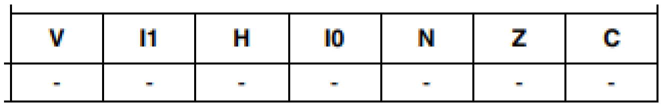
<u>Execute</u>



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.



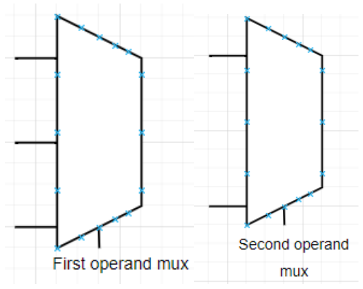
Register Bank – In register bank se afla trei registre de uz general. Acumulatorul este un registru de 8 biti cu rolul de a pastra atat rezultatele operatiilor aritmetice si logice, cat si manipularea datelor. Registrele X si Y sunt registre de 16 biti impartite in cate doua registre de 8 biti. Aceste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



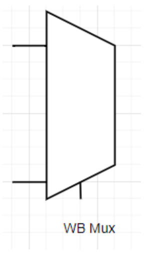
V	I1	H	I0	N	Z	C
-	-	-	-	-	-	-

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.

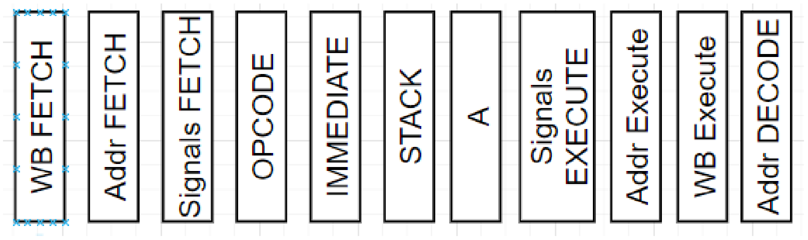


First operand mux/ Second operand mux – Cele doua muxuri controleaza ce linii de date intra in ALU.



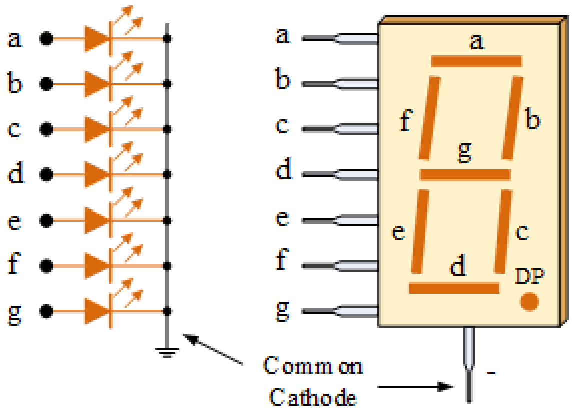
WB Mux – Acest mux controleaza ce linie de date va fi scrisa inapoi in memorie sau in stack.

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.

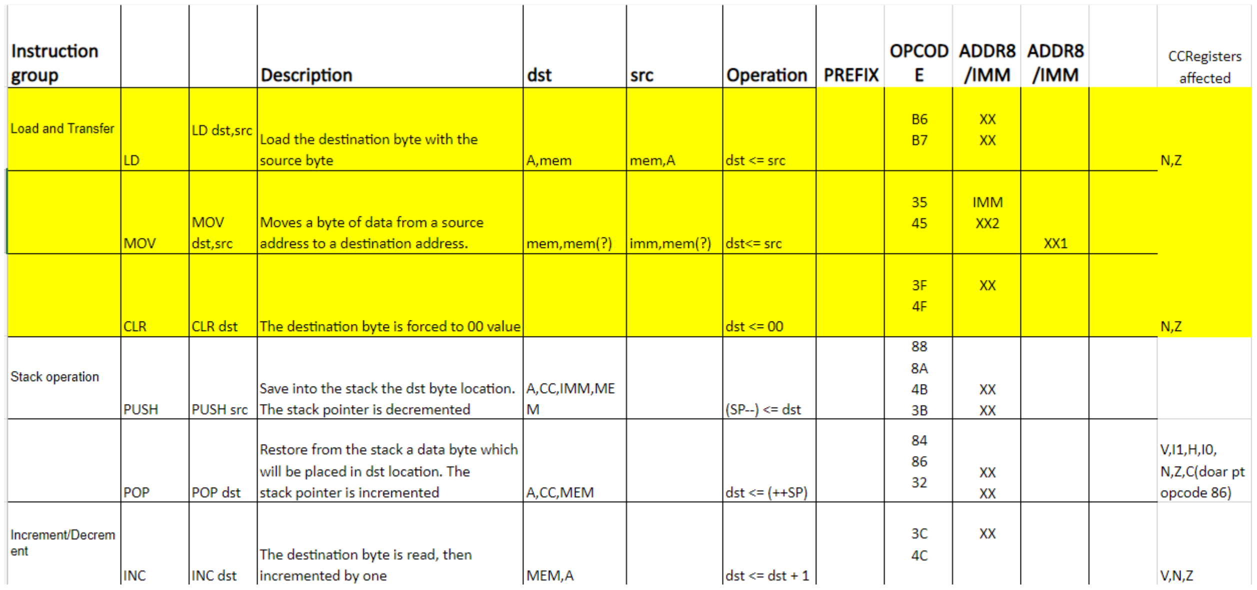
Afisare



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	OPCODE	ADDR8 /IMM	ADDR8 /IMM		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,MEM		(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

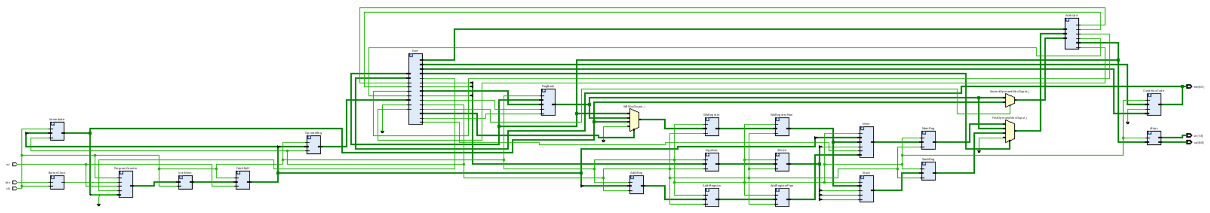
	DEC	DEC dst	The destination byte is read, then decremented by one	MEM,A		dst <= dst - 1		3A 4A	XX			V,N,Z
Compare and tests	TNZ	TNZ ds	The destination byte is tested and both N and Z flags of the Condition Code (CC) register are updated accordingly	MEM, Reg		{N, Z} = Test(dst)		3D 4D	XX			N,Z
	BCP	BCP A,src	The source byte, is ANDed to the contents of the accumulator.	A	MEM	{N, Z} <= A AND src		B5	XX			N,Z
	CP	CP dst,src	The source byte is subtracted from the destination byte and the result is lost. However, N, Z, C flags of Condition Code (CC) register are updated	A	MEM	{N, Z, C} = Test (dst - src)		B1	XX			V,N,Z,C
Logical operations	AND	AND A,src	The source byte, is ANDed with the contents of the accumulator	A	MEM	A <= A AND src		B4	XX			N,Z
	OR	OR A,src	The source byte, is logically ORed with the contents of the accumulator	A	MEM	A <= A OR src		BA	XX			N,Z
	XOR	XOR A,src	The source byte, is logically XORed with the contents of the accumulator	A	MEM	A <= A XOR src		B8	XX			N,Z
Bit Operation	BSET	BSET dst,#pos	Read the destination byte, set the corresponding bit (bit position), and write the result in destination byte	MEM	#pos	dst <= dst OR pos		11		IMM		
	BRES	BRES dst,#pos	Read the destination byte, reset the corresponding bit (bit position), and write the result in destination byte.	MEM	#pos	dst <= dst AND COMPLEMENT pos		12		IMM		
	BCPL	BCPL dst, #pos	Complements the bit position in destination location. Leaves all other bits unchanged.	MEM	#pos	dst(pos) <= 1 - dst(pos)		13		IMM		
Arithmetic operations	NEG	NEG dst	The destination byte is read, then each bit is toggled (inverted), and the result is incremented	MEM,A		dst <= (dst XOR FF) + 1		30 40	XX			V,N,Z,C
	ADD	ADD A,src	The source byte is added to the contents of the accumulator and the result is stored in the accumulator	A	MEM	A <= A+ src		BB	XX			V,H,N,Z,C
	SUB	SUB A,src	The source byte is subtracted from the contents of the accumulator/SP and the result is stored in the accumulator/SP	A,SP	MEM,IMM	A <= A- src		B0 52	XX XX			V,N,Z,C
Shift and rotates	SLL	SLL dst	It perform an unsigned multiplication by 2	Mem.Reg				38 48	XX			N,Z,C
	SRA	SRA dst	It performs an signed division by 2: The sign bit 7 is not modified	Mem,Reg				37 47	XX			N,Z,C
	SRL	SRL dst	It perform an unsigned division by 2	Mem.Reg				34 44	XX			N,Z,C
Unconditional Jump or Call	JP	JP dst	The unconditional jump, simply replaces the content of PC by destination address in same section of memory	PC		PC <=IMM		CC		IMM		
	NOP	NOP	This is a single byte instruction that does nothing.					9D				
Condition Code Flag modification	SIM	I1 = 1, I0 = 1	Set the Interrupt mask of the Condition Code (CC) register, which disables interrupts					9B				I1, I0
	RIM	I1 = 1, I0 = 0	Clear the Interrupt mask of the Condition Code (CC) register, which enable interrupts					9A				I1, I0

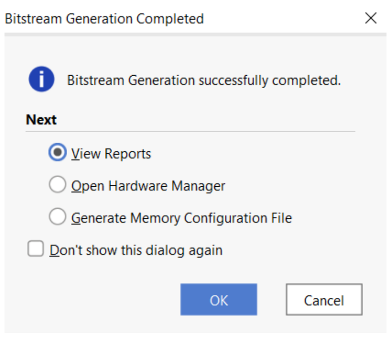
	SCF	C=1	Set the carry flag of the Condition Code (CC) register. It may be used as user controlled flag.					99				C
	RCF	C=0	Clear the carry flag of the Condition Code (CC) register.					98				C
	CCF	CC.C <- CC.C	Complements the Carry flag of the Condition Code (CC) register.					8C				C

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.




Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

[✓ View Reports]

Open Hardware Manager

Generate Memory Configuration File

Don't show this dialog again

OK Cancel

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

https://en.wikipedia.org/wiki/STM8 - Setul de instructiuni al microcontrollerului din familia STM8