

Emulator ISA 13xM

Boțoc Indi (C 1.2), Codreanu Mihai - Constantin (TI 2.1)
Isac Marco-Deian (TI 2.1), Nedelea Maria-Paula (TI 2.1)



CUPRINS

Introducere

slide
3

**Structura
generală**

slide
4

**Setul de
instrucțiuni**

slide
5-10

**Structura
software**

slide
11

Implementare

slide
12-16

**ISA
I3xM**



Introducere



- Acest proiect reprezintă implementarea unui **emulator/simulator pentru un subset de instrucțiuni inspirat de arhitectura RISC-V**, cu scopul de a demonstra conceptele fundamentale de procesare a instrucțiunilor la nivel de procesor. Proiectul este dezvoltat în limbajul C și utilizează biblioteca **ncurses** pentru o interfață terminal-based, care facilitează vizualizarea și interacțiunea cu componentele sistemului, cum ar fi memoria, registrele și stiva.
- **Scopul Proiectului**

Scopul principal al acestui proiect este de a simula funcționarea unui procesor simplificat, incluzând următoarele componente esențiale:

 - ❑ **Memoria** (separată în memorie pentru instrucțiuni, date și stivă).
 - ❑ **Regiștrii** (inclusiv regiștri generali și regiștri speciali, cum ar fi PC – Program Counter și SP – Stack Pointer).
 - ❑ **Decodificarea instrucțiunilor** în funcție de tipurile R, I, S și B.
 - ❑ **Executarea instrucțiunilor**, incluzând operații aritmetice, logice, de salt și de manipulare a memoriei.



Structura generală



Proiectul se compune din:

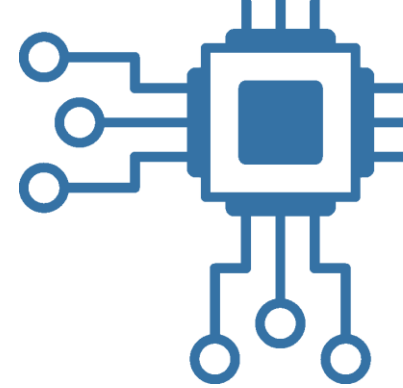
1. Module principale:

- a. Decoder: Decodifică instrucțiunile brute și le mapează în structuri interne.
- b. Encoder: Transformă instrucțiunile în formă binară.
- c. Executor: Evaluează și execută instrucțiunile decodificate.
- d. Memorie: Gestionează accesul și actualizarea memoriei.
- e. Interfață: Afișează informații utilizatorului și preia input-uri.

2. Componente cheie:

- a. Regiștri și Memorie: Sunt reprezentate prin structuri dedicate și vectori pentru eficiență.
- b. Flag-uri: Indicatori de stare ai procesorului pentru execuții condiționate.

Setul de instrucțiuni



RISC-V REFERENCE

- Am folosit setul de instrucțiuni **RISC-V**, a cărui formă completă poate fi vizualizată [aici](#).
- Acest set de instrucțiuni a fost adaptat pentru a putea fi implementat mai ușor.
- Am folosit 3 tipuri de instrucțiuni:
 - ❑ Tipul **R** (instrucțiuni cu ambii operanzi regiștri);
 - ❑ Tipul **I** (instrucțiuni cu operand registru și operand valoare imediată)
 - ❑ Tipul **B** (instrucțiuni de tip branch).

RISC-V Instruction Set

Core Instruction Formats

31	27	26	25	24	20	19	15	14	12	11	7	6	0		
funct7				rs2		rs1		funct3		rd		opcode		R-type	
imm[11:0]								funct3		rd		opcode		I-type	
imm[11:5]				rs2		rs1		funct3		imm[4:0]		opcode		S-type	
imm[12:10:5]				rs2		rs1		funct3		imm[4:1:11]		opcode		B-type	
				imm[31:12]								rd		opcode	U-type
				imm[20:10:11:19:12]								rd		opcode	J-type

RV32I Base Integer Instructions

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srl	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011	0x5		if(rs1 ≥ rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 ≥ rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1101111	0x0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	



Structura unei instrucțiuni



```
#ifndef INSTRUCTION_H
#define INSTRUCTION_H

#include <stdint.h>

// Structura pentru o instructiune decodificata
typedef struct
{
    uint32_t raw;           // Instructiunea bruta
    uint8_t opcode;        // Codul operatiei
    uint8_t funct3;        // Funcție secundara pentru identificare (R/I/B)
    uint8_t funct7;        // Funcție secundara pentru instrucțiuni de tip R
    uint8_t rd;            // Registru destinatie
    uint8_t rs1;           // Primul registru sursa
    uint8_t rs2;           // Al doilea registru sursa
    int16_t imm;           // Valoare imediata
    uint8_t deriv;         // Indicator derivata (doar pentru instrucțiuni B)
    char type;             // Tipul instructiunii (R, I, B)
} Instruction;

#endif
```

Instrucțiuni de tip R

INSTRUCȚIUNE	OPCODE	FUNCT3	FUNCT7	RD	RS1	RS2	COMENTARII
add	011 0011	0x0	0x00	rd	rs1	rs2	Adunare regiștri
sub	011 0011	0x0	0x20	rd	rs1	rs2	Scădere regiștri
mul	011 0011	0x0	0x01	rd	rs1	rs2	Înmulțire regiștri
udv	011 0011	0x5	0x01	rd	rs1	rs2	Câtul împărțirii
mod	011 0011	0x6	0x01	rd	rs1	rs2	Restul împărțirii
lsl	011 0011	0x1	0x00	rd	rs1	rs2	Shiftare logică la stânga
lsr	011 0011	0x5	0x00	rd	rs1	rs2	Shiftare logică la dreapta
cmp	011 0011	0x2	0x00	rd	rs1	rs2	Comparare regiștri
and	011 0011	0x7	0x00	rd	rs1	rs2	Și
xor	011 0011	0x4	0x00	rd	rs1	rs2	SAU exclusiv
or	011 0011	0x6	0x00	rd	rs1	rs2	SAU
mov	011 0011	0x0	0x00	rd	0	rs2	Mutare regiștri

Instrucțiuni de tip I

INSTRUCȚIUNE	OPCODE	FUNCT3	RD	RS1	IMM	COMENTARIU
add	0b0010011	0x0	rd	rs1	imm	Operație de adunare
sub	0b0010011	0x3	rd	rs1	imm	Operație de scădere
xor	0b0010011	0x4	rd	rs1	imm	Operație XOR
or	0b0010011	0x6	rd	rs1	imm	Operație OR
and	0b0010011	0x7	rd	rs1	imm	Operație AND
lsl	0b0010011	0x1	rd	rs1	imm	Shift logic la stânga
lsr	0b0010011	0x5	rd	rs1	imm	Shift logic la dreapta
asr	0b0010011	0x5	rd	rs1	imm (0x2 0 << 5)	Shift aritmetic la dreapta
cmp	0b0010011	0x2	rd	rs1	imm	Comparare implementată cu SLTI
mov	0b0010011	0x0	rd	0	imm	Mutare valoare imediată în registru

Instrucțiuni de tip I*

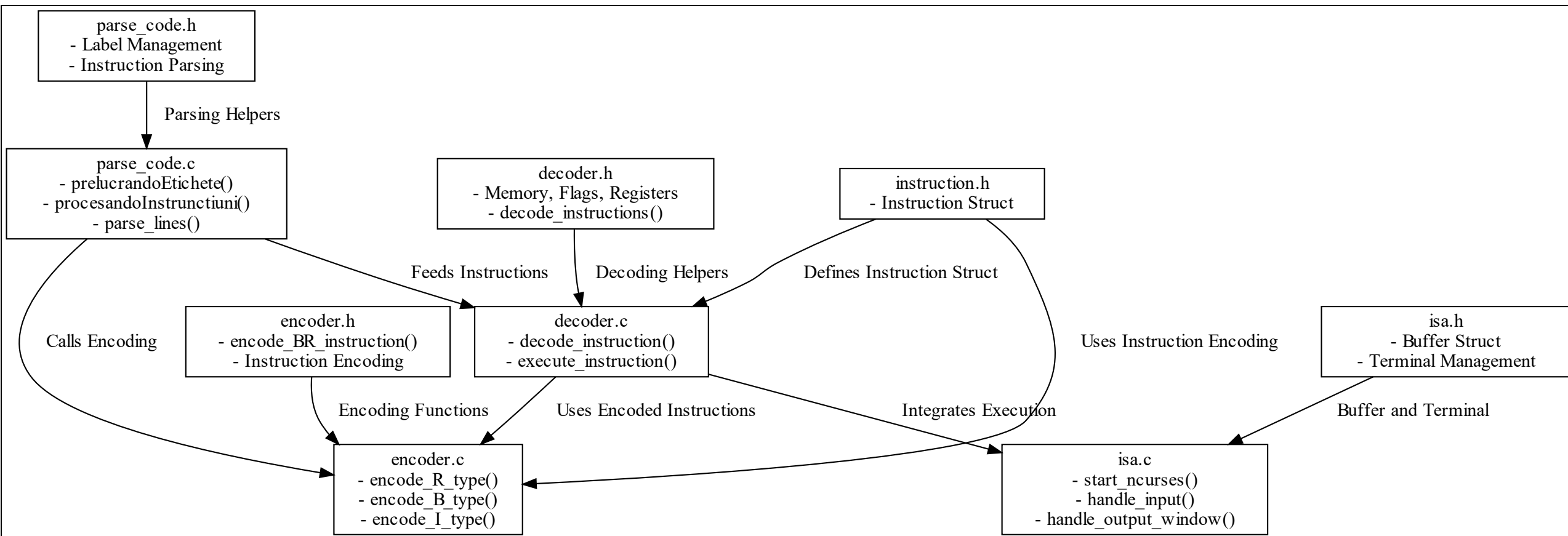
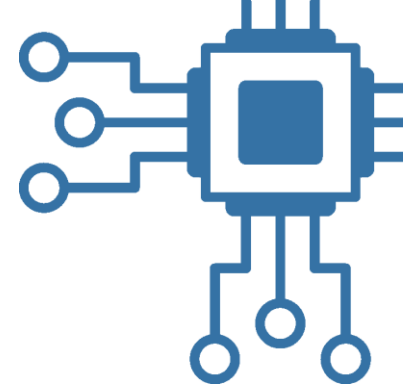
* aceste instrucțiuni nu au neapărat tipul I, dar au fost implementate folosindu-ne de structura lui.

INSTRUCȚIUNE	OPCODE	FUNCT3	RD	RS1	IMM	COMENTARIU
lda	000 0011	0x2	rd	0	imm	Load Address
ldr	000 0011	0x2	rd	rs1	imm	Load Register
str	010 0011	0x2	rd	rs1	imm	Store Register
sta	010 0011	0x2	rd	0	imm	Store Address
psh	010 0011	0x1	rd	0	0	Push pe stivă
pop	000 0011	0x1	rd	0	0	Pop de pe stivă
ret	111 0011	0x1	0	0	0	Return interpretat ca ECALL
hlt	111 0011	0x0	0	0	0	Halt interpretat ca ECALL

Instrucțiuni de tip B

INSTRUCȚIUNE	OPCODE	FUNCT3	RS1	RS2	DERIV	DESCRIERE
bra	110 0011	0x0	0	0	3	Salt absolut
beq	110 0011	0x0	rs1	rs2	0	Salt dacă egal
bne	110 0011	0x1	rs1	rs2	0	Salt dacă diferit
blt	110 0011	0x4	rs1	rs2	0	Salt dacă mai mic
bge	110 0011	0x5	rs1	rs2	0	Salt dacă mai mare sau egal
brz	110 0011	0x0	0	0	1	Salt dacă ZF == 1
brp	110 0011	0x5	0	0	1	Salt dacă ZF == 0 și SF == 0
bmi	110 0011	0x4	0	0	1	Salt dacă SF == 1
bgt	110 0011	0x5	rs1	rs2	2	Salt dacă mai mare
ble	110 0011	0x4	rs1	rs2	2	Salt dacă mai mic sau egal
bvs	110 0011	0x1	0	0	2	Salt dacă OF == 1
bcs	110 0011	0x5	0	0	3	Salt dacă CF == 1
bpl	110 0011	0x5	0	0	4	Salt dacă SF == 0
jms	110 0011	0x0	0	0	4	Salt subrutină

Structura software





Implementarea



ISA

gestionează
terminalul
și interfața

PARSE_CODE


procesează
codul sursă
și etichetele

ENCODER

convertește
instrucțiunile
în cod binar

DECODER

decodifică și
execută
instrucțiunile
și
gestionează
ciclul de
execuție,
memoria,
registrii și
flags-urile



Modulul ISA

```
#ifndef ISA_H
#define ISA_H
```

```
#include "parse_code.h"
#include <ncurses.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
```

```
#define MAX_COLS 256
#define INIT_LINES 100
```

```
typedef struct {
    char **lines;
    int num_lines;
} Buffer;
```

```
// functii pentru interfata
void start_ncurses();
void end_ncurses(WINDOW *menuwin_left, WINDOW *menuwin_right);
```

```
// functii pentru buffer
Buffer *create_buffer();
void insert_char(char *line, int pos, char ch, int max_cols);
void delete_char(char *line, int pos, int max_cols);
void insert_new_line(Buffer *buffer, int line, int max_cols);
void free_buffer(Buffer *buffer);
```

```
// functii pentru diferite lucruri
void display_message(WINDOW *win, const char *format, ...);
void handle_error(WINDOW *win, const char *format, ...);
```

```
// functii pentru ferestre
void init_windows(WINDOW **win1, WINDOW **win2, int height, int width);
void handle_output_window(Buffer *buffer, WINDOW *win2);
void handle_input(Buffer *buffer, WINDOW *win1, WINDOW *win2);
```

```
#endif // ISA_H
```

Funcție pentru inițializarea interfeței

Funcție pentru închiderea interfeței

Funcție pentru afișarea unui mesaj într-o fereastră

Funcție pentru inițializarea ferestrelor

Funcție pentru tratarea scrierii de cod

Funcție pentru împachetarea și
trimiterea codului către parse_code

Modulul PARSE_CODE

```
#ifndef PARSE_CODE_H
#define PARSE_CODE_H
```

```
#include "encoder.h"
#include "isa.h"
#include "decoder.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <ncurses.h>
```

```
#define MAX_ET 512
#define MAX_LINII 512
```

```
typedef struct l {
    int adresa;
    int opcode;
    int regs[3];
    int use_imm;
    int use_dat;
} linie;
```

```
typedef struct ea {
    int adresa;
    char *nume;
} eticheta;
```

```
// variabile globale importante
extern uint32_t *instructiuni;
extern int iCount;
extern WINDOW *win, *win1;
```

```
// functii pentru etichete
void addEticheta(const char *nume, int address);
int getEtichetaAddress(const char *nume);
int inArray(const char *key, char *array[], int size);
int etichetaValida(const char *label);
```

```
// functii pentru linii si instructiuni
// void prelucrando(char *linie);
char* prelucrandoEtichete(char *linie);
void procesandoInstructiuni(char *linie, const int lAcum);
void parse_lines(char **lines, int num_lines, WINDOW *winL, WINDOW *winR);
#endif
```

Structura pentru etichetă

Tabloul de instrucțiuni

Numărul de instrucțiuni

Funcție pentru prelucrarea unei linii
și reținerea etichetelor

Funcție în care prelucrăm instrucțiunile
și le trimitem spre codificare,
apoi le stocăm în tabloul de instrucțiuni

Funcție în care le apelăm pe cele 2 de mai sus și în
care trimitem datele spre decodare și execuție

Modulul ENCODER

```
#ifndef ENCODER_H
#define ENCODER_H
```

```
#include "parse_code.h"
#include "isa.h"
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <stdint.h>
```

```
// functii encoding
```

```
uint32_t encode_R_type(uint8_t opcode, uint8_t rd, uint8_t funct3, uint8_t rs1, uint8_t rs2, uint8_t funct7);
uint32_t encode_I_type(uint8_t opcode, uint8_t rd, uint8_t funct3, uint8_t rs1, int16_t imm);
uint32_t encode_B_type(uint8_t opcode, uint8_t funct3, uint8_t rs1, uint8_t deriv, int16_t imm);
```

```
uint32_t encode_R_instruction(const char *operation, int rd, int rs1, int rs2, int is_mov);
uint32_t encode_I_instruction(const char *operation, int rd, int rs1, const int imm, const int impl);
uint32_t encode_BR_instruction(const char *operation, const char *imm_label, const int lCurenta);
```

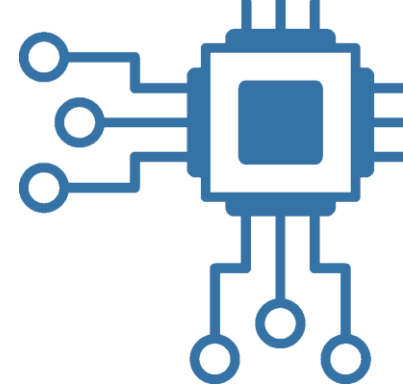
```
#endif // ENCODER_H
```

Funcții pentru codificarea
instrucțiunilor de tip R

Funcții pentru codificarea
instrucțiunilor de tip I

Funcții pentru codificarea
instrucțiunilor de tip B

Modulul DECODER



```
#ifndef DECODER_H
#define DECODER_H
```

```
#include "instruction.h"
#include "isa.h"
#include <stdint.h>
#include <stdio.h>
#include <ncurses.h>
```

```
#define MEMORY_SIZE 3072 // 3KB Total
#define INSTRUCTION_SIZE 1024 // 1KB pentru instructiuni
#define DATA_SIZE 1024 // 1KB pentru date
#define STACK_SIZE 1024 // 1KB pentru stiva
```

```
#define INSTRUCTION_START 0
#define INSTRUCTION_END 1023
```

```
#define DATA_START 1024
#define DATA_END 2047
```

```
#define STACK_START 2048
#define STACK_END 3071
```

```
#define NUM_REGISTERS 8
```

```
typedef struct {
    uint8_t ZF; // Zero Flag
    uint8_t SF; // Sign Flag
    uint8_t OF; // Overflow Flag
    uint8_t CF; // Carry Flag
} CPU_Flags;
```

```
// decodificarea unei instructiuni
Instruction decode_instruction(uint32_t instruction);
```

```
// decodificarea unui vector de instructiuni
void decode_instructions(uint32_t *instructions, int num_instructions, WINDOW *winL, WINDOW *winR);
```

```
#endif
```

Structura de flags

Funcția de
decodificare
a instrucțiunilor

Funcția care
gestionează
fluxul de execuție
al fiecărei instrucțiuni



Bibliografie



- J. L. Hennessy, D. A. Patterson, Computer Architecture: A Quantitative Approach, [acs.pub.ro](https://www.acs.pub.ro)
- RISC-V reference card, [cs.sfu.ca](https://www.cs.sfu.ca)
- Cursurile și laboratoarele de la ITSC, cv.upt.ro
- RISC and CISC in Computer Organization, [geeksforgeeks.org](https://www.geeksforgeeks.org)
- Instruction set architecture, [wikipedia.org](https://www.wikipedia.org)
- What is the instruction set for RISC processors?, [quora.com](https://www.quora.com)
- RISC – Reduced Instruction Set Computer, [lumenci.com](https://www.lumenci.com)
- A Beginner's Guide to RISC and CISC Architectures, [medium.com](https://www.medium.com)