Prosty procesor RISC

Specyfikacja

I. architektura Harvardzka
II. brak adresowania pośredniego
III. brak stosu
IV. pamięć danych RAM: rozmiar 16B, 8-bitowe słowa
V. pamięć programu ROM: rozmiar 1k, 14-bitowe słowa
VI. Rejestry:
    a. A Akumulator 8-bitowy
    b. B Rejestr ogólny 8-bitowy
    c. PC Licznik programu 10-bitowy
    d. IR Rejestr instrukcji 14-bitowy
    e. Z Flaga zera 1-bitowa
    f. C Flaga przeniesienia 1-bitowa
VII. Format instrukcji:
    a. kod operacji 4-bitowy (2^4=16 operacji)
    b. operand 8-bitowy lub
    c. adres 10-bitowy
VIII. Wykonywane operacje
    a. operacje ogólne

	kod	mn.	arg.	opis
1.	1010	mva	#op	załadowanie operandu #op do rejestru A
2.	0001	mvb	#op	załadowanie operandu #op do rejestru B
3.	0010	load	#op	załadowanie bajtu spod adresu RAM[#op] do akumulatora
4.	0011	store	#op	załadowanie bajtu z akumulatora do adresu RAM[#op]
5.	0100	xchg	-	A <-> B wymiana rejestrów A i B
6.	0101	jmp	#op	skok bezwarunkowy do etykiety #op
7.	0110	jz	#op	skok pod warunkiem jedynki w rejestrze zera do etykiety #op
8.	0111	jc	#op	skok pod warunkiem jedynki w rejestrze przeniesienia do etykiety #op
9.	1000	jnz	#op	skok pod warunkiem zera w rejestrze zera do etykiety #op
10.	1001	jnc	#op	skok pod warunkiem zera w rejestrze przeniesienia do etykiety #op
11.	0000	nop	-	brak operacji

    b. operacje logiczne

1.	1011	and	-	A <- A & B	if (A == 0) Z <= 1 else Z <= 0
2.	1100	or	-	A <- A | B	if (A == 0) Z <= 1 else Z <= 0
3.	1101	not	-	A <- ~A	if (A == 0) Z <= 1 else Z <= 0

    c. operacje arytmetyczne

1.	1110	add	-	A <- A + B	if (A==0) Z<=1 else Z<=0	if ((A+B)<A) C<=1 else C<=0
2.	1111	sub	-	A <- A – B	if (A==0) Z<=1 else Z<=0	if ((A+B)>A) C<=1 else C<=0

```
`timescale 1ns / 1ps

module cpu_v1(clk, rst, Aout, Bout, PCout, IRout, Zout, Cout);

input clk;
input rst;

output [7:0]   Aout;
output [7:0]   Bout;
output [9:0]   PCout;
output [13:0]  IRout;
output Zout;
output Cout;

reg [7:0]      A;
reg [7:0]      B;
reg [9:0]      PC;
reg [13:0]     IR;
reg Z;
reg C;

reg [7:0] RAM[0:15];
reg [13:0] ROM[0:1023];



wire [3:0] OPCODE=IR[13:10];
wire [9:0] ADDRESS=IR[9:0];
wire [7:0] OPERATOR=IR[7:0];

assign Aout=A;
assign Bout=B;
assign PCout=PC;
assign IRout=IR;
assign Zout=Z;
assign Cout=C;

always @(posedge clk)
begin
	if (~rst)
		begin
			PC <= 0;
			A <= 0;
			B <= 0;
			Z <= 0;
			C <= 0;
		end
	else
		begin
			IR <= ROM[PC];
			PC <= PC+1;
		end
end

always @(negedge clk)
begin
	case (OPCODE)
		4'b0000:		//nop
			A <= A;
		4'b1010:		//mva arg
			A <= OPERATOR;

		4'b0001:				//mvb arg
			B <= OPERATOR;

		4'b0010:				//load addr
			A <= RAM[ADDRESS];

		4'b0011:				//store addr
			RAM[ADDRESS] <= A;

		4'b0100:				//xchg
			begin
				A <= B;
				B <= A;
			end

		4'b0101:				//jmp addr
			PC <= ADDRESS;

		4'b0110:				//jz addr
			if (Z == 1'b1)
				PC <= ADDRESS;

		4'b0111:				//jc addr
			if (C == 1'b1)
				PC <= ADDRESS;

		4'b1000:		//jnz addr
			if (Z == 1'b0)
				PC <= ADDRESS;


		4'b1001:				//jnc addr
			if (C == 1'b0)
				PC <= ADDRESS;

		4'b1011:				//and
		begin
			A <= A&B;
			if ((A&B) == 8'b0)
				Z <= 1;
			else
				Z <= 0;
		end

		4'b1100:				//or
		begin
			A <= A|B;

			if ((A&B) == 8'b0)
				Z <= 1;
			else
				Z <= 0;
		end

		4'b1101:			//not
		begin
			A <= ~A;
```

```
                        if ((A) == 8'b0)
                                Z <= 1;
                        else
                                Z <= 0;
                end

                4'b1110:                                //add
                begin
                        A <= A + B;

                        if ((A+B) < A)
                                C <= 1;
                        else
                                C <= 0;

                        if ((A+B) == 8'b0)
                                Z <= 1;
                        else
                                Z <= 0;
                end

                4'b1111:                                //sub
                begin
                        A <= A - B;

                        if ((A-B) > A)
                                C <= 1;
                        else
                                C <= 0;

                        if ((A - B) == 8'b0)
                                Z <= 1;
                        else
                                Z <= 0;
                end


        endcase

end


endmodule
```

```cpp
#include <iostream>
#include <algorithm>
#include <string>
#include <sstream>
#include <valarray>
#include <iomanip>
#include <vector>
#include <map>
#include <fstream>
#include <bitset>
#include <set>


using namespace std;

int main(int argc, char **argv)
{
	map<string, bitset<10>> et;

	unsigned lineNumber;
	string line;
	char opcode[20];
	char argument[20];
	fstream i;
	fstream o;
	i.open("in.txt", ios::in);
	o.open("out.txt", ios::out | ios::trunc);


	lineNumber = 0;
	cout << "Faza 1: etykiety -> adresy skoku" << endl;
	while (std::getline(i, line))
	{
		strcpy(opcode, "\0");
		strcpy(argument, "\0");
		sscanf(line.c_str(), "%s %s", &opcode, &argument);

		if (strcmp("mva", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("mvb", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("load", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("store", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("xchg", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("jmp", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("jz", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("jc", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("jnz", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("jnc", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("nop", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("and", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("or", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("not", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("add", opcode) == 0)
		{
			++lineNumber;
		}
		if (strcmp("sub", opcode) == 0)
		{
			++lineNumber;
		}

		switch (opcode[0])
		{
		case '.':
			et[opcode] = lineNumber;
		}
	}


	i.clear();
	i.seekg(0, ios::beg);
	lineNumber = 0;

	cout << "Faza 2: Konwersja mnemonikow" << endl;
	while (std::getline(i, line))
	{
		strcpy(opcode, "\0");
		strcpy(argument, "\0");
		sscanf(line.c_str(), "%s %s", &opcode, &argument);

		std::bitset<8> binarg(atoi(argument));
		std::bitset<10> binaddr(atoi(argument));

		if (strcmp("nop", opcode) == 0)
		{
```

```
        o << "ROM[" << lineNumber << "] = 14'b0000_0000000000";
        o << "; //nop" << endl;
        ++lineNumber;
    }
    if (strcmp("mva", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b1010_00_";
        o << binarg;
        o << "; //mva " << argument << endl;
        ++lineNumber;
    }
    if (strcmp("mvb", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b0001_00_";
        o << binarg;
        o << "; //mvb " << argument << endl;
        ++lineNumber;
    }
    if (strcmp("load", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b0010_";
        o << binaddr;
        o << "; //load " << argument << endl;
        ++lineNumber;
    }
    if (strcmp("store", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b0011_";
        o << binaddr;
        o << "; //store " << argument << endl;
        ++lineNumber;
    }
    if (strcmp("xchg", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b0100_0000000000";
        o << "; //xchg" << endl;
        ++lineNumber;
    }
    if (strcmp("jmp", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b0101_";
        o << et[argument];
        o << "; //jmp " << argument << endl;
        ++lineNumber;
    }
    if (strcmp("jz", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b0110_";
        o << et[argument];
        o << "; //jz " << argument << endl;
        ++lineNumber;
    }
    if (strcmp("jc", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b0111_";
        o << et[argument];
        o << "; //jc " << argument << endl;
        ++lineNumber;
    }
    if (strcmp("jnz", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b1000_";
        o << et[argument];
        o << "; //jnz " << argument << endl;
        ++lineNumber;
    }
    if (strcmp("jnc", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b1001_";
        o << et[argument];
        o << "; //jnc " << argument << endl;
        ++lineNumber;
    }
    if (strcmp("and", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b1011_0000000000";
        o << "; //and" << endl;
        ++lineNumber;
    }
    if (strcmp("or", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b1100_0000000000";
        o << "; //or" << endl;
        ++lineNumber;
    }
    if (strcmp("not", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b1101_0000000000";
        o << "; //not" << endl;
        ++lineNumber;
    }
    if (strcmp("add", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b1110_0000000000";
        o << "; //add" << endl;
        ++lineNumber;
    }
    if (strcmp("sub", opcode) == 0)
    {
        o << "ROM[" << lineNumber << "] = 14'b1111_0000000000";
        o << "; //sub" << endl;
        ++lineNumber;
    }
    switch (opcode[0])
    {
    case '.':
        o << "// " << opcode << endl;
    }
}

o << endl;
cout << lineNumber << " linii" << endl;
i.close();
o.close();

return 0;
}
```