**Міністерство освіти і науки України**

**Національний університет "Львівська політехніка"**

**Кафедра ЕОМ**



**Звіт з лабораторної роботи №5**

**з дисципліни “** **Комп’ютерні системи ”**

**на тему: ” Дослідження програмної моделі RISC CPU ”**

**Виконав: студент .гр. КІ-33**

**Скрипник Д.О.**

**Прийняв: асистент**

**Львів 2020 р.**

**Мета:** Навчитися здійснювати оцінку структури об’єкта (RISC CPU) на існуючій програмній моделі. Навчитись встановлювати структуру інтерфейсів об’єкта.

**ПОРЯДОК ВИКОНАННЯ РОБОТИ**

**Завдання до лабораторної**

1. Дослідити програмну модель RISC CPU;
2. Визначити склад програмної моделі RISC CPU.
3. Визначити призначення блоків у структурі RISC CPU.
4. Визначити зв’язки між структурними блоками RISC CPU (інтерфейси).
5. Визначити структури інтерфейсів між блоками RISC CPU.
6. Визначити окремі потоки у структурі інтерфейсів:

* інформаційні;
* керування.

**Перелік блоків програмної моделі RISC CPU:**

*FETCH\_BLOCK*

*DECODE\_BLOCK*

*EXEC\_BLOCK*

*FLOAT\_BLOCK*

*MMX\_BLOCK*

*BIOS\_BLOCK*

*PAGING\_BLOCK*

*ICACHE\_BLOCK*

*DCACHE\_BLOCK*

*PIC\_BLOCK*

**Призначення блоків у структурі RISC CPU:**

*FETCH\_BLOCK*  – вибирає (розпізнає) команду.

*DECODE\_BLOCK* – вибирає з команди операнди.

*EXEC\_BLOCK*  – блок, в якому команда виконується.

*FLOAT\_BLOCK*  – виконуються операції над числами з плаваючою комою.

*MMX\_BLOCK*  –­ ­виконуються mmx операції.

*BIOS\_BLOCK*  – реалізує bios.

*PAGING\_BLOCK*  – сторінковий блок.

*ICACHE\_BLOCK*  – кеш для інструкцій.

*DCACHE\_BLOCK* – кеш для даних.

*PIC\_BLOCK*  – модуль переривань.

**Вхідні та вихідні інтерфейси для кожного блоку RISC CPU окремо:**







FETCH\_BLOCK

sc\_in<unsigned > ramdata;/ instruction from RAM/ інструкція з оперативної ПАМ'ЯТІ

sc\_in<unsigned > branch\_address; // branch target address/ цільова адреса, що відгалужується

sc\_in<bool> next\_pc; // pc ++

sc\_in<bool> branch\_valid; // branch\_valid

sc\_in<bool> stall\_fetch; // STALL\_FETCH

sc\_in<bool> interrupt; // interrrupt

sc\_in<unsigned> int\_vectno; // interrupt vector number номер вектора переривання

sc\_in<bool> bios\_valid; // BIOS input valid Вхід BIOS, дійсний

sc\_in<bool> icache\_valid; // Icache input valid Вхід Icache, дійсний

sc\_in<bool> pred\_fetch; // branch prediction fetch

sc\_in<unsigned >pred\_branch\_address; // branch target address цільова адреса, що відгалужується

sc\_in<bool> pred\_branch\_valid; // branch prediction fetch

sc\_out<bool> ram\_cs; // RAM chip select

sc\_out<bool> ram\_we; // RAM write enable for SMC

sc\_out<unsigned > address; // address send to RAM адреса відправляють оперативній ПАМ'ЯТІ

sc\_out<unsigned > smc\_instruction; // for self-modifying code для однорідний-змінюючого коду

sc\_out<unsigned> instruction; // instruction send to ID інструкція відправляють ID

sc\_out<bool> instruction\_valid; // inst valid

sc\_out<unsigned > program\_counter; // program counter

sc\_out<bool> interrupt\_ack; // interrupt acknowledge переривання визнають

sc\_out<bool> branch\_clear; // clear outstanding branch

sc\_out<bool> pred\_fetch\_valid; // branch prediction fetch

sc\_out<bool> reset; // reset

sc\_in\_clk CLK;

DECODE\_BLOCK

sc\_in<bool> resetin; // input reset

sc\_in<unsigned> instruction; // fetched instruction неприродна інструкція

sc\_in<unsigned> pred\_instruction; // fetched instruction

sc\_in<bool> instruction\_valid; // input valid вхід, дійсний

sc\_in<bool> pred\_inst\_valid; // input valid

sc\_in<bool> destreg\_write; // register write enable запис регістра вирішують

sc\_in<unsigned> destreg\_write\_src; // which register to write? який регістр написати?

sc\_in<signed> alu\_dataout; // data from ALU

sc\_in<signed> dram\_dataout; // data from Dcache

sc\_in<bool> dram\_rd\_valid; // Dcache read data valid

sc\_in<unsigned> dram\_write\_src; // Dcache data write to which reg

sc\_in<signed> fpu\_dout; // data from FPU

sc\_in<bool> fpu\_valid; // FPU data valid

sc\_in<unsigned> fpu\_destout; // write to which register

sc\_in<bool> clear\_branch; // clear outstanding branch

sc\_in<bool> display\_done; // display to monitor done

sc\_in<unsigned > pc; // program counter from IFU

sc\_in<bool> pred\_on; // branch prediction is on

sc\_out<unsigned > br\_instruction\_address; // branch invoke instruction

sc\_out<bool> next\_pc; // next pc ++ ?

sc\_out<bool> branch\_valid; // branch valid signal

sc\_out<unsigned > branch\_target\_address; // branch target address

sc\_out<bool> mem\_access; // memory access valid

sc\_out<unsigned > mem\_address; // memory physical address

sc\_out<int> alu\_op; // ALU/FPU/MMU Opcode

sc\_out<bool> mem\_write; // memory write enable

sc\_out<unsigned> alu\_src; // destination register number

sc\_out<bool> reg\_write; // not implemented

sc\_out<signed int> src\_A; // operand A

sc\_out<signed int> src\_B; // operand B

sc\_out<bool> forward\_A; // data forwarding to operand A

sc\_out<bool> forward\_B; // data forwarding to operand B

sc\_out<bool> stall\_fetch; // stall fetch due to branch

sc\_out<bool> decode\_valid; // decoder output valid

sc\_out<bool> float\_valid; // enable FPU

sc\_out<bool> mmx\_valid; // enable MMU

sc\_out<bool> pid\_valid; // load process ID

sc\_out<signed> pid\_data; // process ID value

sc\_in\_clk CLK;

EXEC\_BLOCK

sc\_in<bool> reset; // reset not used.

sc\_in<bool> in\_valid; // input valid

sc\_in<int> opcode; // opcode from ID

sc\_in<bool> negate; // not implemented

sc\_in<int> add1; // not implemented

sc\_in<bool> shift\_sel; // not implemented

sc\_in<signed int> dina; // operand A

sc\_in<signed int> dinb; // operand B

sc\_in<bool> forward\_A; // data forwarding A valid

sc\_in<bool> forward\_B; // data forwarding B valid

sc\_in<unsigned> dest; // destination register number

sc\_out<bool> C; // Carry bit

sc\_out<bool> V; // Overflow bit

sc\_out<bool> Z; // Zero bit

sc\_out<signed int> dout; // output data

sc\_out<bool> out\_valid; // output valid

sc\_out<unsigned> destout; // write to which registers?

sc\_in\_clk CLK;

FLOAT\_BLOCK

sc\_in<bool> in\_valid; // input valid bit

sc\_in<int> opcode; // opcode

sc\_in<signed int> floata; // operand A

sc\_in<signed int> floatb; // operand B

sc\_in<unsigned> dest; // write to which register

sc\_out<signed int> fdout; // FPU output

sc\_out<bool> fout\_valid; // output valid

sc\_out<unsigned> fdestout; // write to which register

sc\_in\_clk CLK;

MMX\_BLOCK

sc\_in<bool> mmx\_valid; // MMX unit enable

sc\_in<int> opcode; // opcode

sc\_in<signed int> mmxa; // operand A

sc\_in<signed int> mmxb; // operand B

sc\_in<unsigned> dest; // Destination register number

sc\_out<signed int> mmxdout; // MMX output

sc\_out<bool> mmxout\_valid; // MMX output valid

sc\_out<unsigned> mmxdestout; // destination number

sc\_in\_clk CLK;

BIOS\_BLOCK

sc\_in<unsigned > datain; // modified instruction

sc\_in<bool> cs; // chip select

sc\_in<bool> we; // write enable for SMC

sc\_in<unsigned > addr; // physical address

sc\_out<unsigned > dataout; // ram data out

sc\_out<bool> bios\_valid; // out valid

sc\_out<bool> stall\_fetch; // stall fetch if output not valid

sc\_in\_clk CLK;

PAGING\_BLOCK

sc\_in<unsigned > paging\_din; // input data

sc\_in<bool> paging\_csin; // chip select

sc\_in<bool> paging\_wein; // write enable

sc\_in<unsigned > logical\_address; // logical address

sc\_in<unsigned > icache\_din; // data from BIOS/icache

sc\_in<bool> icache\_validin; // data valid bit

sc\_in<bool> icache\_stall; // stall IFU if busy

sc\_out<unsigned > paging\_dout; // output data

sc\_out<bool> paging\_csout; // output cs to cache/BIOS

sc\_out<bool> paging\_weout; // write enable to cache/BIOS

sc\_out<unsigned > physical\_address; // physical address

sc\_out<unsigned > dataout; // dataout from memory

sc\_out<bool> data\_valid; // data valid

sc\_out<bool> stall\_ifu; // stall IFU if busy

sc\_in\_clk CLK;

ICACHE\_BLOCK

sc\_in<unsigned > datain; // modified instruction

sc\_in<bool> cs; // chip select

sc\_in<bool> we; // write enable for SMC

sc\_in<unsigned > addr; // address

sc\_in<bool> ld\_valid; // load valid

sc\_in<signed> ld\_data; // load data value

sc\_out<unsigned > dataout; // ram data out

sc\_out<bool> icache\_valid; // output valid

sc\_out<bool> stall\_fetch; // stall fetch if busy

sc\_in\_clk CLK;

DCACHE\_BLOCK

sc\_in<signed> datain; // input data

sc\_in<unsigned> statein; // input state bit MESI(=3210)

sc\_in<bool> cs; // chip select

sc\_in<bool> we; // write enable

sc\_in<unsigned > addr; // address

sc\_in<unsigned> dest; // write back to which register

sc\_out<unsigned> destout; // write back to which register

sc\_out<signed> dataout; // dataram data out

sc\_out<bool> out\_valid; // output valid

sc\_out<unsigned> stateout; // state output

sc\_in\_clk CLK;

PIC\_BLOCK

sc\_in<bool> ireq0; // interrupt request 0

sc\_in<bool> ireq1; // interrupt request 1

sc\_in<bool> ireq2; // interrupt request 2

sc\_in<bool> ireq3; // interrupt request 3

sc\_in<bool> cs; // chip select

sc\_in<bool> rd\_wr; // read or write

sc\_in<bool> intack\_cpu; // interrupt acknowledge from CPU

sc\_out<bool> intreq; // interrupt request to CPU

sc\_out<bool> intack; // interrupt acknowledge to devices

sc\_out<unsigned> vectno; // vector number

////////////////////////////////////////////////////////////////////////////

// MAIN PROGRAM

////////////////////////////////////////////////////////////////////////////

sc\_clock clk("Clock", 1, 0.5, 0.0);

printf("// Module : main of CPU Model\n");

printf("// Purpose : This is a simple CPU modeling using SystemC.\n");

printf("// Instruction Set Architecure defined by Martin Wang.\n");

printf("// SystemC (TM) Copyright (c) 1988-2000 by Synopsys, Inc. \n");

fetch IFU("FETCH\_BLOCK");

IFU.init\_param(delay\_cycles);

IFU << ram\_dataout << branch\_target\_address << next\_pc << branch\_valid

<< stall\_fetch << intreq << vectno << bios\_valid << icache\_valid

<< pred\_fetch << pred\_branch\_address << pred\_branch\_valid << ram\_cs << ram\_we

<< addr << ram\_datain << instruction << instruction\_valid << program\_counter

<< intack\_cpu << branch\_clear << pred\_fetch\_valid << reset << clk;

decode IDU("DECODE\_BLOCK");

IDU << reset << instruction << pred\_instruction << instruction\_valid

<< pred\_inst\_valid << out\_valid << destout << dout << dram\_dataout

<< dram\_rd\_valid << destout << fdout << fout\_valid << fdestout

<< branch\_clear << dsp\_data\_valid << program\_counter << pred\_on

<< branch\_instruction\_address << next\_pc << branch\_valid

<< branch\_target\_address << mem\_access << mem\_address << alu\_op

<< mem\_write << alu\_src << reg\_write << src\_A << src\_B << forward\_A

<< forward\_B << stall\_fetch << decode\_valid << float\_valid << mmx\_valid

<< pid\_valid << pid\_data << clk;

exec IEU("EXEC\_BLOCK");

IEU << reset << decode\_valid << alu\_op << negate << add1 << shift\_sel

<< src\_A << src\_B << forward\_A << forward\_B << alu\_src << c << v << z

<< dout << out\_valid << destout << clk;

floating FPU("FLOAT\_BLOCK"); // order dependent

FPU << float\_valid << alu\_op << src\_A << src\_B << alu\_src

<< fdout << fout\_valid << fdestout << clk;

mmxu MMXU("MMX\_BLOCK");

MMXU << mmx\_valid << alu\_op << src\_A << src\_B << alu\_src

<< fdout << fout\_valid << fdestout << clk;

bios BIOS("BIOS\_BLOCK");

BIOS.init\_param(delay\_cycles);

BIOS.datain(ram\_datain); // order independent

BIOS.cs(ram\_cs);

BIOS.we(ram\_we);

BIOS.addr(addr);

BIOS.dataout(ram\_dataout);

BIOS.bios\_valid(bios\_valid);

BIOS.stall\_fetch(stall\_fetch);

BIOS.CLK(clk);

paging PAGING("PAGING\_BLOCK");

PAGING << ram\_datain << ram\_cs << ram\_we << addr << icache\_din

<< icache\_validin << icache\_stall << paging\_dout << paging\_csout

<< paging\_weout << physical\_address << ram\_dataout << icache\_valid

<< stall\_fetch << clk ;

icache ICACHE("ICACHE\_BLOCK");

ICACHE.init\_param(delay\_cycles);

ICACHE << paging\_dout << paging\_csout << paging\_weout

<< physical\_address << pid\_valid << pid\_data << icache\_din << icache\_validin

<< icache\_stall << clk;

dcache DCACHE("DCACHE\_BLOCK");

DCACHE.init\_param(delay\_cycles);

DCACHE << mmic\_datain << mmic\_statein << mmic\_cs << mmic\_we << mmic\_addr

<< mmic\_dest << mmic\_destout << mmic\_dataout << mmic\_out\_valid << mmic\_stateout << clk;

pic APIC("PIC\_BLOCK");

APIC << ireq0 << ireq1 << ireq2 << ireq3 <<intack\_cpu << rd\_wr

<< intack\_cpu << intreq << intack << vectno;

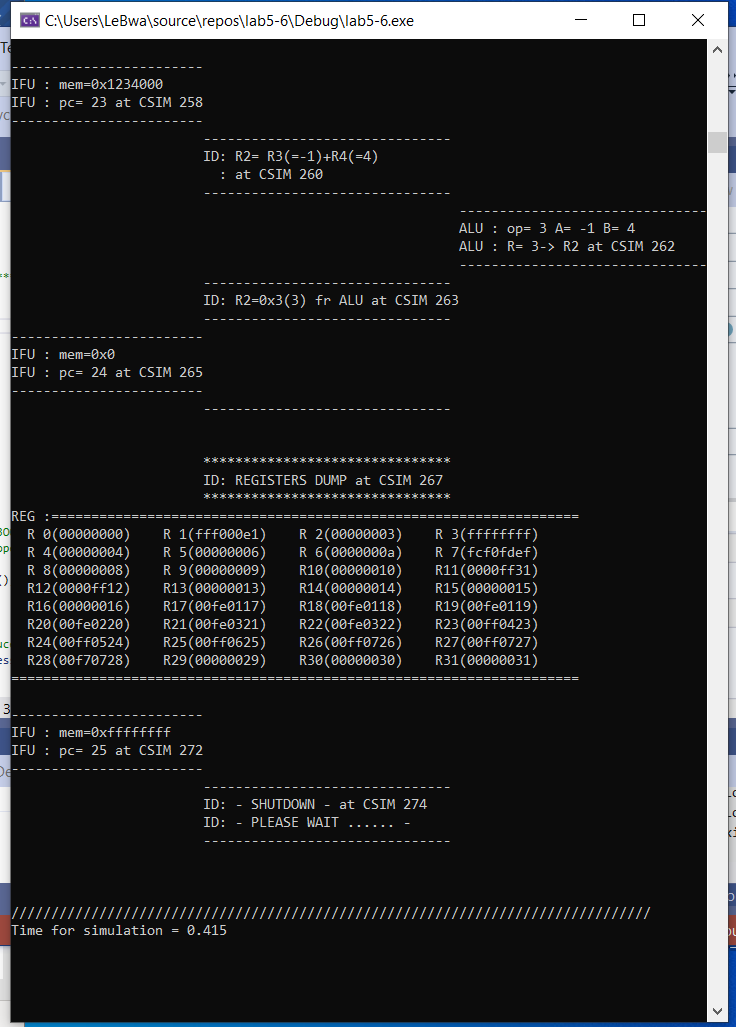
time\_t tbuffer = time(NULL);

sc\_start(clk, -1);

cout << "Time for simulation = " << (time(NULL) - tbuffer) << endl;

**Блок-схема алгоритму:**





*Приклад симуляції*

**Висновок:** на цій лабораторній роботі я навчився здійснювати оцінку структури об’єкту (RISC CPU) на існуючій програмній моделі, а також навчився встановлювати структуру інтерфейсів об’єкту.