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: ст. гр. КІ-32

Кононенко Р.Ю.

Прийняв:

Козак Н.Б.

Львів – 2020

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

**Завдання:**

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

**Хід роботи:**

1. Дослідив програмну модель RISC CPU, її склад та призначення програмних блоків:

• FETCH\_BLOCK – блок, що вибирає команду(операцію);

• DECODE\_BLOCK – блок, що вибирає операнди з КОП і декодує його для подальшого пересилання у один з наступних двох блоків;

• EXEC\_BLOCK – блок виконання команди(арифметичні операції);

• FLOAT\_BLOCK – блок, що виконує операції над числами з плаваючою комою;

• MMX\_BLOCK – виконує MMX-операції(арифметико-логічні: додавання із зсувом і т.д.);

• BIOS\_BLOCK – блок, який реалізує інтерфейс BIOS(початкова ініціалізація машини після увімкнення);

• PAGING\_BLOCK – сторінковий блок;

• ICACHE\_BLOCK – блок, що містить кеш для інструкцій;

• DCACHE\_BLOCK – блок, що містить кеш для даних;

• PIC\_BLOCK – блок, що містить модуль переривань.

1. Визначив зв’язки між структурними блоками(інтерфейсами) RISC CPU:

• FETCH\_BLOCK – залежить від BIOS\_BLOCK, оскільки там зберігаються коди команд(операцій);

• DECODE\_BLOCK – залежить від FETCH\_BLOCK, оскільки очікує на вході код команди для її подальшого декодування і вибірки операндів;

• FLOAT\_BLOCK – залежить від DECODE\_BLOCK, оскільки очікує на вході код команди для виконання відповідної арифметичної операції над операндами з рухомою комою;

• EXEC\_BLOCK – залежить від DECODE\_BLOCK, оскільки очікує на вході код команди для виконання відповідної арифметичної операції над операндами;

• MMX\_BLOCK – залежить від DECODE\_BLOCK, оскільки очікує на вході код команди для виконання відповідної арифметико-логічної операції над операндами;

• BIOS\_BLOCK – не залежить від жодного з блоків, оскільки містить наперед задану інформацію, за допомогою якої відбувається початкова ініціалізація і вибірка команди для FETCH\_BLOCK;

• PAGING\_BLOCK – залежить від BIOS\_BLOCK, оскільки розділяє наявну пам’ять на частини рівного розміру(сторінки);

• ICACHE\_BLOCK – у випадку використання залежить від FETCH\_BLOCK, оскільки містить кеш(буферну пам’ять) для інструкцій, що призначений для правильної роботи BRANCH\_PREDICTION(передбачення переходу при циклічному виконанні);

• DCACHE\_BLOCK – у випадку використання залежить від EXEC\_BLOCK або MMX\_BLOCK, оскільки містить кеш(буферну пам’ять) для даних, що призначений для правильної роботи BRANCH\_PREDICTION(передбачення переходу при циклічному виконанні);

• PIC\_BLOCK – залежить від FETCH\_BLOCK, оскільки перевіряє кожну вибрану команду і, у випадку, якщо таку знайдено, реалізує переривання(завершення) роботи.

1. Визначив структуру інтерфейсів між блоками, що відповідає наведеному опису у п.2, а також виділив окремі потоки структури інтерфейсів(інформаційні/керування) у вигляді коментарів:

FETCH\_BLOCK

sc\_in<unsigned > ramdata; // instruction from RAM

sc\_in<unsigned > branch\_address; // branch target address

sc\_in<bool> interrupt; // interrrupt

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

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

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

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

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

DECODE\_BLOCK

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

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

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

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

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

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

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

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

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

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

EXEC\_BLOCK

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

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

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

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

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<unsigned> destout; // write to which registers?

sc\_in\_clk CLK;

FLOAT\_BLOCK

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

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

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

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

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

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

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

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

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

ICACHE\_BLOCK

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<bool> cs; // chip select

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

sc\_in<unsigned > addr; // address

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

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

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

1. Розробив блок-схему досліджуваної моделі RISC CPU:

Fetch

Icache

Decode

Dcache

Integer Execution

Floating Point Execution

MMX Execution

Software Side

*Рис.1. Блок-схема RISC-CPU*

**Висновок:**

На даній лабораторній роботі навчився здійснювати оцінку структури об’єкта (RISC CPU) на існуючій програмній моделі та встановлювати структуру інтерфейсів об’єкта.