**Міністерство освіти І науки України**

**національний університет “Львівська політехніка”**

****

**Кафедра ЕОМ**

лабораторна робота № 5

“ Дослідження програмної моделі RISC CPU ”

з дисципліни

**"** **Комп’ютерні системи "**

Студента групи КІ-32

Кузишина Назара

Прийняв: Козак Н. Б.

**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:**

Дослідивши програмну модель я визначив:

I) Призначення програмних блоків у структурі RISC CPU;

II) Їхні зв’язки;

III) Структуру інтерфейсів зв’язків, а також її окремі потоки.

I)

1. FETCH\_BLOCK – блок вибору операції;
2. DECODE\_BLOCK – блок декодує операцію і передає операнди;
3. EXEC\_BLOCK – блок виконання арифметичних операцій;
4. FLOAT\_BLOCK – блок виконання операцій з рухомою комою;
5. MMX\_BLOCK – блок виконує операції типу MMX;
6. BIOS\_BLOCK – блок ініціалізації машини після увімкнення;
7. PAGING\_BLOCK – блок сторінок;
8. ICACHE\_BLOCK – блок для зберігання інструкцій в своєму кеші;
9. DCACHE\_BLOCK – блок для зберігання даних в своєму кеші;
10. PIC\_BLOCK – блок модуль переривань.

II)

1. FETCH\_BLOCK залежить від BIOS\_BLOCK
2. DECODE\_BLOCK залежить від FETCH\_BLOCK
3. FLOAT\_BLOCK залежить від DECODE\_BLOCK
4. EXEC\_BLOCK залежить від DECODE\_BLOCK
5. MMX\_BLOCK залежить від DECODE\_BLOCK
6. BIOS\_BLOCK не залежить від жодного з блоків
7. ICACHE\_BLOCK залежить від FETCH\_BLOCK
8. DCACHE\_BLOCK залежить від EXEC\_BLOCK
9. PIC\_BLOCK – залежить від FETCH\_BLOCK

III)

**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