МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра ЕОМ

Лабораторна робота №5

з дисципліни “Комп’ютерні системи”

На тему:

«Дослідження програмної моделі RISC CPU»

Виконав:

ст. гр. КІ-32

Сідлярський А.І.

Перевірив:

Козак Н.Б.

Львів

2020

Мета: 1. Навчитися здійснювати оцінку структури об'єкта (RISC CPU) на існуючій програмній моделі.

2. Навчитись встановлювати структуру інтерфейсів об'єкта.

Варіант: 15

Завдання до лабораторної роботи:

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.

Виконання завдання:

Перелік блоків та їхфункціональне призначення:

FETCH_BLOCK – вибірка команди.
DECODE_BLOCK – декодування команди.
EXEC_BLOCK – блок виконання.
FLOAT_BLOCK – блок для виконання чисел з рухомою 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.

Хедери кожного блоку, де описані вхідні та вихідні порти:

- ***FETCH_BLOCK***

```
sc_in<unsigned > ramdata;           // instruction from RAM
sc_in<unsigned > branch_address; // branch target address
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
sc_in<bool>      bios_valid;        // BIOS input valid
sc_in<bool>      icache_valid;      // Icache input valid
sc_in<bool>      pred_fetch;        // branch prediction fetch
sc_in<unsigned > pred_branch_address;     // branch target address
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
sc_out<unsigned > smc_instruction;        // for self-modifying code
sc_out<unsigned>  instruction;            // instruction send to ID
sc_out<bool>     instruction_valid;// inst valid
sc_out<unsigned > program_counter;        // program counter
sc_out<bool>     interrupt_ack;           // interrupt acknowledge
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
sc_in<unsigned> pred_instruction;             // fetched instruction
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<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
```

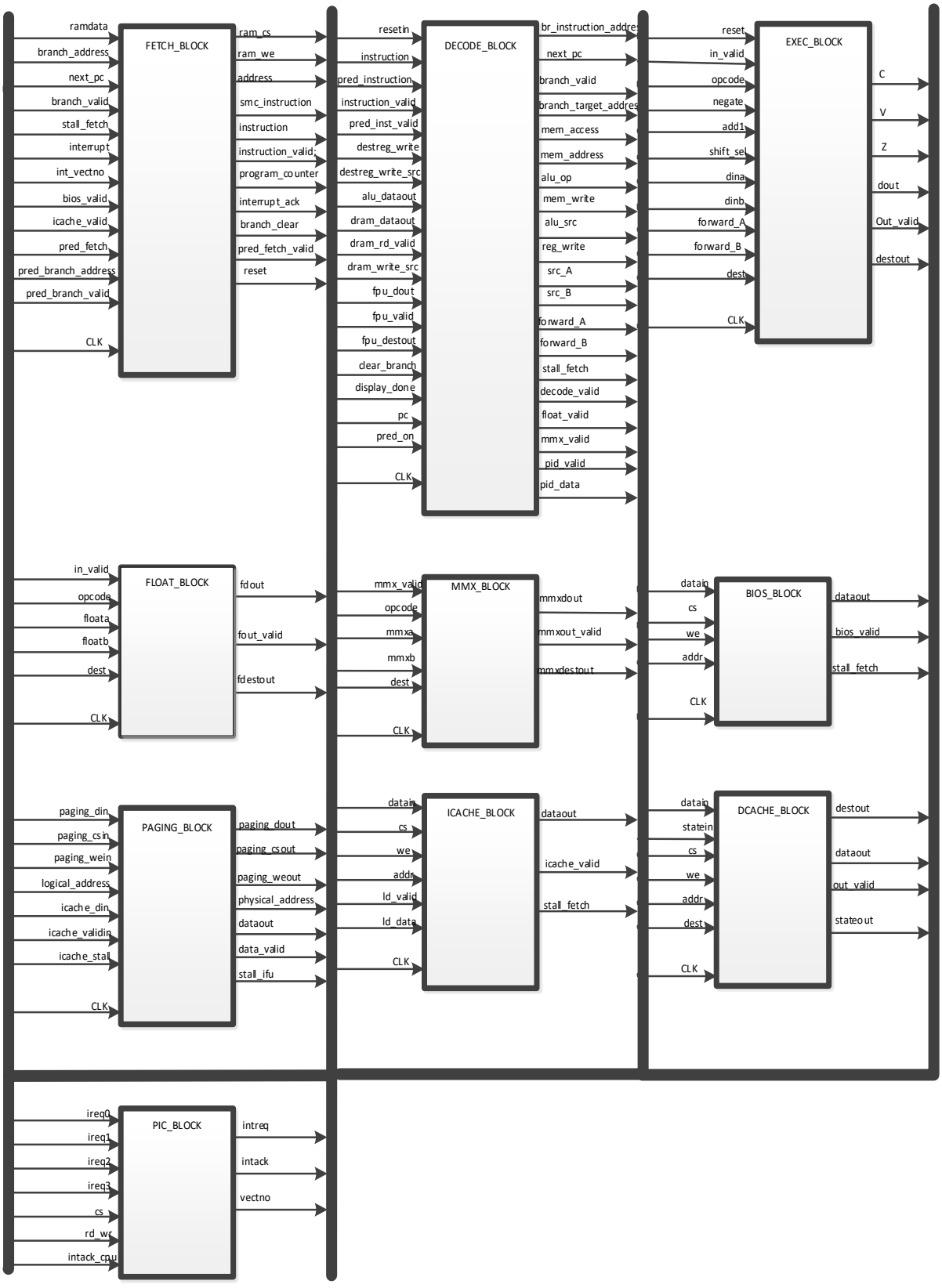

Рис.1 Структурна схема модифікованого конвеєра

Висновок: в ході виконання лабораторної роботи я навчився здійснювати оцінку структури об’єкта (RISC CPU) на існуючій програмній моделі, а також навчився встановлювати структуру інтерфейсів об’єкта.