Дизајн микропроцесора риск архитектуре наставак

Модел контролне јединице

```
254 always @(rst or clk)
255    if(rst==1)
256    begin
257       pc_ena <= `ENABLE;
258       pc_sel <= `PC_SEL_0;
259       step <= `FETCH;
260    end
261    else
```

Процедура која се извршава на сваку промену ресет и такт сигнала. У случају активирања ресет сигнала *PC* регистар се поставља на вредност нула а корак постаља на почетни корак.

```
262    /*Konacni automat (state machine)*/
263    if(clk == 1'b1)
264    case(step)
265       `FETCH:
266          begin
267             a_sel <= `BUS_A_SEL_PC;
268             alu_ena <= `ENABLE;
269             alu_op <= `ALU_INC;
270             mar_ena <= `ENABLE;
271             mdr_ena <= `ENABLE;
272             mdr_sel <= `MDR_SEL_DATA;
273             mem_op <= `MEM_READ;
274             step <= `DECODE;
275          end
```

Од 262 лине кода почиње имплементација коначног аутомата. У првом кораку се врши доношење инструкције из меморије пребацивањем вредности регистра *PC* у *MAR*. У овом кораку се врши и инкрементирање *PC* и на тај начин овај регистра садржи вредност меморијске локације на којој се налази следећа инструкција.

Модел контролне јединице

```
276         `DECODE:
277           begin
278              a_sel <= `BUS_A_SEL_MDR;
279              b_sel <= `BUS_B_SEL_ALU_RES;
280              ir_ena <= `ENABLE;
281              pc_ena <= `ENABLE;
282              pc_sel <= `PC_SEL_BUS_B;
283              step <= `EXECUTE;
284           end
```

У другом кораку текућа инструкција, која је у међувремену донешена из меморије у *MDR* регистар, пребацује у *IR* регистар.

У овом кораку се и инкрементирана вредност *PC* регисата уписује у *PC* регистра.

Модел контролне јединице

```
285         `EXECUTE:
286           case(opcode)
287             `ADD, `SUB, `AND, `OR:
288               begin
289                 a_sel <= `BUS_A_SEL_REGFILE_OUT0;
290                 b_sel <= `BUS_B_SEL_REGFILE_OUT1;
291                 regfile_out0_sel <= src0;
292                 regfile_out1_sel <= src1;
293                 regfile_in_sel <= dst;
294                 psw_ena <= `ENABLE;
295                 alu_ena <= `ENABLE;
296                 alu_op <= opcode[2:0];
297                 step <= `WRITE;
298               end
```

У трећем кораку се генеришу контролни сигнали у зависности који је инструкција у питању *(opcode)*.

У случају да су у питању бинарне аритметичко-логичке операције, извршава се део кода приказан на слици.

Модел контролне јединице

У случају унарних аритметичко-логичких операција, операнд се из скупа регистара опште намене пропушта на магистралу A, а остали контролни сигнали се генеришу исто као код бинарних операција.

```
299             `INC, `NOT, `SLA, `SRA:
300               begin
301                   a_sel <= `BUS_A_SEL_REGFILE_OUT0;
302                   regfile_out0_sel <= src0;
303                   regfile_in_sel <= dst;
304                   psw_ena <= `ENABLE;
305                   alu_ena <= `ENABLE;
306                   alu_op <= opcode[2:0];
307                   step <= `WRITE;
308               end
```

Модел контролне јединице

У случају инструкције за учитавање из меморије, вредност регистра опште који садржи адресу са које се врши учитавање се преко магистрале A пребацује у *MAR* регистар.

Дозвољава се упис у *MDR* registar, а као извор се бира порт преко кога се врши трансфер података са меморије. Затим се одмах садржај овог регистра препушта на магистралу B, тако да ће подаци бити доступни чим се учитају из меморије.

```
309             `LD:
310                begin
311                    a_sel <= `BUS_A_SEL_REGFILE_OUT0;
312                    b_sel <= `BUS_B_SEL_MDR;
313                    regfile_out0_sel <= src0;
314                    regfile_in_sel <= dst;
315                    mar_ena <= `ENABLE;
316                    mdr_ena <= `ENABLE;
317                    mdr_sel <= `MDR_SEL_DATA;
318                    mem_op <= `MEM_READ;
319                    step <= `WRITE;
320                end
```

Модел контролне јединице

У случају инструкције којом се врши упис у меморију, адреса меморије и вредност података која се уписује се пребацују у *MAR* и *MDR* регистре. За меморију се селектује операција којом се врши упис у меморију.

```
321             `ST:
322                begin
323                   a_sel <= `BUS_A_SEL_REGFILE_OUT0;
324                   b_sel <= `BUS_B_SEL_REGFILE_OUT1;
325                   regfile_out0_sel <= dst;
326                   regfile_out1_sel <=src0;
327                   mar_ena <= `ENABLE;
328                   mdr_ena <= `ENABLE;
329                   mdr_sel <= `MDR_SEL_BUS_B;
330                   step <= `WRITE;
331                end
```

Модел контролне јединице

Код инструкције за учитавање константе на магистрале B пропушта константа и селектује се регистар у кога ће бити уписана константа.

```
332             `LDI:
333                begin
334                   b_sel <= `BUS_B_SEL_IMM;
335                   regfile_in_sel <= dst;
336                   step <= `WRITE;
337                end
```

Модел контролне јединице

```
338             `JMP, `JZ, `JB:
339                if((opcode == `JZ && !zero) ||
340                (opcode == `JB && !(!zero && carry)))
341                   step <= `FETCH;
342                else
343                   begin
344                      a_sel <= `BUS_A_SEL_PC;
345                      b_sel <= `BUS_B_SEL_IMM;
346                      alu_ena <= `ENABLE;
347                      alu_op <= `ALU_ADD;
348                      step <= `WRITE;
349                   end
350             default:
351                step <= `FETCH;
352          endcase
```

У случају инструкције условног скока ако услов није испуњен одмах се враћа на први корак и обраду наредне инструкције.

У супротном ако је услов испуњен тада се на једну магистралу пропушта *PC* регистар, а на другу вредност константе и активира се *ALU* јединица и селектује се операција сабирања.

Овде се завршава структура *case* трећег корака.

Модел контролне јединице

У четвртом кораку се у зависности од операције инструкција се генеришу контролни и активациони сигнали.

```
353         `WRITE:
354           case(opcode)
355             `ADD, `INC, `SUB, `NOT, `AND, `OR, `SLA, `SRA:
356               begin
357                 b_sel <= `BUS_B_SEL_ALU_RES;
358                 regfile_in_ena <= `ENABLE;
359                 step <= `FETCH;
360               end
```

Када је у питању аритметичко-логичка инструкција тада се излаз из *ALU* јединица уписује у регистар опште намене.

```
361             `LD:
362               begin
363                 regfile_in_ena <= `ENABLE;
364                 step <= `FETCH;
365               end
```

Када се учитава податак из мемеорије онда се садржај *MDR* регистра уписује у одговарајући регистар опште намене.

Модел контролне јединице

```
366             `ST:
367               begin
368                 mem_op <= `MEM_WRITE;
369                 step <= `FETCH;
370               end
```

Када је у питању упис у меморију тад се активира операција уписа у меморију.

```
332             `LDI:
333               begin
334                 b_sel <= `BUS_B_SEL_IMM;
335                 regfile_in_sel <= dst;
336                 step <= `WRITE;
337               end
```

У случају инструкције уписа константе у регистар, активира се упис у скуп регистара опште намене.

Модел контролне јединице

```
376             `JMP, `JZ, `JB:
377                begin
378                   b_sel <= `BUS_B_SEL_ALU_RES;
379                   pc_ena <= `ENABLE;
380                   pc_sel <= `PC_SEL_BUS_B;
381                   step <= `FETCH;
382                end
383             default:
384                step <= `FETCH;
385          endcase
386          default:
387          step <= `FETCH;
388    endcase
```

У случају инстукција скока, тада се врдност излаза *ALU* јединице уписује у *PC* регистар.

Овде се завршава структура *case* четвртог корака као и структура *case* коначног аутомата.

Модел контролне јединице

else је од условне доделе на линији кода 263 када се испитује да ли је такт позитиван, у случају да услов није испуњен скаче се на део кода приказан на слици испод и врши се деактивирање сигнала који дозвољавају успис у регисте опште и посебне намене, као и деактивирање *ALU* јединице.

```
389     else
390     begin
391         regfile_in_ena <= `DISABLE;
392         alu_ena <= `DISABLE;
393         pc_ena <= `DISABLE;
394         ir_ena <= `DISABLE;
395         psw_ena <= `DISABLE;
396         mar_ena <= `DISABLE;
397         mdr_ena <= `DISABLE;
398     end
399 endmodule
```

Модел комплетног микропроцесора

Модел комплетног микропроцесора служи за увезивање *datapath* и контролне јединице. Његове улазни подаци се прослђују обема једницама, а излазни сигнали су за комуникацију са меморијом.

Листа улазних и излазних сигнала и њихова декларација.

```
413  module risc16(
414  input clk,      // Signal takta.
415  input rst,      // Signal reseta.
416  output mem_op, // Signal odredjivanja pristupa memoriji.
417  output [15:0] address, // Memorijska adresa.
418  inout [15:0] data);    // Podatak koji se razmenjuje sa memorijom.
```

Модел комплетног микропроцесора

Декларација носиоца података који се користе у моделу комплетног микропроцесора.

```
420  wire [1:0] a_sel, b_sel; //Odre]ivanje sta se propusta na magistrale.
421  wire [3:0] regfile_out0_sel, regfile_out1_sel; //Selektorski signali za izlazni skup registara
422  wire [3:0] regfile_in_sel; // Selektrorski signal za upis u skup registara.
423  wire regfile_in_ena; // Kontrolni signal za upis u skup registara.
424  wire alu_ena; // Kontrolni signal ALU jedinice.
425  wire [2:0] alu_op; // Kod operacije za ALU jedinicu.
426  wire pc_ena, pc_sel, ir_ena, psw_ena; //Kontrolni i selektroski signali za PC,IR,PSW.
427  wire mar_ena, mdr_ena, mdr_sel; //Kontrolni i selektorski signali za MDR, MAR.
428  wire [3:0] opcode, dst, src0, src1; //Komponente tekuce instrukcije.
429  wire zero, carry; // Vrednosti Z i C flegova PSW registra.
```

Модел комплетног микропроцесора

Инстанцирање *datapath* компоненте и повезивање портова на основу њихових имена.

```
431 /* Instanciranje datapath jedinice */
432 datapath _datapath (.clk(clk), .rst(rst), .mem_op(mem_op), .a_sel(a_sel), .b_sel(b_sel),
433 .regfile_out0_sel(regfile_out0_sel), .regfile_out1_sel(regfile_out1_sel),
434 .regfile_in_ena(regfile_in_ena), .regfile_in_sel(regfile_in_sel), .alu_ena(alu_ena),
435 .alu_op(alu_op), .pc_ena(pc_ena), .pc_sel(pc_sel), .ir_ena(ir_ena), .psw_ena(psw_ena),
436 .mar_ena(mar_ena), .mdr_ena(mdr_ena), .mdr_sel(mdr_sel), .opcode(opcode), .dst(dst),
437 .src0(src0), .src1(src1), .zero(zero), .carry(carry), .addr(address), .data(data));
```

Инстанцирање контролне јединице и повезивање портова преко уређене листе.

```
439 /* Instanciranje kontrolne jedinice */
440 control _control (clk, rst, opcode, dst, src0, src1, zero, carry, mem_op, a_sel, b_sel,
441 regfile_out0_sel, regfile_out1_sel, regfile_in_ena, regfile_in_sel, alu_ena, alu_op,
442 pc_ena, pc_sel, ir_ena, psw_ena, mar_ena, mdr_ena, mdr_sel);
443 endmodule
```

Симулација дизајна

За симулацију модела користи се *ISE Simulator* компаније Xilinx који је сатавни део *ISE Design Suite* програма коришћеног за опис овог микропроцесора. Код симулације приказан је на сликама испод.

```
25  `define DUMPFILE "program.dump"
```

Макро за дефинисање фајла из кога ће се очитавати меморија. Овај начин имплементације меморије је јако користан јер се има могућност брзе и лаке измене инструкција, а и не мора се дезајнирати меморија.

```
29      // Ulazi signal.
30      reg clk;
31      reg rst;
32
33      // Izlazi signal.
34      wire mem_op;
35      wire [15:0] address;
36
37      // Bidirekcioni signal.
38      wire [15:0] data;
```

Дефинисање улазних и излазних сигнала.

Симулација дизајна

```
40      //Memorija.
41      reg [15:0] mem[0:65535];
```

Регистар који представља меморију и у који ће бити учитане вредности из *dump* фајла. Меморија је имплементирана као поље од 2^{16} речи дужине 16 битова.

```
43      // Instanciranje test komponente.
44      risc16 uut (
45          .clk(clk),
46          .rst(rst),
47          .mem_op(mem_op),
48          .address(address),
49          .data(data)
50      );
```

Инстанцирање тест компоненте, односно микропроцесора дизајн у овом примеру.

Симулација дизајна

```
52    // Inicijalizacija ulaza.
53    initial
54    begin
55        rst <= 1'b0;
56        clk <= 1'b0;
57    end
```

Иницијализација улазних сигнала.

```
59    //Generisanje takt signala.
60    always
61        #15 clk <= ~clk;
```

Процедура *always* за генерисање такта, променом *clk* сигнала на сваких 20 временских јединица.

```
63    //Resetovanje sistema.
64    initial
65    begin
66        rst <= 1'b1;
67        #20;
68        rst <= 1'b0;
69    end
```

На почетку симулације извршава се ресетовање система, и постављање сигнала на иницијалне вредности.

Симулација дизајна

```
71    //Ucitavanje sadrzaja memorije iz fajla.
72    initial
73    $readmemh (`DUMPFILE, mem);
```

За учитавање података из *dump* фајла користи се системска директива *readmemh* коју подржава *Isim.* Сваки ред фајла одговара једној меморијској речи и очекује се да буде дат у хексадецималном формату.

```
75    //Ucitavanje podataka iz memorije
76    assign data = mem_op ? 16'bz : mem[address];
```

Учитавање подата из меморије преко *data* порта микропроцесора.

Симулација дизајна

За упис података у меморију преко *data* порта микропроцесора којисти се *always* процедура.

```
78      //Upis podataka u memoriju.
79      always @(mem_op or address or data)
80          if(mem_op)
81              begin
82                  mem[address] <= data;
83                  $display ($time,": mem[%x] <- %x", address, data);
84              end
85  endmodule
```

Системска директива *display* користи за брз приказ резултата.

Симулација дизајна

Садржај *dump* фајла за симулацију сабирања два броја (2+2) је приказан на слици испод.

```
@000
b003

@002
0002
0002
a002
8100
a003
8200
0312
a001
9030
b0f8
```

Празни редови се игноришум, а специфицирање меморијске адресе се врши знаком @ испред адресе. У овом фајлу је задато да се на адреси 000 налази вредност b003 затим се прескаче адреса 001 на коју ће се смештати резултат и специфицира адреса меморије 002. Од ове адресе па до 000b се налазе вредности дате у последњих десет редова. Меморијске локације које нуси иницијализоване имају непознату вредност у свим својим битовима.

```
LDI R0, 2
LD R1, (R0)
LDI R0, 3
LD R2, (R0)
ADD R3, R1, R2
LDI R0, 1
ST (R0), R3
JMP -8
```

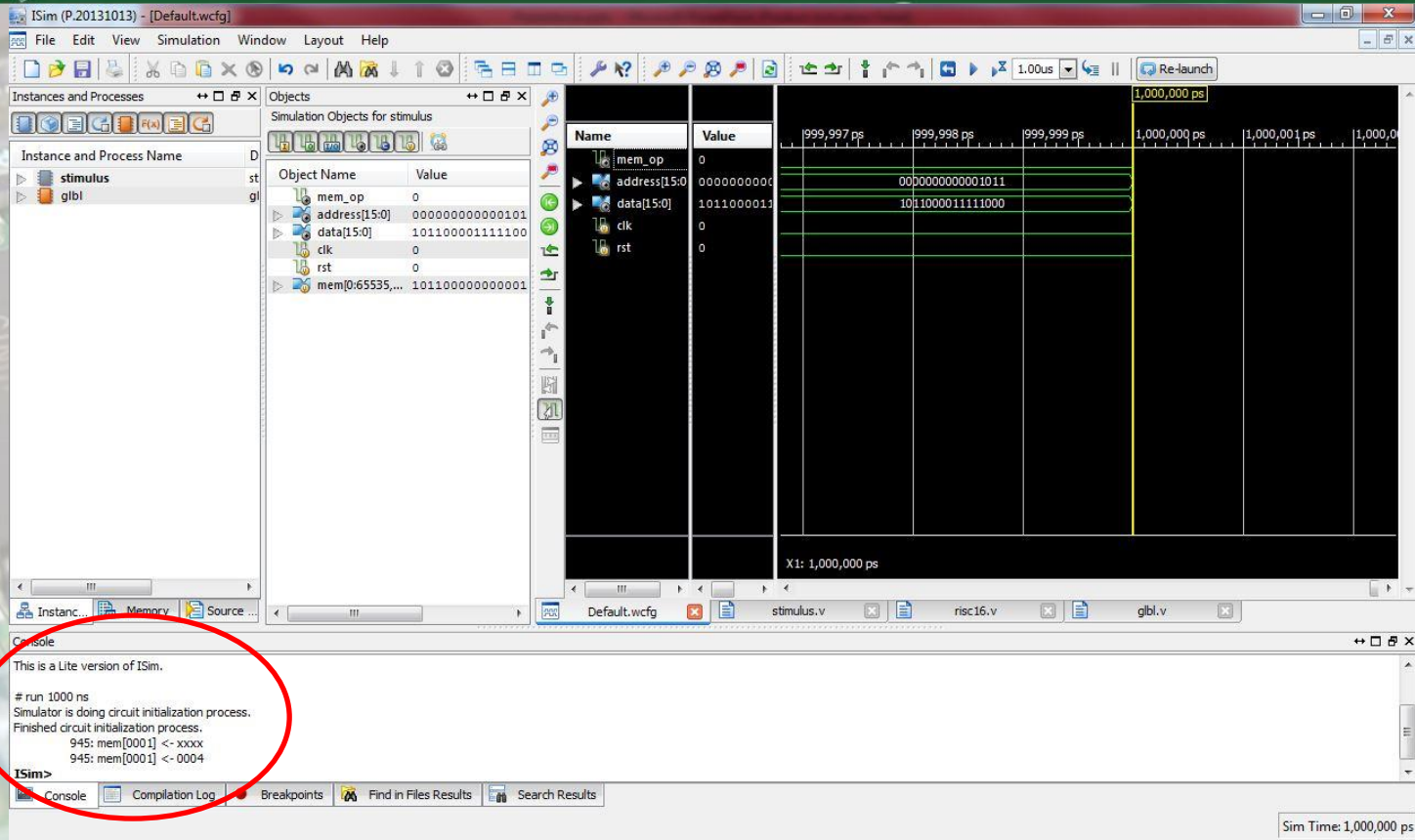
Приказ горе наведеног код у машинском језику у неком хипотетичком асемблеру.

Симулација дизајна

Приказ екрана у *Isim* симулатору.

```
This is a Lite version of ISim.

# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
            945: mem[0001] <- xxxx
            945: mem[0001] <- 0004
```



Закључак

Даље могуће надоградње овог система су многобројне:

- Креирање додатних *dump* фајлова са новим програмима.
- Његово унапређење новим инструкцијама. За почетак је потребна инструкција за заустављање процесора. Због непостојања ове инструкције програм ће се стално вртети у петљи.
- Конструисање асемблера како би се избегло коришћење машинског језика, а затим и писање преводилаца за више програмске језике.
- Дизајнирање осталих компоненти система (системска магистрала, меморија итд.).