



Дизајн микропроцесора риск арихтектура

Архитектура микропроцесора

- 16 регистара опште намене (16-битни)
- 16 адресних линија за комуникацију са меморијом
- 16 линија за податке за читање/упис података из/у меморију
- 14 инструкција



Скуп инструкција

Инструкција	Операција	Сематика операције
ADD d,s0,s1	сабирање	$d \leftarrow s0 + s1$
INC d,s0	инкрементирање	$d \leftarrow s0++$
SUB d,s0,s1	одузимање	$d \leftarrow s0 - s1$
NOT d,s0	битска негација	$d \leftarrow \sim s0$
AND d,s0,s1	битска коњукција	$d \leftarrow s0 \& s1$
OR d,s0,s1	битска дисјункција	$d \leftarrow s0 s1$
SLA d,s0	померање улево за 1	$d \leftarrow s0 \ll 1$
SRA d,s0	померање удесно за 1	$d \leftarrow s0 \gg 1$
LD d,s0	учитавање из меморије	$d \leftarrow (s0)$

Скуп инструкција

ST d,s0	упис у меморију	$(d) \leftarrow s0$
LDI d,imm	учитавање броја	$d \leftarrow s0 + s1$
JMP imm	безусловни скок	goto PC+imm
JZ imm	условни скок (ако је једнако)	If(Z) goto PC+imm
JB imm	условни скок (ако је мање)	If(!Z && C) goto PC+imm

- d, s0, s1 су неки од регистара опште намене,
- знак $\leftarrow$ представља упис у неки од регистра или у меморију,
- заграде () представљају да регистар садржи адресу неке меморијске локације,
- imm је 8-битни означен број који је задат у инструкцији,
- PC (program counter) регистар у коме се чува адреса текуће инструкције,
- Z i C su zero i carry flegovi.

Организација микропроцесора

Основне компоненте микропроцесора су:

- аритметичко-логичка јединица
- контролна јединица
- скуп регистара
- меморијска јединица
- улазно-излазна јединица



Опис модела разматраног система дат је у Verilog HDL језику.

Модел аритметичко-логичке јединице

```
26 `define ALU_ADD 3'b000
27 `define ALU_INC 3'b001
28 `define ALU_SUB 3'b010
29 `define ALU_NOT 3'b011
30 `define ALU_AND 3'b100
31 `define ALU_OR 3'b101
32 `define ALU_SLA 3'b110
33 `define ALU_SRA 3'b111
```

Макрои убрзавају писање *HDL* описа и повећавају његову прегледност.

Дефинисање макро коришћених у *ALU* модулу.

Модул је основни градивни елемент који чини основу пројектовања у *Verilog* језику, а портови представљају спремне тачке преко којих модул међусобно комуницирају.

Дефинисање *ALU* модула и његове листе портова.

```
85 ///////////////////////////////////////////////////////////////////
86 // MODEL ARITMETICKO-LOGICKE JEDINICE //
87 ///////////////////////////////////////////////////////////////////
88 module alu (
89     input ena,                // Aktiviranje komponente.
90     input [15:0] in0,         // Prvi ulazni operand.
91     input [15:0] in1,         // Drugi ulazni operand.
92     input [2:0] op,           // Kod operacije.
93     output reg [15:0] out,     // Izlazni port(rezultat operacije).
94     output zero,              // Zero fleg.
95     output reg carry);        // Carry fleg.
```

Модел аритметичко-логичке јединице

```
97 always @(ena or in0 or in1 or op)
98     if(ena == 1)
99         case(op)
100             `ALU_ADD: {carry, out} <= in0 + in1;
101             `ALU_INC: {carry, out} <= in0 + 16'd1;
102             `ALU_SUB: {carry, out} <= in0 - in1;
103             `ALU_NOT: {carry, out} <= {1'b0, ~in0};
104             `ALU_AND: {carry, out} <= {1'b0, in0 & in1};
105             `ALU_OR: {carry, out} <= {1'b0, in0 | in1};
106             `ALU_SLA: {carry, out} <= {in0, 1'b0};
107             `ALU_SRA: {out, carry} <= {in0[15], 1'b0};
108         endcase
109         assign zero = {out == 16'd0};|
110     endmodule
```

Главни део кода *ALU* модула где се користи процедура типа *always*, условна додела *if*, структура *case*, континуална додела *assign*. Знак *<=* представља неблокирајуће доделе.

Always процедура се извршавати кад гок дође до промене неког од *ena*, *in0*, *in1* или *op* сигнала.

Модул скупа регистара опште намене

```
////////////////////////////////////  
//////// MODEL SKUPA REGISTARA OPSTE NAMENE //////////  
////////////////////////////////////  
module regfile(  
input [3:0] sel_out0, // Signal selekcije registra cija ce vrednost biti predstavljena na izlazu out0.  
input [3:0] sel_out1, // Signal selekcije registra cija ce vrednost biti predstavljena na izlazu out1.  
input ena_in, // Kontrolni registar za upis u registar.  
input [3:0] sel_in, // Signal selekcije registra u koji ce biti upisana vrednost sa ulaza in.  
input [15:0] in, // Vrednost koja se upisuje u selektovani registar.  
output [15:0] out0, // Vrednost selektovanog registra sa portom sel_out0.  
output [15:0] out1); // Vrednost selektovanog registra sa portom sel_out1.  
  
reg [15:0] regs[15:0]; // Skup registra opste namene mikroporocesora.  
|  
assign out0 = regs[sel_out0];  
assign out1 = regs[sel_out1];  
  
always @(ena_in or sel_in or in)  
    if(ena_in)  
        regs[sel_in] <= in;  
endmodule
```

Модел *datapath* компоненте

- Ова компонента врши пренос података унутар микропроцесора.
- Обједињује имплементацију магистрале, наменских регистара и рудиментарне меморије.
- Магистрала представља два супа од по 16 жица.
- Наменски регистри који се налазе у овој компоненти су:
 - MAR(memory address register)
 - MDR(memory data register)
 - PC(program counter)
 - IR(instruction register)
 - PSW(Program Status Word)

Модел *datapath* компоненте

Листа улазних портова и њихова декларација.

```
148 module datapath(  
149     input clk,                // Signal takta.  
150     input rst,                // Signal reseta.  
151     input mem_op,             // Odredjuje pristup memoriji (upus ili citanje).  
152     input [1:0] a_sel,        // Odredjuje sta se pusta na magistralu a.  
153     input [1:0] b_sel,        // Odredjuje sta se pusta na magistralu a.  
154     input [3:0] regfile_out0_sel, // Selektovanje izlaznog out0 registra iz skupa registara.  
155     input [3:0] regfile_out1_sel, // Selektovanje izlaznog out0 registra iz skupa registara.  
156     input regfile_in_ena,      // Signal dozvole upisa u registra iz skupa registara.  
157     input [3:0] regfile_in_sel, // Selektovanje registra iz skupa registra u koji ce biti upusana vrednost.  
158     input alu_ena,             // Aktiviranje aritmeticko-logicke jedinice.  
159     input [2:0] alu_op,        // Kod operacije za alu jedinicu.  
160     input pc_ena,             // Dozvola upisa PC registar.  
161     input pc_sel,             // Selektovanje ulaza u PC registar.  
162     input ir_ena,             // Dozvola upisa u IR registar.  
163     input psw_ena,            // Dozvola upisa u PSW registar.  
164     input mar_ena,            // Dozvola upisa u MAR registar.  
165     input mdr_ena,            // Dozvola upisa u MDR registar.  
166     input mdr_sel,            // Selektovanje ulaza u MDR registar.
```

Модел *datapath* компоненте

Листа излазних портова и њихова делкарација.

data порт је бидирекциони порт.

```
167 output [3:0] opcode,           // Kod operacije tekuće instrukcije.
168 output [3:0] dst,             // Adresa odredisnog registra tekuće instrukcije.
169 output [3:0] src0,            // Adresa prvog izvršnog registra tekuće instrukcije.
170 output [3:0] src1,            // Adresa drugog izvršnog registra tekuće instrukcije.
171 output zero,                  // Vrednost Z flega PSW registra.
172 output carry,                 // Vrednost C flega PSW registra.
173 output [15:0] addr,           // Memorijska adresa.
174 inout [15:0] data);          // Podatak koji se razmenjuje sa memorijom.
```

Декларација носиоца податка.

```
176 wire [15:0] bus_a, bus_b;     // Magistrale mikroprocesora.
177 reg [15:0] pc, ir;             // Registri PC i IR.
178 reg [1:0] psw;                 // Registar PSW.
179 reg [15:0] mar, mdr;           // Registri MAR i MDR.
180 wire [15:0] regfile_out0, regfile_out1; // Izlazi skupa registara opšte namene.
181 wire [15:0] alu_res;           // Rezultat racunanja u ALU komponente.
182 wire alu_zero, alu_carry;      // Flegovi ALU komponente.
```

Модел *datapath* компоненте

Селектовање сигнала који се прослеђују на магистрале А и В.

```
184 assign bus_a = a_sel[1] ? pc : (a_sel[0] ? mdr : regfile_out0);
185 assign bus_b = b_sel[1] ? (b_sel[0] ? alu_res : {{8{ir[7]}},
186 ir[7:0]}):(b_sel[0] ? mdr : regfile_out1);
```

Разбијање неких регистара на чланове ради сталне њихове доступности када год се неки промени. За то разбијање се користи операција придруживања. На порт *data* се прослеђују вредност *MDR* регистра само ако је у питању упис у меморију, а на порт *addr* се прослеђује вредност регистра *MAR* када год се догоди његова промена.

```
209 assign {opcode, dst, src0, src1} = ir;
210 assign {carry, zero} = psw;
211 assign addr = mar;
212 assign data = mem_op ? mdr : 16'dz;
```

Модел *datapath* компоненте

```
188 always @(pc_ena or pc_sel or bus_b)
189     if(pc_ena)
190         if(pc_sel)
191             pc <= bus_b;
192         else
193             pc <= 16'd0;
194 always @(ir_ena or bus_a)
195     if(ir_ena)
196         ir <= bus_a;
197 always @(psw_ena or alu_zero or alu_carry)
198     if(psw_ena)
199         psw = {alu_carry, alu_zero};
200 always @(mar_ena or bus_a)
201     if(mar_ena)
202         mar <= bus_a;
203 always @(mdr_ena or mdr_sel or bus_b or data)
204     if(mdr_ena)
205         case(mdr_sel)
206             1'b0: mdr <= data;
207             1'b1: mdr <= bus_b;
208         endcase
```

Креирање процеса који контролишу упис у *PC*, *IR*, *PSW*, *MAR*, *MDR* регистре.

Упис у *PC* и *MDR* регистре се контролише мултиплексорима јер постоје два извора са којих долази вредност за упис у ове регистре.

Модел *datapath* компоненте

У овој компоненти се инстанцирају објекти модела скупа опште намене и компонента аритметичко-логичке јединице. Постоје два начина повезивања портова инстанцираних модела са сигнаlima из окружења:

- повезување преко уређене листе,
- повезивање портова на основу њихових имена.

Овде је употребљена други начин повезивања.

```
214 /* Instanciranje skupa registara opste namene i aritmeticko-logicke jedinice*/
215 regfile _regfile(.sel_out0(regfile_out0_sel), .sel_out1(regfile_out1_sel),
216 .ena_in(regfile_in_ena), .sel_in(regfile_in_sel), .in(bus_b),
217 | .out0(regfile_out0), .out1(regfile_out1));
218
219 alu _alu(.ena(alu_ena), .in0(bus_a), .in1(bus_b), .op(alu_op),
220 .out(alu_res), .zero(alu_zero), .carry(alu_carry));
221 endmodule
```

Модел контролне јединице

Дефинисање макроа коришћених у моделу контролне јединице.

```
35  /* Koraci pri izvršenju instrukcija. */
36  `define FETCH      2'b00
37  `define DECODE     2'b01
38  `define EXECUTE    2'b10
39  `define WRITE      2'b11

62  /* Operacije sa memorijom. */
63  `define MEM_READ   1'b0
64  `define MEM_WRITE 1'b1

66  /* Selektroski signali za magistralu A. */
67  `define BUS_A_SEL_REGFILE_OUT0 2'b00
68  `define BUS_A_SEL_MDR           2'b01
69  `define BUS_A_SEL_PC            2'b10
```

```
42  /* Kodovi pojedinačnih instrukcija. */
43  `define ADD 4'b0000
44  `define INC 4'b0001
45  `define SUB 4'b0010
46  `define NOT 4'b0011
47  `define AND 4'b0100
48  `define OR  4'b0101
49  `define SLA 4'b0110
50  `define SRA 4'b0111
51  `define LD  4'b1000
52  `define ST   4'b1001
53  `define LDI 4'b1010
54  `define JMP 4'b1011
55  `define JZ  4'b1100
56  `define JB  4'b1101
```

Модел контролне јединице

```
71  /* Selektroski signali za magistralu B. */
72  `define BUS_B_SEL_REGFILE_OUT1 2'b00
73  `define BUS_B_SEL_MDR          2'b01
74  `define BUS_B_SEL_IMM          2'b10
75  `define BUS_B_SEL_ALU_RES      2'b11
```

```
77  /* Selektorski signali za PC registar. */
78  `define PC_SEL_0      1'b0
79  `define PC_SEL_BUS_B 1'b1
```

```
81  /* Selktorski signali za MDR registar. */
82  `define MDR_SEL_DATA  1'b0
83  `define MDR_SEL_BUS_B 1'b1
```

```
58  /* Konstante koje ativiraju tj. deaktiviraju odredjene signale. */
59  `define ENABLE  1'b1
60  `define DISABLE 1'b0
```

Модел контролне јединице

Контролна јединица се понаша као коначни аутомат са 4 стања, где стања представљају одговарајући кораћи при извршењу инструкције. Корази при извршењу инструкције су, доношење, декодирање, извршење и упис.

Листа улазних портова и њихова делкарација. Поред сигнала такта и ресета улазни сигнали су и сигнали од *datapath* компоненте који су потребни за генерисање контролних сигнала.

```
223 ///////////////////////////////////////////////////////////////////
224 /////////////////////////////////////////////////////////////////// MODEL KONTROLNE JEDINICE ///////////////////////////////////////////////////////////////////
225 ///////////////////////////////////////////////////////////////////
226 module control (
227     input clk,           // Signal takta.
228     input rst,           // Reset signal.
229     input [3:0] opcode,  // Komponenta tekuce instrukcije: opkod.
230     input [3:0] dst,      // Komponenta tekuce instrukcije: adresa odredisnog registra.
231     input [3:0] src0,     // Komponenta tekuce instrukcije: adresa izvornog registra 0.
232     input [3:0] src1,     // Komponenta tekuce instrukcije: adresa izvornog registra 1.
233     input zero,          // Vrednost zero flega psw registra.
234     input carry,         // Vrednost carry flega psw registra.
```

Модел контролне јединице

Листа излазних портова и њихова делкарација. Сви излазни сигнали представљају контролне сигнале који се повезују на одговарајуће улазе *datapath* јединице.

```
235 output reg mem_op,           // Tip pristupa memoriji (upis ili citanje).
236 output reg [1:0] a_sel,      // Odredjivanje sta se propusta na a magistralu.
237 output reg [1:0] b_sel,      // Odredjivanje sta se propusta na b magistralu.
238 output reg [3:0] regfile_out0_sel, // Selektroski signal za out0 izlaz skupa registara.
239 output reg [3:0] regfile_out1_sel, // Selektroski signal za out1 izlaz skupa registara.
240 output reg regfile_in_ena,    // Kontrolni signal upisa u registar iz skupa registara.
241 output reg [3:0] regfile_in_sel, //Registra iz skupa registra u koji ce biti upisana vrednost.
242 output reg alu_ena,          // Ukljucenje aritmeticko-logicke jedinice.
243 output reg [2:0] alu_op,      // Kod operacije za aritmeticko-logicku jedinicu.
244 output reg pc_ena,           // Kontrolni signal za PC registar.
245 output reg pc_sel,           // Selektroski signal za PC registar.
246 output reg ir_ena,           // Kontrolni signal za IR registar.
247 output reg psw_ena,          // Kontrolni signal za PSW registar.
248 output reg mar_ena,          // Kontrolni signal za MAR registar.
249 output reg mdr_ena,          // Kontrolni signal za MDR registar.
250 output reg mdr_sel);         // Selektroski signal za MDR registar.

252 reg [1:0] step; // Koraci pri izvršenju, tj. stanje konacnog automata.
```

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
```

```
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
```

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.

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* регистра 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 А, 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 читавање из меморије, вредност регистра опште који садржи адресу са које се врши читавање се преко магистрале А пребацује у *MAR* регистар.

Дозвољава се упис у *MDR* регистар, а као извор се бира порт преко кога се врши трансфер података са меморије. Затим се одмах садржај овог регистра препушта на магистралу В, тако да ће подаци бити доступни чим се читају 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 податка која се уписује се пребацују у *MAR* и *MDR* регистре. За меморију се селекује 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 читавање константе на магистрале В пропушта константа и селекује 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
```

Овде се завршава структура *case* трећег корака.

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 селекује се операција сабирања.

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 меморије 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 инструкција скока, тада се вредност 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 упис 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; //Određivanje 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; // Selektorski 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 selektorski 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 tekuće 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 саставни 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 дизајнирати 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 // Bidirekcionni 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 уčitане вредности из *dump* фајла. Меморија је имплементирана као поље од 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 (`DUMPFIL, 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 *dumpr* 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 нису 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
```

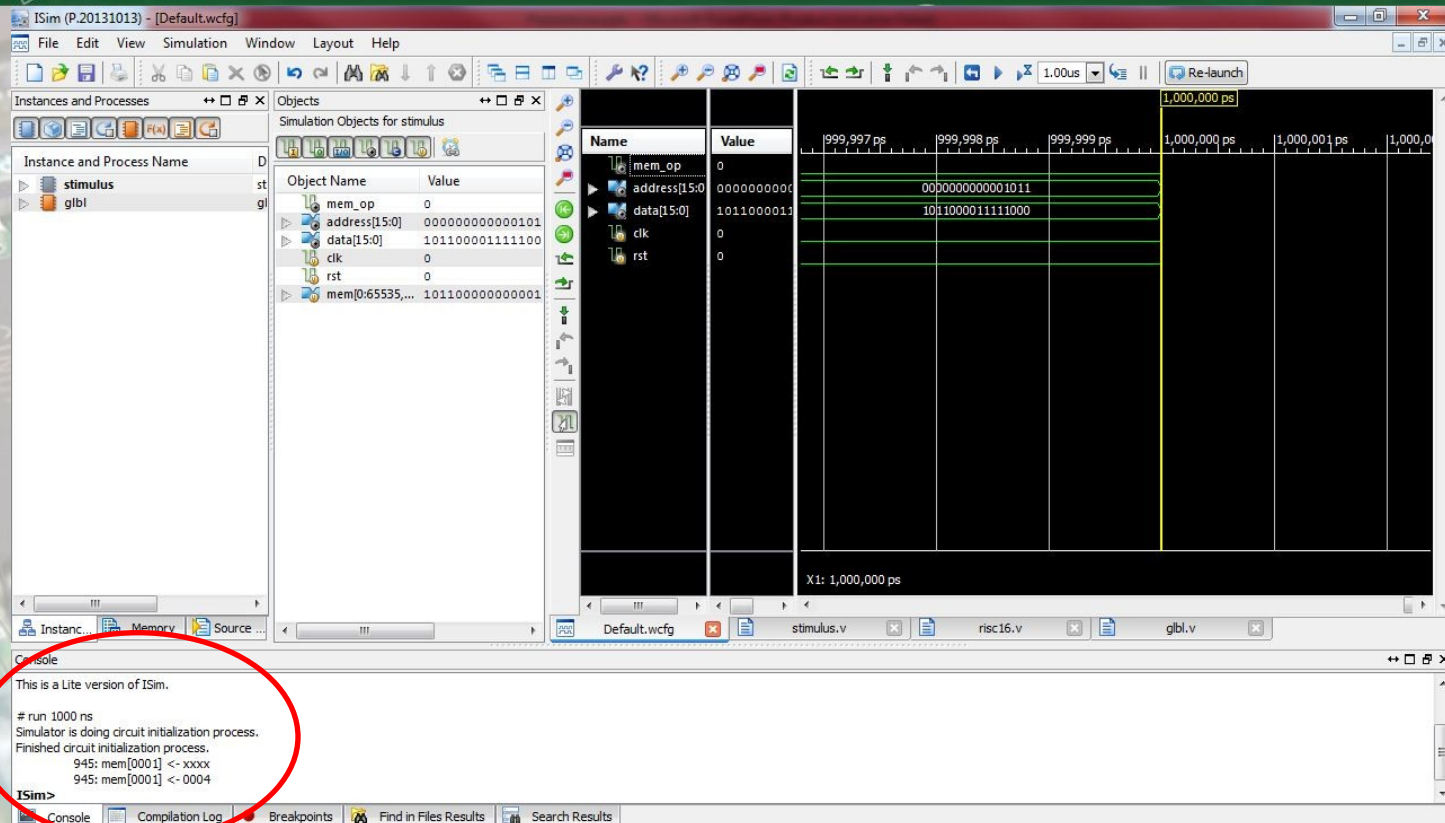
Приказ горе наведеног код у машинском језику у неком хипотетичком асемблеру.

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.).