Univerzitet u Beogradu Elektrotehnički fakultet
katedra za računarsku tehniku i informatiku

Sistemsko programiranje projekat

picoComputer - emulator

Student: Petar Parabucki
Br. indeksa: 115/06

Januar 2015.

Profesor: dr Dragan Bojić
Asistent: Saša Stojanović

Sadržaj:

Opis problema 3
Kratak opis rešenja i uputstvo za pokretanje programa 4
Primeri ulaznih podataka i dobijenih izlaza 5
Listing programa: 10

Opis problema

Napisati emulator za picoComputer (računar objašnjen na predmetu Programiranje 1). Emulator treba da bude implementiran kao statički rekompilator. Ulaz emulatora je fajl koji se dobija na izlazu asemblera za picoComputer.

Kratak opis rešenja i uputstvo za pokretanje programa

Emulator je izrađen korišćenjem programskog jezika C i C++.

Program se sastoji od jednog fajla: **recompiler**.c

U fajlu **recompiler.c** se nalazi glavni program, koji poziva metodu ***main().*** Glavni deo emulatora, koji se odnosi na interpretiranje i izvršavanje komandi, realizovan je korišćenjem ***switch*** structure, u okviru koje se vrši učitavanje jedne po jedne instrukcije koje se učitavaju iz ulaznog fajla koji je u .HEX formatu.

Pri pokretanju je potrebno navesti dva parametra. Nakon naziva programa, kao drugi parametar treba uneti naziv .HEX test fajla u kome se nalazi heksadecimalni kod test programa. Kao prvi parametar treba navesti **ime programa prevedenog pomoću komande ./comp** .

Na primer, ukoliko se ulazni test program nalazi u fajlu **test.hex**, za pokretanje emulatora je u komandnoj liniji potrebno otkucati: ***./emu ./test.hex*** **.** Test fajl treba da se nalazi u istom folderu kao i izvršni fajl **emu**.

Program će prijaviti grešku ukoliko se ne unese naziv fajla koji sadrži ulazni test program, ili ukoliko ne postoji fajl čije je ime navedeno kao parametar.

Prevođenje se vrši pozivanjem komande : **./comp** , koja u sebi sadrži sledeći kod:
gcc -w recompiler.c -L/usr/local/lib/ -llightning -Wl,-rpath -Wl,/usr/local/lib/ -o emu

Primeri ulaznih podataka i dobijenih izlaza

Za potrebe testiranja emulatora korišćeni su programi koji su dati u skripti iz vežbi iz predmeta Programiranje 1 .

Test 1 : ZADATAK 2

Sastaviti program na mašinskom jeziku pC, koji učitava dva cela broja sa tastature i ispisuje ih po nerastućem redosledu.

PICO COMPUTER KOD	HEX KOD
P=0	
A=1	**0008**
B=2	**7102**
ORG 8	**6128**
IN A,2	**0010**
BGT A,B,KRAJ	**5128**
BEQ A,B,KRAJ	**0010**
MOV P,A	**0010**
MOV A,B	**0120**
MOV B,P	**0200**
KRAJ: STOP A,B	**F120**

Test 2: ZADATAK 4

Sastaviti program na mašinskom jeziku pC, koji učitava N celih brojeva sa tastature, a zatim izračunava i ispisuje celobrojni deo aritmetičke sredine tih brojeva.

PICO COMPUTER KOD	HEX KOD
N=1	**0008**
TEK=2	**7101**
S=3	**0208**
I=4	**0064**
ORG 8	**7A81**
IN N	**0308**
MOV TEK,100	**0000**
IN (TEK),N	**0408**
MOV S,0	**0000**
MOV I,0	**133A**
PETLJA: ADD S,S,(TEK)	**9220**
ADD TEK,TEK,1	**0001**
ADD I,I,1	**9440**
BGT N,I,PETLJA	**0001**
DIV S,S,N	**6148**
STOP S	**0010**
	4331
	F300

Test 3: ZADATAK 5

Sastaviti program na mašinskom jeziku pC, za izračunavanje zbira prvih N prirodnih brojeva, i zbira kvadrata prvih N prirodnih brojeva.

PICO COMPUTER KOD	HEX KOD
N=1	
S1=2	**0008**
S2=3	**7101**
K=4	**0208**
ORG 8	**0000**
IN N	**0308**
MOV S1,0	**0000**
MOV S2,0	**1221**
PETLJA: ADD S1,S1,N	**3411**
MUL K,N,N	**1334**
ADD S2,S2,K	**A110**
SUB N,N,1	**0001**
BGT N,0,PETLJA	**6108**
STOP S1,S2	**000D**
	F230

Test 4: ZADATAK 6

Sastaviti program na mašinskom jeziku pC kojim se na osnovu dva data niza brojeva A[i] I B[i] formira novi niz C[i] , tako da C[i]=A[i]+B[i] {i=0….N-1}

PICO COMPUTER KOD	HEX KOD
A=100	**0008**
B=200	**7001**
C=300	**0108**
N=0	**0064**
adrA=1	**0208**
adrB=2	**00C8**
adrC=3	**0308**
I=4	**012C**
ORG 8	**7980**
IN N	**7A80**
MOV adrA, #A	**0400**
MOV adrB, #B	**1B9A**
MOV adrC, #C	**9110**
IN (adrA),N	**0001**
IN (adrB),N	**9220**
MOV I,N	**0001**
DALJE: ADD (adrC),(adrA),(adrB)	**9330**
ADD adrA,adrA,1	**0001**
ADD adrB,adrB,1	**A440**
ADD adrC,adrC,1	**0001**
SUB I,I,1	**6408**
BGT I,0,DALJE	**0012**
MOV adrC, #C	**0308**
OUT (adrC),N	**012C**
STOP	**8B80**
	F000

Test 5: ZADATAK 7

Koje vrednosti ispisuje priloženi program za pC. Tačan odgovor je 2 3 2.

PICO COMPUTER KOD	HEX KOD
X=1	
Y=2	**0008**
Z=3	**0108**
ORG 8	**0002**
MOV X,#Y	**9210**
ADD Y,X,#X	**0001**
MOV (Y),#Y	**0A08**
STOP X,Y,Z	**0002**
	F123

Test 6: ZADATAK 8

Sastaviti program na mašinskom jeziku pC, kojim se iz datog niza celih brojeva izostavljaju svi elementi koji su parni.

PICO COMPUTER KOD	HEX KOD
A=200	**0008**
N=1	**7101**
adrI=2	**0208**
adrJ=3	**00C8**
K=4	**0320**
P=5	**7A81**
ORG 8	**0410**
IN N	**C5A0**
MOV adrI,#A	**0002**
MOV adrJ,adrI	**B550**
IN (adrI),N	**0002**
MOV K,N	**55A8**
DALJE: DIV P,(adrI),2	**0017**
MUL P,P,2	**0BA0**
BEQ P,(adrI),PAR	**9330**
MOV (adrJ),(adrI)	**0001**
ADD adrJ,adrJ,1	**9220**
PAR: ADD adrI,adrI,1	**0001**
SUB K,K,1	**A440**
BGT K,0,DALJE	**0001**
SUB N,adrJ,#A	**6408**
BEQ N,0,KRAJ	**000E**
MOV adrI,#A	**A130**
OUT (adrI),N	**00C8**
KRAJ: STOP	**5108**
	0024
	0208
	00C8
	8A81
	F000

Test 7: ZADATAK 9

Sastaviti potprogram na mašinskom jeziku pC, za izračunavanje N!.

PICO COMPUTER KOD	HEX KOD
N=1	**0008**
F=2	**7101**
ORG 8	**6018**
DALJE: IN N	**0010**
BGT 0,N,KRAJ	**D000**
JSR NF	**0011**
OUT F	**8201**
BEQ N,N,DALJE	**5118**
KRAJ: STOP	**0008**
NF: MOV F,1	**F000**
BEQ N,0,GOTOVO	**0208**
PETLJA: MUL F,F,N	**0001**
SUB N,N,1	**5108**
BGT N,0,PETLJA	**001A**
GOTOVO: RTS	**3221**
	A110
	0001
	6108
	0015
	E000

Test 8: ZADATAK 10

Po startovanju sledećeg programa nap C , redom se unose sledeće vrednosti: 22, 16, 2. Šta će biti ispisano?

Odgovor : 5 , 4 , 2

PICO COMPUTER KOD	HEX KOD
X=1	**0008**
Y=2	**0408**
N=3	**0000**
K=4	**0508**
M=5	**0003**
ORG 8	**7301**
MOV K,0	**D000**
MOV M,#N	**0014**
L1: IN N	**9440**
JSR SBR	**0001**
ADD K,K,1	**6548**
BGT M,K,L1	**000C**
STOP	**F000**
	0108
SBR: MOV X,0	**0000**
L2: ADD X,X,1	**9110**
MUL Y,X,X	**0001**
BGT N,Y,L2	**3211**
OUT X	**6328**
RTS	**0016**
	8101
	E000

Dati .HEX kodovi su dobijeni kao izlaz iz picoComputer asemblera, kada su mu prosleđeni programi koji su u skladu sa picoComputer sintaksom.

Listing programa:

```
// Project: Sistemsko Programiranje
// File:  staticRecompiler.c
// Date:  December 2014 - January 2015
// Author:  Petar Parabucki 115/06

#include <stdio.h>
#include <lightning.h>
#include <stdbool.h>

#define MAX_NUM_OF_INSTRUCTIONS 100
#define MEMMORY_SIZE 0x10000         // 2^16 je velicina memorije pC

#define MASK_KO 0xF000
#define MASK_KO_SH 12
#define MASK_A1 0x700
#define MASK_A1_SH 8
#define MASK_A2 0x70
#define MASK_A2_SH 4
#define MASK_A3 0x7
#define MASK_A3_SH 0
#define MASK_i1 0x800
#define MASK_i1_SH 11
#define MASK_i2 0x80
#define MASK_i2_SH 7
#define MASK_i3 0x8
#define MASK_i3_SH 3
#define MASK_X  0xF00
#define MASK_X_SH 8
#define MASK_Y  0xF0
#define MASK_Y_SH 4
#define MASK_N  0xF
#define MASK_N_SH 0




static jit_state_t *_jit;
jit_state_t* jit_states[MAX_NUM_OF_INSTRUCTIONS]; // states of jit , max instructions


//-------------------------------------------------------------
//              FUNKCIJE
//-------------------------------------------------------------

typedef int (*MOV)    (int, int);      /* X:=Y */
typedef int (*MOV_C)  (int, int);      /* X:=C */
typedef int (*MOV_AN) (int, int);      /* X[j]:=Y[j] N-1 */
typedef int (*MOV_AC) (int, int);      /* X[j]:=Y[j] C-1*/
```

```
typedef int (*ADD)    (int, int, int);
typedef int (*ADD_I)  (int, int, int);
typedef int (*SUB)    (int, int, int);
typedef int (*SUB_I)  (int, int, int);
typedef int (*MUL)    (int, int, int);
typedef int (*MUL_I)  (int, int, int);
typedef int (*DIV)    (int, int, int);
typedef int (*DIV_I)  (int, int, int);

typedef int (*STOP)   ();
//-------------------------------------------------------------


FILE* getFile(char* argv){

  FILE *ifp;

  ifp = fopen(argv, "r");  // open file for read

  if (ifp == NULL) {
    printf("Nije moguce otvoriti fajl - sledi izlazak iz programa!\n");
    printf("%s \n",argv);
    exit(1);
  }

  return ifp;
}

//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

void coolPrint(){

  printf("*****************************************************************************\n");
  printf(" ____ ____          ____          _     _        _ \n");
  printf("/ ___||  _ \\         |  _ \\ _ __ ___ (_) ___| | ____ _|  |_\n");
  printf("\\___ \\| |_| |  _____  | |_) | '__/ _ \\| |/ _ \\ |/ / _` | __|\n");
  printf(" ___) |  __/  |_____| |  __/| | | (_) | |  __/   < (_| | |_\n");
  printf("|____/|_|             |_|   |_|  \\___// |\\___|_|\\_\\__,_|\\__|\n");
  printf("                                    |__/\n");
  printf("*****************************************************************************\n");
}

//-------------------------------------------------------------

int main(int argc, char* argv[]) {


  FILE *fajl;
  int rec;
  short int numOfI=0;
```

```
  short int numOfIJIT=0;
  bool isIns = true;


// Struktura masinske instrukcije (16 bita) :
// u komentaru stoji kako ce biti ucitavani pocevsi od index-a 0
  char kodOp = 0;  // biti 12 13 14 15
  char i1 = 0;     // biti 11
  char a1 = 0;     // biti 8 9 10
  char i2 = 0;     // biti 7
  char a2 = 0 ;    // biti 4 5 6
  char i3 = 0;     // biti 3
  char a3 = 0;     // biti 0 1 2
//------------------------------

// fiksne adrese
 // potrebno 2^16

  int arg1,arg2,arg3;
  int adr[MEMMORY_SIZE] = {};  // oduzeto 8 zbog toga sto fiksne adrese cuvam u fAdr
  int fAdr[8] ={}  ;
  int startAdr = 0;            // pokazuje od koje adrese pocinje program u memoriji
  int PC = 0;                  // pokazuje do koje instrukcije smo stigli
  int SP = MEMMORY_SIZE-1;       // pokazuje na prvu slobodnu adresu

  jit_node_t *sp;

  char uX,uY,uN,izX,izY,izN;

// funkcije :
  MOV     mov;
  MOV_C   mov_c;
  MOV_AN  mov_an;
  MOV_AC  mov_ac;

  ADD     add;
  ADD_I   add_i;
  SUB     sub;
  SUB_I   sub_i;
  MUL     mul;
  MUL_I   mul_i;
  DIV     divv;
  DIV_I   divv_i;

  STOP    stop;

// da se definise samo jednom

  bool    dMov    = true;
  bool    dMov_c  = true;
  bool    dMov_an = true;
```

```
  bool    dMov_ac = true;

  bool    dAdd    = true;
  bool    dAdd_i  = true;
  bool    dSub    = true;
  bool    dSub_i  = true;
  bool    dMul    = true;
  bool    dMul_i  = true;
  bool    dDiv    = true;
  bool    dDiv_i  = true;

  coolPrint();

//-------------------------------------------------------------
//              Dohvatanje fajla
//-------------------------------------------------------------

  fajl = getFile(argv[1]);
  printf("Fajl %s je dohvacen, sledi ucitavanje isntrukcija.\n",argv[1]);

//-------------------------------------------------------------

//-------------------------------------------------------------
//              Ucitavanje instrukcija
//-------------------------------------------------------------
  fscanf(fajl, "%d", &startAdr);  // od kojeg mesta u memoriji pocinje kod programa
  while (fscanf(fajl, "%x", &adr[numOfI+startAdr]) != EOF ) {
    numOfI++;
  }

  printf("Ukupan broj ucitanih isntrukcija je : %d \n",numOfI);

//-------------------------------------------------------------

//-------------------------------------------------------------
//              Ispis instrukcija
//-------------------------------------------------------------

 printf("Sledi ispis ucitanih isntrukcija:\n\n");
int i=0;
  while(i<numOfI){
    printf("\t%2.d. instrukcija %.4x \n",i+1,adr[i+startAdr] );
    i++;
  }
  printf("\n");

//-------------------------------------------------------------


//-------------------------------------------------------------
//              Dekodovanje instrukcija
```

```
//------------------------------------------------------------

i=0;
PC = startAdr;

init_jit(argv[0]);

while((PC-startAdr)<numOfI){

//------------------------------------------------------------
//            Instrukcijska rec razlozena
//------------------------------------------------------------

  kodOp = (adr[PC] & MASK_KO)>>MASK_KO_SH;
  a1    = (adr[PC] & MASK_A1)>>MASK_A1_SH;
  a2    = (adr[PC] & MASK_A2)>>MASK_A2_SH;
  a3    = (adr[PC] & MASK_A3)>>MASK_A3_SH;
  i1    = (adr[PC] & MASK_i1)>>MASK_i1_SH;
  i2    = (adr[PC] & MASK_i2)>>MASK_i2_SH;
  i3    = (adr[PC] & MASK_i3)>>MASK_i3_SH;


//------------------------------------------------------------

  /*
  printf("isntrukcija %x\n",adr[i] );
  int j;
  printf("kodOP : %x\n",kodOp);
  printf("i1 : %x ",i1);
  printf("a1 : %x\n",a1);
  printf("i2 : %x ",i2);
  printf("a2 : %x\n",a2);
  printf("i3 : %x ",i3);
  printf("a3 : %x\n",a3);
  printf("\n");
  */


  if( ! isIns ){ // OVDE ISPITUJEM DA LI JE REC ADRESA, ILI KONSTANTA, A NE INSTRUKCIJA
    if( ((adr[PC-1] & MASK_KO)>>MASK_KO_SH) == 0x5 ||
        ((adr[PC-1] & MASK_KO)>>MASK_KO_SH) == 0x6) {
      printf("OVO JE ADRESA SKOKA : %x\n",adr[PC]);
    }

  }
  else
  switch(kodOp){

    //  OPERACIJA MOV
    case 0x0 :
            printf("OPERACIJA MOV\n");
```

```
    if(i3==0x0 && a3 == 0x0 ){
      printf("X:=Y\n");

      uX  = (adr[PC] & MASK_A1)>>MASK_A1_SH;
      izY = (adr[PC] & MASK_A2)>>MASK_A2_SH;

      if(dMov==true){

      jit_states[numOfIJIT] = _jit = jit_new_state();

      jit_prolog   ();
arg1 = jit_arg();  // uX
arg2 = jit_arg();  // PC+1
      jit_getarg   (JIT_R0, arg1);
      jit_getarg   (JIT_R1, arg2);
      jit_movr     (JIT_R0,JIT_R1);
      jit_retr     (JIT_R0);

     //kod za generisanje prve funkcije
      mov = jit_emit();
      jit_clear_state();

      numOfIJIT++;
      dMov=false;

      }

      if(i1==0x1 && i2==0x0){
        //printf("W1 INDIREKTNO!!!\n");
        adr[adr[uX]] = mov(adr[uX],adr[izY]); // sledeca rec je konstanta
      }else if(i1==0x0 && i2==0x0){
        //printf("DIREKTNO!!!\n");
        adr[uX] = mov(adr[uX],adr[izY]); // sledeca rec je konstanta
      }else if(i1==0x0 && i2==0x1){
        //printf("W2 INDIREKTNO!!! %d\n");
        adr[uX] = mov(adr[uX],adr[adr[izY]]); // sledeca rec je konstanta
      }



    }else if(i3==0x1 && a3 == 0x0 ){
      // DVE RECI
      printf("X:=C , sledeca rec je konstanta : %x\n",adr[PC+1]);  // konstanta

      uX = (adr[PC] & MASK_A1)>>MASK_A1_SH;

      if(dMov_c==true){
      jit_states[numOfIJIT] = _jit = jit_new_state();

      jit_prolog   ();
```

```
arg1 = jit_arg();  // uX
arg2 = jit_arg();  // PC+1
      jit_getarg   (JIT_R0, arg1);
      jit_getarg   (JIT_R1, arg2);
      jit_movr     (JIT_R0,JIT_R1);
      jit_retr     (JIT_R0);

     //kod za generisanje prve funkcije
      mov_c = jit_emit();
      jit_clear_state();

      numOfIJIT++;
      dMov_c = false;
     }

      //printf("adr[adr[PC]] = %d , PC = %d adr[PC]=%d\n",adr[adr[PC]] ,PC,adr[PC]);

      if(i1==0x1){
        //printf("W1 INDIREKTNO!!!\n");
        adr[adr[uX]] = mov_c(adr[uX],adr[PC+1]); // sledeca rec je konstanta
      }else{
        //printf("W1 DIREKTNO!!!\n");
        adr[uX] = mov_c(adr[uX],adr[PC+1]); // sledeca rec je konstanta
      }


      PC++;

    }else if(i3==0x0 && a3 != 0x0 ){

      printf("X[j]:=Y[j] , j=0,...,N-1 , max 7\n");  // niz sa brojem koji je konstanta

      uX  = (adr[PC] & MASK_X)>>MASK_X_SH;
      izY = (adr[PC] & MASK_Y)>>MASK_Y_SH;
      izN = (adr[PC] & MASK_A3)>>MASK_A3_SH;

      if(dMov_an==true){

      jit_states[numOfIJIT] = _jit = jit_new_state();

      jit_prolog   ();
arg1 = jit_arg();  // uX
arg2 = jit_arg();  // PC+1
      jit_getarg   (JIT_R0, arg1);
      jit_getarg   (JIT_R1, arg2);
      jit_movr     (JIT_R0,JIT_R1);
      jit_retr     (JIT_R0);

      //kod za generisanje prve funkcije
      mov_an = jit_emit();
      jit_clear_state();
```

```
          numOfIJIT++;
          dMov_an = false;
        }


          adr[uX] = mov_an(adr[uX],adr[izY]); // sledeca rec je konstanta



        }else if(i3==0x1 && a3 == 0x7 ){
          // DVE RECI

          // TODO !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
          // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

          printf("X[j]:=Y[j] , j=0,...,C-1 , sledeca rec je konstanta\n");  // niz sa brojem koji je konstanta
          PC++;
        }

break;

//  OPERACIJA ADD
case 0x1 :
        printf("OPERACIJA ADD A1 = A2 + A3 (format)\n");

        if(dAdd == true){

        jit_states[numOfIJIT] = _jit = jit_new_state();

        jit_prolog   ();
  arg1  = jit_arg();
  arg2  = jit_arg();
  arg3  = jit_arg();
        jit_getarg   (JIT_R0, arg1);
        jit_getarg   (JIT_R1, arg2);
        jit_getarg   (JIT_R2, arg3);
        jit_addr     (JIT_R0,JIT_R1,JIT_R2);
        jit_retr     (JIT_R0);

  //kod za generisanje prve funkcije
        add = jit_emit();
        jit_clear_state();
        numOfIJIT++;
        dAdd = false;

        }
        //printf("a1 = %d ,a2 = %d ,a3 = %d  \n",a1,a2,a3 );
        adr[a1] = add(adr[a1],adr[a2],adr[a3]);
        //printf("ADD %d\n",adr[a1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a2],adr[a3]);
```

```
break;

//  OPERACIJA ADD INDIREKTNO
case 0x9 :
        printf("OPERACIJA ADD A1:= A2 + val(val(PC)) ili A1:= val(val(PC)) + A3 , mem indirektno\n");

        if(dAdd_i == true){

        jit_states[numOfIJIT] = _jit = jit_new_state();

        jit_prolog   ();
 arg1  = jit_arg();
 arg2  = jit_arg();
 arg3  = jit_arg();
        jit_getarg   (JIT_R0, arg1);
        jit_getarg   (JIT_R1, arg2);
        jit_getarg   (JIT_R2, arg3);
        jit_addr     (JIT_R0,JIT_R1,JIT_R2);
        jit_retr     (JIT_R0);

 //kod za generisanje prve funkcije
        add_i = jit_emit();
        jit_clear_state();
        numOfIJIT++;
        dAdd_i = false;
       }
        //printf("a1 = %d ,a2 = %d ,a3 = %d  \n",a1,a2,a3 );
        if(a2==0){

          adr[a1] = add_i(adr[a1],adr[a3],adr[PC+1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a3],adr[PC+1]);
        }
        else{
          adr[a1] = add_i(adr[a1],adr[a2],adr[PC+1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a2],adr[PC+1]);
        }
        //printf("ADD %d\n",adr[a1]);

        PC++;

break;

//  OPERACIJA SUB
case 0x2 :
        printf("OPERACIJA SUB\n");

        if(dSub == true){
```

```
        jit_states[numOfIJIT] = _jit = jit_new_state();

        jit_prolog   ();
  arg1  = jit_arg();
  arg2  = jit_arg();
  arg3  = jit_arg();
        jit_getarg   (JIT_R0, arg1);
        jit_getarg   (JIT_R1, arg2);
        jit_getarg   (JIT_R2, arg3);
        jit_subr     (JIT_R0,JIT_R1,JIT_R2);
        jit_retr     (JIT_R0);

  //kod za generisanje prve funkcije
        sub = jit_emit();
        jit_clear_state();
        numOfIJIT++;
        dSub = false;

        }
        //printf("a1 = %d ,a2 = %d ,a3 = %d  \n",a1,a2,a3 );
        adr[a1] = sub(adr[a1],adr[a2],adr[a3]);
        //printf("ADD %d\n",adr[a1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a2],adr[a3]);

break;

//  OPERACIJA SUB INDIREKTNO
case 0xA :
        printf("OPERACIJA SUB, mem indirektno\n");

        if(dSub_i == true){

        jit_states[numOfIJIT] = _jit = jit_new_state();

        jit_prolog   ();
  arg1  = jit_arg();
  arg2  = jit_arg();
  arg3  = jit_arg();
        jit_getarg   (JIT_R0, arg1);
        jit_getarg   (JIT_R1, arg2);
        jit_getarg   (JIT_R2, arg3);
        jit_subr     (JIT_R0,JIT_R1,JIT_R2);
        jit_retr     (JIT_R0);

  //kod za generisanje prve funkcije
        sub_i = jit_emit();
        jit_clear_state();
        numOfIJIT++;
        dSub_i = false;
       }
        //printf("a1 = %d ,a2 = %d ,a3 = %d  \n",a1,a2,a3 );
```

```
        if(a2==0){

          adr[a1] = sub_i(adr[a1],adr[a3],adr[PC+1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a3],adr[PC+1]);
        }
        else{
          adr[a1] = sub_i(adr[a1],adr[a2],adr[PC+1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a2],adr[PC+1]);
        }
        //printf("ADD %d\n",adr[a1]);

        PC++;
break;

//  OPERACIJA MUL
case 0x3 :
        printf("OPERACIJA MUL\n");

        if(dMul == true){

        jit_states[numOfIJIT] = _jit = jit_new_state();

        jit_prolog   ();
 arg1  = jit_arg();
 arg2  = jit_arg();
 arg3  = jit_arg();
        jit_getarg   (JIT_R0, arg1);
        jit_getarg   (JIT_R1, arg2);
        jit_getarg   (JIT_R2, arg3);
        jit_mulr     (JIT_R0,JIT_R1,JIT_R2);
        jit_retr     (JIT_R0);

 //kod za generisanje prve funkcije
        mul = jit_emit();
        jit_clear_state();
        numOfIJIT++;
        dMul = false;

        }
        //printf("a1 = %d ,a2 = %d ,a3 = %d  \n",a1,a2,a3 );
        adr[a1] = mul(adr[a1],adr[a2],adr[a3]);
        //printf("ADD %d\n",adr[a1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a2],adr[a3]);

break;

//  OPERACIJA MUL INDIREKTNO
case 0xB :
        printf("OPERACIJA MUL, mem indirektno\n");
        if(dMul_i == true){
```

```
        jit_states[numOfIJIT] = _jit = jit_new_state();

        jit_prolog   ();
  arg1  = jit_arg();
  arg2  = jit_arg();
  arg3  = jit_arg();
        jit_getarg   (JIT_R0, arg1);
        jit_getarg   (JIT_R1, arg2);
        jit_getarg   (JIT_R2, arg3);
        jit_mulr     (JIT_R0,JIT_R1,JIT_R2);
        jit_retr     (JIT_R0);
  //kod za generisanje prve funkcije
        mul_i = jit_emit();
        jit_clear_state();
        numOfIJIT++;
        dMul_i = false;
       }
        //printf("a1 = %d ,a2 = %d ,a3 = %d  \n",a1,a2,a3 );
        if(a2==0){

          adr[a1] = mul_i(adr[a1],adr[a3],adr[PC+1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a3],adr[PC+1]);
        }
        else{
          adr[a1] = mul_i(adr[a1],adr[a2],adr[PC+1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a2],adr[PC+1]);
        }
        //printf("ADD %d\n",adr[a1]);
        PC++;
break;

//  OPERACIJA DIV
case 0x4 :
        printf("OPERACIJA DIV\n");

        if(dDiv == true){

        jit_states[numOfIJIT] = _jit = jit_new_state();

        jit_prolog   ();
  arg1  = jit_arg();
  arg2  = jit_arg();
  arg3  = jit_arg();
        jit_getarg   (JIT_R0, arg1);
        jit_getarg   (JIT_R1, arg2);
        jit_getarg   (JIT_R2, arg3);
        jit_divr     (JIT_R0,JIT_R1,JIT_R2);
        jit_retr     (JIT_R0);

  //kod za generisanje prve funkcije
        divv = jit_emit();
```

```
        jit_clear_state();
        numOfIJIT++;
        dDiv = false;

        }
        //printf("a1 = %d ,a2 = %d ,a3 = %d  \n",a1,a2,a3 );
        adr[a1] = divv(adr[a1],adr[a2],adr[a3]);
        //printf("ADD %d\n",adr[a1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a2],adr[a3]);

break;

//  OPERACIJA DIV INDIREKTNO
case 0xC :
        printf("OPERACIJA DIV, mem indirektno\n");
        if(dDiv_i == true){

        jit_states[numOfIJIT] = _jit = jit_new_state();

        jit_prolog   ();
 arg1  = jit_arg();
 arg2  = jit_arg();
 arg3  = jit_arg();
        jit_getarg   (JIT_R0, arg1);
        jit_getarg   (JIT_R1, arg2);
        jit_getarg   (JIT_R2, arg3);
        jit_divr     (JIT_R0,JIT_R1,JIT_R2);
        jit_retr     (JIT_R0);

 //kod za generisanje prve funkcije
        divv_i = jit_emit();
        jit_clear_state();
        numOfIJIT++;
        dDiv_i = false;
       }
        //printf("a1 = %d ,a2 = %d ,a3 = %d  \n",a1,a2,a3 );
        if(a2==0){

          adr[a1] = divv_i(adr[a1],adr[a3],adr[PC+1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a3],adr[PC+1]);
        }
        else{
          adr[a1] = divv_i(adr[a1],adr[a2],adr[PC+1]);
        //printf("ADD : %d=%d+%d\n", adr[a1],adr[a2],adr[PC+1]);
        }
        //printf("ADD %d\n",adr[a1]);

        PC++;
break;

//  OPERACIJA BEQ
```

```
case 0x5 :
        printf("OPERACIJA BEQ\n");
break;

// OPERACIJA BGT
case 0x6 :
        printf("OPERACIJA BGT\n");
break;

// OPERACIJA IN
case 0x7 :
        printf("OPERACIJA IN\n");
break;

// OPERACIJA OUT
case 0x8 :
        printf("OPERACIJA OUT\n");
break;

// OPERACIJA JSR
case 0xD :
        printf("OPERACIJA JSR\n");
break;

// OPERACIJA RTS
case 0xE :
        printf("OPERACIJA RTS\n");
break;

// OPERACIJA STOP
case 0xF :
        printf("OPERACIJA STOP\n");

        jit_states[numOfIJIT] = _jit = jit_new_state();

        jit_prolog   ();
arg1     = jit_arg();
arg2     = jit_arg();
arg3     = jit_arg();
        jit_getarg   (JIT_R0, arg1);
        jit_getarg   (JIT_R1, arg2);
        jit_getarg   (JIT_R2, arg3);

        short int numPrint=0;

        if(adr[a1]!=0){
          numPrint++;
        }
        if(adr[a2]!=0){
          numPrint++;
        }
```

```
          if(adr[a3]!=0){
            numPrint++;
          }

          switch(numPrint){

            case 1: jit_pushargi("STOP - %d\n"); break;
            case 2: jit_pushargi("STOP - %d , %d\n"); break;
            case 3: jit_pushargi("STOP - %d , %d , %d\n"); break;
            default: jit_pushargi("STOP - nema argumenata\n"); break;
          }

          jit_ellipsis();

          if(adr[a1]!=0){
            jit_pushargr(JIT_R0);
          }
          if(adr[a2]!=0){
            jit_pushargr(JIT_R1);
          }
          if(adr[a3]!=0){
            jit_pushargr(JIT_R2);
          }
          jit_finishi(printf);
          jit_ret();
          stop = jit_emit();
          jit_clear_state();

          stop(adr[a1],adr[a2],adr[a3]);
          numOfIJIT++;
   break;

   default: printf("ERROR\n"); break;
  }
  PC++;
}
printf("FIKSNA MEMORIJA, STAMPA:\n");
for(i = 0; i<8; i++){
  printf("adr[%d]=%d \n",i,adr[i]);
  //if(i%10==0) printf("\n");
}
printf("\n");
printf("numOfIJIT = %d:\n",numOfIJIT);
for(i = 0; i<numOfIJIT; i++){
  _jit = jit_states[i];
  jit_destroy_state();
}

  finish_jit();
return 0;
}
```