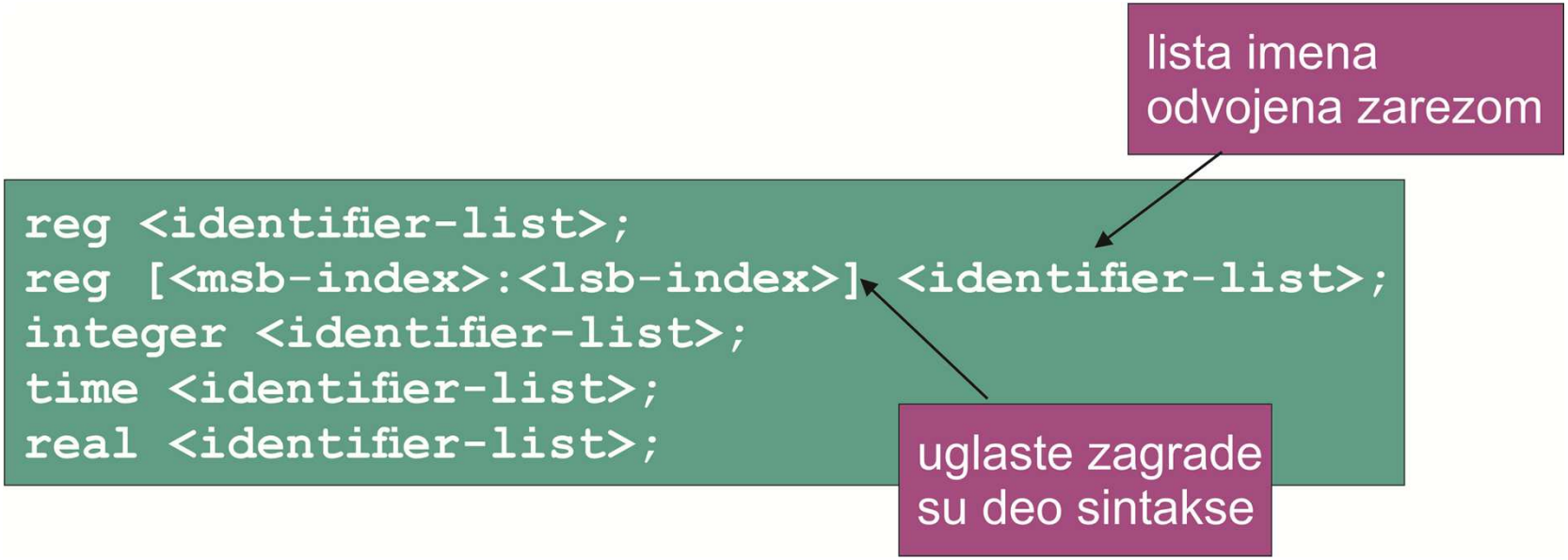
PROJEKTOVANJE INTEGRISANIH KOLA SA MEŠOVITIM SIGNALIMA 2

Registri

- Registri su promenljive koje služe da smeštaju vrednosti
- Registri mogu biti:
  - Skalarni (1 bit)
  - Vektorski (*n* bita)
  - Integer (32 bita)
  - Time (64 bita) - smešta vreme simulacije
  - Real (64 bita) - floating-point vrednosti

Registri

- Sintaksa:



Registri

- Deklariše se u modulu

```
module MODEL;

integer i;

always
begin
    i = i + 1;
end

endmodule
```

initial i *always* blokovi
mogu da pristupaju deklaraciji
na nivou modula

Registri sa više bitova

- Neoznačena vrednost, MSB je sa leve strane
- Indeksi bitova su proizvoljni
  -moraju biti >=0
- Primeri:

```
reg [7    : 0] BYTE;        8 bita
reg [1    :16] WORD;       16 bita
reg [63   :32] DWORD;      32 bita
reg [1023: 0] LFSR;        1k bita
```

- Napomena: koristiti format [N:0]

Pristupanje bitovima u vektorima

- Deklaracija:

```
reg [15:0] WORD;
```

- Može se pristupiti celom vektoru:

```
WORD
```

- Može se pristupiti pojedinačnom bitu:

```
WORD[4]
WORD[i]
WORD[row*4+col]
```

- Može se pristupiti uzastopnim bitovima:
  - ‘selekcija dela vektora’
  - Indeksi moraju biti konstante
  - Indeksi moraju da prate pravac deklaracije

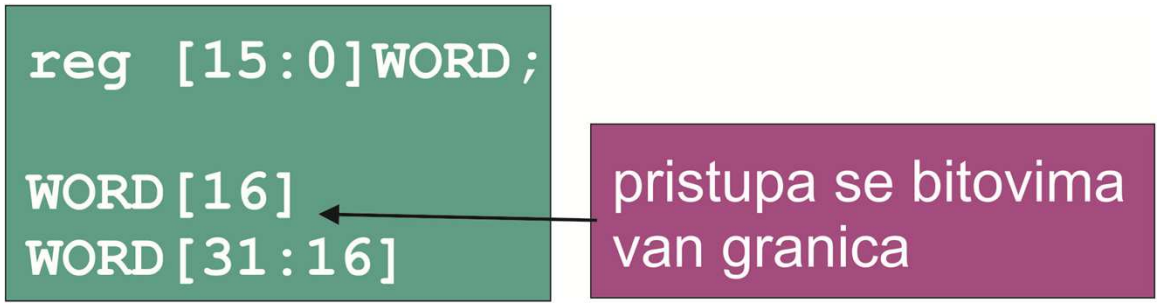
```
WORD[7:0]
WORD[15:14]
WORD[i+3:i]
WORD[0:15]
```

greška

Pristupanje bitovima u vektorima

- Ne postoje provere granica- moramo biti pažljivi sa indeksima!
  - Nema run-time ili grešaka u kompajliranju!

- Rezultat je nedefinisan
  - Nekad je ‘bx
  - Nekad je ‘b1
  - Nekad je ‘b0



Dodeljivanje vrednosti registrima

- Sintaksa:

```
<target> = <expression>;
```

- Target može biti:
  - Čitav registar
  - Bit vektora
  - Deo vektora
  - Konkatenacija svega prethodnog
    - Rezultat je desno poravnjan (right-justified)

```
WORD = <expr>;
WORD[i] = <expr>;
WORD[7:0] = <expr>;
```

```
{WORD[7:0], WORD[15:8]} = WORD;
```

zamena bitova

```
{OPCODE, {MODE, ADDR}, CPUREG} = INSTR;
```

dekodiranje instrukcije

  - ako je veličina instrukcije veća od izraza sa leve strane, biće desno poravnjana

Memorije

- Memorije su polja registara
  - Mogu biti deklarisane samo za reg, integer i time tipove podataka
  - Za real ne
- Imaju samo jednu dimenziju
- Sintaksa:

```
<type> <name> [<addr1>:<addr2>];
```

- Primeri:

```
reg          MEM1 [255:0];
reg [7:0]    MEM2 [16'hFFFF:0];
integer      MEM3 [5:7];
time         MEM4 [4:1];
```

Memorije

- Memorijama se pristupa tako što se pristupa jednoj lokaciji u jednom trenutku
- Nema vremenskih provera niti provera granica polja

Neka je:

```
reg [31:0]  RAM [0:255];
```

`RAM[0] = RAM[1] >> 4;`	OK
`RAM[0:1] = 64'hFFFFFFFFFFFFFFFF;`	nema deljenja mem
`{RAM[0], RAM[1]} = 64'h...`	OK
`RAM[256]`	!!!

Memorije

- Ne može se proizvoljno pristupiti bitovima u memoriji
- Mora se koristiti read-modify-write operator

Neka je:

```
reg [7:0]  RAM [0:255];
```

```
RAM[0][3:0] = 4'b0000;
```

Greška!!!
Pristupanje bitovima u mem

```
reg[7:0]TMP;
TMP = RAM[0];
TMP[3:0] = 4'b0000;
RAM[0] = TMP;
```

Mora se koristiti privremena promenljiva i read-modify-write

Veličina operacija

- Operacije i rezultati operacija imaju specifičnu veličinu
- Izrazi se procenjuju korišćenjem najveće veličine
- Operandi i međuvrednosti se odsecaju ili dopunjavaju nulama ako je to potrebno
  - nema proširivanja sa Z, X, ili znakom
- Mogu se dogoditi neočekivani rezultati

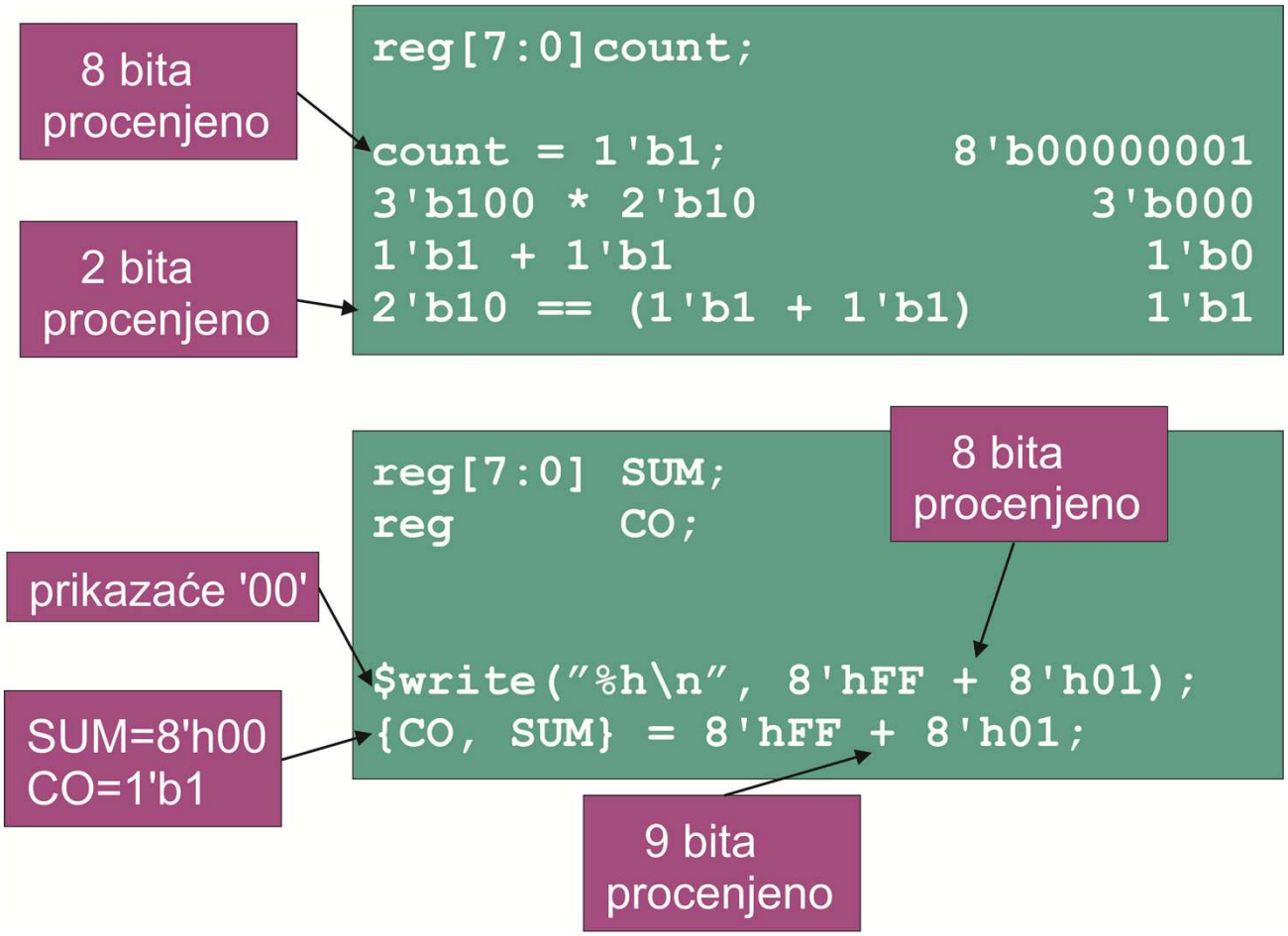
- Napomena: konstante neodređene veličine imaju po difoltu 32 bita

Veličina operacija

• i *op* j	+,-,*,/,%,&, |, ^,^~,~^	max(size(i), size(j))
• *op* i	-, ~	size(i)
• i *op* j	===, !===, ==, !=, &&, ||, >, >=, <, <=	1
• *op* i	&, |, ^	1
• i *op* j	<<, >>	size(i)
• i ? j : k		max(size(j), size(k))
• {i,…j}		size(i)+…+size(j)
• {i{j}}		i*size(j)
• i *assign* j	=, <=	size(i)

Veličina operacija

- Primeri:



Veličina operacija

- Pitanje: Koja je konačna vrednost ‘BUS’-a?

```
reg [63:0] BUS;

BUS = `bz
```

Veličina operacija

- Pitanje: Koja je konačna vrednost ‘BUS’-a?

```
reg [63:0] BUS;

BUS = `bz
```

- Odgovor:

```
`bz:                    32'hZZZZZZZZ
size(`bz):              32
size(BUS):              64
size(expr):             64
resize(`bz, 64):        64'h00000000ZZZZZZZZ
Konačna vrednost:       64'h00000000ZZZZZZZZ
```

$write iskaz

- Prikazuje formatirane vrednosti
- Format specificiran specifikatorima formata
- Specifikatori formata:

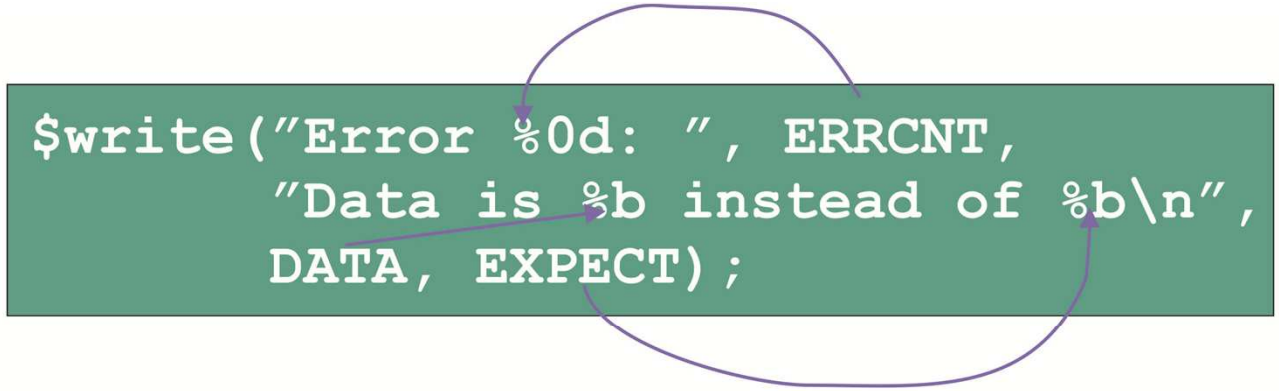
`%b`	`binarni`
`%o`	`oktalni`
`%d`	`decimalni`
`%h`	`heksadecimalni`
`%f`	`realni`

- Treba specificirati %0b, %0d, %0h, itd. da bi se prikazivali u minimalnoj širini polja
  - inače se koristi maksimalna zahtevana širina

$write iskaz

- Argument je sekvenca koja se sastoji od:
  - format stringa
  - podataka koji se formatiraju

- Primeri:




```
$write("Error %0d:Data is %b instead of %b\n",
       ERRCNT, DATA, EXPECT);
```

Iskazi za kontrolu toka

if iskaz

- Kontrola toka se zasniva na Bulovom izrazu

- *else* je opciono
- *else* se povezuje se najbližim *if* koje je bez *else*

- Sintaksa:

```
if (<expr>) <statement>
[else       <statement>]
```

if iskaz

- Primeri:

```
if (RST == 1'b1) Q = 1'b0;
```

```
if (RST == 1'b1) Q = 1'b0;
else             Q = D;
```

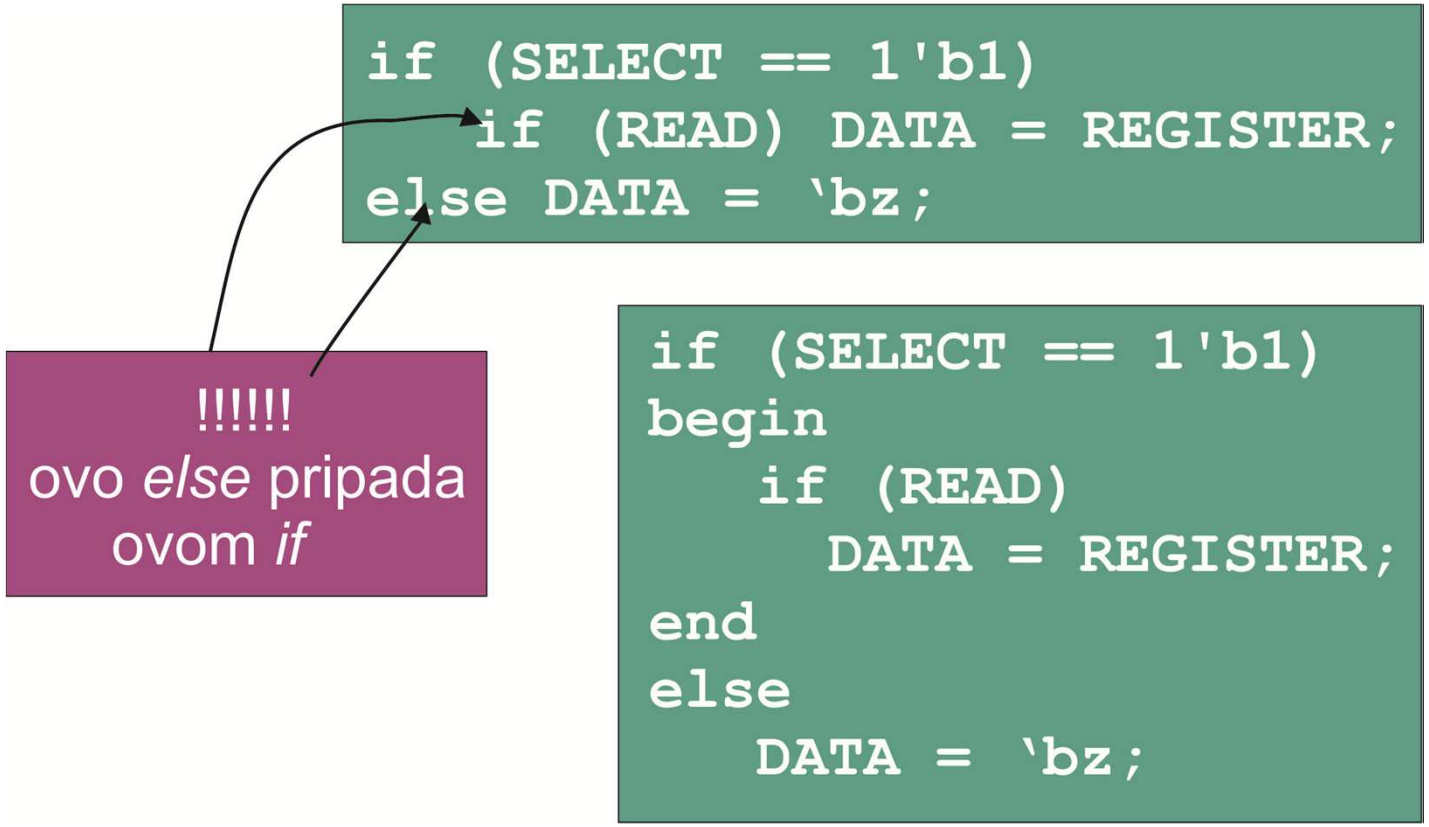
```
if (RST == 1'b1)       Q = 1'b0;
else if (SET == 1'b1)  Q = 1'b1;
else                   Q = D;
```

ovo *else* pripada ovom *if*

```
if (DATA !== EXPECT) begin
   $write("ERROR: unexpected data\n");
   ERRORS = ERRORS + 1;
end
```

if iskaz

- Napomena: koristiti *begin/end* da bi se odvojili delovi *if* iskaza



if iskaz

- Uputstva:
  - ne oslanjati se da je nešto što nije nula TRUE
  - uvek eksplicitno upoređivati
  - koristiti logičke operatore, ne bitwise operatore, kada se izvršavaju logičke operacije

```
if (~A & B & ~C) ...
```

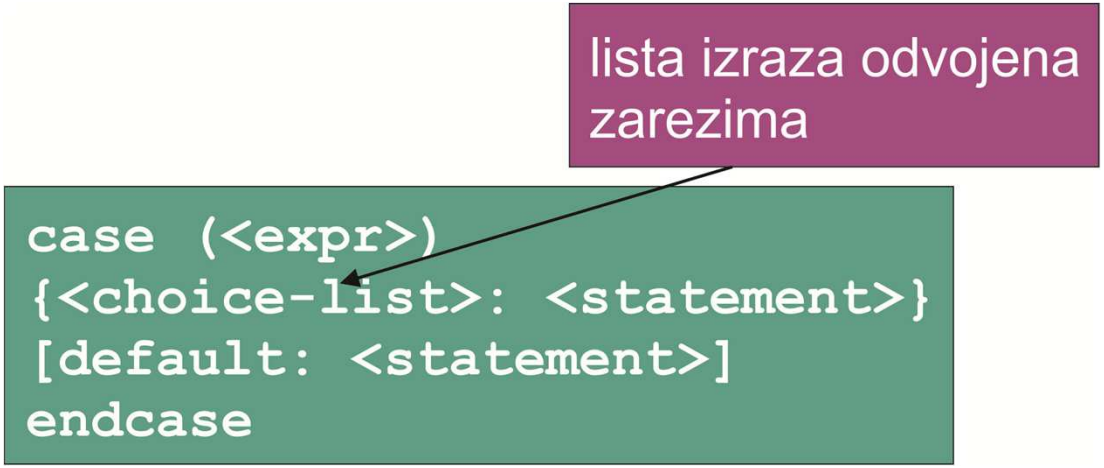
BOLJE →

```
if (A == 1'b0 && B == 1'b1 && C == 1'b0) ...
```

case iskaz

- Kontrola toka zasniva se na višestrukim izborima
  - bolje je nego dugi niz *if/else* iskaza
- *default* može biti specificiran kao ‘svašta’
- ako se ništa ne poklapa, *case* iskaz se ignoriše
  - ‘x’ se poklapa sa ‘x’, ‘z’ se poklapa sa ‘z’

- Sintaksa:



case iskaz

- Primeri:

1'b1 i 1'b0 biće ignorisani

```
case (BIT)
   1'bz: $write("High-impedance\n");
   1'bx: $write("Contention\n");
endcase
```

višestruki izbori

uhvatiće svaki *x* ili *z*

```
case (PARITY)
   2'b01: P = ODD;
   2'b00: P = EVEN;
   2'b10,
   2'b11: P = NONE;
   Default: $write("Invalid parity\n");
endcase
```

case iskaz

- case iskaz je prava zamena za *if/else*
- izvršava se prva opcija koja se poklapa (ispunjava uslov)
- ostale opcije se ignorišu
  - može da ima redundantne unose
  - izbori mogu biti izrazi

case iskaz

- Primeri:

```
case (BIT)
   1'bz: $write(”High-impedance\n”);
   1'bx: $write(”Contention\n”);
   1'b1: $write(”Asserted high\n”);
   1'b0: $write(”Asserted low\n”);
endcase
```

```
case (1'b1)
   RST:     Q = 1'b0;
   SET:     Q = 1'b1;
   default: Q = D;
endcase
```

ekvivalentno sa:

```
if (1'b1 === RST)       Q = 1'b0;
else if (1'b1 === SET)  Q = 1'b1;
else                    Q = D;
```

case iskaz

- Uputstva:
  - uvek koristiti *default*
  - uzeti u obzir sve moguće vrednosti selektora
  - ako nema šta da se izvrši, koristiti *null* iskaz

```
case (OPCODE)
2'b00: <statement>;
2'b01: <statement>;
2'b10: //invalid opcodes - ignore
2'b11: <statement>;
default: $write("Opcode has x's or z's\n");
endcase
```

casez iskaz

- Kao običan *case* iskaz
- Ako ima ‘z’ ili ‘?’ u selektoru, kao da ga nema

- Primer:

```
casez (OPCODE)
4'b0001: <statement1>
4'b001?: <statement2>
4'b01??: <statement3>
4'b1???: <statement4>
default: <statement5>
endcase
```

Koji iskaz će se izvršiti ako je OPCODE jednak:?

```
4'b0101
4'b0z01
4'b0?01
4'b0x01
```

3

1

1

5

casex iskaz

- Kao običan *case* iskaz
- Ako ima ‘z’, ‘?’ ili ‘x’ u selektoru, kao da ga nema

- Primer:

```
casez (OPCODE)
4'b0001: <statement1>
4'b001?: <statement2>
4'b01??: <statement3>
4'b1???: <statement4>
default: <statement5>
endcase
```

Koji iskaz će se izvršiti ako je OPCODE jednak:?

`4'b0101`	3
`4'b0z01`	1
`4'b0?01`	1
`4'b0x01`	1

Petlje

forever iskaz

- Ponavlja iskaz beskonačno
- Sintaksa:

```
forever <statement>
```

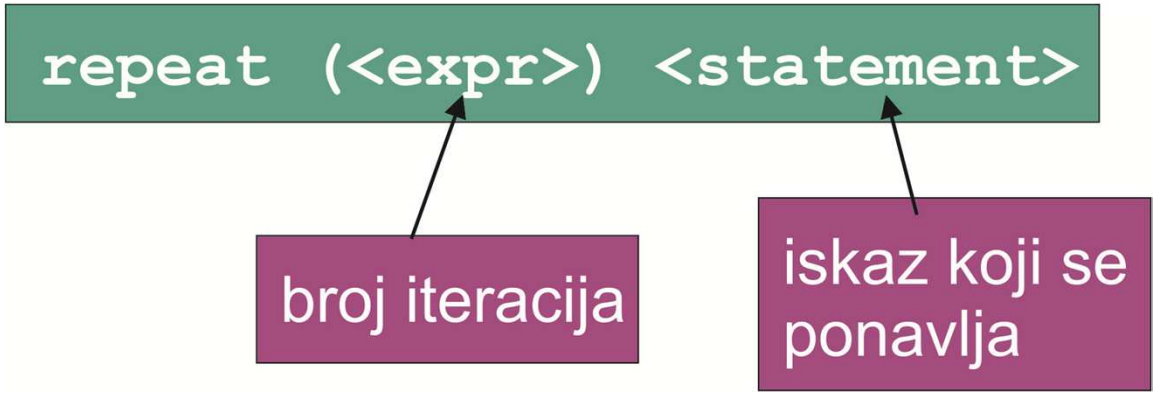
- Primer:

```
forever begin
  CLK = ~CLK;
end
```

repeat iskaz

- Ponavlja iskaz fiksni broj puta
- Broj ponavljanja je određen na početku petlje
  - ne može biti promenjen u toku iteracija

- Sintaksa:



repeat iskaz

- Primeri:

```
repeat (100) $write("Napolju je hladno\n");
```

```
i = 0;
repeat (16) begin
    WORDA[i] = WORDB[15 - i];
    i = i + 1;
end
```

while iskaz

- Ponavlja iskaz sve dok je izraz tačan
- Izraz se procenjuje na početku svake iteracije
- Upozorenje: može da napravi beskonačnu petlju

- Sintaksa:

```
while (<expr>) <statement>
```

while iskaz

- Primer:

```
i =0;
while (i < 16) begin
    WORDA[i] = WORDB[15 - i];
    i = i + 1;
end
```

- Pitanje: Šta se dešava ako je *i* deklarisano kao:

```
reg [3:0] i;
```

for iskaz

- Skraćena forma za neke *while* petlje
- Bolje je za brojače
- Sintaksa:

```
for (<reg-assign1>; <expr>; <reg-assign2>)
    <statement>
```

- Ekvivalentno sa:

```
<reg-assign1>;
while (<expr>) begin
    <statement>
    <reg-assign2>)
end
```

for iskaz

- Primer:

```
for (i = 0; i < 16; i = i + 1) begin
   WORDA[i] = WORDB[15 - i];
end
```

- Uputstvo: Koristiti *repeat* iskaz ukoliko se ne koristi iterator u telu petlje

```
for (i = 0; i < 4; i = i + 1) CLK = ~CLK;
```

BOLJE →

```
repeat (4) CLK = ~CLK;
```

Potprogrami

Funkcije

- Koriste se za enkapsuliranje računanja koja se ponavljaju
  - nema iskaza kašnjenja
- Moraju da vrate vrednost
  - mogu da vrate reg, integer, time ili real
  - vraćaju vrednost dodeljivanjem imenu funkcije
- Argumenti:
  - samo ulaz
  - samo jednobitni i višebitni registri
  - moraju da imaju bar jedan argument

Funkcije

- Sintaksa:

```
function [<type-or-bitrange>] <fctname>;
   {input [bitrange] <argname>;}
   [{<local-reg-decl>}]
   <statement>
endfunction
```

- Deklarisano u modulu:

```
module MODEL;
   ...
   function ...
      ...
   endfunction
   ...
endmodule
```

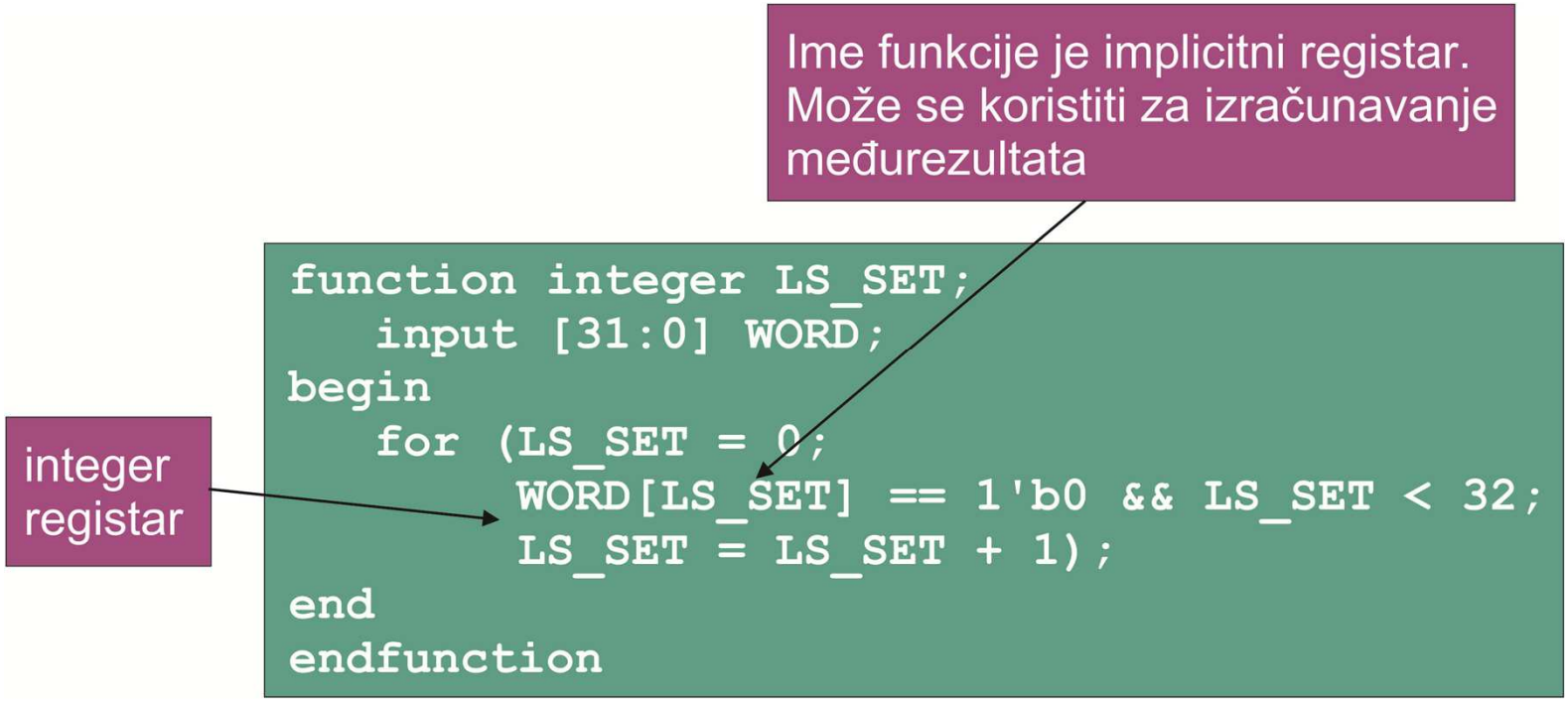
Funkcije

- Primer: Izračunati parnu i neparnu parnost 32-bitne reči

```
function PARITY;
    input [31:0] WORD;
    input        ODD;
begin
    if (ODD) PARITY = ~^WORD;
    else     PARITY = ^WORD;
end
endfunction
```

Funkcije

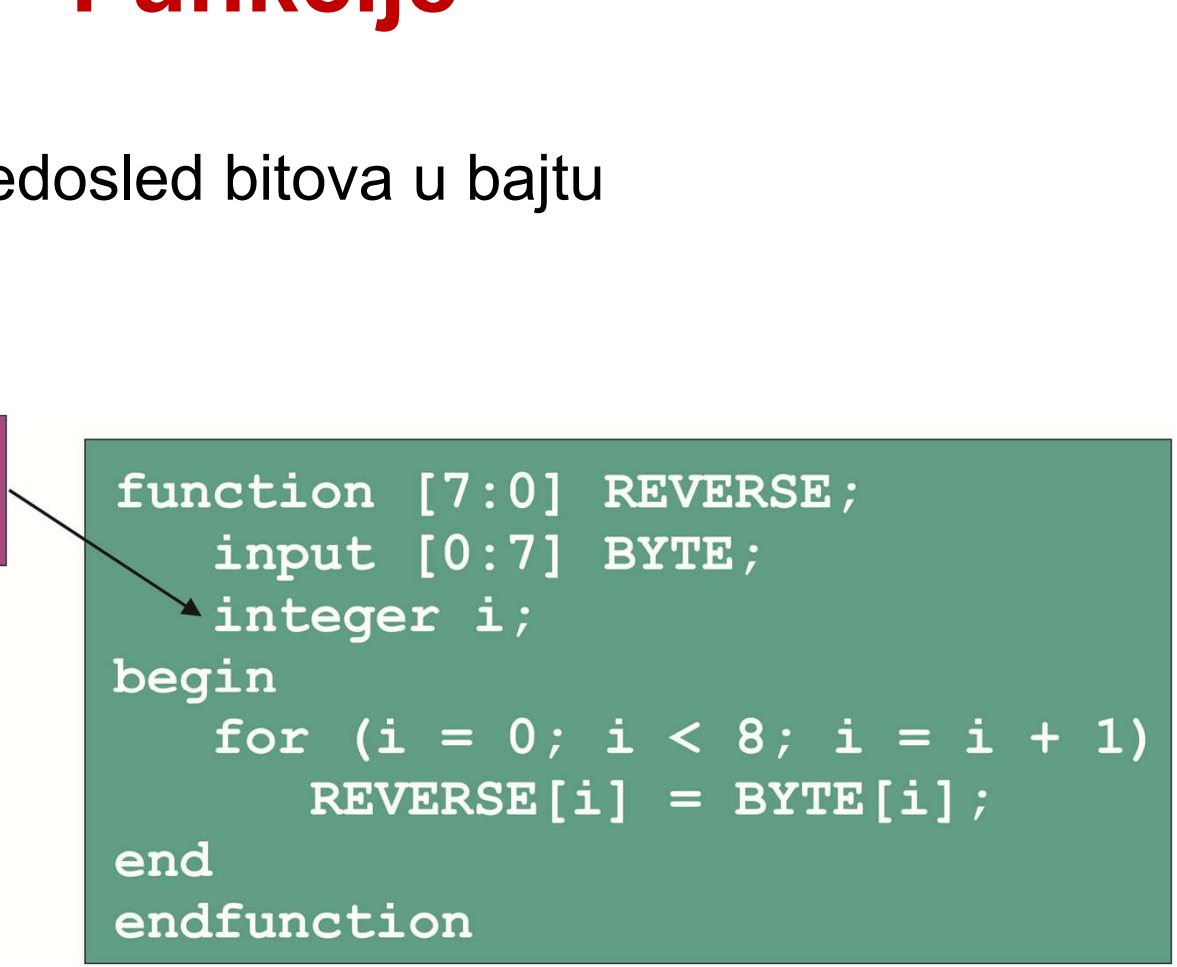
- Primer: Naći indeks bita najmanje težine



Funkcije

- Primer: Obrnuti redosled bitova u bajtu

mogu da se deklarišu lokalni registri



```
function [7:0] REVERSE;
   input [0:7] BYTE;
   integer i;
begin
   for (i = 0; i < 8; i = i + 1)
      REVERSE[i] = BYTE[i];
end
endfunction
```

Pozivanje funkcija

- Funkcije se pozivaju u izrazima
- Postaju novi operatori
- Primeri:

```
if (PARITY(DATA, 1'b1) !== DATA_IN) begin
   $write("Parity error!\n");
end
```

```
DATA = REVERSE(DATA);
```

Zadaci - Tasks

- Koriste se za enkapsulaciju iskaza koji se ponavljaju
- Ne mogu da vrate vrednost
- Zadaci se koriste samo u bihevioralnom kodu
- Argumenti:
  - input, output, inout
  - jednobitni i višebitni registri
    - ne integer, time i real

Zadaci - Tasks

- Sintaksa:

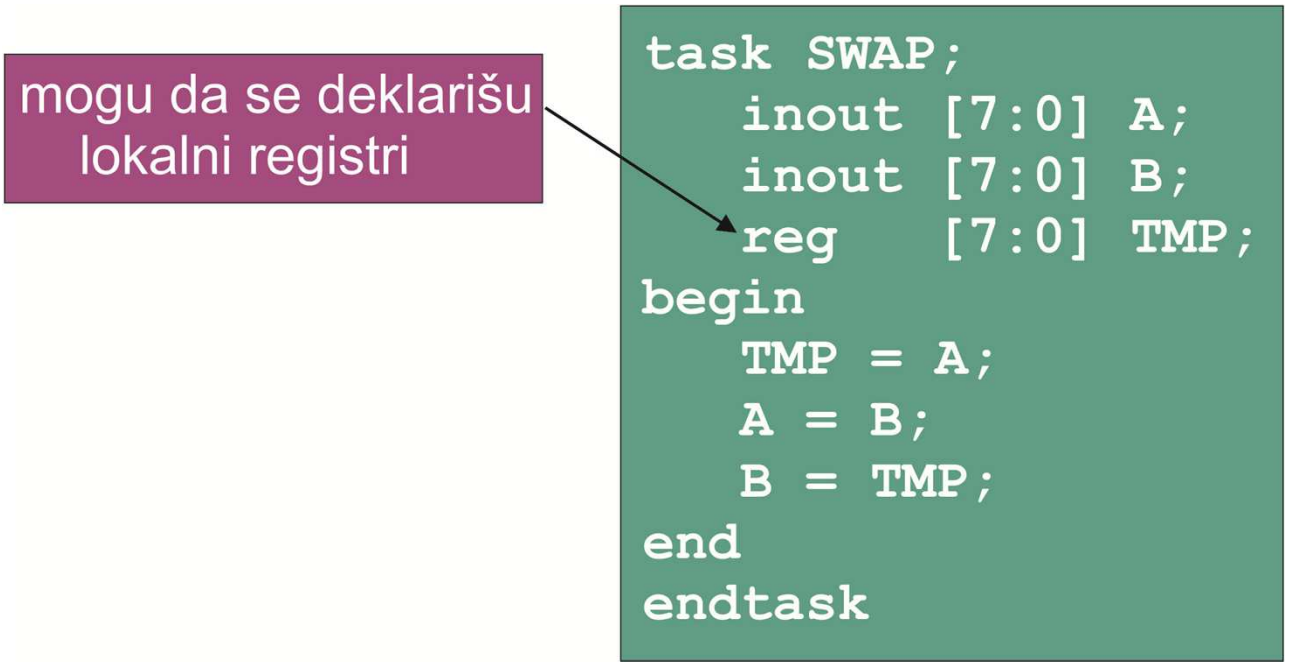
```
task <taskname>;
   {<direction> [bitrange] <argname>;}
   [{<local-reg-decl>}]
   <statement>
endfunction
```

- Deklarisano u modulu:

```
module MODEL;
   ...
   task ...
      ...
   endtask
   ...
endmodule
```

Zadaci - Tasks

- Primer: Zameniti mesta dva bita



Pozivanje zadataka

- Zadaci se pozivaju kao iskaz
- Postaju novi iskazi
- Primer:

```
$write("%h\n", WORD);
SWAP(WORD[15:8], WORD[7:0]);
$write("%h\n", WORD);
```