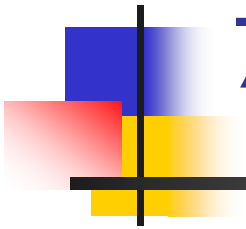
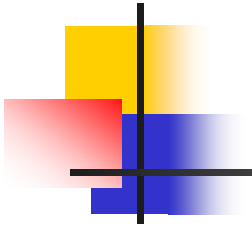


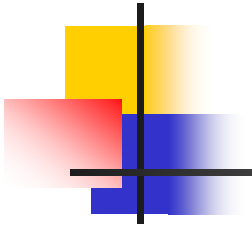
7. Standardni kombinacijski moduli

- kombinacijski moduli
- dekoder i demultipleksor
- multipleksor
- prioritetni koder
- pretvornik koda
- komparator
- opis u jeziku VHDL



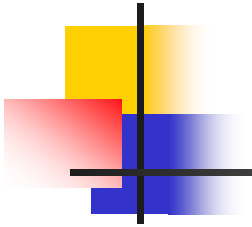
Kombinacijski moduli

- dekompozicija sustava:
  - identifikacija češće korištenih podsustava/sklopova ~ *moduli*
  - *kombinacijski* moduli:
    - izlazi = f(ulazi)
    - ostvarivanje *složenije* (od I, ILI, NE) funkcije
  - tipične izvedbe:
    - MSI i LSI
    - čipovi/dijelovi čipa



Kombinacijski moduli

- općenita podjela kombinacijskih modula:
  - specijalni:
    - ciljano projektirani za zadani sustav
    - *optimalna* izvedba
  - standardni:
    - "opće namjene" (engl. general purpose)
    - proizvodnja u velikim serijama
      ~ niska cijena
    - široko korištene funkcije
  - univerzalni
    ~ ostvarivanje proizvoljne Booleove funkcije



Kombinacijski moduli

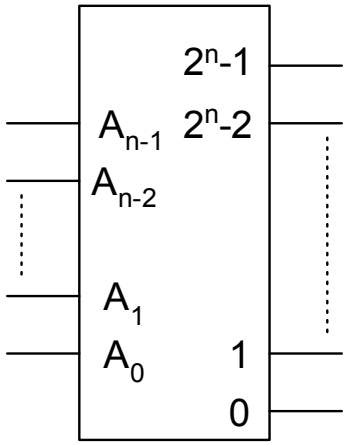
- ostvarivanje složenijih kombinacijskih funkcija:
  - dekoderi i pretvornici koda
  - sklopovi za odabir podataka
  - koderi
  - komparatori
  - aritmetički moduli

Dekoder

- funkcija *dekodiranja*
  ~ identificiranje kodne riječi nekog koda
- *dekoder*
  ~ aktivan *samo jedan* izlaz,
  onaj koji "odgovara" narinutoj kodnoj riječi

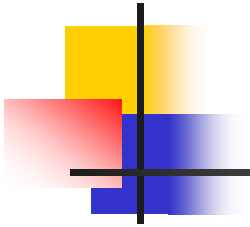
  n ulaza $\rightarrow 2^n$ izlaza

$$Z_i = \begin{cases} 1 & \text{za} \quad i = A_{n-1} \ldots A_0 \quad (\wedge E = 1) \\ 0 & \text{za} \quad i \neq A_{n-1} \ldots A_0 \quad (\vee E = 0) \end{cases}$$



- tipična oznaka:

  <broj adresa>/2<broj adresa>



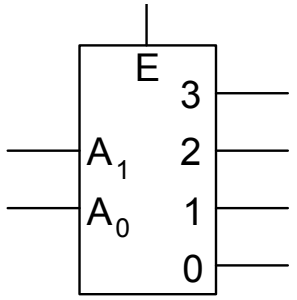
Dekoder

- podjela dekodera:
  - *binarni* dekoderi:
    - n = 2, 3, 4, ... ulaza $\rightarrow$ "1-od-2^n" izlaza
    - usmjeravanje informacije na jedan od izlaza
  - *dekadski* dekoderi:
    - n = 4 ulaza $\rightarrow$ "1-od-10" izlaza
    - dekodiranje binarnih kodova za prikaz dekadskih znamenki npr. BCD, XS-3

Binarni dekoder

Primjer: binarni dekoder 2/4

$$\overline{A}_1\overline{A}_0 \equiv 00_2 = 0_{10}$$
$$\overline{A}_1 A_0 \equiv 01_2 = 1_{10}$$
$$A_1\overline{A}_0 \equiv 10_2 = 2_{10}$$
$$A_1 A_0 \equiv 11_2 = 3_{10}$$



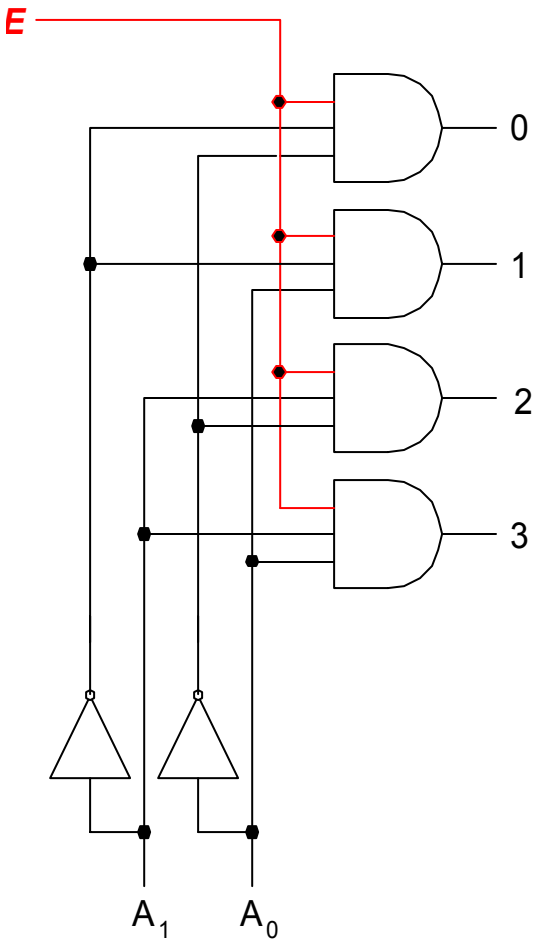
- upravljanje sklopom
  ~ *ulaz za omogućavanje*
  E (engl. enable)

E	A_1	A_0	"0"	"1"	"2"	"3"
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Binarni dekoder

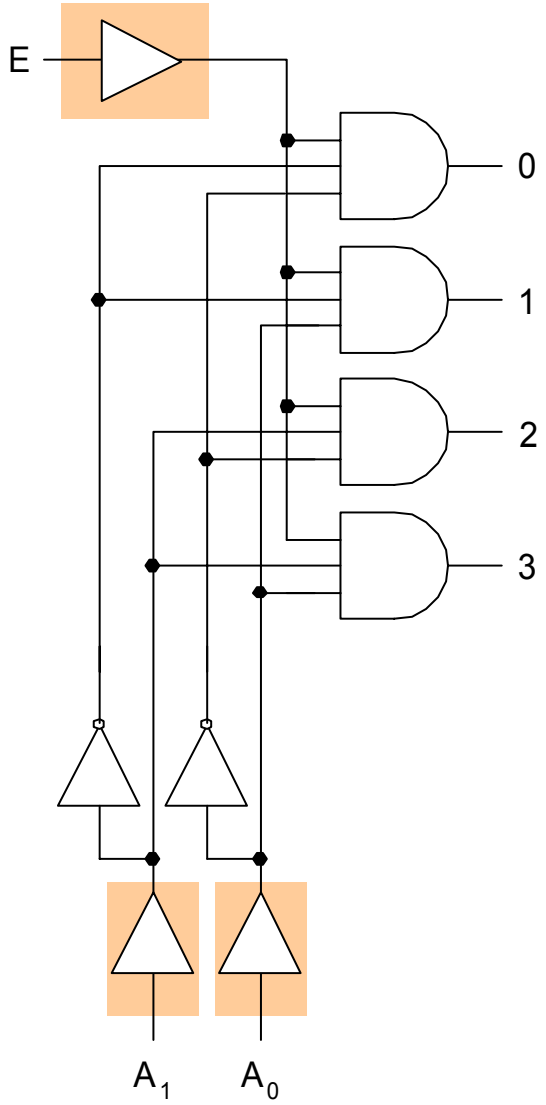
- *struktura* dekodera 2/4:

$$\overline{A}_1\overline{A}_0 \equiv 00_2 = 0_{10}$$
$$\overline{A}_1 A_0 \equiv 01_2 = 1_{10}$$
$$A_1\overline{A}_0 \equiv 10_2 = 2_{10}$$
$$A_1 A_0 \equiv 11_2 = 3_{10}$$



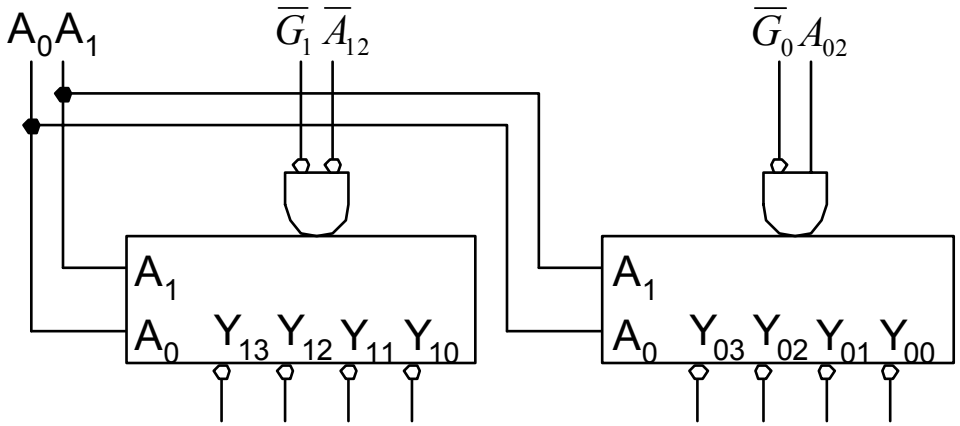
Binarni dekoder

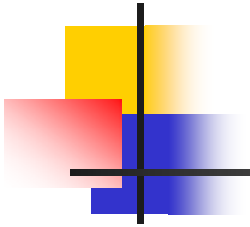
- *stvarna* izvedba:
  - ograničiti opterećenje ulazima
  - odvojni sklopovi ~ *jediničnom* opterećenju
  - funkcijski ekvivalentna izvedba



Binarni dekoder

- izvedbe dekodera
  ~ MSI modul; npr. 74155 (dvostruki četveroizlazni)
  - zajedničke adrese A_1, A_0
  - oznake:
    $\overline{G}_1, \overline{G}_0$: ulazi za omogućavanje
  - dekoder 3-bitnih riječi: $A_{02} = \overline{A}_{12} = A_2$





Binarni dekoder

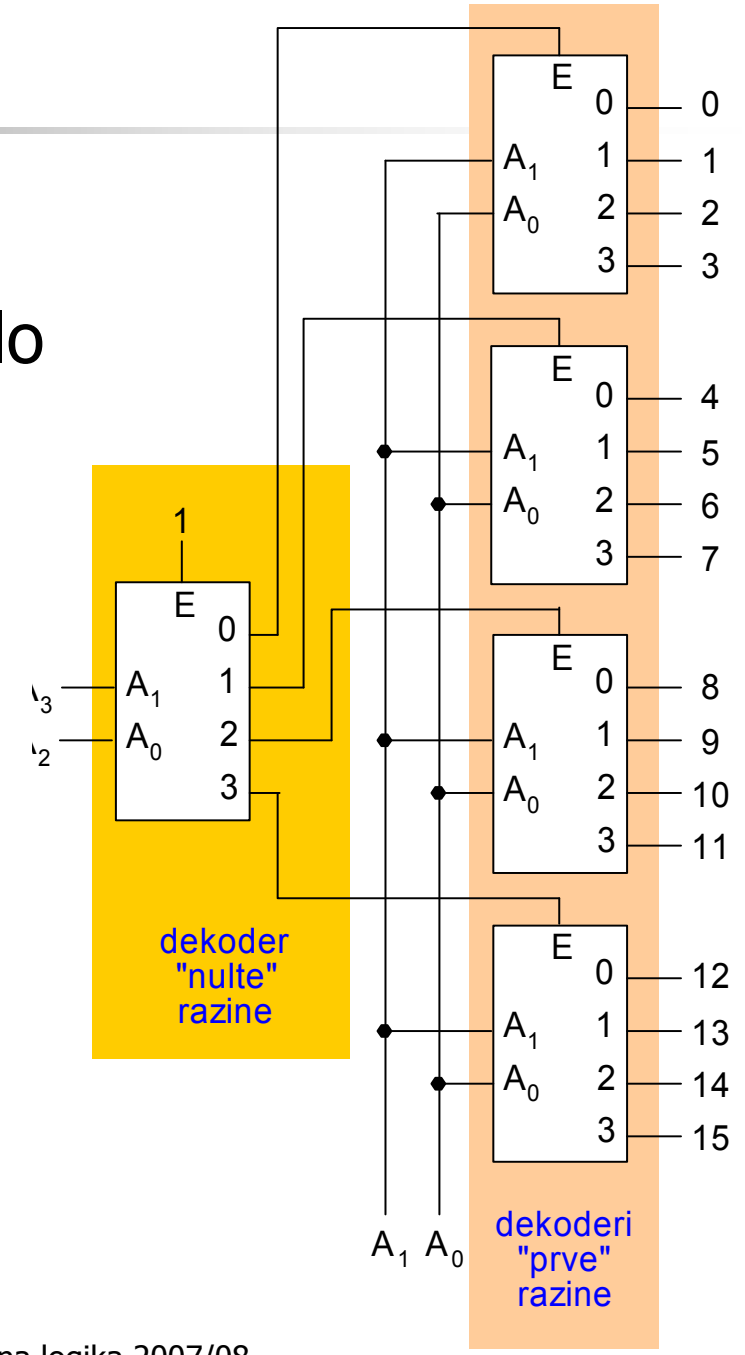
- dekoderi s većim (n > 16) brojem izlaza:
  - izvedba jednim modulom *nepraktična*
    ~ presloženi MSI modul
  - radije *kaskadiranje*
    ~ *dekodersko stablo* (engl. decoder tree):
    - općenita metoda
    - vrijedi za proizvoljno složeni modul;
      npr. izvedba dekodera sklopovima I

Binarni dekoder

Primjer: dekoder 4/16
kao dekodersko stablo

A_3	A_2	A_1	A_0	
0	0	0	0	D_0^1
		0	1	
		1	0	
		1	1	
0	1	0	0	D_1^1
		0	1	
		1	0	
		1	1	
1	0	0	0	D_2^1
		0	1	
		1	0	
		1	1	
1	1	0	0	D_3^1
		0	1	
		1	0	
		1	1	

D^0



Ostvarivanje funkcija dekoderom

- zapažanje:
  - izlazi dekodera = *potpuno dekodirane* kodne riječi ~ *mintermi*

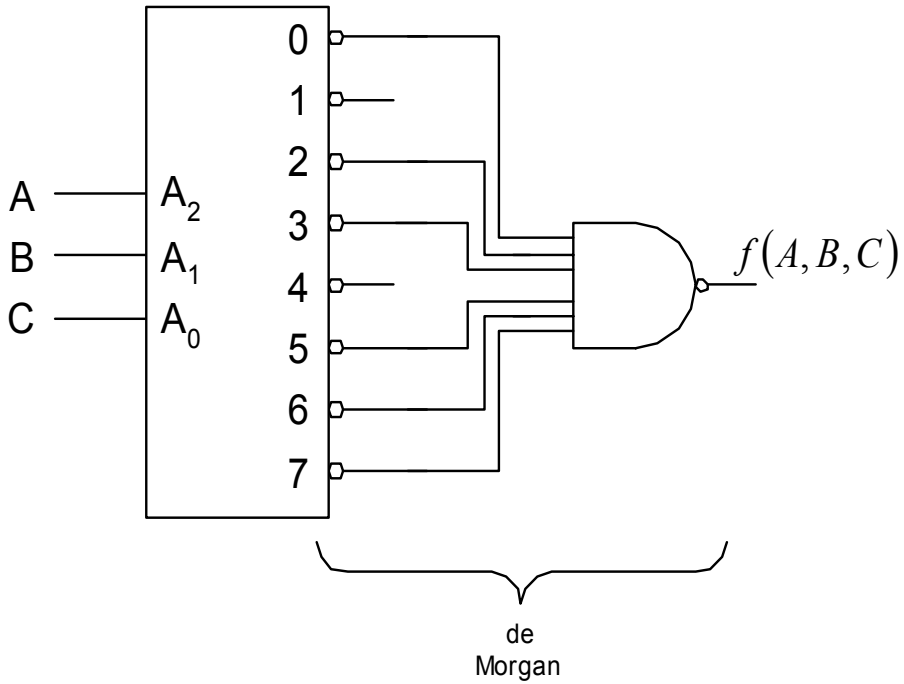
$$\overline{A}_1\overline{A}_0 \equiv 00_2 = 0_{10}$$
$$\overline{A}_1 A_0 \equiv 01_2 = 1_{10}$$
$$A_1\overline{A}_0 \equiv 10_2 = 2_{10}$$
$$A_1 A_0 \equiv 11_2 = 3_{10}$$

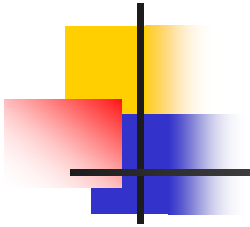
- *ostvarivanje logičkih funkcija* dekoderom:
  - funkcija u kanonskom *disjunktivnom* obliku
  - "pokupiti" (funkcija ILI) izlaze koji odgovaraju mintermima zastupljenim u definiciji funkcije

Ostvarivanje funkcija dekoderom

Primjer: $f(A,B,C)=\sum m(0,2,3,5,6,7)$

- f(A, B, C) $\rightarrow$ dekoder 3/8
- uočiti: ILI°I = (NI°NE)°I





Dekadski dekoder

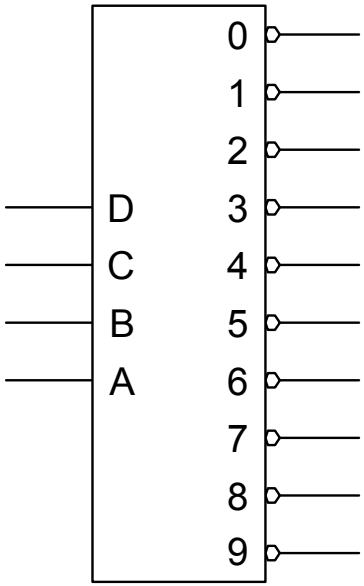
- dekodiranje koda s $N < 2^n$
  ~ "nepotpuno" dekodiranje
  - broj izlaza $< 2^{\text{broj ulaza}}$
  - tipični slučaj
    ~ dekodiranje binarno kodiranih dekadskih znamenki
    - BCD
    - XS-3
    - 2421
    - itd.

Dekadski dekoder

Primjer: BCD-dekadski dekoder 7442

- oznake: D ~ najviša težina (2^3)
- izlazi invertirani:
  zgodno kod kombiniranja sklopova
- uzorci 0 i 1 koji nisu kodne riječi:
  - ne prepoznaju se!
  - minimizacija sklopa
- varijante:

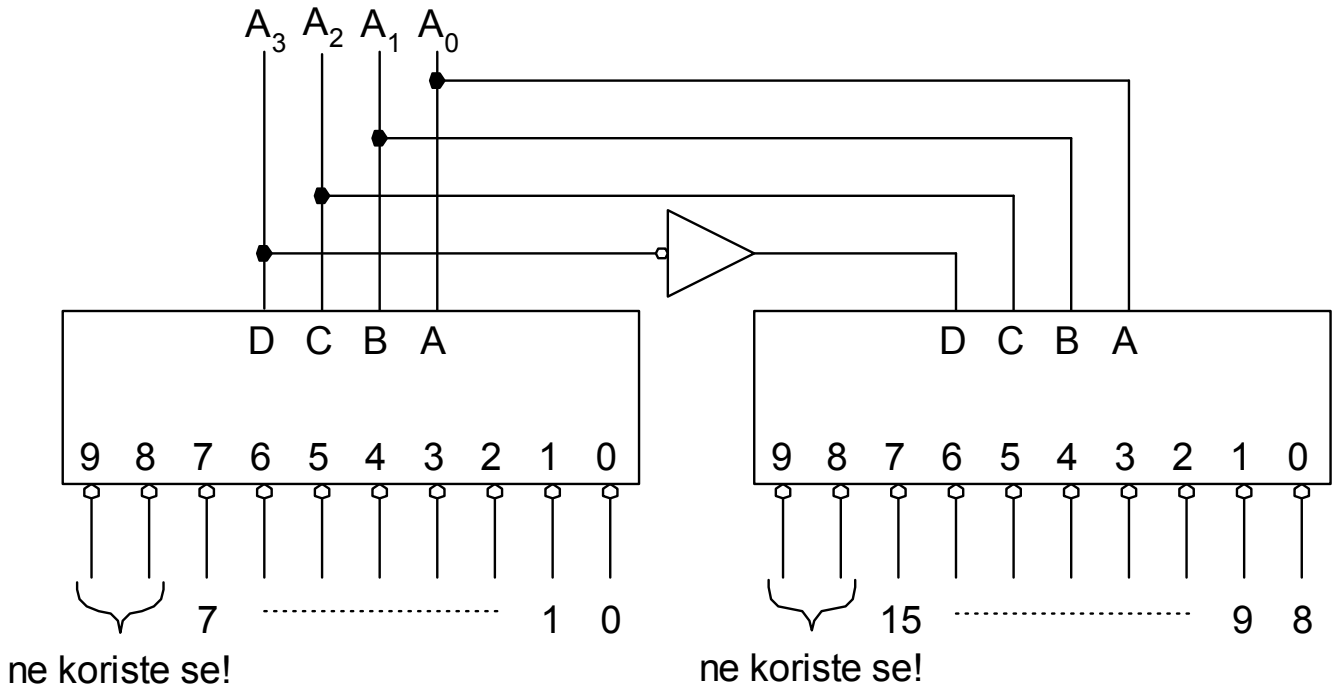
  7443: XS-3
  7444: XS-3 Gray



Dekadski dekoder

Primjer: izvedba dekodera "1-od-16" od dva 7442 :

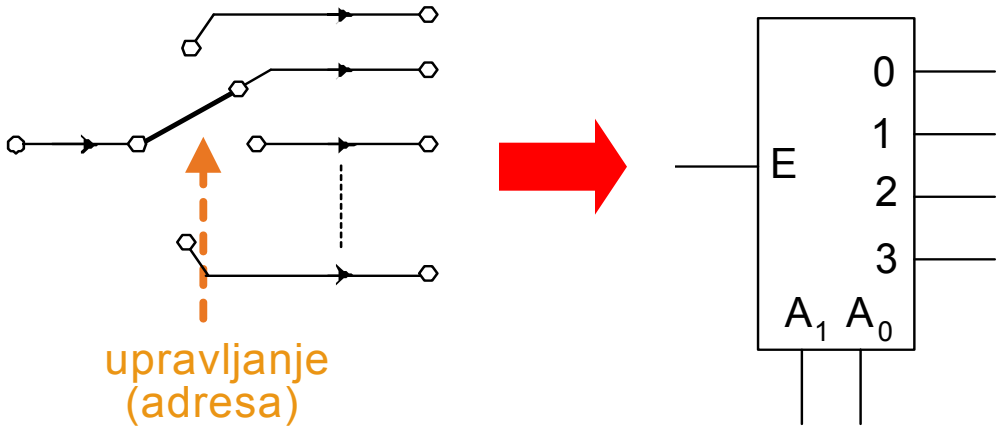
- D : odabir jednog od dva dekodera 1/8



Demultipleksor

- demultipleksor:
  - ulaz za omogućavanje dekodera
    ~ funkcija *ulaza za podatke*
  - "usmjeravanje"/"raspodjela" ulaza na odabrani izlaz
    ~ "demultipleksiranje"

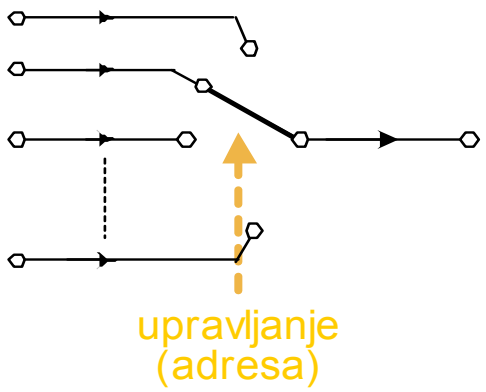
A_1	A_0	"0"	"1"	"2"	"3"
0	0	E	0	0	0
0	1	0	E	0	0
1	0	0	0	E	0
1	1	0	0	0	E

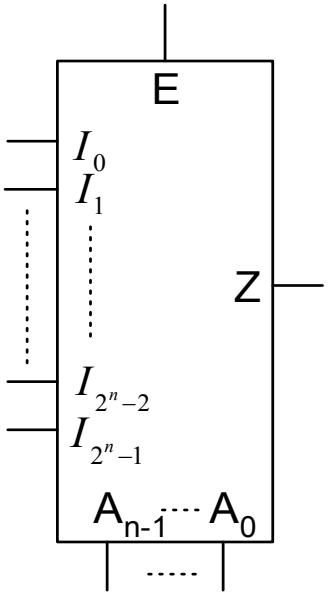


Multipleksor

- multipleksor:
  - odabir podataka
    ~ "multipleksiranje"
  - funkcija *upravljane preklopke*:

$$Z_i = \begin{cases} I_i & \text{za} \quad i = A_{n-1} \ldots A_0 \quad (\wedge E = 1) \\ 0 & \text{za} \quad E = 0 \end{cases}$$



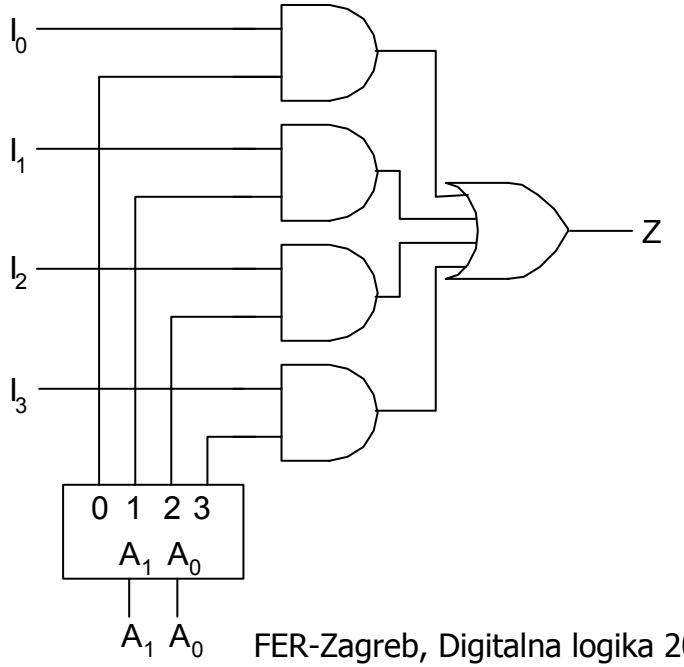


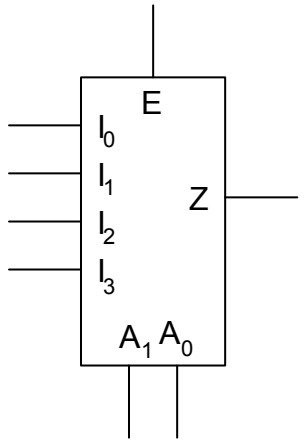
Multipleksor

Primjer: multipleksor 4/1

- adrese se dekodiraju
- izlazi iz dekodera *koincidiraju* s ulazima I_i ~ propuštaju *samo jedan* od ulaza na izlazni ILI sklop

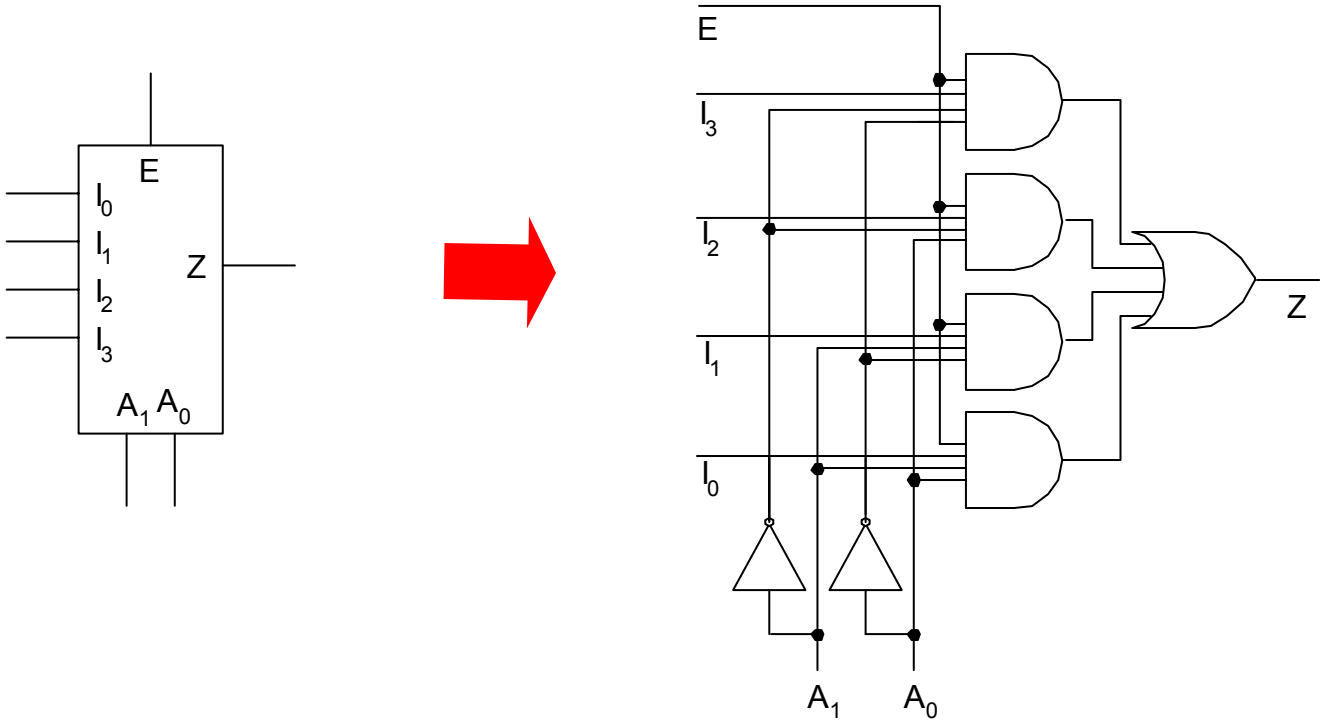
E	A_1	A_0	Z
0	X	X	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3





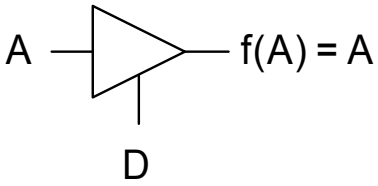
Multipleksor

- funkcija I je *asocijativna*
  ~ "grupirati" I sklopove iz dekodera i koncidenciju

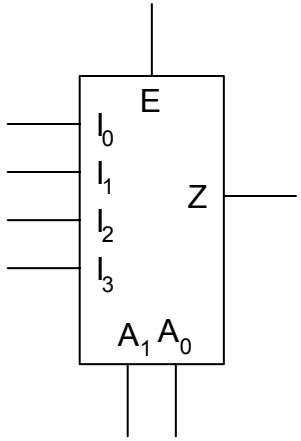


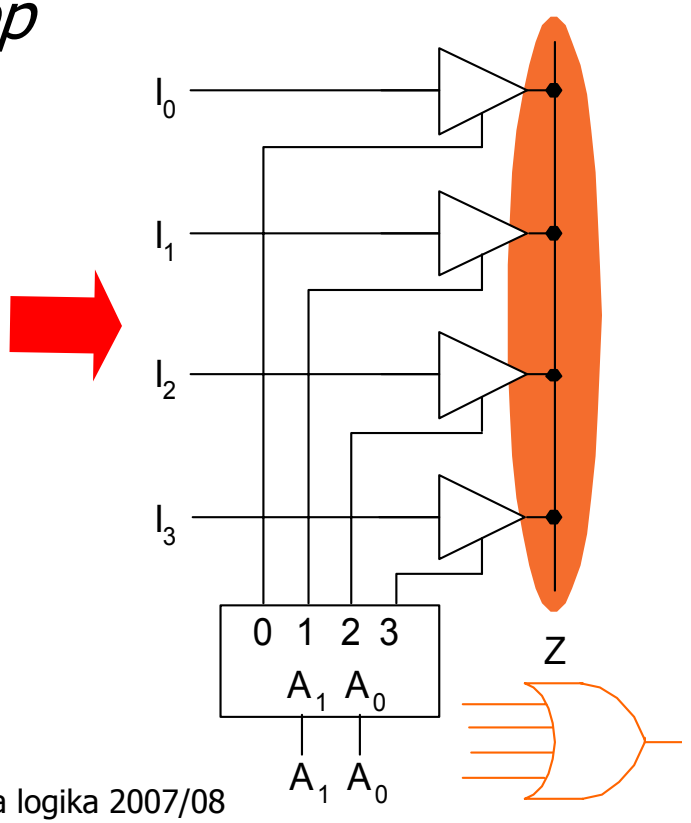
Multipleksor

- izvedba funkcije multipleksiranja *na sabirničkoj liniji*:
  - izlazni ILI → "upravljani spojeni ILI"
    ~ *samo jedan* sklop definira vrijednost V/N
  - *sklopovi s tri stanja* upravljani izlazima iz dekodera
  - (upravljani) *odvojni sklop*



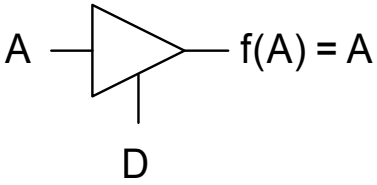
D	A	f
0	X	Z
1	0	0
1	1	1

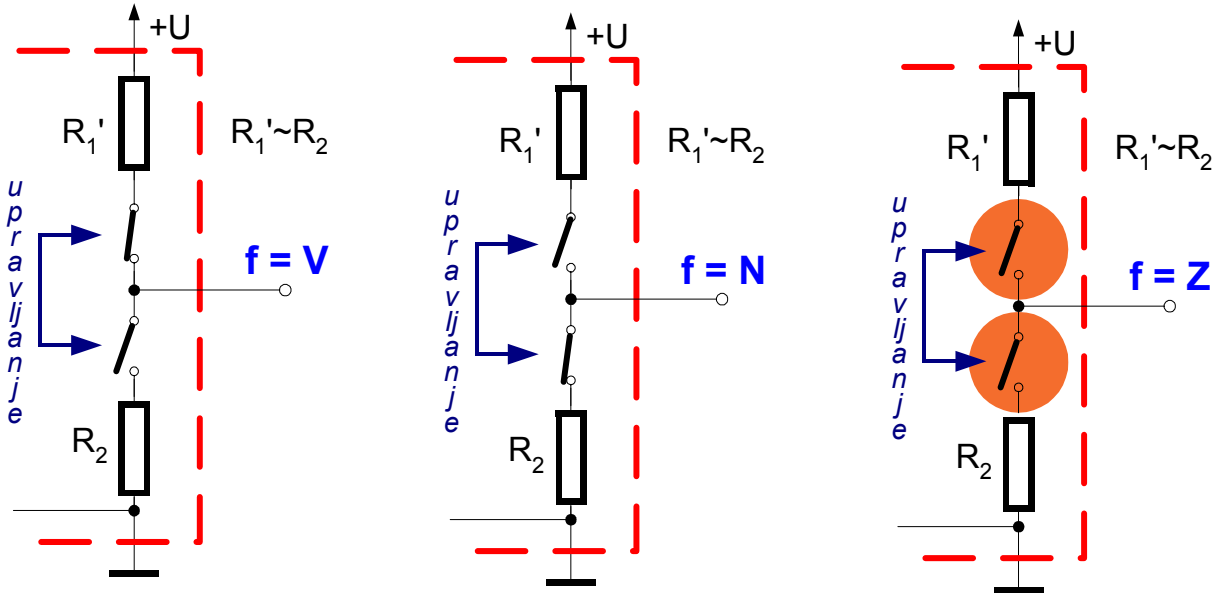




Multipleksor

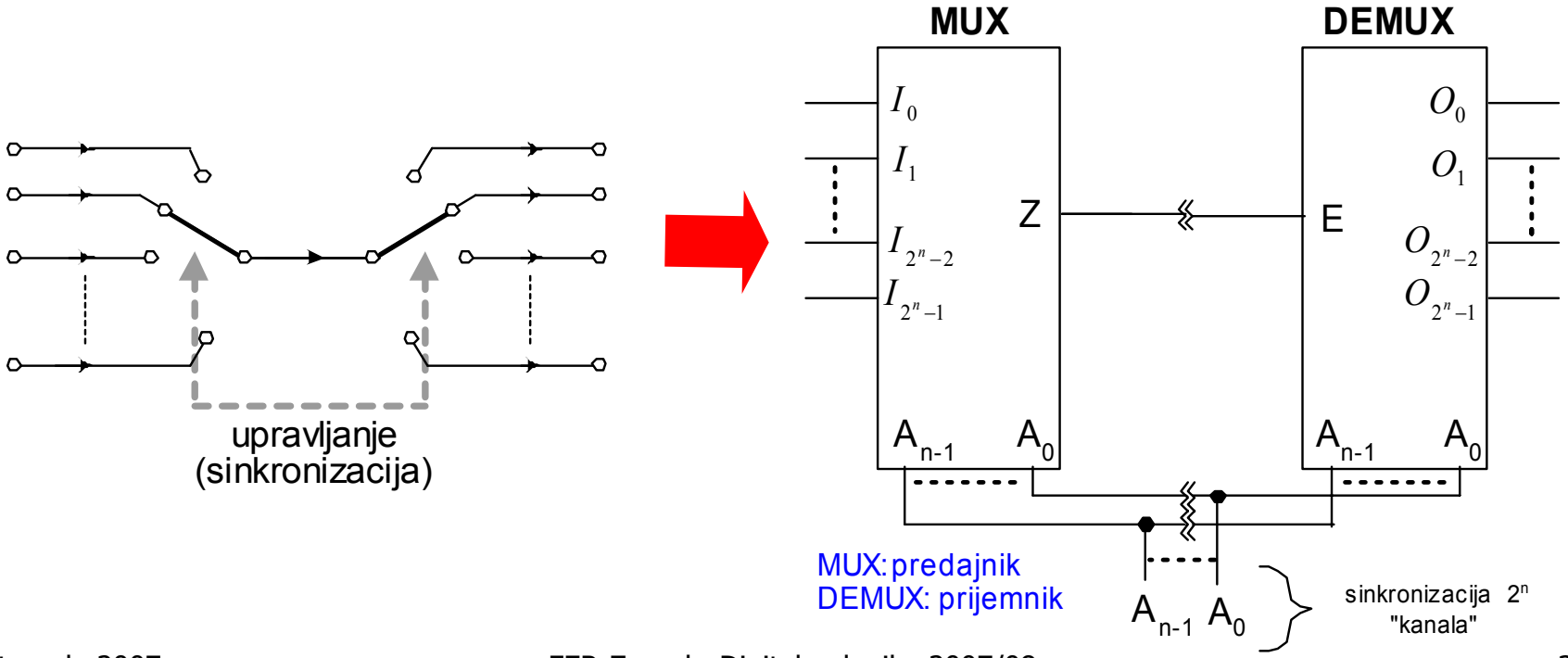
- (upravljani) odvojni sklop → izlaz *s tri stanja*
  - visoko (V)
  - nisko (N)
  - "stanje visoke impedancije" (Z)
    ~ obje izlazne sklopke *isključene* :
    nema pritezanja ni prema V, a niti prema N





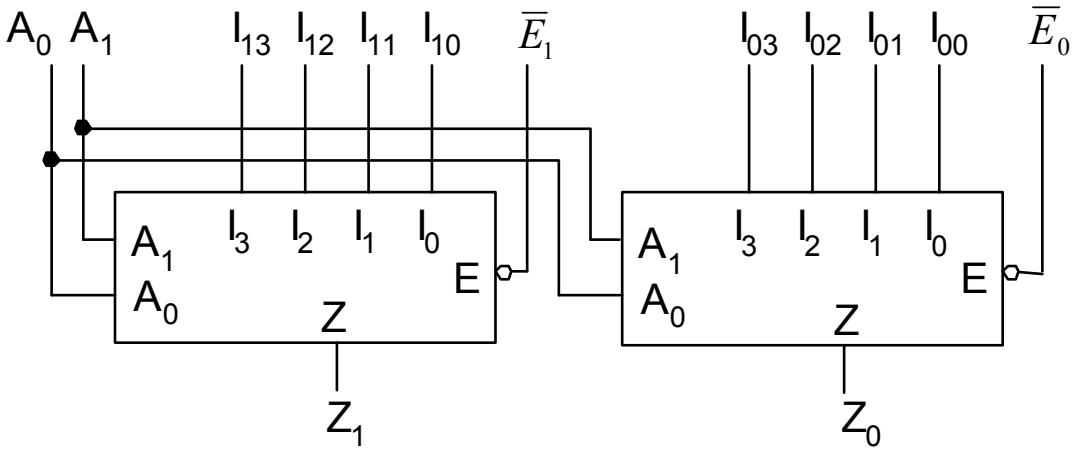
Multipleksor

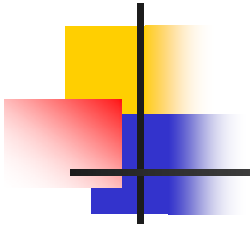
- funkcija multipleksiranja:
  ~ višestruko iskorištenje spojnih puteva
  - prijenos različitih podataka *istim* fizičkim spojnim putem
    ~ "više logičkih kanala preko jedne fizičke linije"
  - vremenska podjela (vremenski multipleks)



Multipleksor

- izvedbe multipleksora
  ~ MSI modul; npr. 74153 (dvostruki četveroulazni)
  - zajedničke adrese A_1, A_0
  - *izdvojeni* ulazi za omogućavanje: $\overline{E_1}, \overline{E_0}$



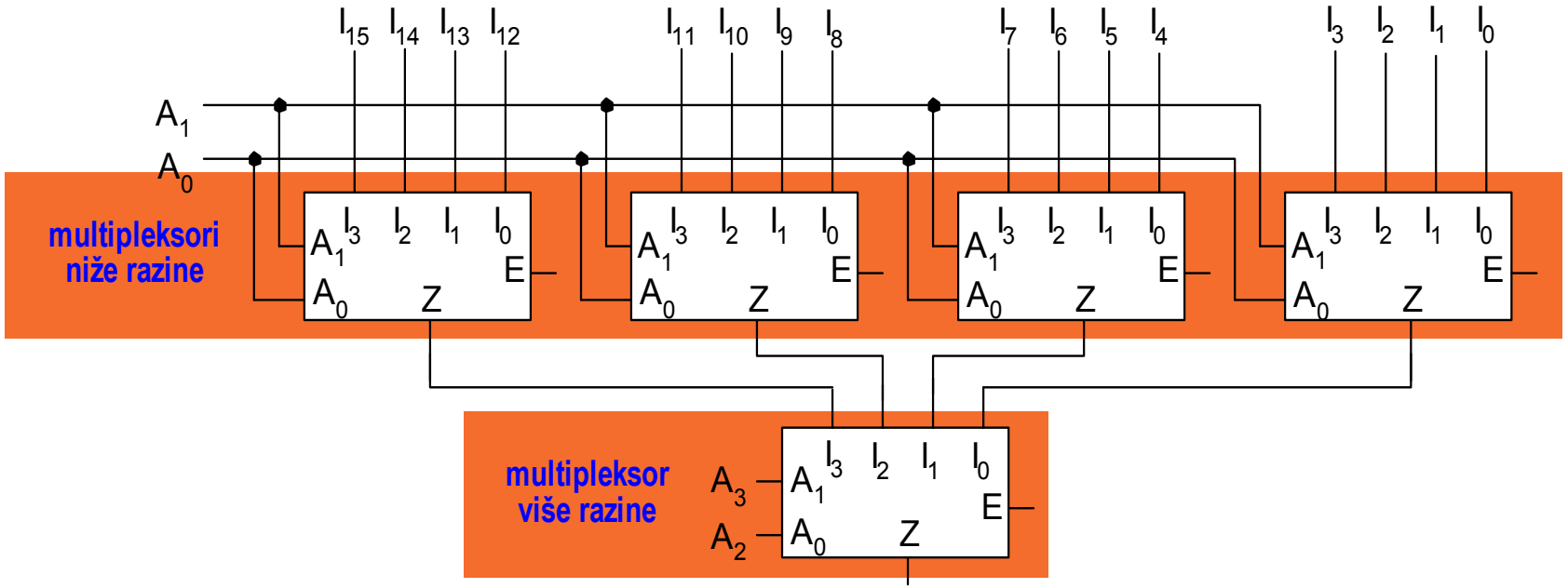


Multipleksor

- multipleksori s većim (n > 16) brojem ulaza:
  - izvedba jednim modulom *nepraktična*
    ~ presloženi MSI modul
  - radije *kaskadiranje*
    ~ *multipleksorsko stablo* (engl. multiplexer tree)
  - izgradnja multipleksorskog stabla:
    - podjela tablice definicije funkcija u podtablice
      ~ ulazi u MUX više razine
    - varijable viših težina na MUX "više razine"

Multipleksor

Primjer: multipleksor 16/1
kao multipleksorsko stablo



Ostvarivanje funkcija multipleksorom

- ostvarivanje logičkih funkcija multipleksorom:
  - funkcija multipleksiranja:
    m_i: minterm predstavlja adresu

$$Z = \sum_{i=0}^{2^n-1} I_i \cdot m_i$$

  - definicija funkcije od n varijabli u kanonskom disjunktivnom obliku :

$$f(x_{n-1},\ldots,x_0) = \sum_{i=0}^{2^n-1} \alpha_i \cdot m_i$$

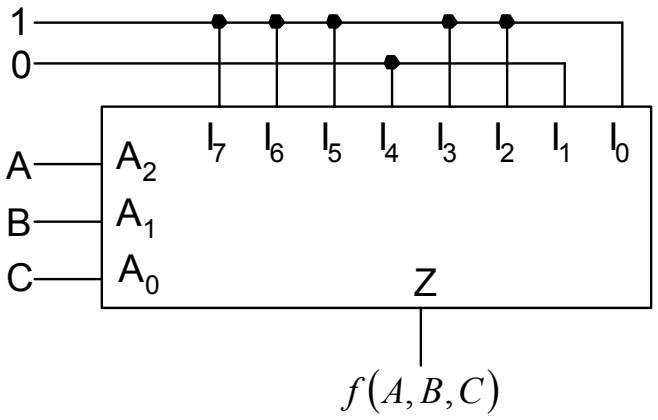
  - pridruživanje: $\forall A_i = x_i, I_i = \alpha_i \Rightarrow Z = f(x_{n-1},\ldots,x_0)$

Ostvarivanje funkcija multipleksorom

Primjer: ostvarivanje funkcije tri varijable

$$f(A,B,C)=\sum m(0,2,3,5,6,7)$$

- "simulacija rada permanentne memorije (ROM)"
- neefikasno! ($\forall$ m_i $\exists$ I_i)



Ostvarivanje funkcija multipleksorom

- rješenje *trivijalnim rezidualnim funkcijama*
  ~ efikasnije rješenje:
  - za $f(x_{n-1},\ldots,x_0)$ MUX s n-1 adresa
    ~ 2^{n-1} "informacijskih" ulaza
  - na ulaze MUX dovoditi funkcije varijable *najmanje* težine:

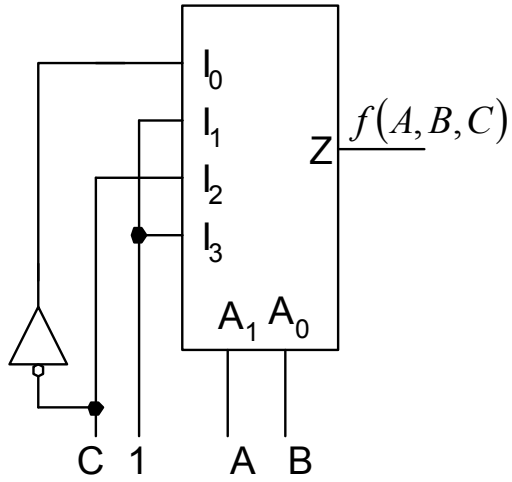
    $$\varphi(x_0)=\{0,1,x_0,\bar{x}_0\}$$

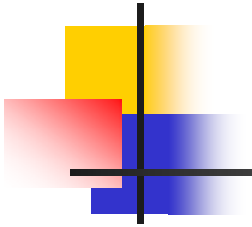
  - 2^{n-1} informacijskih ulaza
    ~ 2^{n-1} funkcija ostatka, *rezidualnih funkcija*
  - rezidualne funkcije od jedne varijable
    ~ *trivijalne* rezidualne funkcije

Ostvarivanje funkcija multipleksorom

Primjer: $f(A,B,C)=\sum m(0,2,3,5,6,7)$

A_1 A	A_0 B	C	ADRESIRANI ULAZ	f	
0	0	0	I_0	1	$\overline{C}$
0	0	1		0	
0	1	0	I_1	1	1
0	1	1		1	
1	0	0	I_2	0	C
1	0	1		1	
1	1	0	I_3	1	1
1	1	1		1	





Ostvarivanje funkcija multipleksorom

- rješenje *netrivijalnim* rezidualnim funkcijama:
  - "netrivijalne" rezidualne funkcije (>1 varijable)
    ~ obično presloženo rješenje
  - (također) kanonski oblik funkcije
    ~ *nema* minimizacije
  - pojednostavljivanje rješenja
    ~ odabir prikladnog pridruživanja varijabli adresnim ulazima

Ostvarivanje funkcija multipleksorom

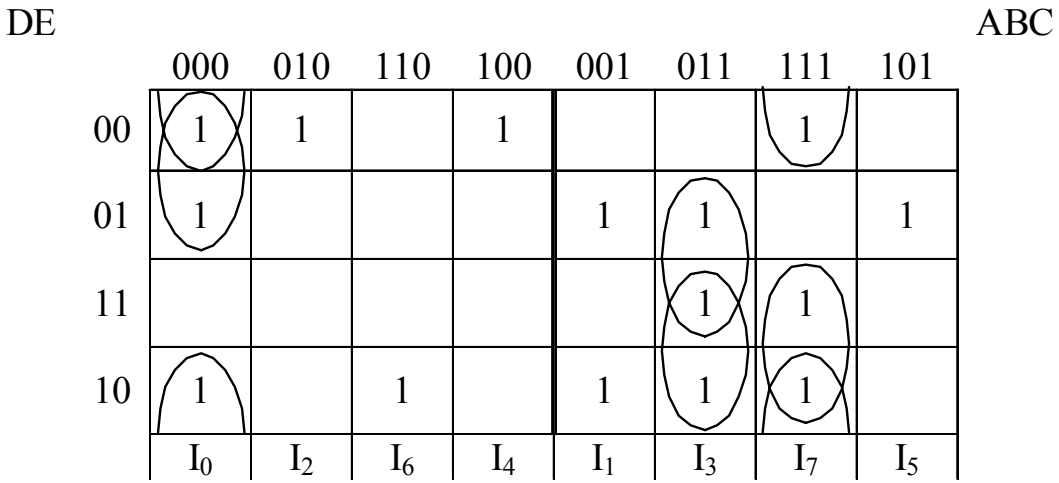
Primjer: $f(A,B,C,D,E) = \sum m(0,1,2,5,6,8,13,14,15,16,21,26,28,30,31)$

- ostvarenje 16-ulaznim MUX
  ~ standardno rješenje
  trivijalnim rezidualnim funkcijama
- ostvarenje 8-ulaznim MUX
  ~ rezidualne funkcije od 2 varijable

Ostvarivanje funkcija multipleksorom

- ostvarenje 8-ulaznim MUX
  ~ rezidualne funkcije od 2 varijable

$$f(A,B,C,D,E) = \sum m(0,1,2,5,6,8,13,14,15,16,21,26,28,30,31)$$

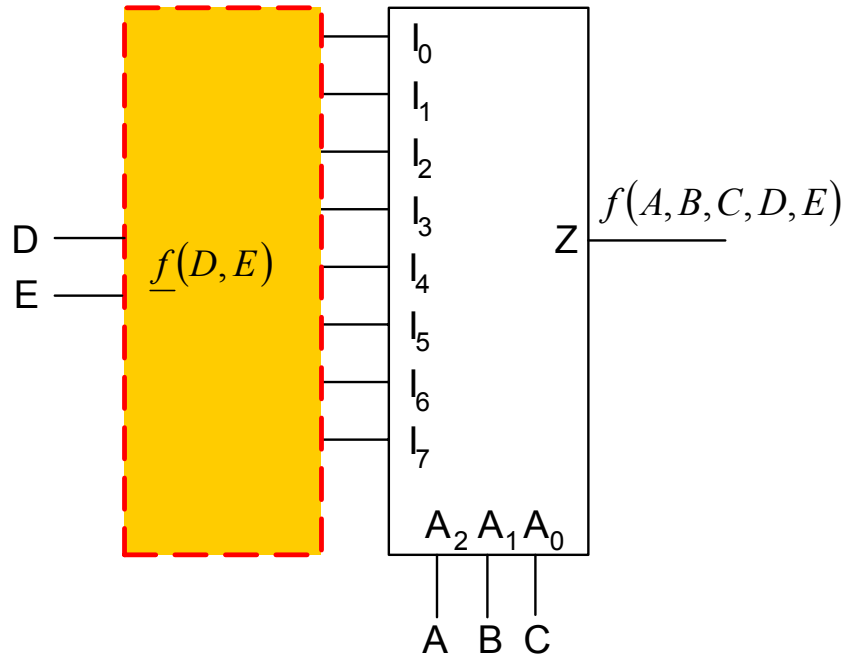


Ostvarivanje funkcija multipleksorom

- ostvarenje 8-ulaznim MUX
  ~ rezidualne funkcije od 2 varijable: $ABC \rightarrow A_2 A_1 A_0$

$$f(A,B,C,D,E) = \sum m(0,1,2,5,6,8,13,14,15,16,21,26,28,30,31)$$

i	A	B	C	$f_{res}=I_i$
	A_2	A_1	A_0	
0	0	0	0	$\overline{DE}$
1	0	0	1	$D \oplus E$
2	0	1	0	$\overline{D}\overline{E}$
3	0	1	1	$D + E$
4	1	0	0	$\overline{D}\overline{E}$
5	1	0	1	$\overline{D}E$
6	1	1	0	$D\overline{E}$
7	1	1	1	$D + \overline{E}$

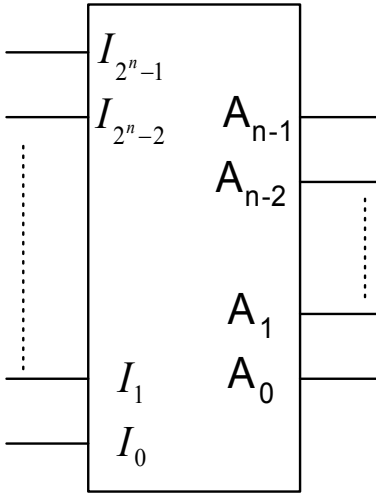


Prioritetni koder

- funkcija *kodiranja*
  ~ generiranje *binarne* kodne riječi nekog koda
- *koder*
  ~ aktivan samo jedan ulaz (npr. $I_i = 1$)

  2^n ulaza $\rightarrow$ n izlaza
- tipična oznaka:

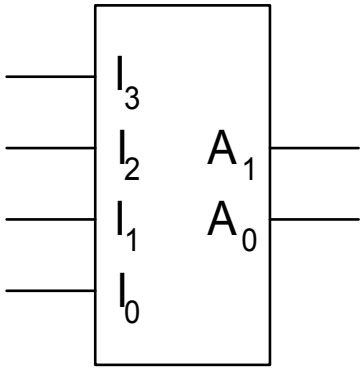
  2<broj adresa>/<broj adresa>



Prioritetni koder

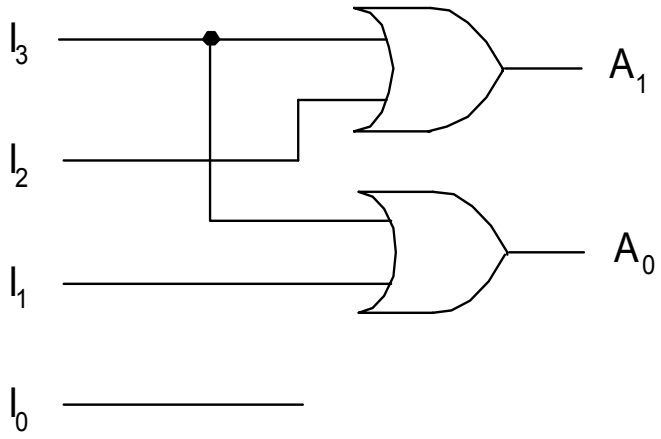
Primjer: koder 4/2

- ograničenje: uzorci ulaza s više 1 *ne mogu* se pojaviti
- simbol "0" (~ I_0) daje $A_1A_0 = 00$ ~ ne utječe!



I_3	I_2	I_1	I_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$A_1 = I_3 + I_2$$
$$A_0 = I_3 + I_1$$

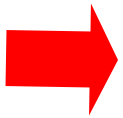


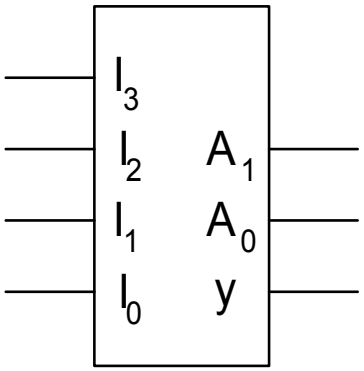
Prioritetni koder

- *prioritetni koder* (engl. priority encoder)
  ~ rješenje problema više aktivnih ulaza
  - djeluje ulaz najvišeg prioriteta
  - svi ulazi = 0 ?
    ~ poseban izlaz:

    y = **if** I_3 **or** I_2 **or** I_1 **or** I_0 **then** 1 **else** 0

I_3	I_2	I_1	I_0	A_1	A_0	y
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

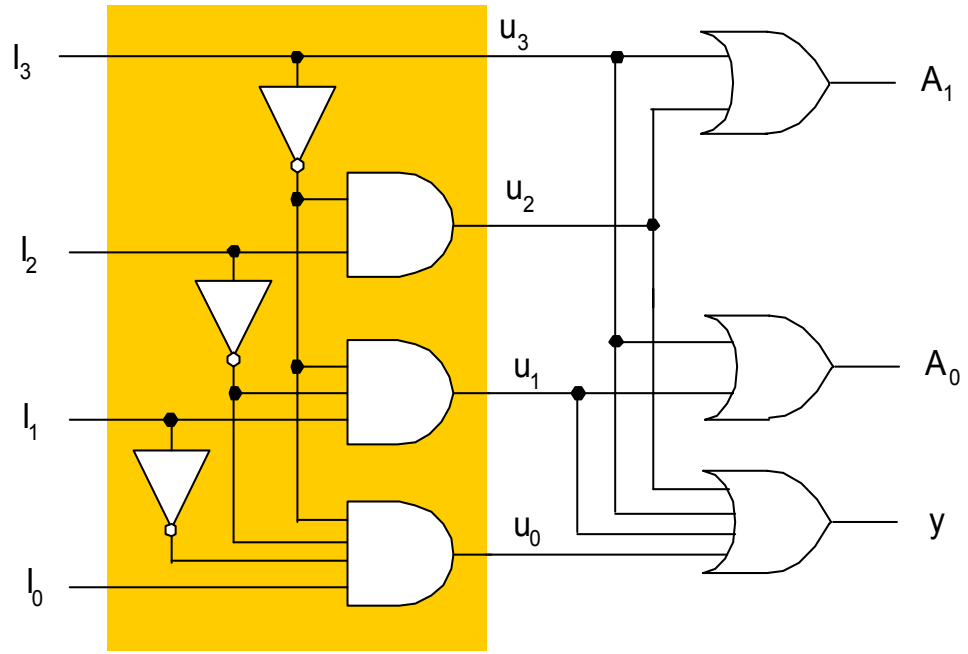




Prioritetni koder

Primjer: prioritetni koder 4/2 [Brown i Vranešić, 2000]

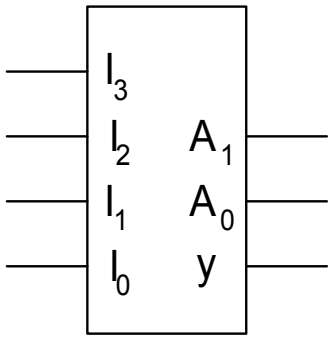
I_3	I_2	I_1	I_0	A_1	A_0	y
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

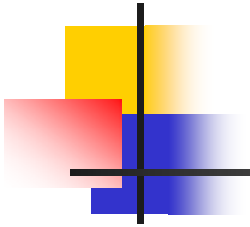


$$u_3 = I_3$$
$$u_2 = \overline{I_3} \cdot I_2$$
$$u_1 = \overline{I_3} \cdot \overline{I_2} \cdot I_1$$
$$u_0 = \overline{I_3} \cdot \overline{I_2} \cdot \overline{I_1} \cdot I_0$$
$$y = u_3 + u_2 + u_1 + u_0$$

$$\Rightarrow$$

$$A_1 = u_3 + u_2$$
$$A_0 = u_3 + u_1$$





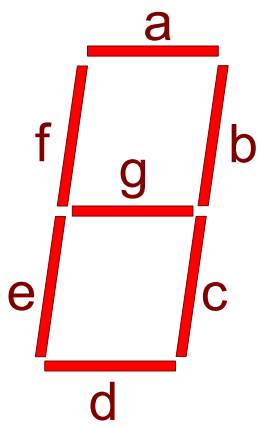
Pretvornik koda

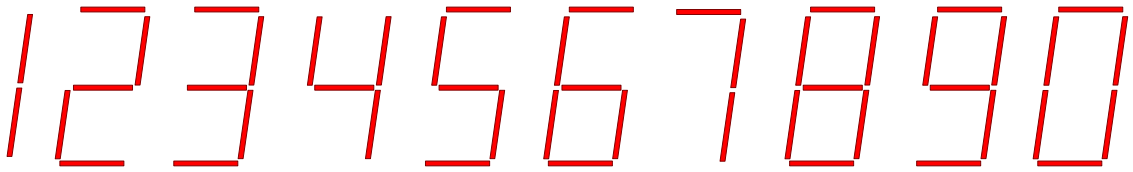
- *pretvornik koda* (engl. code converter):
  - pretvorba kodnih riječi *dvaju različitih* kodova
  - isti princip kao kod dekodera i kodera:
    - dekoder
      ~ kodna riječ $\rightarrow$ 1 aktivni izlaz
    - koder
      ~ 1 aktivni ulaz $\rightarrow$ kodna riječ
  - različiti tipovi MSI modula
    + mogućnost *kaskadiranja* (2-dimenzijske strukture)

Pretvornik koda

Primjer: pretvornik BCD koda u 7-segmentni

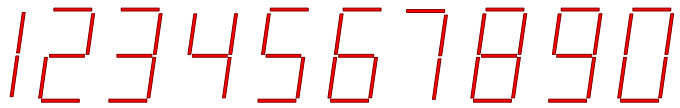
- vrlo raširena primjena
  ~ prikaz BCD znamenki
- element za prikaz
  ~ *7-segmentni prikaz*
  (engl. 7-segment display)

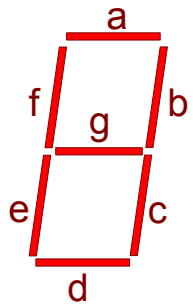




Pretvornik koda

- tablica pretvorbe BCD u 7-segmentni kod

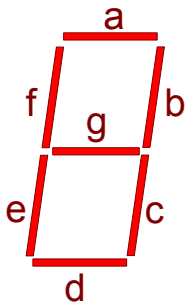


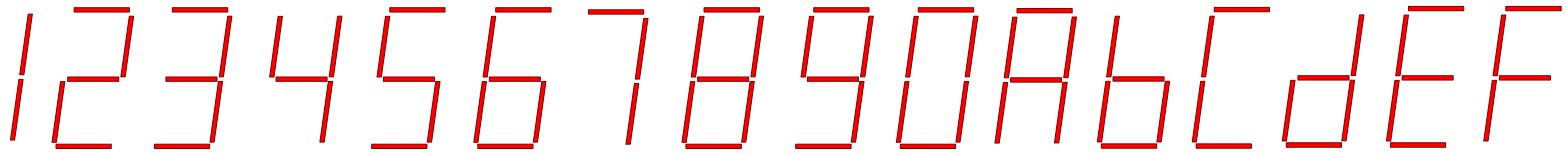


	D_3 2^3 8	D_2 2^2 4	D_1 2^1 2	D_0 2^0 1	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Pretvornik koda

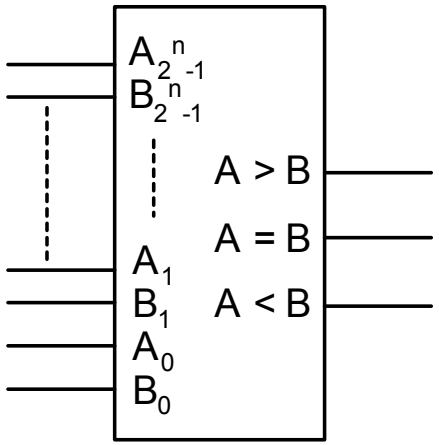
- napisati minimalne izraze za a, b, …, g
- nacrtati sklop
- napisati VHDL ponašajni *i* strukturni model
- ponoviti sve za pretvornik *heksadekadskih brojeva* u 7-segmentni kod





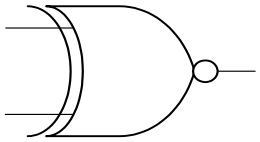
Komparator

- *komparator*
  ~ sklop za usporedbu dva *n*-bitna broja (npr. A i B)
  - obično cijeli brojevi *bez* predznaka
  - mogućnosti:
    - A = B
    - A > B
    - A < B
  - MSI modul
    ~ 4-bitni
    + mogućnost *kaskadiranja*



Komparator

- izgradnja komparatora (jedna mogućnost):
  - usporedba po bitovima
    ~ sklop EX-NILI



  - izlaz A = B
    ~ I funkcija usporedbi po bitovima
  - izlaz A > B
    ~ dominira prvi bit sa svojstvom $A_i > B_i$
    (počev od bita najviše težine)
  - izlaz A < B
    ~ **not** (($A_i > B_i$) **or** (A = B))

Komparator

Primjer: 4-bitni komparator

- usporedba po bitovima: $u_i = \overline{a_i \oplus b_i}, \quad i = 0,...,3$
- izlaz A = B: $"A = B" = u_3 \cdot u_2 \cdot u_1 \cdot u_0$
- izlaz A > B

  ~ *rekurzivno* utvrđivanje $a_i > b_i$, od bita najviše težine:

$$"a_3 > b_3" = a_3 \cdot \overline{b_3}$$
$$"a_2 > b_2" = a_2 \cdot \overline{b_2} \cdot u_3$$
$$"a_1 > b_1" = a_1 \cdot \overline{b_1} \cdot u_3 \cdot u_2$$
$$"a_0 > b_0" = a_0 \cdot \overline{b_0} \cdot u_3 \cdot u_2 \cdot u_1$$

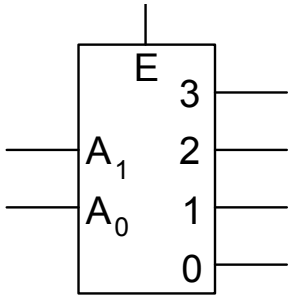
$$\Rightarrow \quad "A > B" = "a_3 > b_3" + "a_2 > b_2" + "a_1 > b_1" + "a_0 > b_0"$$

- izlaz A < B: $"A < B" = \overline{"A = B" + "A > B"}$

Opis dekodera u jeziku VHDL

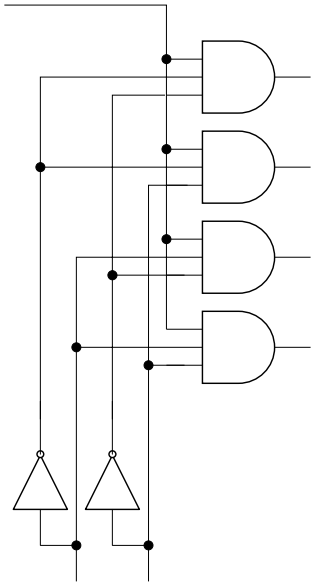
- VHDL *ponašajni* model binarnog dekodera 2/4

```
library ieee;
use ieee.std_logic_1164.all;

entity dekoder24e is
  port (d: out std_logic_vector(0 to 3);
        a: in std_logic_vector(1 downto 0);
        e: in std_logic);
end dekoder24e;

architecture ponasajna of dekoder24e is
begin
  process (a,e)
  begin
    if(e = '0')
      then d <= "0000";
      else case a is
             when "00"   => d <= "1000";
             when "01"   => d <= "0100";
             when "10"   => d <= "0010";
             when "11"   => d <= "0001";
             when others => d <= "0000";
           end case;
    end if;
  end process;
end ponasajna;
```



Opis dekodera u jeziku VHDL

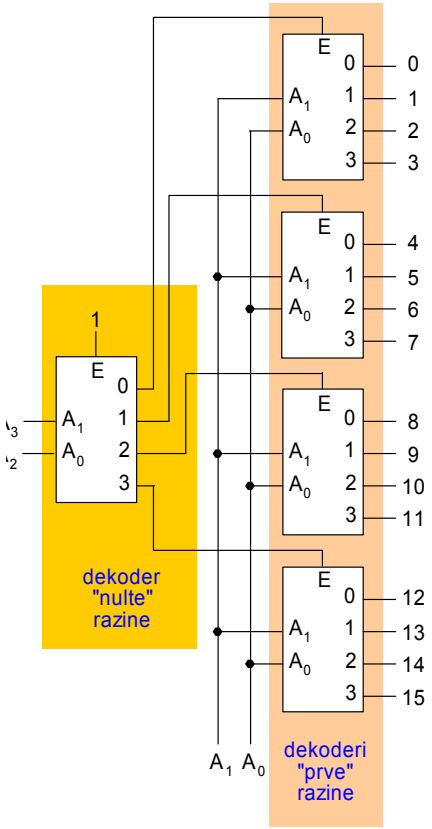
- VHDL *strukturni* model binarnog dekodera 2/4



```
library ieee;
use ieee.std_logic_1164.all;

entity dekoder24e is
  port (d: out std_logic_vector(0 to 3);
        a: in std_logic_vector(1 downto 0);
        e: in std_logic);
end dekoder24e;

architecture strukturna of dekoder24e is
  signal a1_komplement, a0_komplement: std_logic;
  begin
    sklop1: entity work.sklopNOT  port map (a(1),a1_komplement);
    sklop2: entity work.sklopNOT  port map (a(0),a0_komplement);
    sklop3: entity work.sklopAND3 port map (e,a1_komplement,a0_komplement,d(0));
    sklop4: entity work.sklopAND3 port map (e,a1_komplement,a(0),d(1));
    sklop5: entity work.sklopAND3 port map (e,a(1),a0_komplement,d(2));
    sklop6: entity work.sklopAND3 port map (e,a(1),a(0),d(3));
end strukturna;
```

Opis dekodera u jeziku VHDL

- VHDL *strukturni* model dekoderskog stabla 4/16




```
library ieee;
use ieee.std_logic_1164.all;

entity dekoder416e is
  port (d: out std_logic_vector(0 to 15);
        a: in  std_logic_vector(3 downto 0);
        e: in  std_logic);
end dekoder416e;

architecture strukturna of dekoder416e is
  signal e_tmp: std_logic_vector(0 to 3);
  begin
    sklop1: entity work.dekoder24e port map (e_tmp, a(3 downto 2), e);
    sklop2: entity work.dekoder24e port map (d(0 to 3), a(1 downto 0), e_tmp(0));
    sklop3: entity work.dekoder24e port map (d(4 to 7), a(1 downto 0), e_tmp(1));
    sklop4: entity work.dekoder24e port map (d(8 to 11), a(1 downto 0), e_tmp(2));
    sklop5: entity work.dekoder24e port map (d(12 to 15), a(1 downto 0), e_tmp(3));
end strukturna;
```

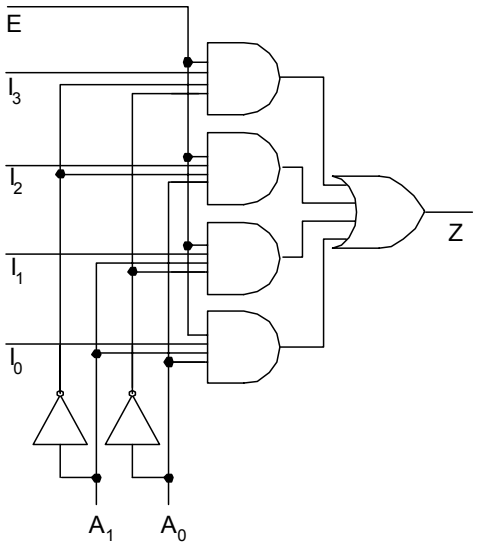
Opis multipleksora u jeziku VHDL

- *VHDL ponašajni model* multipleksora 4/1

```
entity mux41e is
  port (i: in std_logic_vector(0 to 3);
        a: in std_logic_vector(1 downto 0);
        e: in std_logic;
        z: out std_logic);
end mux41e;

architecture ponasajna of mux41e is
begin
  process (i,a,e)
  begin
    if(e = '0')
      then z <= '0';
      else
        case a is
          when "00" =>   z <= i(0);
          when "01" =>   z <= i(1);
          when "10" =>   z <= i(2);
          when "11" =>   z <= i(3);
          when others => z <= '0';
        end case;
    end if;
  end process;
end ponasajna;
```

Opis multipleksora u jeziku VHDL

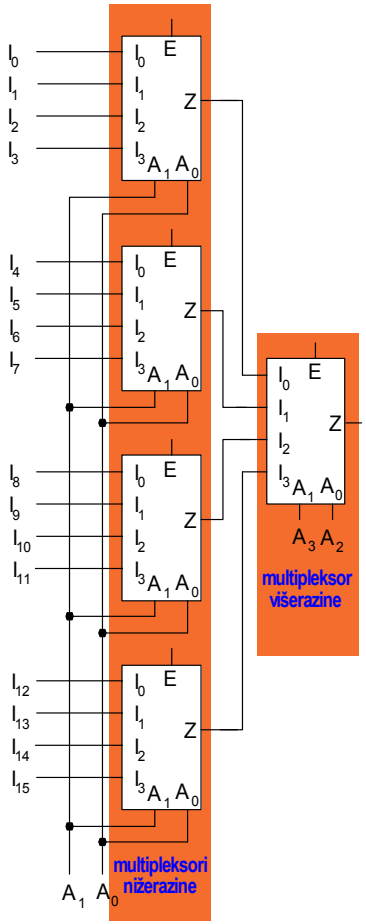
- *VHDL strukturni model* multipleksora 4/1




```
entity mux41e is
  port (i: in std_logic_vector(0 to 3);
        a: in std_logic_vector(1 downto 0);
        e: in std_logic;
        z: out std_logic);
end mux41e;

architecture strukturna of mux41e is
  signal a1_komplement, a0_komplement: std_logic;
  signal rez: std_logic_vector(0 to 3);
  begin
   sklop1: entity work.sklopNOT  port map (a(1),a1_komplement);
   sklop2: entity work.sklopNOT  port map (a(0),a0_komplement);
   sklop3: entity work.sklopAND4 port map (e,a1_komplement,a0_komplement,i(0),rez(0));
   sklop4: entity work.sklopAND4 port map (e,a1_komplement,a(0),i(1),rez(1));
   sklop5: entity work.sklopAND4 port map (e,a(1),a0_komplement,i(2),rez(2));
   sklop6: entity work.sklopAND4 port map (e,a(1),a(0),i(3),rez(3));
   sklop7: entity work.sklopOR4  port map (rez(0),rez(1),rez(2),rez(3),z);
end strukturna;
```

Opis multipleksora u jeziku VHDL

- VHDL *strukturni* model multipleksorskog stabla 16/1:

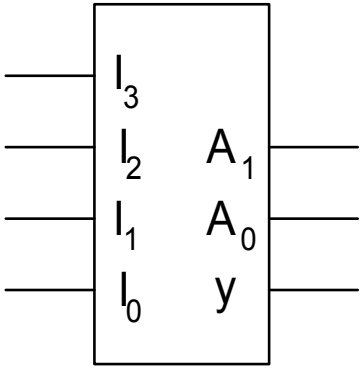



```
library ieee;
use ieee.std_logic_1164.all;

entity mux161e is
  port (i: in std_logic_vector(0 to 16);
        a: in std_logic_vector(3 downto 0);
        e: in std_logic;
        z: out std_logic);
end mux161e;

architecture strukturna of mux161e is
  signal rez: std_logic_vector(0 to 3);
  begin
    sklop1: entity work.mux41e port map (i(0 to 3),a(1 to 0),e,rez(0));
    sklop2: entity work.mux41e port map (i(4 to 7),a(1 to 0),e,rez(1));
    sklop3: entity work.mux41e port map (i(8 to 11),a(1 to 0),e,rez(2));
    sklop4: entity work.mux41e port map (i(12 to 15),a(1 to 0),e,rez(3));
    sklop5: entity work.mux41e port map (rez(0 to 3),a(3 to 1),e,z);
end strukturna;
```

Opis prioritetnog kodera u jeziku VHDL

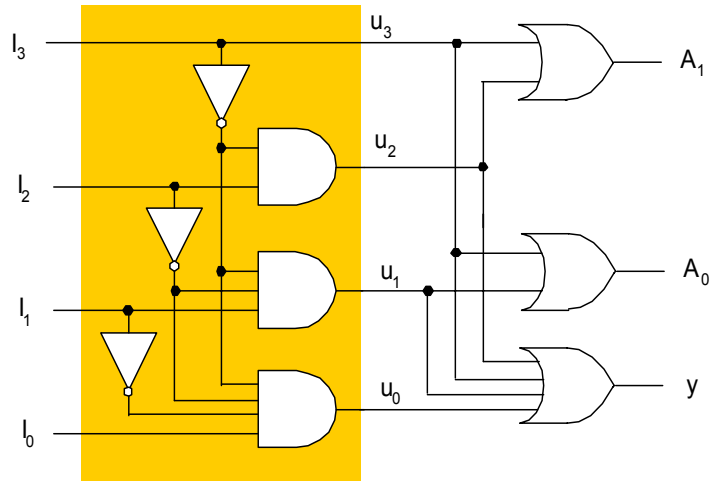
- VHDL *ponašajni* model prioritetnog kodera 2/4

```
library ieee;
use ieee.std_logic_1164.all;

entity priorityEncoder is
  port (I: in std_logic_vector(3 downto 0);
        A: out std_logic_vector(1 downto 0);
        y: out std_logic);
end priorityEncoder;

architecture ponasajna of priorityEncoder is
  begin
    A <= "11" when I(3)='1' else "10"
              when I(2)='1' else "01"
              when I(1)='1' else "00";
    y <= '0'  when I="0000" else '1';
end ponasajna;
```



Opis prioritetnog kodera u jeziku VHDL

- VHDL *strukturni* model prioritetnog kodera 2/4




```
library ieee;
use ieee.std_logic_1164.all;

entity priorityEncoder is
  port (I: in  std_logic_vector(3 downto 0);
        A: out std_logic_vector(1 downto 0);
        y: out std_logic );
  end priorityEncoder;

architecture strukturna of priorityEncoder is
  signal n3,n2,n1,u3,u2,u1,u0: std_logic;
  begin
    sklop1: entity work.sklopNOT  port map (I(3),n3);
    sklop2: entity work.sklopNOT  port map (I(2),n2);
    sklop3: entity work.sklopNOT  port map (I(1),n1);
    u3 <= I(3);
    sklop4: entity work.sklopAND2 port map (n3,I(2),u2);
    sklop5: entity work.sklopAND3 port map (n3,n2,I(1),u1);
    sklop6: entity work.sklopAND4 port map (n3,n2,n1,I(0),u0);
    sklop7: entity work.sklopOR2  port map (u3,u2,A(1));
    sklop8: entity work.sklopOR3  port map (u3,u2,u1,A(0));
    sklop9: entity work.sklopOR4  port map (u3,u2,u1,u0,y);
end strukturna;
```

Opis komparatora u jeziku VHDL

- nacrtati sklop 4-bitnog komparatora
- napisati tablicu kombinacija
- napisati VHDL ponašajni model sklopa
- napisati VHDL strukturni model sklopa