

1.1 Princípy návrhu číslicových systémov

Abstrakcia

Disciplína










Hierarchia

Modularita

Jednotnosť

Abstrakcia

- Údaje su skryté, ak nie sú potrebné
- Systém je možné pozorovať z viacerých levelov abstrakcii (napríklad Prezident sa na výsledok volieb pozerá inak ako štatistický úrad)

Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

Levely abstrakcie v počítačových systémoch

Disciplína

- Zámerne obmedzenie návrhových možností
- Napríklad: Digitálna disciplína (0 a 1, true a false) - bit
 - diskrétné napätie namiesto nepretžitého (napätie, oscilácie) využívaného v Analógovej disciplíne
 - jednoduchšie na konštrukciu ako analógové obvody – môžeme postaviť viac sofistikované systémy
 - Digitálny systém nahrádza analógových predchodcov (napríklad Digitálne kamery, digitálna TV,...)

Hierarchia- Systém rozdelenia do modulov a podmodulov pre jednoduchšie pochopenie problému

Modularita- Správne definované funkcie a rozhrania pre predídenie neočakávaným problémom

Jednotnosť- Podporuje jednotnosť modulov, tak, že moduly je možné použiť znova

Kompozícia - Správne rozloženie prvkov v systéme

Príklad

Hierarchia

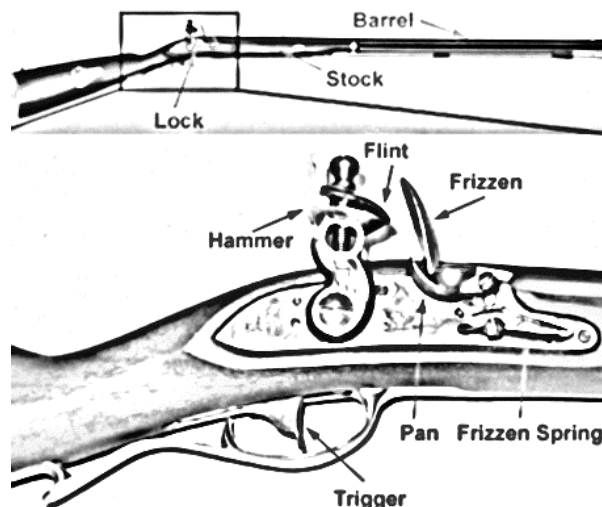
- Tri hlavné moduly:
 - lock, stock, barrel
- Podmoduly lock-u:
 - hammer, flint, frizzen,...

Modularita

- Funkcia stock-u:
 - nasadiť barrel a lock
- Rozhranie stock-u:
 - rozmery a osadenie častí na nasadenie

Jednotnosť

- Vymeniteľné časti pušky



Kompozícia

- Jednotlivé časti sa musia nachádzať na svojom mieste

Charakteristika číslicových systémov

- Číslicový systém chápeme ako formálny model, ktorý možno definovať nad reálnym (napr. elektronickým) číslicovým zariadením.
- Číslicový systém je dynamický systém, ktorý možno charakterizovať týmto spôsobom:
 - Premenné sú číslicové, čo znamená, že nadobúdajú hodnoty iba z konečných množín hodnôt.
 - Správanie sa systému je definované v diskretnom čase
- V digitálnych systémoch sa najčastejšie využíva binárny a hexadecimálny ČS

Dekadický ČS	Dvojkový (Binárny) ČS	Hexadecimálny ČS
<div>1's column 10's column 100's column 1000's column</div> $5374_{10} = 5 \times 10^3 + 3 \times 10^2 + 7 \times 10^1 + 4 \times 10^0$ <div>five thousands three hundreds seven tens four ones</div>	<div>1's column 2's column 4's column 8's column</div> $1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$ <div>one eight one four no two one one</div>	<div>1's column 16's column 256's column</div> $2ED_{16} = 2 \times 16^2 + E \times 16^1 + D \times 16^0 = 749_{10}$ <div>two hundred fifty six's fourteen sixteens thirteen ones</div>

Číselné sústavy - Pozičné - Nepozičné - Symetrické - Nesymetrické

Pozičné číslicové sústavy – hodnota každej číslice je daná jej pozíciou v sekvencii symbolov, kde každá takáto číslica má svoju váhu na výpočet celkovej hodnoty

Diskrétna reprezentácia čísel – zobrazuje možné hodnoty ktoré pozícia v ČS môže nadobúdať
bin (0,1), dek (0-9), hex (0-9A-F)

1.2 Stavebné prvky číslicových systémov:

Klasifikácia základných prvkov

1. Obvody na riadenie prenosu údajov a vnútorných komunikácií:

- Prenos informácií: elektrický, optický, rádiový a pod.
- Prenos informácií je riadený proces

2. Obvody na pamätanie údajov sú určené na ukladanie (pamätanie) údajov v tvare jedno a viacbitových slov:

- preklápacie obvody
- registre
- pamäte

3. Obvody na realizáciu predikátov:

- komparátor
- generátor parity
- generátor priority

4. Obvody na realizáciu operácií

- Obvody na vykonávanie operácií nad obsahmi registrov, zberníc a iných funkčných prvkov sú najčastejšie komponentmi operačnej časti procesora ČP
- Vo všeobecnosti sa tieto operácie rozdeľujú do troch skupín
 - celočíselné aritmetické operácie (sčítanie, odčítanie, násobenie a delenie)
 - logické operácie (logický súčet a súčin, operácie XOR, maskovanie, logické posuvy a pod.)
 - operácie špeciálnej aritmetiky (desiatková aritmetika, aritmetika s pohyblivou rádovou čiarkou špeciálne funkcie a pod.).

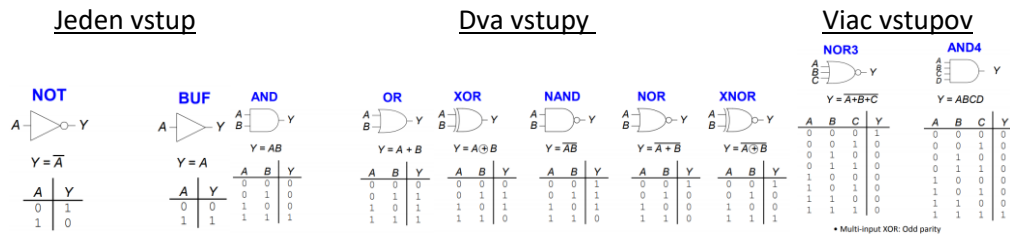
Hradlo

- je základný logický člen mikroarchitektúry ČP, ktorého priechodnosť medzi vstupným (X) a výstupným (Y) kanálom označeným rovnako pomenovanou vstupnou a výstupnou premennou X a Y.

$$Y = \begin{cases} X \Leftarrow R = f(R_1, R_2, \dots, R_k) = 1 & X - \text{je vstupná premenná v kanále } X, \\ 0 \Leftarrow R = f(R_1, R_2, \dots, R_k) = 0 & Y - \text{výstupná premenná v kanále } Y, \end{cases}$$

- R - riadiaci signál definovaný ako funkcia f riadiacich premenných R1, R2, ..., Rk,
t.j. $Y = X \wedge R$, resp. $Y = X \wedge 1 = X$.
- Významnou aplikáciou hradla pri riadení komunikačných operácií je riadenie prenosu údajov z viacerých zdrojov do jedného cieľového zdroja, ktorým je zbernica
- na základe riadiacej informácie buď prepája vstup na výstup (je uvoľnené resp. priepustné), alebo informáciu zo vstupu na výstup neprenáša (hradlo je zablokované resp. nepriepustné).
- na výstupe musí byť nejaká hodnota fyzikálnej veličiny
- v prípade, ak je hradlo zablokované, sa jeho výstup uvedie do tzv. tretieho stavu (stavu vysokej impedancie). V tomto stave má výstup vlastnosť vstupu s vysokou impedanciou.
- okrem jednosmerných hradiel sa používajú aj obojsmerné hradia, kde vstup môže byť v inom okamihu použitý ako výstup a naopak
- vykonáva logické funkcie: NOT, AND, OR, NAND, NOR, XOR, ...
- jeden vstup – NOT, buffer

- dva vstupy – AND, OR, XOR, NAND, NOR, XNOR
- viac vstupov



Spojité veličiny – označujú hodnotu alebo stav (výška, čas, veľkosť,...)

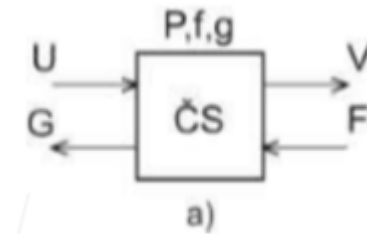
Základ ČS:

- Digitálny systém: Má definovaný počet bitov niekedy môže nastať prepĺnenie (číslo je priveľké a teda sa nezmesť do príslušných bitov)
- Skladá sa z logických obvodov
- Popisujú sa pomocou programovacích jazykov pre modelovanie, simuláciu a syntézu
- Príkladom sú dnešné PC
- Rozdeľujeme:
 - Aplikačno špecifické systémy – vykonávajú špecializovanú činnosť nemeniteľné
 - Univerzálne PC – sú programovateľné realizáciou (mechanicky, elektornicky..)
- Číslicovým systémom je aj *Číslicový počítač* - zložitý univerzálny číslicový systém(automat) určený na samočinné vykonávanie postupnosti operácií(výpočtov) nad údajmi zobrazenými číslicovým kódom, na základe vopred pripraveného a v pamäti uloženého programu(algoritmu)
- je definovaný sedmicou ČS = (U, V, P, F, G, f, g) kde U V F G sú premenné definované vo vonkajších kanáloch ČS určených na jeho styk s okolím ktoré môžu byť:
 - údajové (zobrazujú spracúvané informácie),
 - komunikačné (zobrazujú informácie o spracúvaných údajoch), prezentované ako riadiaceľ informačné a stavové premenné,

- *P* – premenná, vyjadrujúca stav ČS

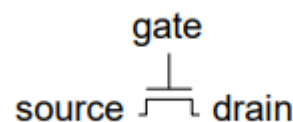
- *f, g* – funkčné vzťahy ktoré vyjadrujú reakciu ČS na podnety definované hodnotami nasledujúcich premenných, pôsobiacich v príslušných kanáloch: $V = f(U, F)$, $G = g(P)$

- Každý ČS chápaný chápeme, ako formálny model definovaný nad reálnym číslicovým zariadením ktorý je možné z hľadiska jeho funkcie dekomponovať na operačnú časť (OČ) a riadiacu časť (RČ).



Tranzistory typu CMOS

- Tranzistor je elektricky ovládateľný spínač ktorý mení stav na 0 alebo 1 ak je na ovládací terminál privedený prúd
- Poznáme Bipolar a MOSFET (metal-oxide-semiconductor field effect transistors) tranzistor MOS (metal oxide silicon)

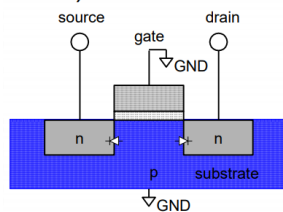


- Polykrémikové hradlo (Polysilicon)
- Oxidový izolátor (SiO₂)

nMOS

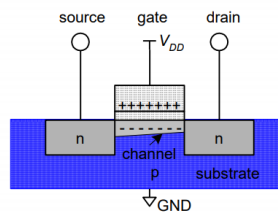
Gate = 0

OFF (no connection between source and drain)



Gate = 1

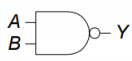
ON (channel between source and drain)



CMOS (Complementary MetalOxide-Semiconductor Logic)

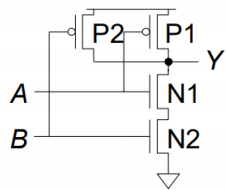
- takmer nulový prúd pretekajúci obvodom v statickom stave
- sú úspornejšie ako nMOS

NAND



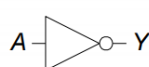
$$Y = \overline{AB}$$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



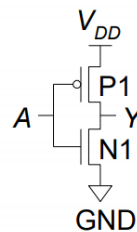
A	B	P1	P2	N1	N2	Y
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

NOT

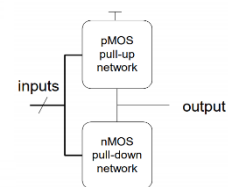


$$Y = \overline{A}$$

A	Y
0	1
1	0



A	P1	N1	Y
0	ON	OFF	1
1	OFF	ON	0



CMOS Štruktúra

2.1 Opis algoritmu celočíselného násobenia

- Postup je podobný ako v prípade dekadického počítania

- Nech $A[n-1:0]$, $B[n-1:0]$, $Q[n-1:0]$, $C[1]$ a PC sú registre. Nech PC slúži ako počítadlo cyklov. Zavedme $C\#Q\#A$ na označenie zreťazenia registrov. Potom

Algoritmus celočíselného násobenia

```
BEGIN
  C#Q := 0 || PC := n
  WHILE (PC > 0)
    IF A(0) = 1 THEN
      C#Q := Q + B
    ENDIF
    C#Q#A := 0 # SR1(C#Q#A) ||
    PC := PC - 1
  END
END
```

Kroky celočíselného násobenia $(+13)_{10} \times (-5)_{10}$ v BinČS

□ $A = |(+13)_{10}| = |(1101)_2|$, $A_z = 0$; $B = |(-5)_{10}| = |(0101)_2|$, $B_z = 1$

□ Znamienko výsledku = $A_z \oplus B_z = 0 \oplus 1 = 1$

C	#	Q	#	A	PC	Poznámka
0		0000		1101	4	$C\#Q := 0 \parallel PC := \lceil \log_2(\max(A , B) + 1) \rceil$ $A(0) = 1 \Rightarrow C\#Q := Q + B$
	+	0101				
0		0101		1101		
0		0010		1110	3	$C\#Q\#A := 0 \# SR1(C\#Q\#A) \parallel PC := PC - 1$
0		0001		0111	2	$C\#Q\#A := 0 \# SR1(C\#Q\#A) \parallel PC := PC - 1$ $A(0) = 1 \Rightarrow C\#Q := Q + B$
	+	0101				
0		0110		0111		
0		0011		0011	1	$C\#Q\#A := 0 \# SR1(C\#Q\#A) \parallel PC := PC - 1$ $A(0) = 1 \Rightarrow C\#Q := Q + B$
	+	0101				
0		1000		0011		
0		0100		0001	0	$C\#Q\#A := 0 \# SR1(C\#Q\#A) \parallel PC := PC - 1$

1 1 0 1	(13) ₁₀	Multiplicand
× 1 0 1 1	(11) ₁₀	Multiplier
1 1 0 1		Partial products
1 1 0 1		
0 0 0 0		
1 1 0 1		
1 0 0 0 1 1 1 1	(143) ₁₀	Product

Opis algoritmu sčítania reálnych čísel vyjadrených v PRČ

- dostaneme 2 vstupy s n-bit a vrátime výstup s n-bit

Example 1. Use the algorithm to compute the sum $A = +1.101 \times 2^2$ a $B = +1.011 \times 2^3$. The precision is $n = 4$.

Solution

The exponent, significand and sign bit of operands are:

$E_A := 2$, $S_A := 1.101$, sign $A := 0$, $E_B := 3$, $S_B := 1.011$, sign $B := 0$.

- $d := 2 - 3 = -1 < 0 \Rightarrow E_A < E_B \Rightarrow A \leftrightarrow B \Rightarrow S_A := 1.011$, $E_A := 3$, $S_B := 1.101$, $E_B := 2$, $E := E_A = 3$.
- sign $A = \text{sign } B \Rightarrow$ we don't create the two's complement of B
- $|d| = +1 \Rightarrow$ shift to the right by one bit of $S_B = 1.101 \Rightarrow S_B := 0.110$ $\lfloor 1 \Rightarrow g := 1$, $r := 0$, $s := 0$.
- $C.S := S_A + S_B$

$$\begin{array}{r} 1.011 \\ + 0.110 \\ \hline \end{array}$$
 $[1]0.001 \Rightarrow S = 0.001$, $C = 1$, $S(n-1) = S(3) = 0$.
- $C = 1$, $S(3) = 0 \Rightarrow S := C \# SR1(S) = 1.000 \Rightarrow E := E + 1 = 3 + 1 = 4$.
- $s := g \vee r \vee s = 1 \vee 0 \vee 0 = 1$, $r := S(0) = 1$.
- $(r \wedge S(0)) \vee (r \wedge s) = (1 \wedge 0) \vee (1 \wedge 1) = 1 \Rightarrow \text{SUM} := S + S(0)_{-1} = 1.000 + 0.001 = 1.001$.
- sign $A = \text{sign } B \Rightarrow \text{sign SUM} := \text{sign } A = 0$

SUM := +1.001 × 2⁴ = +18 (The expected result is +17.5).

Example 2. Use the algorithm to compute the sum $A = -1.110 \times 2^2$ and $B = +1.101 \times 2^{-1}$. The precision is $n = 4$.

Solution

The exponent, significand and sign bit of operands are:

$E_A := 2$, $S_A := 1.110$, sign $A := 1$, $E_B := -1$, $S_B := 1.101$, sign $B := 0$.

- $d := 2 - (-1) = 3 \geq 0 \Rightarrow E_A \geq E_B \Rightarrow$ we don't swap A and B , $E := E_A = 2$.
- sign $A \neq \text{sign } B \Rightarrow$ we create the two's complement of $B \Rightarrow S_B := 0.011$.
- $|d| = 3 \Rightarrow S_B := 1.110 \lfloor 011$, $g := 0$, $r := 1$, $s := 1$.
- $C.S := S_A + S_B$

$$\begin{array}{r} 1.110 \\ + 1.110 \lfloor 011 \\ \hline \end{array}$$
 $[1]1.100 \lfloor 011 \Rightarrow S = 1.100$, $C = 1$, $S(n-1) = S(3) = 1$
- $C = 1$, $S(3) = 1 \Rightarrow$ shifting of S is not necessary
- because S wasn't shifted $\Rightarrow S := R \vee S = 1 \vee 1 = 1$, $R := G = 0$.
- $(r \wedge S(0)) \vee (r \wedge s) = (0 \wedge 0) \vee (0 \wedge 1) = 0 \Rightarrow$ rounding is not necessary $\Rightarrow \text{SUM} := S$.
- $d \geq 0$, $A < 0$, $B \geq 0 \Rightarrow \text{SUM} < 0 \Rightarrow \text{sign SUM} := 1$

SUM := -1.1 × 2² = -6 (The expected result is -6.1875 without rounding).

$$\begin{array}{r} 1.10011 \\ + 0.00001 \\ \hline 1.10100 \end{array}$$

2.2 Opis algoritmu celočíselného delenia

- Snažíme sa o čo najmenší počet operácii ako je len možné
- Nech $A[n-1:0]$, $B[n-1:0]$, $Q[n-1:0]$ a PC sú registre. Nech PC slúži ako počítadlo cyklov. Zavedme $Q\#A$ na označenie zreteženia registrov. Potom

Algoritmus celočíselného delenia

```

BEGIN
  C#Q := 0 || PC := n
  WHILE(PC > 0)
    C#Q#A := SL1(C#Q#A)#0
    IF (Q ≥ B) THEN
      C#Q := Q - B || A(0) := 1
    ENDIF
    PC := PC - 1
  END
END

```

Kroky celočíselného delenia $(+13)_{10} / (-5)_{10}$ v BinČS

- $A = |(+13)_{10}| = |(1101)_2|$, $A_z = 0$; $B = |(-5)_{10}| = |(0101)_2|$, $B_z = 1$
- Znamienko výsledku = $A_z \oplus B_z = 0 \oplus 1 = 1$

Q	#	A	PC	Poznámka
0000		1101	4	$C\#Q := 0 PC := \lceil \log_2(\max(A , B) + 1) \rceil$
0001		1010		$C\#Q\#A := SL1(C\#Q\#A)\#0$
0001		1010	3	$Q < B \Rightarrow PC := PC - 1$
0011		0100		$C\#Q\#A := SL1(C\#Q\#A)\#0$
0011		0100	2	$Q < B \Rightarrow PC := PC - 1$
0110		1000		$C\#Q\#A := SL1(C\#Q\#A)\#0$
- 0101				$Q > B \Rightarrow C\#Q := Q - B,$
0001		1001	1	$A(0) := 1, PC := PC - 1$
0011		0010		$C\#Q\#A := SL1(C\#Q\#A)\#0$
0011		0010	0	$PC := PC - 1$

```

1100011 : 10010 = 101,1
- 10010
00110
1101
11011
- 10010
1001
10010
- 10010
00000

```

Násobenie v PHRČ

- Najjednoduchšou operáciou pohyblivej rádovej čiarky je násobenie
- Operácia násobenia násobenca s násobiteľom sa skladá z niekoľkých častí:
 - násobenie mantís podľa algoritmu celočíselného násobenia bezznamienkových, resp. znamienkových čísel (Boothov algoritmus)
 - násobenie sa uskutočňuje na základe algoritmu celočíselného bezznamienkového násobenia -
 - zaokrúhľenie výsledku, pretože výsledkom násobenia dvoch n -rádových čísel je $2n$ -rádový, resp. $(2n-1)$ -rádový výsledok, ktorého n najnižších rádov sa použije v procese zaokrúhľovania výsledku 1
 - výpočet exponenta výsledku, ktorý zahŕňa odčítanie bázy posunutia od súčtu exponentov s posunutou nulou.

Example 1a. In binary with $p = 5$, show how the multiplication algorithm computes the product $A = (+6.25)_{10}$ a $B = (+3.5)_{10}$.

Solution:

$$A = 6.25 = (110.01)_2 = 1.1001 \times 2^2, S_A := 11001, E_A := 2, \text{sign}A := 0$$

$$B = 3.5 = (11.1)_2 = 1.1100 \times 2^1, S_B := 11100, E_B := 1, \text{sign}B := 0$$

$$\text{The exponent of the result is } E := E_A + E_B = 3$$

C	#	Q	#	A	PC	Note
0		00000		11001	5	$C\#Q := 0 \parallel PC := \lceil \log_2(\max(A , B) + 1) \rceil$
		+ 11100				$A(0) = 1 \Rightarrow C\#Q := C\#Q + B$
0		<u>11100</u>		11001		
0		01110	01100	±	4	$C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$
0		00111	00110	0	3	$A(0) = 0 \Rightarrow C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$
0		00011	10011	0	2	$A(0) = 0 \Rightarrow C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$
		+ 11100				$A(0) = 1 \Rightarrow C\#Q := C\#Q + B$
0		<u>11111</u>		10011		
0		01111	11001	±	1	$C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$
		+ 11100				$A(0) = 1 \Rightarrow C\#Q := C\#Q + B$
1		<u>01011</u>		11001		
0		10101	11100	±	0	$C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$

$$g := A(n-1) = 1$$

$$r := A(n-2) = A(5-2) = 1$$

$$s := A(n-3) \vee A(n-4) \vee \dots \vee A(0) = A(5-3) \vee A(5-4) \vee A(5-5) = 1 \vee 0 \vee 0 = 1$$

if $Q(4) = 1$ **then** *rounding (rule 2)*
 $s := r \vee s = 1 \vee 1 = 1$
 $r := g = 1$
 $E := E + 1 = 4$ add 1 to the exponent

endif

if $(r \wedge s) = 1$ **then**
 $P(\text{roduct}) := Q + 1 = 10101 + 1 = 10110$

endif

$$\text{sign}P = \text{sign}A \oplus \text{sign}B = 0 \oplus 0 = 0$$

The result after rounding is $+ 1.0110 \times 2^4 = (+10110)_2 = 22$ (relative error is -0,125; the expected result without rounding is 21.875).

2.3 Booth-ov algoritmus

- zaobchádza s +- číslami jednotne tak že využíva fakt že reťazec s 0 a 1 v multiplikátore nevyžadujú sčítavanie iba otáčanie

- Nech sú dané čísla A a B vyjadrené v binárnom doplnkovom kóde na n-bitoch.

- $A = a_{n-1}a_{n-2} \dots a_1a_0$ (násobenec)

$B = b_{n-1}b_{n-2} \dots b_1b_0$ (násobiteľ).

- Platí

a_i	a_{i-1}	$a_{i-1} - a_i$	Operácia
0	0	0	NOP
1	1	0	NOP
1	0	-1	Odčítaj B od Q
0	1	1	Pripočítaj B k Q

kde $i = 0$

Let $A = (+13)_{10}$ and $B = (-5)_{10}$. Then $A = (01101)_{2\text{'s complement}}$; $B = (-5)_{10} = (11011)_{2\text{'s complement}}$;
 $(-B) = (+5)_{10} = (00101)_{2\text{'s complement}}$

C	#	Q	#	A	PC	
0		00000		01101	0	5
		+ 00101				$C\#Q := 0 \parallel PC := \lceil \log_2(\max(A , B) + 1) \rceil + 1$
0		00101		01101		$A(0) > A(-1) \Rightarrow C\#Q := (C\#Q - B)_{2DK}$
0		00010		10110	1	4
		+ 11011				$C\#Q\#A := C \# SR1(C\#Q\#A) \parallel PC := PC - 1$
1		11101		10110		$A(0) < A(-1) \Rightarrow C\#Q := (C\#Q + B)_{2DK}$
1		11110		11011	0	3
		+ 00101				$C\#Q\#A := C \# SR1(C\#Q\#A) \parallel PC := PC - 1$
0		00011		11011		$A(0) > A(-1) \Rightarrow C\#Q := (C\#Q - B)_{2DK}$
0		00001		11101	1	2
0		00000		11110	1	1
		+ 11011				$C\#Q\#A := C \# SR1(C\#Q\#A) \parallel PC := PC - 1$
1		11011		11110		$A(0) < A(-1) \Rightarrow C\#Q := (C\#Q + B)_{2DK}$
1		11101		11111	0	0
						$C\#Q\#A := C \# SR1(C\#Q\#A) \parallel PC := PC - 1$

Note: 2DK means 2's complement

Násobenie v PHRČ

- Najjednoduchšou operáciou pohyblivej rádovej čiarky je násobenie

Operácia násobenia násobenca s násobiteľom sa skladá z niekoľkých častí:

- násobenie mantís podľa algoritmu celočíselného násobenia bezznamienkových, resp. znamienkových čísel (Boothov algoritmus)

- násobenie sa uskutočňuje na základe algoritmu celočíselného bezznamienkového násobenia -

- zaokrúhlenie výsledku, pretože výsledkom násobenia dvoch n-rádových čísel je $2n$ -rádový, resp. $(2n-1)$ -rádový výsledok, ktorého n najnižších rádov sa použije v procese zaokrúhľovania výsledku 1

- výpočet exponenta výsledku, ktorý zahŕňa odčítanie bázy posunutia od súčtu exponentov s posunutou nulou.

Example 1a. In binary with $p = 5$, show how the multiplication algorithm computes the product $A = (+6.25)_{10}$ a $B = (+3.5)_{10}$.

Solution:

$$A = 6.25 = (110.01)_2 = 1.1001 \times 2^2, S_A := 11001, E_A := 2, \text{sign}A := 0$$

$$B = 3.5 = (11.1)_2 = 1.1100 \times 2^1, S_B := 11100, E_B := 1, \text{sign}B := 0$$

$$\text{The exponent of the result is } E := E_A + E_B = 3$$

C	#	Q	#	A	PC	Note
0		00000		11001	5	$C\#Q := 0 \parallel PC := \lceil \log_2(\max(A , B) + 1) \rceil$
		+ 11100				$A(0) = 1 \Rightarrow C\#Q := C\#Q + B$
0		<u>11100</u>		11001		
0		01110	01100	±	4	$C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$
0		00111	00110	0	3	$A(0) = 0 \Rightarrow C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$
0		00011	10011	0	2	$A(0) = 0 \Rightarrow C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$
		+ 11100				$A(0) = 1 \Rightarrow C\#Q := C\#Q + B$
0		<u>11111</u>		10011		
0		01111	11001	±	1	$C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$
		+ 11100				$A(0) = 1 \Rightarrow C\#Q := C\#Q + B$
1		<u>01011</u>		11001		
0		10101	11100	±	0	$C\#Q\#A := 0\#SR1(C\#Q\#A) \parallel PC := PC - 1$

$$g := A(n-1) = 1$$

$$r := A(n-2) = A(5-2) = 1$$

$$s := A(n-3) \vee A(n-4) \vee \dots \vee A(0) = A(5-3) \vee A(5-4) \vee A(5-5) = 1 \vee 0 \vee 0 = 1$$

if $Q(4) = 1$ **then** *rounding (rule 2)*
 $s := r \vee s = 1 \vee 1 = 1$
 $r := g = 1$
 $E := E + 1 = 4$ add 1 to the exponent

endif

if $(r \wedge s) = 1$ **then**
 $P(\text{roduct}) := Q + 1 = 10101 + 1 = 10110$

endif

$$\text{sign}P = \text{sign}A \oplus \text{sign}B = 0 \oplus 0 = 0$$

The result after rounding is $+ 1.0110 \times 2^4 = (+10110)_2 = 22$ (relative error is -0,125; the expected result without rounding is 21.875).

3.1 Boolova algebra

- algebrická štruktúra s logickými operáciami
- abstraktný formálny systém obsahujúci množinu prvkov (a, b, c, ...), nad ktorou sú definované dve binárne operácie symbolizované pomocou znakov \wedge a \vee .
- Boolova algebra je komplementárny a distributívny zväz

Axiomy – sú nedokázateľné v rovnakom zmysle ako definície v algebre..sú ale jednoduchšie (2 stavy 0,1)

Teorémy – vznikajú z axiémov

Table 2.1 Axioms of Boolean algebra

Axiom	Dual	Name
A1 $B = 0$ if $B \neq 1$	A1' $B = 1$ if $B \neq 0$	Binary field
A2 $\overline{0} = 1$	A2' $\overline{1} = 0$	NOT
A3 $0 \cdot 0 = 0$	A3' $1 + 1 = 1$	AND/OR
A4 $1 \cdot 1 = 1$	A4' $0 + 0 = 0$	AND/OR
A5 $0 \cdot 1 = 1 \cdot 0 = 0$	A5' $1 + 0 = 0 + 1 = 1$	AND/OR

Table 2.2 Boolean theorems of one variable

Theorem	Dual	Name
T1 $B \cdot 1 = B$	T1' $B + 0 = B$	Identity
T2 $B \cdot 0 = 0$	T2' $B + 1 = 1$	Null Element
T3 $B \cdot B = B$	T3' $B + B = B$	Idempotency
T4 $\overline{\overline{B}} = B$		Involution
T5 $B \cdot \overline{B} = 0$	T5' $B + \overline{B} = 1$	Complements

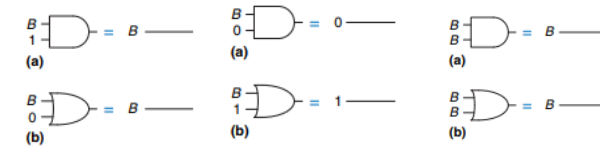


Figure 2.14 Identity theorem in hardware: (a) T1, (b) T1'

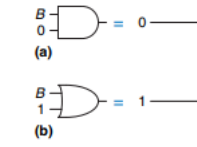


Figure 2.15 Null element theorem in hardware: (a) T2, (b) T2'

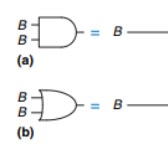


Figure 2.16 Idempotency theorem in hardware: (a) T3, (b) T3'

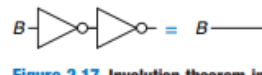


Figure 2.17 Involution theorem in hardware: T4

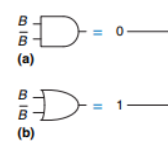
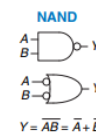
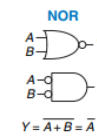


Figure 2.18 Complement theorem in hardware: (a) T5, (b) T5'

Theorem	Dual	Name
T6 $B \cdot C = C \cdot B$	T6' $B + C = C + B$	Commutativity
T7 $(B \cdot C) \cdot D = B \cdot (C \cdot D)$	T7' $(B + C) + D = B + (C + D)$	Associativity
T8 $(B \cdot C) + B \cdot D = B \cdot (C + D)$	T8' $(B + C) \cdot (B + D) = B + (C \cdot D)$	Distributivity
T9 $B \cdot (B + C) = B$	T9' $B + (B \cdot C) = B$	Covering
T10 $(B \cdot C) + (B \cdot \overline{C}) = B$	T10' $(B + C) \cdot (B + \overline{C}) = B$	Combining
T11 $(B \cdot C) + (\overline{B} \cdot D) + (C \cdot D) = B \cdot C + \overline{B} \cdot D$	T11' $(B + C) \cdot (\overline{B} + D) \cdot (C + D) = (B + C) \cdot (\overline{B} + D)$	Consensus
T12 $\overline{B_0 \cdot B_1 \cdot B_2 \dots} = (\overline{B_0} + \overline{B_1} + \overline{B_2} \dots)$	T12' $\overline{B_0 + B_1 + B_2 \dots} = (\overline{B_0} \cdot \overline{B_1} \cdot \overline{B_2} \dots)$	De Morgan's Theorem



A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 2.19 De Morgan equivalent gates

George Bool: Predstavil tri základné logické operácie AND OR NOT

Claude Shannon – implementácia – elektromechanické relé – obvody

Elementárne logické funkcie

Logický súčin - AND

Logický súčet - OR

Negácia - NOT

Boolovské rovnice

- Funkcionálna špecifikácia výstupov v zmysle vstupov

- Pracujú s hodnotami TRUE a FALSE

- Axiomy a Teoremy uľahčujú Boolovské rovnice

Komplement – premenná s čiarou - opak A', B'

Literal – premenná alebo komplement A, A', B, B'

Implikant – produkt literalov ABC'

Miniterm – produkt obsahujúci vstupy AB'C'

Maxterm – produkt obsahujúci vstupy (A+B'+C)

Sum of products (SOP) form = Disjunktívna forma

- Všetky rovnice je možné zapísať vo forme SOP

- Každý stĺpec má miniterm

Odvožené (vektorové) logické funkcie

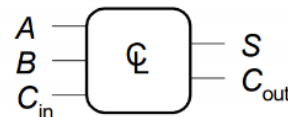
- Negácia logického súčinu - NAND

- Negácia logického súčtu – NOR

- Nerovnosť - XOR; Rovnosť NXOR

$$S = F(A, B, C_{in})$$

$$C_{out} = F(A, B, C_{in})$$



$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

A	B	Y	minterm	minterm name
0	0	0	$\overline{A} \overline{B}$	m_0
0	1	1	$\overline{A} B$	m_1
1	0	0	$A \overline{B}$	m_2
1	1	1	$A B$	m_3

$$Y = F(A, B) = \overline{A}B + AB = \Sigma(1, 3)$$

- Minterm je produktom (AND) literálov $Y = AB$
- Každý minterm je TRUE iba pre vlastný stĺpec
- Výsledkom je suma OR z produktov AND $Y = AB + A'B'$

Product of Sums (POS) form = Konjunktívna forma

- Všetky rovnice je možné zapísať vo forme POS
- Každý stĺpec ma maxterm
- Maxterm je sumou (OR) literálov $Y = A+B'$
- Každý maxterm je FALSE iba pre vlastný stĺpec
- Výsledkom je predukt AND zo súm OR $Y = (A+B')(A+B)$

A	B	Y	maxterm	maxterm name
0	0	0	$A + B$	M_0
0	1	1	$A + \overline{B}$	M_1
1	0	0	$\overline{A} + B$	M_2
1	1	1	$\overline{A} + \overline{B}$	M_3

$$Y = F(A, B) = (A + B)(A + \overline{B}) = \Pi(0, 2)$$

Karnaugové mapy

- Boolovské výrazy môžeme minimalizovať kombinovaním požiadavok
- KM minimalizujú rovnice graficky

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Y	AB	00	01	11	10
C					
0		1	0	0	0
1		1	0	0	0

Y	AB	00	01	11	10
C					
0		$\overline{A}\overline{B}\overline{C}$	$\overline{A}B\overline{C}$	$AB\overline{C}$	$A\overline{B}\overline{C}$
1		$\overline{A}\overline{B}C$	$\overline{A}BC$	ABC	$A\overline{B}C$

Pravidlá- Príslušné hodnoty (0,1) musia byť zakrúžkované minimálne raz

- Každé zakrúžkovanie musí obopínať bunky 2^n (1,2,4,8,...)
- Každé zakrúžkovanie musí byť tak veľké ako je to možné
- Môže prechádzať aj cez hrany
- Nepotrebné hodnoty (v zakázanej zóne..menia sa s prúdom, teplotou,...indikujú chybu ich reálna hodnota je niekde medzi 0 a 1) sú krúžkované len v prípade že znížia počet rovníc

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1

Y	AB	00	01	11	10
CD	00	1	0	X	1
01	0	X	X	1	

3.2 Boolovské rovnice

- Funkcionálna špecifikácia výstupov v zmysle vstupov
- Pracujú s hodnotami TRUE a FALSE
- Axiomy a Teoremy uľahčujú Boolovské rovnice
- Komplement – premenná s čiarou - opak A' , B'
- Literal – premenná alebo komplement A , A' , B , B'
- Implikant – produkt literalov ABC'
- Minterm – produkt obsahujúci vstupy $AB'C'$
- Maxterm – produkt obsahujúci vstupy $(A+B'+C)$

Sum of products (SOP) form = Disjunktívna forma

- Všetky rovnice je možné zapísať vo forme SOP
- Každý stĺpec má minterm
- Minterm je produktom (AND) literalov $Y = AB$
- Každý minterm je TRUE iba pre vlastný stĺpec
- Výsledkom je suma OR z produktov AND $Y = AB + A'B'$

Product of Sums (POS) form = Konjunktívna forma

- Všetky rovnice je možné zapísať vo forme POS
- Každý stĺpec má maxterm
- Maxterm je sumou (OR) literalov $Y = A+B'$
- Každý maxterm je FALSE iba pre vlastný stĺpec
- Výsledkom je predukt AND zo súm OR $Y = (A+B')(A+B)$

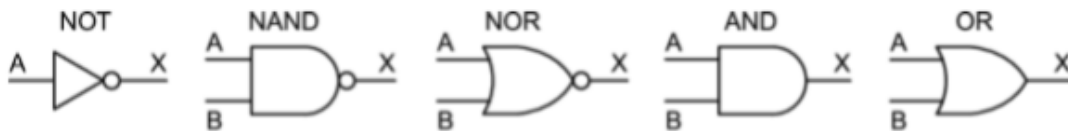
Elementárne logické členy

Logický súčin - AND

Logický súčet - OR

Negácia - NOT

Symbolické označenie LČ



Pravdivostná tabuľka LČ

A	X
0	1
1	0

(a)

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(c)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

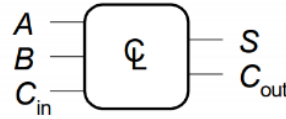
(d)

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(e)

$$S = F(A, B, C_{in})$$

$$C_{out} = F(A, B, C_{in})$$



$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

A	B	Y	minterm	minterm name
0	0	0	$\bar{A} \bar{B}$	m_0
0	1	1	$\bar{A} B$	m_1
1	0	0	$A \bar{B}$	m_2
1	1	1	$A B$	m_3

$$Y = F(A, B) = \bar{A}B + AB = \Sigma(1, 3)$$

A	B	Y	maxterm	maxterm name
0	0	0	$A + B$	M_0
0	1	1	$A + \bar{B}$	M_1
1	0	0	$\bar{A} + B$	M_2
1	1	1	$\bar{A} + \bar{B}$	M_3

$$Y = F(A, B) = (A + B)(A + \bar{B}) = \Pi(0, 2)$$

Odvozené (vektorové) logické členy

- Negácia logického súčinu - NAND

- Negácia logického súčtu - NOR

- Nerovnosť - XOR; Rovnosť NXOR

Schémy viacúrovňových LO

- Logika v SOP = DF sa nazýva dvoj-úrovňová logika pozostáva z viacerých literálov pripojených na LO
- Keďže systémy sa stavajú o viac ako dvoch LO tak pre zníženie HW požiadavok sa používajú viacúrovňové LO

Trojstupový XOR $\rightarrow Y = A'B'C + A'BC' + AB'C' + ABC == A \oplus B \oplus C$

Figure 2.30 Three-input XOR:
(a) functional specification and
(b) two-level logic implementation

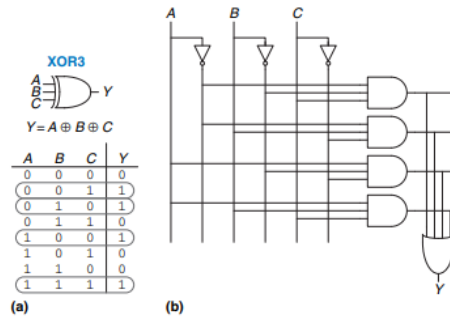


Figure 2.31 Three-input XOR
using two two-input XORs

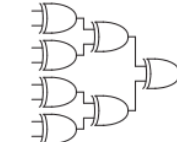
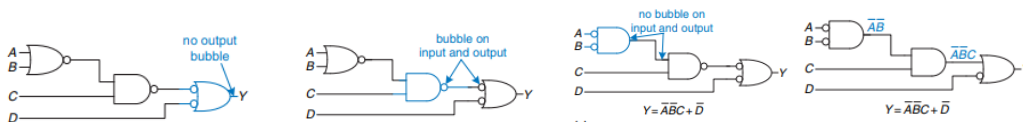


Figure 2.32 Eight-input XOR using
seven two-input XORs

Bubble pushing \rightarrow používa sa na prekreslenie obvodov NOT bubliny sa tak medzi sebou zrušia



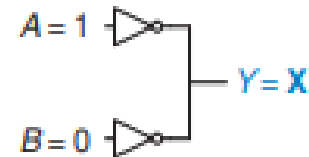
Viachodnotová logika je logický systém, ktorého výrazy nadobúdajú v interpretácii viac ako dve pravdivostné hodnoty.

Význam X a Z

- Boolovské hodnoty sú obmedzené na 0 a 1
- LO môžu nadobúdať nepovolené (X) a float hodnoty (Z)

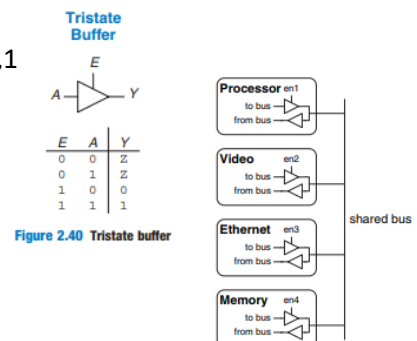
X – indikuje že obvod má neznámu alebo zakázanú hodnotu – „don't care“

- zvyčajne sa to stáva ak má obvod hodnotu 0 a 1 zároveň
- situácia sa nazýva spor (contention) a je považovaná za chybu je treba sa jej vyhnúť
- hodnota prúdu je teda niekde medzi 0 a Vdd záleží na hradlách nachádza sa v zakázanej zóne a môže vyústiť v poškodenie obvodu



Z – indikuje že uzol nieje ani v hodnote HIGH ani LOW

- mylnou predstavou je že je to to isté ako 0, ale v skutočnosti je to 0,1 alebo nejaká hodnota medzi
- nieje priamo považované za chybu
- príkladom vzniku je predstava že nepripojený vstup je 0
- jednou z možností využitia je trojstavový buffer má tri možnosti výstupu 0, 1, Z ak je $E = 1$ tak obvod povoľuje výstup Z toto sa využíva v zberniciach aby mohol so zdieľanou pamäťou kedykoľvek ktokoľvek komunikovať



Karnaughové mapy

- Boolovské výrazy môžeme minimalizovať kombinovaním požiadavok
- KM minimalizujú rovnice graficky

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Y	AB	00	01	11	10
C	0	1	0	0	0
1	1	1	0	0	0

Y	AB	00	01	11	10
C	0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

Pravidlá- Príslušné hodnoty (0,1) musia byť zakrúžkované minimálne raz

- Každé zakrúžkovanie musí obopínať bunky 2^x (1,2,4,8,...)
- Každé zakrúžkovanie musí byť tak veľké ako je to možné
- Môže prechádzať aj cez hrany
- Nepotrebné hodnoty (v zakázanej zóne..menia sa s prúdom, teplotou,...indikujú chybu ich reálna hodnota je niekde medzi 0 a 1) sú krúžkované len v prípade že znížia počet rovníc

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1

Y	AB	00	01	11	10
CD	00	1	0	X	1
01	1	0	X	X	1

3.3 Kombinačné LO

- Výstup kombinacného obvodu závisí len od skutočnej okamžitej kombinácii vstupných hodnôt
- Je opakom sekvencnej logiky ktorá má pamäť, kombinacná pamäť nemá keďže závisí len od okamžitej kombinácii vstupných parametrov

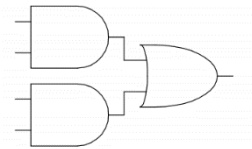
- môžeme sa na ne pozeráť ako na sústavu jedného alebo viacerých vstupov/výstupov, funkcionálnu špecifikáciu opisujúcu vzťah medzi vstupmi a výstupmi, časovú špecifikáciu opisujúcu oneskorenie
- sú zložené z uzlov a prvkov

Prvok – je samotný obvod so vstupmi, výstupmi a špecifikáciou

Uzol – je kábel ktorého napätie má hodnotu. Sú klasifikované ako vstup (prijíma dáta), výstup (dáva dáta) a vnútorné uzly (prepoj ktorý nie je vstup ani výstup)

- každý prvok obvodu je kombinačný
- každý uzol je vstupom alebo výstupom v obvode
- neobsahuje cyklické cesty
- využíva sa na stavbu komplexnejších systémov
- Funkcia logického kombinačného obvodu môže byť zadaná pomocou:

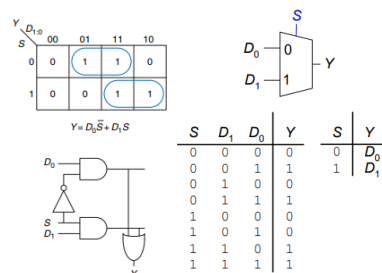
ústní definície (mějme např. zadaná čísla 4270/1563),
matematického výrazu,
seznamu indexů,
názvu kódu (binární, hexadecimální, oktalový, BCD, BCD+3, Gray, Johnson, Aiken, 1 z 10...),
pravdivostní tabulkou



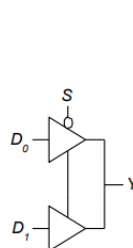
Multiplexor

- Je to kombinačný logický obvod, ktorý pracuje ako riadený elektronický prepínač. Podľa kombinácie bitov na adresových vstupoch (na obr. sú to vstupy C1 a C2) prepína jeden z N údajových vstupov (na obr. sú to vstupy S1 až S4) na jeden údajový výstup (D).
- sú jednými z najviac používaných KO
- vyberajú si výstup z N možných vstupov podľa hodnoty signálu

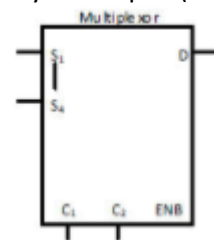
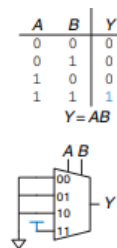
2:1 Mux



Tristate



4:1 Mux



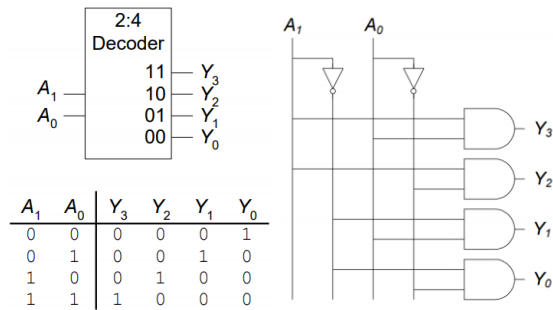
Demultiplexor

- Je opakom multiplexoru. Má jeden vstupný kanál a niekoľko výstupných kanálov. Podobne, ako v prípade Mux, sa distribúcia jednobitového vstupného slova X na jeden z jeho výstupných kanálov uskutočňuje prostredníctvom riadiacich vstupov C1, C2, ..., Cr, kde $r = \log_2 n$.

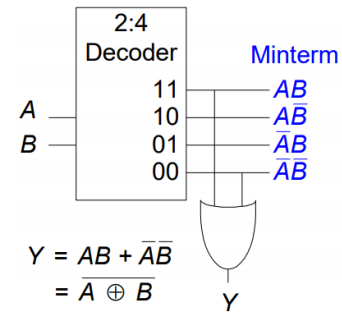
Dekóder

- Má N vstupov a 2^N výstupov

- V jednom okamihu je HIGH iba jeden výstup



Tento OR koncept využíva ROM



Časové charakteristiky KO

- čas medzi zmenou vstupu a zmenou výstupu
- využívajú sa na synchronizáciu obvodov a na stavbu rýchlych obvodov
- každému výstupu trvá nejaký čas kým vstup spracuje – oneskorenie

Oneskorenie je spôsobené

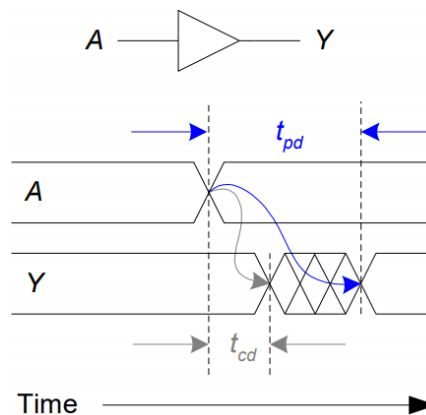
- odporom v obvode
- obmedzuje ho tiež rýchlosť svetla

Propagation delay: t_{pd} = maximálne oneskorenie zo vstupu na výstup

Contamination delay: t_{cd} = minimálne oneskorenie zo vstupu na výstup

Časy t_{pd} a t_{cd} sa líšia aj z

- dôvodom zvyšovania a znižovania oneskorenia,
- viacero vstupov a výstupov ma za následok že niektoré sa spracujú skôr ako iné,
- fyzikálne obmedzenia (obvod je rýchlejší ak je chladný)



4.1 Stavebné prvky pamäťových blokov

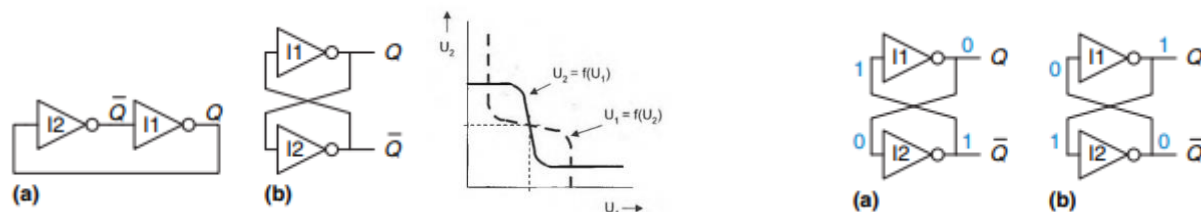
Pamäte

- na zapamätanie polí binárnych reťazcov
- funkcia pamäte v operačnej časti ČS
- statická pamäť typu RAM (Random Access Memory)
- dynamická pamäť typu DRAM (Dynamic RAM)

Funkcia pamäte v riadiacej časti ČS

- statická pamäť typu ROM (Read Only Memory)
- jednorázovo programovateľná pamäť typu PROM (Programmable ROM),
- preprogramovateľná pamäť typu EPROM (Erasable PROM),
- RWM (Read Write Memory)

- výstup sekvenčnej logiky závisí od aktuálnych a predchádzajúcich vstupoch – majú pamäť
 - stav = informácia o obvode potrebná na definovanie budúceho správania
 - Latch a flip-flop = stavové prvky ukladajúce jeden bit stavu
 - synchronne sekvenčné obvody = kombinačná logika sprevádzajúca flip-flops
 - bistabilný prvky – prvok s dvoma stabilnými stavmi
- a = cyklický menič b = rovnaký menič využívajúci krížové prepojenie
vstup I1 je výstupom I2 a naopak



Preklápacie obvody (PO)

- sú určené na pamätanie jednobitových slov
- jeden alebo viac vstupov I1, I2, ..., In (vstupné kanály)
- dva navzájom komplementárne výstupy Q' a Q (výstupné kanály).
- pamäťové správanie rôznych typov PO je interpretované spravidla Mooreovým automatom.
- astabilné, monostabilné, bistabilné
- synchronne vs. Asynchronne

Poda typu synchronizácie

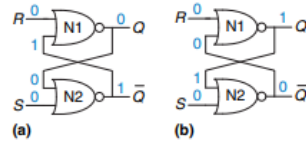
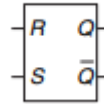
- Synchronizácia úrovňou hodinového signálu (úrovňová alebo hladinová synchronizácia) (Level Triggered Latch)
- Synchronizácia nábežnou hranou hodinového signálu (Positive edge triggered flip-flop)
- Synchronizácia zostupnou hranou hodinového signálu (Negative edge triggered flip-flop)

Asynchronne preklápacie obvody

- Najjednoduchšie sekvenčné súčiastky
- Reagujú na zmenu vstupných signálov okamžite
- Preklápací obvod vznikne spojením dvoch negujúcich log. členov do okruhu. Takto je vytvorená spätnoväzobná slučka s celkovým fázovým posunom 360°, t.j. ide o kladnú spätnú väzbu.

SR Latch

- obsahuje 2 krížové prepojenia cez NOR hradlá
- má 2 vstupy S a R ktoré menia a resetujú stav, a 2 výstupy



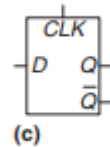
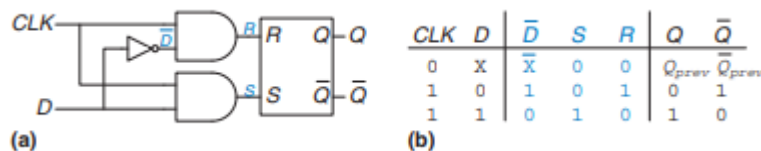
Case	S	R	Q	\bar{Q}
IV	0	0	Q_{prev}	\bar{Q}_{prev}
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

Synchronne preklápacie obvody

- Pre správnu činnosť PO je nutné dodržať niektoré dynamické parametre vstupných signálov: *doba predstihu, doba presahu, minimálna strmosť hrany hodinového impulzu, minimálna doba jeho trvania*
- Nedodržanie niektorých z týchto podmienok môže vyvolať metastabilný stav, kedy výsledné preklopenie je náhodné.
- Rozhodujúci je stav na vstupoch v tesnom okolí nábežnej hrany hodinového impulzu.
- V iných okamžikoch sú zmeny na vstupoch bezvýznamné.
- Nevhodné časovanie hodinových impulzov a vstupných signálov môže vyvolať metastabilný stav.
- Metastabilný stav má za následok nespoľahlivú funkciu preklápacieho obvodu. PO sa môže rozkmitať, alebo sa môže preklopiť do nesprávneho stavu, alebo sa môže preklopiť do správneho stavu s veľkým oneskorením. Dobu oneskorenia nie je možné predvídať.

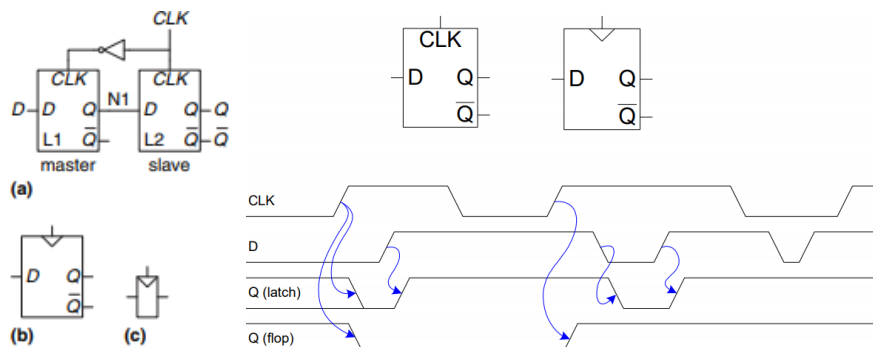
D Latch

- obsahuje dátový D (aký bude ďalší stav) a hodinový CLK signál ktorý rozhoduje o tom kedy sa ma stav zmeniť



D preklápací obvod

- obsahuje dátový D (aký bude ďalší stav) a hodinový CLK signál ktorý rozhoduje o tom kedy sa ma stav zmeniť pozostáva z dvoch D Latch obvodov jeden je slave druhý master tie sú prepojené uzlom N1.

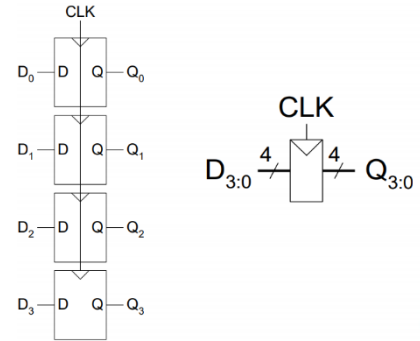


Register

- je usporiadaný súbor preklápacích obvodov (PO)
- je určený na zápis a pamätanie slov binárnych informácií
- pamäťový register vs. registrov všeobecného použitia
- označenie registra → ak $X[n-1 : 0]$, resp. $X[n-1 .. 0]$, potom $RGX[n-1 : 0]$, resp. $RGX[n-1 .. 0]$, alebo RGX

Dátový register

- Je usporiadaný súbor PO D
- Spoločný rozvod hodinových impulzov, prípadne aj spoločné nulovanie a nastavenie.
- Hladinou riadený dátový register = súbor hladinou riadených PO D
- Hranou riadený dátový register = súbor hranou riadených PO D



Posuvný register

- Kaskádne zapojenie PO D so spoločným rozvodom hodinových impulzov.

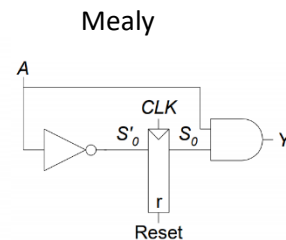
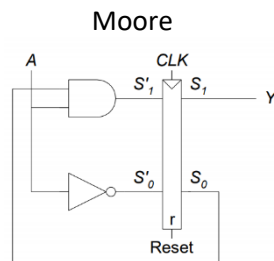
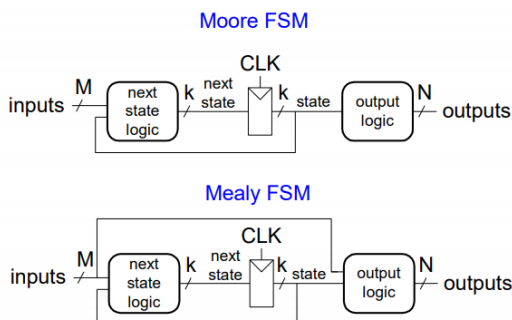
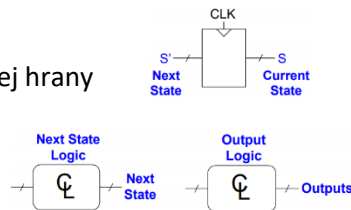
Základné operácie:

- | | |
|-----------------------------|---|
| ZS) $RGX := X$ | zápis slova X |
| NL) $RGX := 0$ | nulovanie registra RGX |
| NS) $RGX := 5$ | nastavenie registra RGX na dekadickú hodnotu slova $X = (5)_{10} = (101)_2$ |
| CT) $DO := RGX$ | čítanie obsahu RG do výstupného kanálu DO v priamom kóde |
| NCT) $DO := \overline{RGX}$ | čítanie obsahu RG do výstupného kanálu DO v obrátenom (inverznom) kóde (resp. $NCT) DO := \overline{RGX}$) |

Konečnostavové automaty (FSM)

Pozostávajú zo:

- stavový register:
 - ukladá aktuálny stav
 - naplní nasledujúci stav hodinovej hrany
- kombinačná logika:
 - vypočíta nasledujúci stav
 - vypočíta výstupy
- Nasledujúci stav je stanovený podľa aktuálnych stavov a vstupov
- Dva typy konečnostavových automatov sa odlišujú podľa výstupnej logiky
 - Moore: výstupy závisia od aktuálneho stavu
 - Mealy: výstupy závisia od aktuálnych stavov a vstupov



Navrch konecnostavovych automatov	
Identifikácia vstupov a výstupov	
Návrh stavových prechodových diagramov	
Písanie stavových prechodových tabuliek	
Výber kódovania	Moore: <ol style="list-style-type: none"> 1. Prepísanie stavovej prechodovej tabuľky so stavovým kódovaním 2. Napísanie výstupnej tabuľky
	Mealy: <ol style="list-style-type: none"> 1. Prepísanie skombinovej stavovej prechodovej tabuľky a výstupov so stavovým kódovaním
Písanie boolovských rovníc pre ďalšie stavy a výstupy	
Nákres schémy	

4.2 Synchronné sekvenčné LO

- kombinačná logika sprevádzaná flip-flops
- stav je synchronizovaný podľa času
- každý prvok obvodu je buď registrom alebo kombinačným obvodom
- aspoň jeden prvok obvodu je registrom
- všetky registre dostávajú jednotný hodinový signál
- každá cyklická cesta obsahuje aspoň jeden register

Sekvenčné obvody

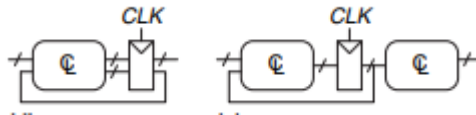
- dávajú udalostiam nejakú sekvenciu
- majú pamäť (krátkodobú)
- využívajú spätnú väzbu od výstupov pre vstupy na uloženie informácií

Dve známe SSLO

- konečnostavové automaty
- pipelines

Hazardy:

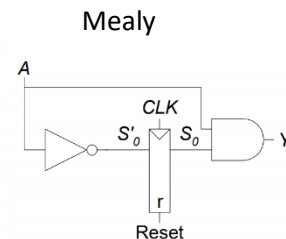
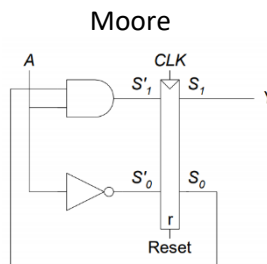
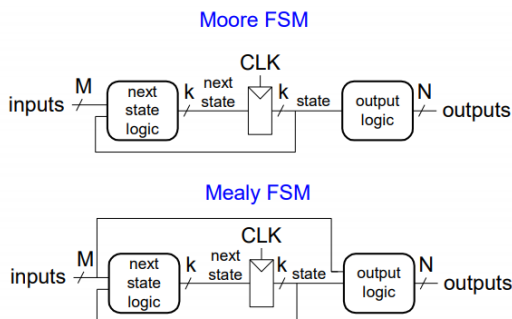
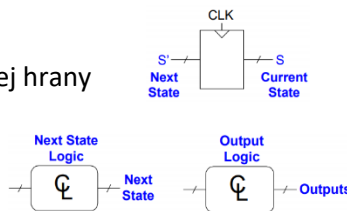
- hodiny musia byť pomalé aby sa predišlo predbiehaniu
- zložitejšie na pochopenie ľahšie na implementáciu ako asynchrónne obvody
- sú dostatočné ale preukázalo sa že sú obmedzujúce. Pri rýchlych mikroprocesoroch môžu niektoré registre prijať oneskorené hodinové cykly



Konečnostavové automaty (FSM)

Pozostávajú zo:

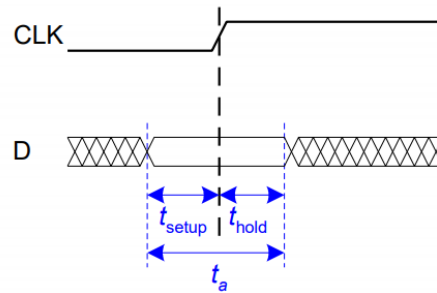
- stavový register:
 - ukladá aktuálny stav
 - naplní nasledujúci stav hodinovej hrany
- kombinačná logika:
 - vypočíta nasledujúci stav
 - vypočíta výstupy
- Nasledujúci stav je stanovený podľa aktuálnych stavov a vstupov
- Dva typy konečnostavových automatov sa odlišujú podľa výstupnej logiky
 - Moore: výstupy závisia od aktuálneho stavu
 - Mealy: výstupy závisia od aktuálnych stavov a vstupov



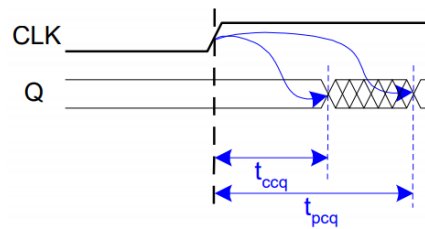
Navrch konecnostavovych automatov	
Identifikácia vstupov a výstupov	
Návrh stavových prechodových diagramov	
Písanie stavových prechodových tabuliek	
Výber kódovania	Moore: 3. Prepísanie stavovej prechodovej tabuľky so stavovým kódovaním 4. Napísanie výstupnej tabuľky
	Mealy: 2. Prepísanie skombinovej stavovej prechodovej tabuľky a výstupov so stavovým kódovaním
Písanie boolovských rovníc pre ďalšie stavy a výstupy	
Nákres schémy	

4.3 Časové charakteristiky sekvenčných obvodov

- Hodinový signál musí v SO musí byť dostatočne dlhý aby sa usadili všetky signály
- Ak nieje môžu nastať rôzne iné stavy a obvod sa stane nestabilným
- Vstupné časové obmedzenia:
 - plánovaný čas [t_{setup}] – čas kým data hodinovej hrany musia byť stabilné (nemenia sa)
 - zdržiavací čas [t_{hold}] – čas po skočení hodinovej hrany kedy dáta musia byť stabilné
 - čas prierezu [t_{aperture}] – čas v okolí hodinovej hrany kedy dáta musia byť stabilné = ($t_{\text{setup}} + t_{\text{hold}}$)

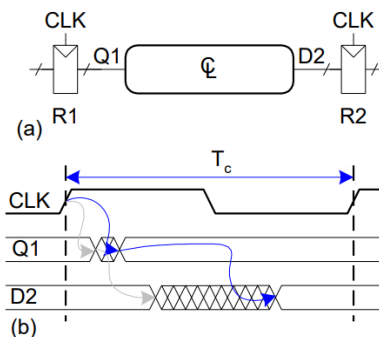


Propagation delay: t_{pd} = čas po skočení hodinovej hrany ktorý garantuje že výstup Q sa nebude meniť
Contamination delay: t_{cd} = čas po skočení hodinovej hrany ktorý umožňuje Q byť nestabilné, meniť stav



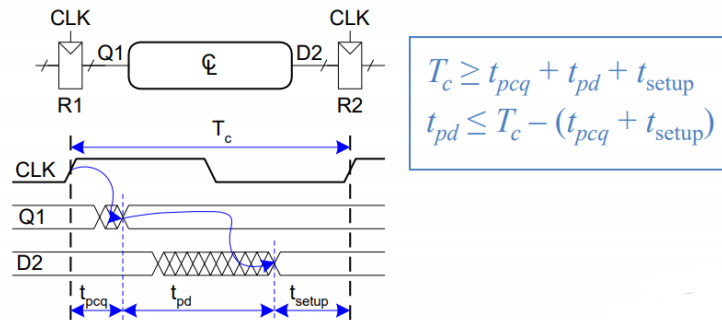
Dynamicky

- Vstupy SSLO musia byť stabilné počas t_{aperture} v okolí hodinovej hrany. Presnejšie, vstupy musia byť stabilné. Aspoň t_{setup} pred hodinovou hranou a aspoň kým t_{hold} po hodinovej hrane
- Oneskorenie medzi registrami má minimálny a maximálny delay, ktorý závisí na oneskoreniach prvkov obvodov



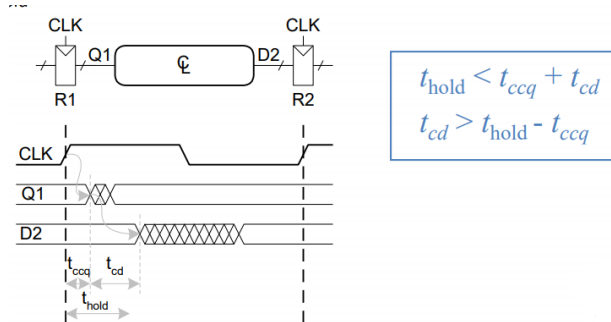
Systémové časové obmedzenia

- Závisí na maximálnom oneskorení registrov R1 cez CL na R2
- Vstup pre R2 musí byť stabilný aspoň pre t_{setup}

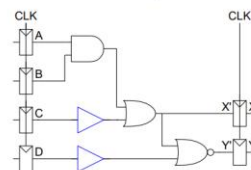


Časové obmedzenia pre Hold

- Závisí na minimálnom oneskorení registrov R1 cez CL na R2
- Vstup pre R2 musí byť stabilný aspoň pre t_{hold}



Add buffers to the short paths:



$$t_{pd} = 3 \times 35 \text{ ps} = 105 \text{ ps}$$

$$t_{cd} = 2 \times 25 \text{ ps} = 50 \text{ ps}$$

Setup time constraint:

$$T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps}$$

$$f_c = 1/T_c = 4.65 \text{ GHz}$$

Timing Characteristics

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$t_{\text{hold}} = 70 \text{ ps}$$

$$\text{per gate} \begin{cases} t_{pd} = 35 \text{ ps} \\ t_{cd} = 25 \text{ ps} \end{cases}$$

Hold time constraint:

$$t_{ccq} + t_{cd} > t_{\text{hold}} ?$$

$$(30 + 50) \text{ ps} > 70 \text{ ps} ? \text{ Yes!}$$

Hodinový sklz

- rozdiel medzi dvoma hodinovými hranami
- hodiny neprichádzajú na všetky registre v rovnaký čas
- hľadáme najhoršiu možnú analýzu pre garanciu že dynamická disciplína nieje porušená pre žiaden register – v systéme je mnoho registrov

Paralélne spracovanie

Token: Skupina vstupov spracovaných na vyprodukovanie skupiny výstupov

Latencia: Čakacia doba/ Čas za ktorý token prejde zo začiatku na koniec

Priepustnosť: Počet tokenov vyprodukovaných za jednotku času

- Paralelizmus zvyšuje priepustnosť

Typy paralelizmu:

- priestorový → duplikovaný HW robí viaceré úlohy naraz

- dočasný → úloha je rozdelená na viacero fáz, taktiež nazývaný pipelining, napríklad assembly line

- Ben Bitdiddle bakes cookies to celebrate traffic light controller installation
- 5 minutes to roll cookies
- 15 minutes to bake
- What is the latency and throughput without parallelism?

- What is the latency and throughput if Ben uses parallelism?

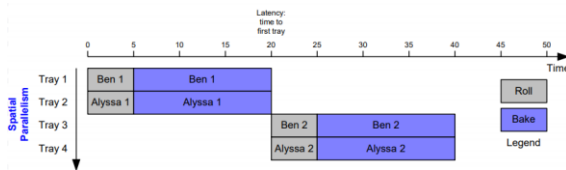
– **Spatial parallelism:** Ben asks Alyssa P. Hacker to help, using her own oven

– **Temporal parallelism:**

- two stages: rolling and baking
- He uses two trays
- While first batch is baking, he rolls the second batch, etc.

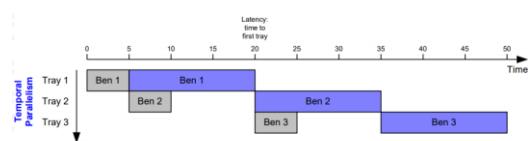
Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughput = 1 tray/ 1/3 hour = **3 trays/hour**



Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughput = 2 trays/ 1/3 hour = **6 trays/hour**



Latency = 5 + 15 = 20 minutes = **1/3 hour**

Throughput = 1 trays/ 1/4 hour = **4 trays/hour**

Using both techniques, the throughput would be **8 trays/hour**

5.1 Uvod do jazyka VHDL

HDL – z angl. Hardware description language je jazyk popisujúci štruktúru a správanie elektrických obvodov.

2 najpoužívanejšie HDL –

- VHDL 2008 = vznik 1981 Ministerstvo Obrany US
- SystemVerilog = vznik 1984 Gateway Design Automation

Dve hlavné účely HDL sú logické simulácie a syntézy. Počas simulácie, vstupy sú aplikované na modul a výstupy sú overované za účelom overenia, že modul pracuje tak ako by mal.

Počas syntézy, textový popis modulu je transformovaný na logické hradlá (gates). Logický syntetizér môže obvod optimalizovať, t.j. zredukovať počet zariadení. Netlist môže byť textový súbor alebo to môže byť schema.

Simulácia :

- Vstupy aplikované na modul
- Výstupy kontrolujúce správnosť fungovania modulu
- milióny dolárov ušetrené ladením v simulácii namiesto hardvéru

Syntéza :

- Transformuje HDL kód do netlistu popisujúceho hardvér (t. j. zoznam hradíel a káblov ktoré ich spájajú)
- Logický syntetizér optimalizuje množstvo potrebného hardware

Modul – je to blok hardveru so vstupmi a výstupmi. Napríklad AND hradlo, multiplexor alebo generator priority sú všetko príklady hardverových modulov.

Moduly – poznáme 2 typy ktorými popisujeme funkcionality modulov

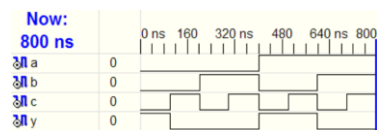
- Behavioral – popisujú, čo modul robí
- Structural – popisujú štruktúru modelu

VHDL syntax

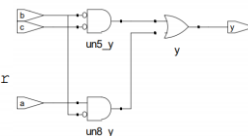
- o tento jazyk rozlišuje veľkosť písmen
- o premenné začínajúce číslom nie sú povolené
- o komentáre sa označujú nasledovne

--pre jeden riadok

/* viac riadkov */



y <= (not a and not b and not c) or
(a and not b and not c) or
(a and not b and c);



Kompozícia Behavioral VHDL:

Pripojenie knižnice

Deklarovanie portov (premenných)

Hlavný program – begin, program, end

Statements – if/else, case -> musia byť vnútri process vyzrazu

Kombinačné logické obvody

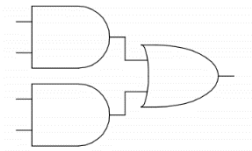
- Vystup kombinacneho obvodu zavisí len od skutočnej okamžitej kombinácii vstupnych hodnot
- Je opakom sekvencnej logiky ktorá ma pamat, kombinacna pamat nema kedze zavisí len od okamžitej kombinácii vstupnych pamametrov
- môžeme sa na ne pozerat ako na sústavu jedného alebo viacerých vstupov/výstupov, funkcionálnu špecifikáciu opisujúcu vzťah medzi vstupmi a výstupmi, časovú špecifikáciu opisujúcu oneskorenie
- sú zložené z uzlov a prvkov

Prvok – je samotný obvod so vstupmi, výstupmi a špecifikáciou

Uzol – je kábel ktorého napätie má hondotu. Sú klasifikované ako vstup (prijíma dáta), výstup (dáva dáta) a vnútorné uzly (prepoj ktorý nieje vstup ani výstup)

- každý prvok obvodu je kombinačný
- každý uzol je vstupom alebo výstupom v obvode
- neobsahuje cyklické cesty
- využíva sa na stavbu komplexnejších systémov
- Funkcia logického kombinačného obvodu môže byť zadaná pomocou:

ústní definice (mějme např. zadaná čísla 4270/1563),
matematického výrazu,
seznamu indexů,
názvu kódu (binární, hexadecimální, oktalový, BCD, BCD+3, Gray, Johnson, Aiken, 1 z 10...),
pravdivostní tabulkou



- Postup návrhu – vytvoríme pravdivostnu tabulku, minimalizujeme funkciu (karnaughova mapa), oznacíme slucky v mape, slucku zapiseme v minimalizovanej podobe, ak je treba tak slucku zapiseme v pozadovanej forme logickeho obvodu (vacsinou NAND), nakreslime schemu
- Vyrazy v kombinacnej logike – process statement, case statement, vyuziva Reduction operators(skratenie zapisu), delays a pod

5.2 strukturalny opis modulov

Výstupy sekvenčnej logiky závisia od aktuálnych i predchádzajúcich vstupných hodnotach.

Preto sekvenčná logika má pamäť. Sekvenčná logika si môže explicitne

Pamätať niektoré predchádzajúce vstupy, alebo moze destilovať predchádzajúce vstupy na menšie množstvo informácií nazývané stav systému. Stav

digitálneho sekvenčného obvodu je súbor bitov nazývaných stavové premenné, ktoré obsahujú všetky informácie o minulosti potrebné na vysvetlenie správania okruhu v budúcnosti.

VHDL používa Idiomy na popis latches, Flip-flops a FSM

- idiomy = špecifické spôsoby popisu rôznych tried logiky
- Iné štýly kódovania môžu sice správne simulovať ale vytvárajú nesprávny hardvér

latches a flip – flops su jednoduche sekvenčne logicke obvody ktore uchovavaju hodnotu stavu o veľkosti jedného bitu.

Latch pozostáva z dvoch NOR hradiel prepojených do kríža

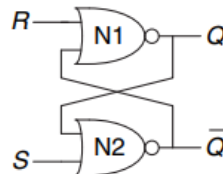


Figure 3.3 SR latch schematic

Rozdiel medzi nimi je v tom, že latch je level-sensitive (level triggering) to znamená že obvod beží len keď je signál napríklad 1 alebo 0. Flip flop beží len pri zmene stavu, tj keď sa mení 1 na 0 alebo 0 sa mení na 1, čiže je edge-triggering. Tu je to fajn vysvetlene - <https://www.youtube.com/watch?v=m1QBxTeVaNs>

Flip-flops = preklapacie obvody

Preklápací obvod (alebo bežnejšie, hoci nespisovne **klopný obvod**, skr. **KO**) je elektronický obvod s niekoľkými *stabilnými* alebo *nestabilnými stavmi*, medzi ktorými sa dokáže (na základe zmeny elektrickej veličiny na niektorom vstupe alebo vnútornej spätnej väzby) prepínať – *preklápať*. Skladá sa z niekoľkých tranzistorov, [logických hradiel](#), alebo iných [aktívnych súčiastok](#).

Preklápacie obvody majú v elektronike široké využitie ako generátory impulzov, [oscilátory](#), [statické pamäte](#), oneskorovače, [časovače](#), [čítače](#), deliče kmitočtu a pod. Na preklápacích obvodoch sú založené [elektronický digitálne obvody](#), tvoriace základ digitálnych [počítačov](#).

Vo všeobecnosti je naručne analyzovať sekvenčne obvody

Univerzálna štruktúra –

```
Process (sensitivity list)
begin
    statement;
end process;
```

Pravidla pre signalove priradenia

Synchronizujúca sekvenčná logika : používa process(clk) a neblokujúce priradenia (<=)

```
process (clk)
begin
```

```

        if rising_edge(clk) then q <= d; // nonblocking
    end if;
end process;

```

=> dobrý synchronizer využíva neblokujúce priradenia – nonblocking assignments – ktoré sa zapisujú nasledovne pomocou <=

Neblokujúce priradenia sa vykonávajú paralelne

=> zlý synchronizer využíva blokujúce priradenia ktoré sa zapisujú pomocou :=

Blokujúce priradenia blokujú vykonávanie ďalších príkazov, dokým nie je aktuálny príkaz vykonaný

Návod ako používať blokujúce a neblokujúce priradenia

VHDL

1. Use process(clk) and nonblocking assignments to model synchronous sequential logic.

```

process(clk) begin
    if rising_edge(clk) then
        n1 <= d; -- nonblocking
        q <= n1; -- nonblocking
    end if;
end process;

```

2. Use concurrent assignments outside process statements to model simple combinational logic.

```

y <= d0 when s = '0' else d1;

```

3. Use process(all) to model more complicated combinational logic where the process is helpful. Use blocking assignments for internal variables.

```

process(all)
    variable p, g: STD_LOGIC;
begin
    p := a xor b; -- blocking
    g := a and b; -- blocking
    s <= p xor cin;
    cout <= g or (p and cin);
end process;

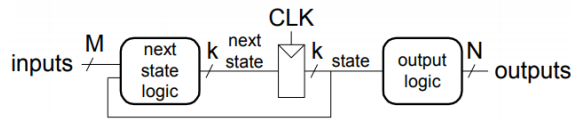
```

4. Do not make assignments to the same variable in more than one process or concurrent assignment statement.

5.3 návrh konecnostavovych automatov

Konecne stavove automaty – tri bloky :

- next state logic
- state register
- output logic



- Doteraz všetky naše moduly mali vstupy a výstupy s pevnou šírkou. Napríklad sme museli definovať samostatné moduly pre 4- a 8-bitové širokopásmové multiplexory 2: 1.
- HDL umožňujú variabilnú šírku bitov pomocou parametrizovaných modulov.

The generics a generate statements.

Testbenche – hdl s ktore testuju ine moduly

- HDL, ktorý testuje ďalší modul (nazývany jednotka pod testovaním /testovaná jednotka (uut) unit under test)
- Nie je možné syntetizovať, dá sa iba simulovať

Typy: - Jednoduché

- Sebakontroluje
- Sebakontroluje s testovacími vektormi

Testbench s Testvectors

Súbor Testvector: vstupy a očakávané výstupy

- Testbench:
 1. Generovanie hodín na priradenie vstupov, čítanie výstupov
 2. nacistie testvector suboru do pola
 3. Priradenie vstupov, očakávaných výstupov
 4. Porovnanie výstupov s očakávanými výstupmi a hlásenie chyby
- Testbench hodiny:
 - priradiť vstupy (na stúpajúcom okraji)
 - porovnať výstupy s očakávanými výstupmi (na klesajúcom okraji).
- Hodiny testbench sa tiež používajú ako hodiny pre synchronne sekvenčných obvodov

6.1 digitalne logicke obvody

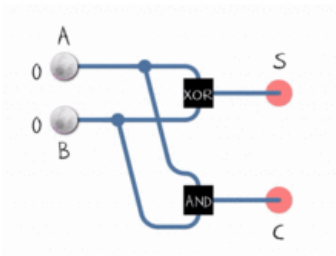
- Digitálne stavebné prvky:
 - brány, multiplexory, dekodéry, registre, aritmetické obvody, čítače, pamäťové pole, logické súbory
- Stavebné bloky demonštrujú hierarchiu, modularitu a pravidelnosť:
 - Hierarchia jednoduchších komponentov
 - dobre definované rozhrania a funkcie
 - Pravidelná štruktúra sa ľahko rozšíri na rôzne veľkosti

Sčítačky môžu byť – polovičné (half adder) alebo kompletne (full adder)

Half adder spočítava 2 samostatne binárne čísla. Má dva výstupy –

prvý je S, t.j. výsledok spočítania a druhý je C ako carry a to predstavuje hodnotu vyššieho rádu. S je operácia XOR a C je operácia AND nad dvoma hodnotami. T.j. napríklad ak spočítavame bity 1 a 1, tak výsledok sčítania S je 0 a C je 1 (teda do ďalšieho sčítavania prechádza jednotka). Pozri wikipédiu Adder (electronics), tam to je pekne vysvetlené

half adder teda pozostáva z XOR a AND hradla



Full adder má 3 vstupy kde C_{in} predstavuje prenesenú jednotku, ktorá môže byť ako výstup z half addera

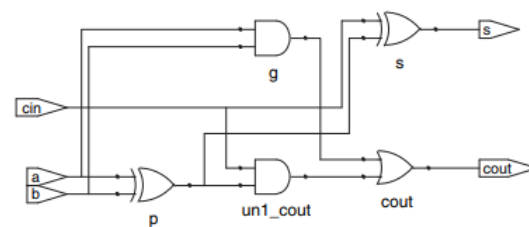
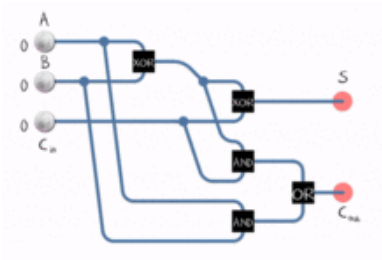


Figure 4.8 fulladder synthesized circuit

HDL Example 4.7 FULL ADDER

SystemVerilog

In SystemVerilog, internal signals are usually declared as logic.

```
module fulladder(input  logic a, b, cin,
                 output logic s, cout);

    logic p, g;

    assign p = a ^ b;
    assign g = a & b;

    assign s = p ^ cin;
    assign cout = g | (p & cin);
endmodule
```

VHDL

In VHDL, *signals* are used to represent internal variables whose values are defined by *concurrent signal assignment statements* such as $p \leq a \text{ xor } b$;

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fulladder is
    port(a, b, cin: in  STD_LOGIC;
          s, cout: out STD_LOGIC);
end;

architecture synth of fulladder is
    signal p, g: STD_LOGIC;
begin
    p <= a xor b;
    g <= a and b;

    s <= p xor cin;
    cout <= g or (p and cin);
end;
```

Odcitacky – môžu byť tiež half subtractor a full subtractor

Half subtractor odcitacka je kombinálny obvod, odcitava dva bity navzájom

Ma 2 vstupy a 2 výstupy - D ako difference a B ako borrow

D je výsledok po odčítaní a borrow je hodnota ktorú sme si „pozicali“. B nadobúda hodnotu 1 iba v jednom prípade odpocítavania a to pri vstupoch 0 a 1.

Komparátor – môže byť komparátor **rovnosti** alebo **menší ako**

Je to zariadenie ktoré porovnáva 2 hodnoty a jeho výstupom je podľa typu komparátora buď či sú rovné alebo či prvé je menšie ako druhé

Aritmetická logická jednotka (ALU) je kombinálny obvod ktorý vykonáva aritmetické a bitové operácie na celociselných binárnych číslach. Je to opakom k floating-point unit (FPU) ktorá operuje na číslach s radovou čiarkou.

6.2 Reprezentacia desatinnych cisel v pocitaci

Floating-Point Representation 1

1. Convert decimal to binary (**don't reverse steps 1 & 2!**):

$$228_{10} = 11100100_2$$

2. Write the number in "binary scientific notation":

$$11100100_2 = 1.11001_2 \times 2^7$$

3. Fill in each field of the 32-bit floating point number:

- The sign bit is positive (0)
- The 8 exponent bits represent the value 7
- The remaining 23 bits are the mantissa

1 bit	8 bits	23 bits
0	00000111	11 1001 0000 0000 0000 0000

Floating-Point Representation 2

- First bit of the mantissa is always 1:
 - $228_{10} = 11100100_2 = 1.11001 \times 2^7$
- So, no need to store it: *implicit leading 1*
- Store just fraction bits in 23-bit field

1 bit	8 bits	23 bits
0	00000111	110 0100 0000 0000 0000 0000
Sign	Exponent	Fraction

Floating-Point Representation 3

- *Biased exponent*: bias = 127 (01111111_2)
 - Biased exponent = bias + exponent
 - Exponent of 7 is stored as:

$$127 + 7 = 134 = 0x10000110_2$$
- The **IEEE 754 32-bit floating-point representation** of 228_{10}

1 bit	8 bits	23 bits
0	1000110	110 0100 0000 0000 0000 0000
Sign	Biased Exponent	Fraction

in hexadecimal: **0x43640000**



Počítadlo -

V digitálnej logike a výpočtovej technike je počítadlo zariadenie, ktoré uchováva (a niekedy zobrazuje) počet prípadov, kedy nastala konkrétna udalosť alebo proces, často vo vzťahu k hodinovému signálu. Najčastejším typom je sekvenčný digitálny logický obvod so vstupnou čiarou nazývanou hodiny a viacero výstupných riadkov. Hodnoty na výstupných riadkoch predstavujú číslo v binárnom alebo BCD čísle. Každý impulz aplikovaný na vstupy hodín zvyšuje alebo znižuje počet v počítadle.

Registre – su to obvody ktore vacsinou pozostavaju z flip – flops, casto s mnohymi charakteristikami podobne ku pamati. Maju schopnost citat a zapisovat viacero bitov v case a taktiez pouzivaju adresy k tomu aby vybrali konkretny register. Na rozdiel od pamate je ze maju specialne hardverove funkcie ktore sa lisia od beznych funkcii pamate. Cize sa da povedat ze registre us pamate s pridanymi hardver funkciami. Registre us na rozhrani softveru a periferie, kedy softver do nich zapisuje informacie pre ine zariadenia a ine zariadenia zase zapisuju do registrov z ktorych softver cita

PLA a FPGA –

- PLA (programovateľné logické súbory)
 - ✓ AND pole nasledované OR polom
 - ✓ Kombinovaná logika
 - ✓ Pevné interné pripojenia
- FPGA (programovateľné polia brány)
 - ✓ pole logických prvkov (LEs)
 - ✓ Kombinovaná a sekvenčná logika
 - ✓ Programovateľné interné pripojenia

FPGA -

- Zložený z: - LE (Logické prvky): vykonavaju logiku sú prepojené s maticou

- IOE (vstupné / výstupné prvky): rozhranie s vonkajším svetom obsahujú register, budič, multiplexor a ochranné obvody

- Programovateľné prepojenie: spája LEs a IOEs

- Niektoré FPGA obsahujú iné stavebné prvky ako napr multiplikátory a RAM

- Programovateľná prepojovacia matica (Switch Matrix, Switch Boxes)

Funkcie FPGA:

- Funkciu definuje „programovateľná logika“

- Sú väčšinou volatilné zariadenia

FLASH pamäť

po pripojení na zdroj stiahne sa konfigurácia do FPGA

FPGA nie je okamžite po zapnutí funkčné a trvá niekoľko ms (~10-100) kým „nabehne“

- generátor niekoľkých hodinových signálov pomocou fázového závesu PLL (Phase Locked Loop) alebo DLL (Delay Locked Loop)

- zabudované procesorové jadro apod

Moduly

- Prídavné bloky / moduly

Embedded processors (Microblaze, Nios)

DSP bloky

Distribúovaná pamäť (On-chip Memory)

Bloky na generovanie a správu hodín (PLL a DCM)

6.3 Pamäťové obvody

Účinne ukladajú veľké množstvo dát

• 3 bežné typy:

- Dynamická pamäť s náhodným prístupom (DRAM)

- statická pamäť s náhodným prístupom (SRAM)

- Pamäť len na čítanie (ROM)

• M-bitová hodnota dát čítaná / zapísaná na každej jedinej N-bitovej adrese

RAM – • Prchavé: stratí údaje po vypnutí

• Čítanie a písanie rýchlo

• Hlavná pamäť v počítači je RAM

(DRAM)

ROM – (nejkeje cigan)

• Neprchavé: uchováva údaje pri vypnutí

• Číta rýchlo, ale písanie nie je možné

• Pamäť Flash v kamerách a digitálne fotoaparáty sú všetky ROM

Typy RAM –

○ DRAM (Dynamic random access memory)

○ SRAM (Static random access memory)

Rozdiel v tom, ako ukladajú údaje:

- DRAM používa kondenzátor

- SRAM používa meniče so striedavými spojkami

DRAM

DRAM bola vynajdena v roku 1966 ujom Dennardom ktorý pracoval v IBM. Ľudia boli zpoiatku skeptickí ale do roku 1970 DRAM používali všetky počítače

- Dátové bity sú uložené na kondenzátore
- Dynamická, pretože hodnota sa musí obnoviť (prepísať) pravidelne a po prečítaní:
- Únik nabíjania z kondenzátora degraduje hodnotu
- Čítanie ničí uloženú hodnotu

SRAM je drahšia ako DRAM, väčšinou sa využíva pre CPU cache. Nevýhody sú cena, veľkosť a kapacita. Výhody – jednoduchosť, spoľahlivosť a nízka spotreba elektriny

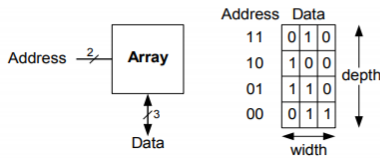
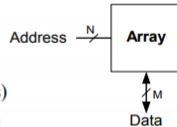
Flash pamäť

Vyniesol ju ujo Masuoka ktorý v rokoch 1971 až 1994 pracoval v Toshiba. Spolu s ním na tom pracoval len počas voľných chvíľ po pracovnej dobe ale neskôr to prerástlo do projektu ktorý ročne zarabal 25 miliónov dolárov.

- je rýchlou EEPROM pamäťou umožňuje mazať a zapisovať rýchlejšie ako EEPROM info ostáva aj po vypnutí elektriny

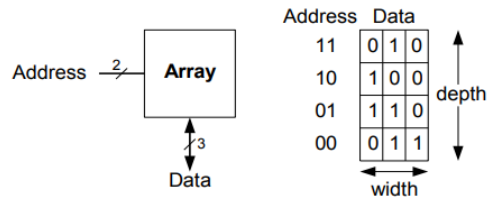
Memory Arrays

- 2-dimensional array of bit cells
- Each bit cell stores one bit
- N address bits and M data bits:
 - 2^N rows and M columns
 - **Depth:** number of rows (number of words)
 - **Width:** number of columns (size of word)
 - **Array size:** depth \times width = $2^N \times M$



Memory Array Example

- $2^2 \times 3$ -bit array
- Number of words: 4
- Word size: 3-bits
- For example, the 3-bit word stored at address 10 is 100



© 2000 Intel Corporation

PLA a FPGA –

- PLA (programovateľné logické súbory)

- ✓ AND pole nasledované OR polom
- ✓ Kombinovaná logika
- ✓ Pevné interné pripojenia

- FPGA (programovateľné polia brány)

- ✓ pole logických prvkov (LEs)
- ✓ Kombinovaná a sekvenčná logika
- ✓ Programovateľné interné pripojenia

FPGA -

- Zložený z: - LE (Logické prvky): vykonávajú logiku sú prepojené s maticou
- IOE (vstupné / výstupné prvky): rozhranie s vonkajším svetom obsahujú register, budič, multiplexor a ochranné obvody
- Programovateľné prepojenie: spája LEs a IOEs
- Niektoré FPGA obsahujú iné stavebné prvky ako napr multiplikátory a RAM
- Programovateľná prepojovacia matica (Switch Matrix, Switch Boxes)

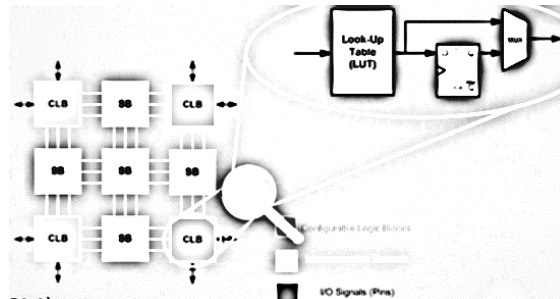
Funkcie FPGA:

- Funkciu definuje „programovateľná logika“
- Sú väčšinou volatilné zariadenia
 - FLASH pamäť
 - po pripojení na zdroj stiahne sa konfigurácia do FPGA
 - FPGA nie je okamžite po zapnutí funkčná a trvá niekoľko ms (~10-100) kým „nabehne“
- generátor niekoľkých hodinových signálov pomocou fázového závesu PLL (Phase Locked Loop) alebo DLL (Delay Locked Loop)
- zabudované procesorové jadro apod

Moduly

- Prídavné bloky / moduly

- Embedded processors (Microblaze, Nios)
- DSP bloky
- Distribuovaná pamäť (On-chip Memory)
- Bloky na generovanie a správu hodín (PLL a DCM)



7.1 Instrukčno orientovaná architektura počítače

- Instrukcie: příkazy v počítači Jazyk
 - Jazyk Assembly: ľudsky čitateľný formát inštrukcie
 - jazyk počítača: formát čitateľný počítačom (1 a 0)
- architektúra MIPS: - Vypracoval John Hennessy a jeho kolegovia na Stanford a v 80-tych rokoch. - Používa sa v mnohých komerčných systémoch vrátane Silicon Graphics, Nintendo a Cisco. Akonáhle sa naučíte jednu architektúru, ľahko sa naučíte ostatným.

Základné konštrukčné princípy, ako sú vyjadrené v Hennessy a Patterson:

1. Jednoduchosť uprednostňuje pravidelnosť
2. Rýchlym bežným prípadom
3. Smaller je rýchlejší
4. Dobrý dizajn vyžaduje dobré kompromisy

Instructions: Addition

C Code

```
a = b + c;
```

MIPS assembly code

```
add a, b, c
```

- **add:** mnemonic indicates operation to perform
- **b, c:** source operands (on which the operation is performed)
- **a:** destination operand (to which the result is written)

Multiple Instructions

- More complex code is handled by multiple MIPS instructions.

C Code

```
a = b + c - d;
```

MIPS assembly code

```
add t, b, c # t = b + c  
sub a, t, d # a = t - d
```

Machine language –

Binárne zobrazenie pokynov

- Počítače chápu len 1 a 0
- 32-bitové inštrukcie - jednoduchosť uprednostňuje pravidelnosť: 32-bitové dáta & inštrukcie • 3

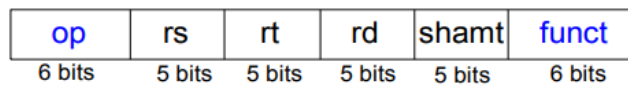
formáty inštrukcií:

- Typ R: registračné operandy ,operuje na troch registroch
- I-Type: okamžitý operand, operuje na dvoch registroch
- J-Type: na skákanie,okamzite

R-Type

- *Register-type*
- 3 register operands:
 - rs, rt: source registers
 - rd: destination register
- Other fields:
 - op: the *operation code* or *opcode* (0 for R-type instructions)
 - funct: the *function*
with opcode, tells computer what operation to perform
 - shamt: the *shift amount* for shift instructions, otherwise it's 0

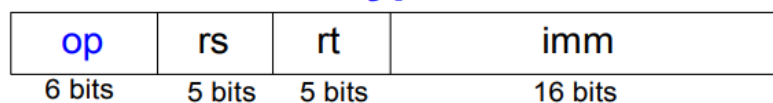
R-Type



I-Type

- *Immediate-type*
- 3 operands:
 - rs, rt: register operands
 - imm: 16-bit two's complement immediate
- Other fields:
 - op: the opcode
 - Simplicity favors regularity: all instructions have opcode
 - Operation is completely determined by opcode

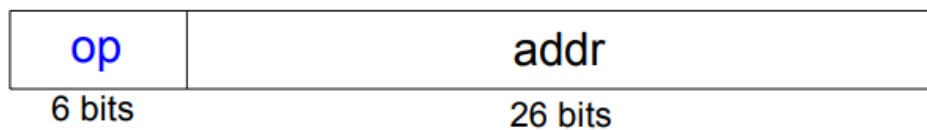
I-Type



Machine Language: J-Type

- *Jump-type*
- 26-bit address operand (`addr`)
- Used for jump instructions (`j`)

J-Type



PROGRAMOVANIE ARCHITEKTURY

Logické instrukcie –

Logical Instructions

- **and, or, xor, nor**
 - `and`: useful for **masking** bits
 - Masking all but the least significant byte of a value:
 $0xF234012F \text{ AND } 0x000000FF = 0x0000002F$
 - `or`: useful for **combining** bit fields
 - Combine `0xF2340000` with `0x000012BC`:
 $0xF2340000 \text{ OR } 0x000012BC = 0xF23412BC$
 - `nor`: useful for **inverting** bits:
 - `A NOR $0 = NOT A`
- **andi, ori, xori**
 - 16-bit immediate is zero-extended (*not* sign-extended)
 - `nor` not needed

Logical Instructions Example 1

Source Registers

\$s1	1111	1111	1111	1111	0000	0000	0000	0000
\$s2	0100	0110	1010	0001	1111	0000	1011	0111

Assembly Code

Result

and \$s3, \$s1, \$s2	\$s3	0100	0110	1010	0001	0000	0000	0000
or \$s4, \$s1, \$s2	\$s4	1111	1111	1111	1111	1111	0000	1011
xor \$s5, \$s1, \$s2	\$s5	1011	1001	0101	1110	1111	0000	1011
nor \$s6, \$s1, \$s2	\$s6	0000	0000	0000	0000	0000	1111	0100

Aritmetické instrukcie – patria sem add, sub, mult, div, atd...

7.2 Programovanie architektury MIPS

Vetvenie (branching) – moze byt podmienene alebo nepodmienenene

Vykonaju sa pokyny mimo poradia

- Druhy vetvenia:

- podmienené

- vetva, ak je rovná (beq)
- vetva, ak nie je rovnaká (bne)

- nepodmienenene

- skok (j) jump
- skokový register (jr) jump register
- skok a prepojenie (jal) jump and link

If statement - Príkaz if hodnotí testovací výraz v zátvorkách.

Ak je skúšobný výraz vyhodnotený ako true (nonzero), vyhlásenie (y) vnútri tela if je vykonané.

Ak je testovací výraz vyhodnotený ako false (0), vyhlásenie (y) vnútri tela if sa preskočí z vykonania.

While loop – prikaz sa vykonava pokym je zadana podmienka pravdiva

For – podobne While, ved vies....

- inicializácia: spustí sa pred začiatkom cyklu
- stav: testuje sa na začiatku každej iterácie
- operácia slučky: vykoná sa na konci každej iterácie
- vyhlásenie: vykoná sa vždy, keď je splnená podmienka

Zivotny cyklus programu –

Fázy životného cyklu programu sú fázy, ktoré počítačový program prechádza, od počiatočného vytvorenia po nasadenie a vykonávanie. Fázy sú čas úpravy, čas zostavenia, čas spojenia, čas distribúcie, čas inštalácie, čas načítania a čas spustenia.

Fázy životného cyklu nemusia nevyhnutne prebiehať v lineárnom poradí a môžu sa navzájom prepojiť rôznymi spôsobmi. Napríklad pri úprave programu môže vývojár softvéru opakovane upravovať, kompilovať, inštalovať a vykonávať na svojom počítači, aby zabezpečil dostatočnú kvalitu predtým, než bude distribuovaný používateľom; kópie upraveného programu sú potom stiahnuté, nainštalované a vykonané používateľmi v ich počítačoch.

Adresovacie mody –

Ako sa adresujeme operandy?

- Len registrovať
- Okamžite
- Základné adresovanie
- PC-Relatívny
- Pseudo Direct
 - ➔ určujú, ako pokyny jazyka počítača v tejto architektúre identifikujú operand (y) každej inštrukcie. Adresovací mod určuje, ako vypočítať efektívnu pamäťovú adresu operandu pomocou informácií uchovávaných v registroch a / alebo konštantách obsiahnutých v inštrukcii stroja alebo inde.

Addressing Modes

Register Only

- Operands found in registers
 - **Example:** add \$s0, \$t2, \$t3
 - **Example:** sub \$t8, \$s1, \$0

Immediate

- 16-bit immediate used as an operand
 - **Example:** addi \$s4, \$t5, -73
 - **Example:** ori \$t3, \$t7, 0xFF

MIPS pouziva 5 adresovacich modov :

1. register-only
2. immediate
3. base
4. PC-relative
5. Pseudo-direct

Prve tri definuju mody citania a zapisu operandov. Posledne dve definuju mody pisania programoveho pocitadla.

Register-only adresovanie pouziva registre pre vsetky zdrojove a cielove operandy. Vsetky instrukcie typu R pouzivaju register-only adresovanie

Immediate Addressing – pouziva 16 bit priamo spolocne s registrami ako operandami. Niektore I-Typy instrukcii, ako napríklad add immediate (addi) a load upper immediate(lui) pouzivaju priame adresovanie

Base Addressing – instrukcie s pristupom do pamate ako napríklad load word (lw) a store word (sw) pouzivaju base adresovanie. Efektivna adresa pamate operandu je najdena pridaním základnej adresy v registri RS do „sign-extended 16 bit offset“ najdeny v priamom poli.

PC-relative addressing – instrukcie podmienkových vetiev pouzivaju toto adresovanie za ucelom specifikovania novej hodnoty PC keď je vetva zobrať. Označený offset v priamom poli je pridaný do PC za ucelom obdržať nový PC, preto adresa cieľovej vetvy sa považuje za relatívnu vzhľadom na aktuálny PC

7.3 Programovanie architektury MIPS

Pole –

Prístup k veľkým množstvám podobných údajov

- Index: prístup ku každému prvku
- Veľkosť: počet prvkov
- Základná adresa = adresa prvého prvku, array [0]
- Prvý krok pri prístupe do poľa: načítajte základnú adresu do registra

Volanie funkcií (podprogramov) – pozostava z dvoch účastníkov. Prvý je volajúci (v našom prípade väčšinou funkcia main) a volany (nami implementované funkcie) volajúci:

- odovzdáva argumenty volajúcemu
- skoky na volajúceho

volany:

- vykonáva funkciu
- vracia výsledok volajúcemu
- vráti sa do bodu volania

Nesmie prepísať registre alebo pamäť, ktoré potrebuje volajúci

Zasobník –

- Pamäť sa dočasne používa na uloženie premenných
- Expanduje : používa sa viac pamäte keď je potrebný väčší priestor
- Stahuje sa : používa menej pamäte keď už nie je priestor potrebný

Rastie smerom dole (z vyššej do nižšej pamäte adresy)

- Pointer zasobníka: \$ sp ukazuje na vrchol zásobníka

Rekurzia -

Rekurzia v informatike je metóda riešenia problému, kde riešenie závisí od riešení menších prípadov toho istého problému (na rozdiel od iterácie). Tento prístup sa môže uplatniť na mnohé typy problémov a rekurzia je jednou z hlavných myšlienok počítačovej vedy.

Pri rekurzii funkcia vo svojom tele vola samu seba.

bAdresovacie mody –

Ako sa adresujeme operandy?

- Len registrovať
- Okamžité
- Základné adresovanie
- PC-Relatívny
- Pseudo Direct

➔ určujú, ako pokyny jazyka počítača v tejto architektúre identifikujú operand (y) každej inštrukcie. Adresovací mod určuje, ako vypočítať efektívnu pamäťovú adresu operandu pomocou informácií uchovávaných v registroch a / alebo konštantách obsiahnutých v inštrukcii stroja alebo inde.

Addressing Modes

Register Only

- Operands found in registers
 - **Example:** add \$s0, \$t2, \$t3
 - **Example:** sub \$t8, \$s1, \$0

Immediate

- 16-bit immediate used as an operand
 - **Example:** addi \$s4, \$t5, -73
 - **Example:** ori \$t3, \$t7, 0xFF

MIPS pouziva 5 adresovacich modov :

6. register-only
7. immediate
8. base
9. PC-relative
10. Pseudo-direct

Prve tri definuju mody citania a zapisu operandov. Posledne dve definuju mody pisania programoveho pocitadla.

Register-only adresovanie pouziva registre pre vsetky zdrojove a cielove operandy. Vsetky instrukcie typu R pouzivaju register-only adresovanie

Immediate Addressing – pouziva 16 bit priamo spolocne s registrami ako operandami. Niektore I-Typy instrukcii, ako napríklad add immediate (addi) a load upper immediate(lui) pouzivaju priame adresovanie

Base Addressing – instrukcie s pristupom do pamate ako napríklad load word (lw) a store word (sw) pouzivaju base adresovanie. Efektivna adresa pamate operandu je najdena pridaním základnej adresy v registri RS do „sign-extended 16 bit offset“ najdeny v priamom poli.

PC-relative addressing – instrukcie podmienkových vetiev pouzivaju toto adresovanie za ucelom specifikovania novej hodnoty PC keď je vetva zobrana. Označený offset v priamom poli je pridaný do PC za ucelom obdržať nový PC, preto adresa cieľovej vetvy sa považuje za relatívnu vzhľadom na aktuálny PC