7. Digitální logické obvody

Prozatím jsme se stále seznamovali se světem fyzikálních jevů / napětí, proud, teplota, tlak, změna vzdáleností, atd. /. Viděli jsme, že tyto jevy z jasných fyzikálních příčin mají spojitý charakter v čase i okamžité hodnotě. Říkali jsme jim analogové veličiny a po převodu na elektrické veličiny analogové signály $u(t) = f(t)$. Současně jsme ale zjistili, že přenos a zpracování takto definovaných veličin není jednoduché a dochází ke zkreslení signálu, a to z nejrůznějších příčin - omezená přesnost analogových měřících přístrojů a jejich „čtení“, frekvenční zkreslení v důsledku neideální amplitudové a fázové charakteristiky, nelineárních jevů a vzniku vyšších, případně kombinovaných kmitočtů a i dalších vlivů. Na druhé straně jsme již poznali, že existuje metoda, pomocí které můžeme tyto signály převést na signály diskrétního charakteru, udávající amplitudu analogového signálu v čase t_k a přitom zachováme spektrum původního signálu.

Převedeme-li informaci o amplitudě v čase t_k pomocí nějakého kódu do světa čísel / viz kapitola o A/D a D/A převodu /, můžeme pro zpracování původních analogových signálů užít zcela jinou metodiku. Musíme tedy naučit elektronické obvody pracovat s čísly, tato čísla si pamatovat, provádět s nimi operace a nakonec je i zobrazovat.

Číslo může být vyjádřeno v různých číselných soustavách. Od útlého dětství většina lidí užívá desítkovou soustavu - máme deset prstů na rukou a můžeme pomocí nich zobrazit příslušné číslo 0 až 9 . Pro větší množství užíváme desítkové soustavy - čísla 0 až 9 a poloha číslice v čísle určuje jeho váhu, a tedy např. $524.6_{10} = 5 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0 + 6 \cdot 10^{-1}$. První elektronický počítač na světě - ENIAC - počítal skutečně v této desítkové soustavě. Záhy se však přišlo na to, že pro elektronické obvody je nejvýhodnější soustava dvojková, která má pouze čísla 0 a 1 a odpovídá dvěma mezním stavům nějakého obvodu - napětí na výstupu buď není nebo je.

Tyto dva stavy můžeme nazývat různě : 1, Ano, Yes, True, High, ON, Closed ; 0, Ne, No, False, Low, OFF, Open .

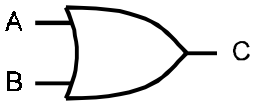
Naše číselná soustava má tedy za základ číslo 2 s možnými číslicemi 0 a 1. Potom tedy např. $11011_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 27_{10}$. Stav nějakého obvodu na výstupu může nabývat pouze dvou hodnot 0 a 1 a tento stav je ovšem závislý na stavu několika veličin vstupujících do tohoto obvodu. Bude-li počet vstupů n, bude počet možných různých stavů 2^n. Předpis, kterým se přiřazuje stav n vstupů k výstupnímu stavu, nazýváme logická funkce n - proměnných. Zřejmě je možné vytvářet složitější funkce kombinací nejjednodušších funkcí, které mají jenom dvě vstupní logické proměnné.

To nám ovšem dává možnost vytvořit jakousi základní jednotku a pro složitější funkce tuto jednotku opakovaně užít. Navíc je nanejvýš vhodné takové jednotky vytvářet jako integrované obvody pomocí monolitické technologie.

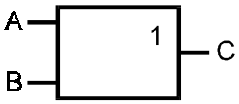
Za základní logické funkce byl vzat logický součet a logický součin. Logický součet je rozhodování typu NEBO / OR / a pro dvě logické proměnné platí rovnice $A + B = C$. Proměnné A a B mohou nabývat pouze dvou hodnot 0 a 1 , a proto proměnná C může nabývat $2^2 = 4$ stavy. Tyto stavy můžeme vyjádřit také tzv. pravdivostní tabulkou.

A	B	C = A+B
0	0	0
0	1	1
1	0	1
1	1	1

Obvod, který splňuje rovnici A + B = C , a proto má stavy podle uvedené pravdivostní tabulky, značíme ve schématech takto / viz Obr. 7-1 /.

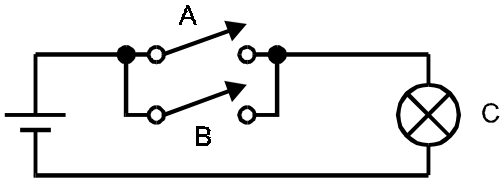


ale někdy též takto



Obr. 7-1

Činnost obvodu OR si můžeme představit podle Obr. 7-2 .

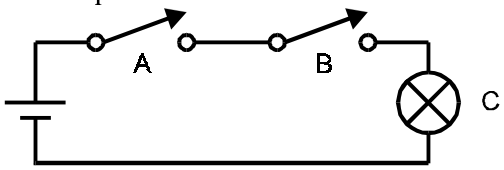


Obr. 7-2

Druhou základní logickou operací je logický součin čili rozhodování typu A / AND / , kdy platí následující pravdivostní tabulka :

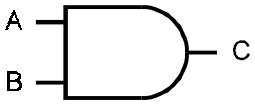
A	B	C = A·B
0	0	0
0	1	0
1	0	0
1	1	1

Činnost si opět můžeme představit podle Obr. 7-3 .



Obr. 7-3

Obvod nazýváme obvodem AND a jeho značka pro schémata je uvedena na Obr. 7-4 .

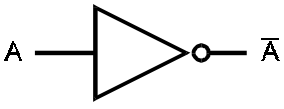


ale též

Obr. 7-4

Kromě těchto dvou základních operací se dvěma proměnnými je nutné zavést ještě operaci negace pro jednu proměnnou. Negace znamená obrácený stav $\overline{A}$ - můžeme číst A NOT . Pravdivostní tabulka a značka jsou na Obr. 7-5 .

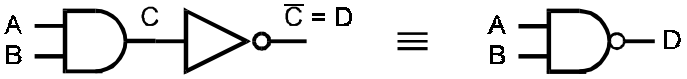
A	$\overline{A}$
0	1
1	0



Obr. 7-5

Důležitým požadavkem z hlediska realizace těchto obvodů je to, že úroveň výstupů /pro 1 a 0/ musí odpovídat úrovni vstupů /pro 1 a 0/, abychom mohli bez dalších prostředků obvody

spojovat a vytvářet tak složitější logické sítě. Provedeme takové spojení na příkladu dle Obr. 7-6 .

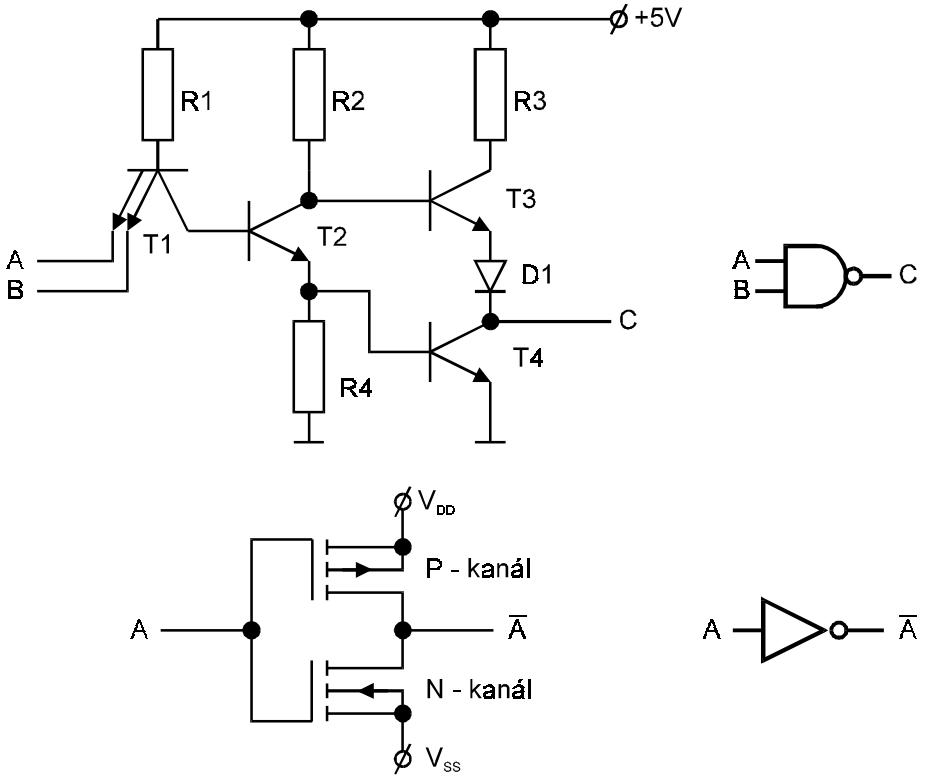


Obr. 7-6

Logickou funkci, kterou jsme tak získali, nazýváme funkcí NAND / NOT AND / a můžeme pro ni napsat pravdivostní tabulku.

A	B	C	D
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Existuje řada realizací logických obvodů, které jsou známé svými zkratkami / DCTL, RTL, DTL, TTL, I^2L, STTL, ECL, MOS, CMOS / . Z řady způsobů realizací si všimněme dvou základních - TTL / transistor - transistor logic / a CMOS / complementary metal - oxide semiconductor / - viz Obr. 7-7 .



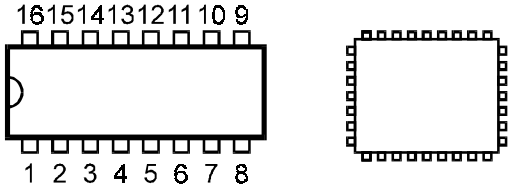
Obr. 7-7

Výhodou TTL / T^2L / logiky je jednoduché napájení V_{CC} = 5 V (+- 0.25 V) a poměrně značná rychlost změny stavů ; logika není také náchylná k statické elektřině ; nevýhodou je poměrně značná spotřeba energie a omezená hustota integrace, protože izolaci jednotlivých prvků vytváříme pomocí p - n přechodu v inverzním směru.

Pro urychlení rychlosti přeměny stavů se užívají Schottkyho diody pro ochranu před saturací a vzniká tak logika STTL, LSTTL, ASTTL .

Výhody CMOS značně převyšují nevýhody. Mezi výhody jednoznačně patří napájení 3 ÷ 15 V, podstatně omezená spotřeba / proud prochází pouze v okamžicích změny stavu, snadnější

výroba - neužívají se prvky typu rezistor a soustava je částečně samoizolující, z čehož plyne zvýšená hustota integrace, dále jsou podstatně menší nároky na napájecí zdroje z hlediska stability, vysoká vstupní impedance a výborné teplotní vlastnosti. Nevýhodou je nižší rychlost - avšak se zdokonalováním technologických procesů a vytvářením menších struktur hradel tato nevýhoda postupně mizí.
Často bývá několik jednodušších logických obvodů uzavřeno v jednom pouzdře s vývody po dvou stranách / Dual in Line Package / nebo u složitějších obvodů po všech stranách nejčastěji čtvercového pouzdra - viz Obr. 7-8 / pohled shora / .



Obr. 7-8

Zákonitosti logických funkcí nám dává tzv. **Booleova algebra** / George Boole 1815 - 1864 / :

Komutativní zákony :

1/ $A + B = B + A$

2/ $A \cdot B = B \cdot A$

Asociativní zákony :

3/ $(A + B) + C = A + (B + C)$

4/ $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

Distributivní zákon :

5/ $A \cdot (B + C) = A \cdot B + A \cdot C$

6/ $A \cdot 0 = 0$

7/ $A \cdot 1 = A$

8/ $A \cdot A = A$

9/ $A \cdot \overline{A} = 0$

10/ $A + 0 = A$

11/ $A + 1 = 1$

12/ $A + A = A$

13/ $A + \overline{A} = 1$

14/ $A \cdot (A + B) = A$

15/ $A + A \cdot B = A$

16/ $A + \overline{A} \cdot B = A + B$

17/ $\overline{\overline{A}} = A$

De Morganovy zákony :

18/ $\overline{A \cdot B \cdot C} = \overline{A} + \overline{B} + \overline{C}$

19/ $\overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$

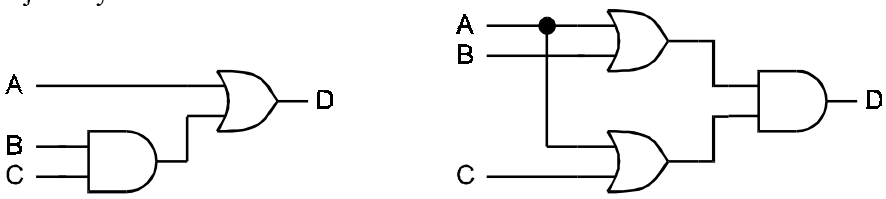
20/ $A + B \cdot C = (A + B) \cdot (A + C)$

Na základě platnosti Booleovy algebry realizace logických funkcí není jednoznačná. Ukažme si to na vztahu 20/, kdy platí $A+(B\cdot C)=(A+B)\cdot(A+C)$. Platnost si ukážeme takto :

$$(A+B)\cdot(A+C)=(A\cdot A)+(A\cdot B)+(A\cdot C)+(B\cdot C)=A+(B\cdot C)$$

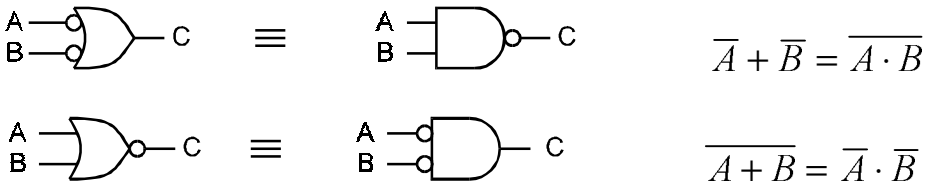
neboť $A\cdot A=A \qquad A+A\cdot B=A \qquad A+A\cdot C=A$

a dvě realizace jsou tyto - viz Obr. 7-9 .



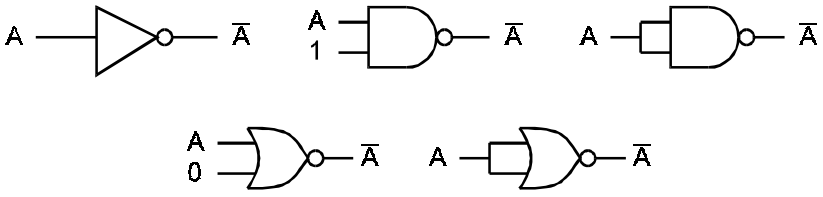
Obr. 7-9

Je možné provést tzv. „minimalizaci logických funkcí“ - pro čtyři logické proměnné /A,B,C,D/ můžeme užít tzv. Veitchových diagramu a pro větší počet proměnných počítačové programy, ale je otázka, zda tato minimalizace logické funkce bude nejvýhodnější z hlediska realizace. Proto se těmito metodami zatím zabývat nebudeme.
Z De Morganova teorému nám vyplývají následující ekvivalence - viz Obr. 7-10 .



Obr. 7-10

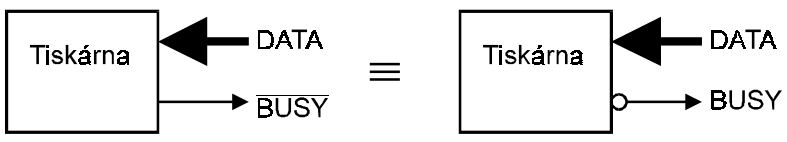
Jako invertory mohou sloužit i hradla typu NAND a NOR podle Obr. 7-11 .



Obr. 7-11

O pravdivosti těchto ekvivalencí se můžeme snadno přesvědčit na základě pravdivostních tabulek nebo rovnic Booleovy algebry.

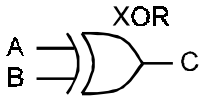
Jak jsme si již mohli všimnout, invertovanou funkci A značíme $\overline{A}$ a inverzi ve schématech kroužkem (bublinkou). Potom se mohou vyskytovat i ve schématech dvě ekvivalentní značení - viz Obr. 7-12 / například / :



Obr. 7-12

V aritmetice počítačů je důležitý ještě obvod, kterému říkáme EXLUSIVE-OR (XOR). Tento obvod se dvěma vstupy dává na výstupu 1, pokud na některém z jeho vstupů je jednička, ale dává 0, pokud je 1 na obou vstupech. Tento úkon značíme $A \oplus B = C$ a pravdivostní tabulka bude

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



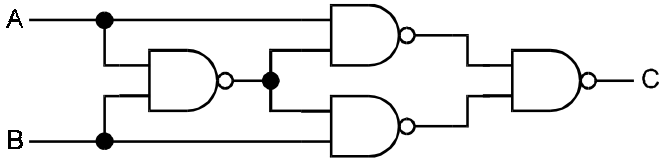
Tuto funkci můžeme realizovat na základě nám již známých funkcí takto :

$$A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B \qquad (7.1)$$

Vytvořit tuto funkci bychom mohli různým způsobem, ale nejjednodušší způsob vyplývá z této úpravy základní rovnice

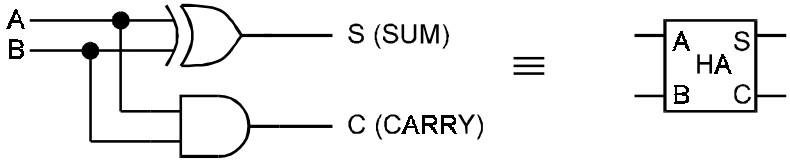
$$C = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B = A \cdot \overline{B} + A \cdot \overline{A} + B \cdot \overline{A} + B \cdot \overline{B}$$
$$C = A \cdot (\overline{B} + \overline{A}) + B \cdot (\overline{A} + \overline{B}) = A \cdot (\overline{A \cdot B}) + B \cdot (\overline{A \cdot B}) \qquad (7.2)$$

a realizace je pak na Obr. 7-13 .



Obr. 7-13

Hradlo EXLUSIVE-OR nám vlastně umožňuje provádět sčítání ve dvojkové soustavě / a tím i s dalšími obvody umožňuje odečítání, dělení, násobení a další operace /, ale potřebujeme ještě informaci o přenosu do vyššího řádu, což můžeme učinit dle Obr. 7-14 a dostáváme tzv. poloviční sčítačku - HALF ADDER (HA).



Obr. 7-14

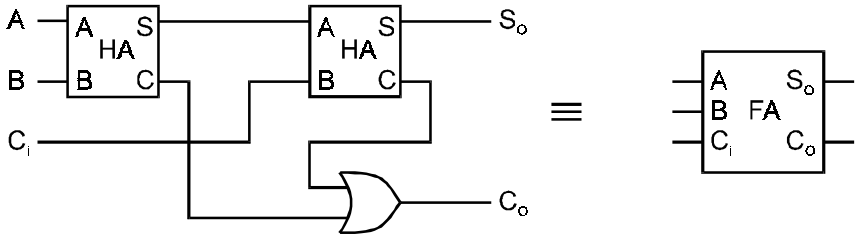
Zřejmě platí $S = A \cdot \overline{B} + \overline{A} \cdot B \qquad C = A \cdot B$ (7.3)

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Úplnou sčítačku, která má vstup pro přenos, můžeme realizovat takto - Obr. 7-15 , kdy

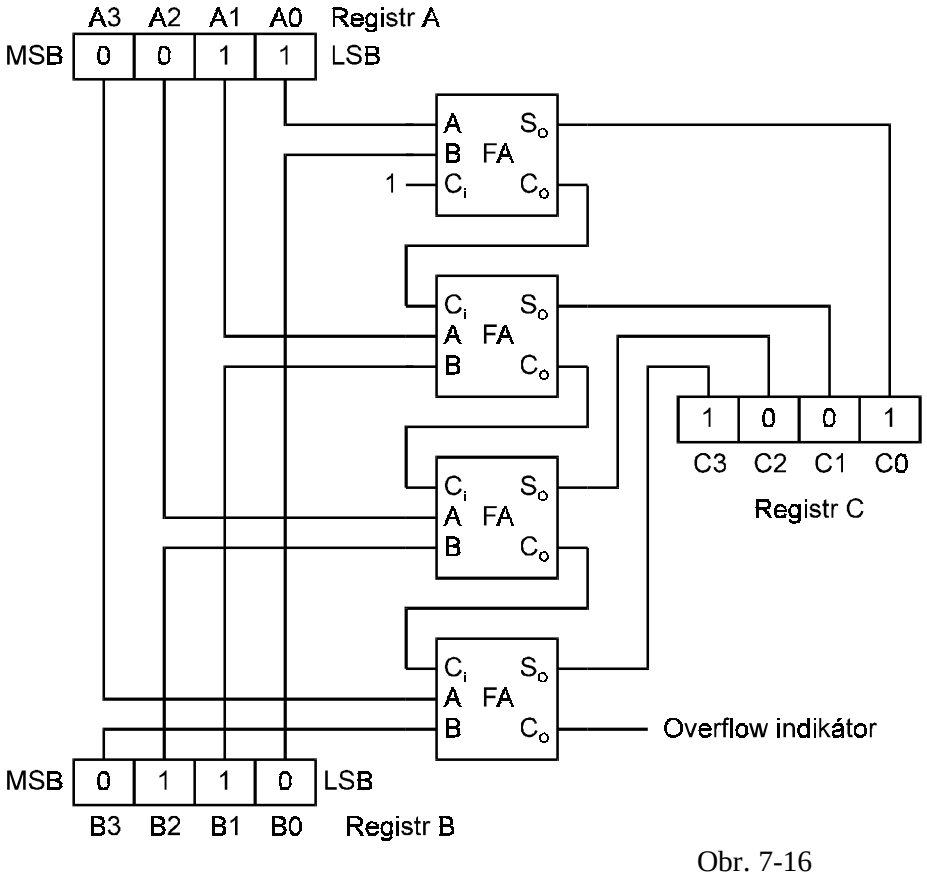
$$S_O = A \oplus B \oplus C_i \qquad C_O = A \cdot B + C_i \cdot (A \oplus B) \tag{7.4}$$

C_i = CARRY IN ; S_O = SUM OUT ; C_O = CARRY OUT



Obr. 7-15

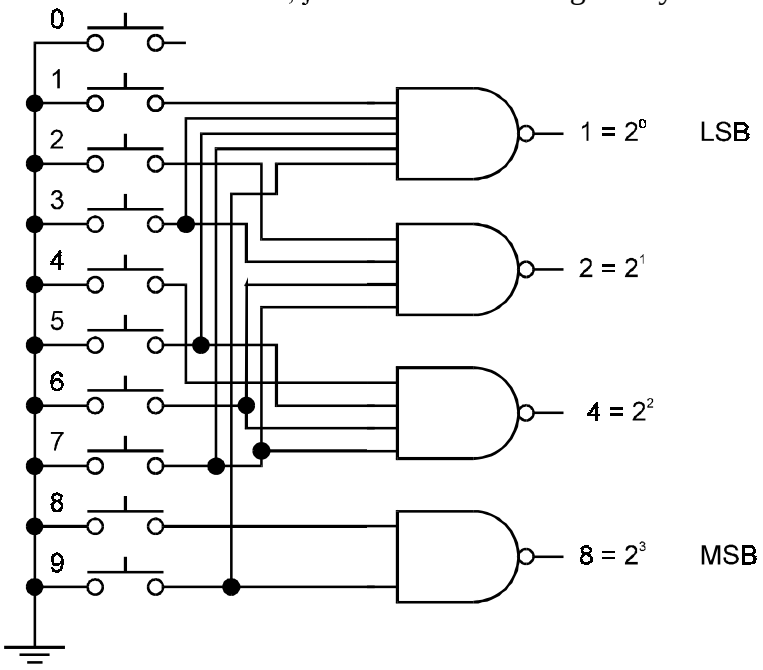
Tyto obvody je možné sdružovat do jednoho pouzdra a může tak vzniknout např. čtyřbitová paralelní sčítačka / paralelní protože užívá pro každý bit vlastní úplnou sčítačku - jiným typem je sériová sčítačka, v níž je užita jenom jedna úplná sčítačka a bity se sčítají postupně /. Čtyřbitová paralelní sčítačka je nakreslena na Obr. 7-16 / typ 7483 /. Ještě k významu zkratek na tomto obrázku : LSB = least significant bit / nejnižší významový bit / ; MSB = most significant bit / nejvyšší významový bit /.



Obr. 7-16

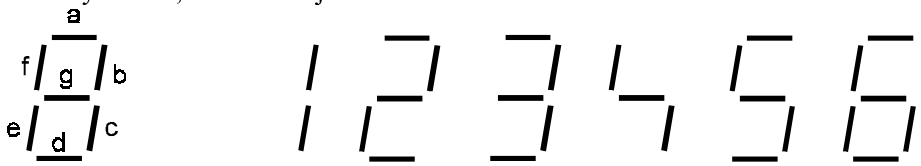
Určitě bychom neocenili kalkulačku, která by kromě tlačítek pro matematické operace měla z čísel pouze 0 a 1 a my museli pracně na papíře si převádět desítková čísla na binární / toto za

nás mohou naše jednoduché obvody vhodným způsobem zapojení uskutečnit /. Obvod, který může převádět desítková čísla na binární, je na Obr. 7-17 / integrovaný obvod 74 147 /.



Obr. 7-17

Stiskneme-li např. tlačítko označené 7, budeme mít výstup 0111 / volný nepřipojený vstup znamená jedničku /. Mám-li číslo v desítkové soustavě např. 1023_{10}, bude vyjádřeno v dvojkové soustavě $1023_{10} = 1111111111_2$. Jinak mohu číslo v desítkové soustavě vyjádřit také v kódu BCD / binary coded decimal /, kdy převádíme jednotlivé číslice : $1023_{10} = (0001)(0000)(0010)(0011)_{BCD}$. Na druhé straně potřebujeme i obrácený systém. Máme totiž jako výstup - třeba výsledek výpočtu kalkulačky - signál v BCD kódu a potřebujeme ho převést na desítkovou číslici 0 ÷ 9. K tomu slouží běžně tzv. sedmisegmentový displej - viz Obr. 7-18 , ze kterého lze rozsvícením jednotlivých segmentů vytvořit čísla 0 ÷ 9 / lze vytvořit i několik dalších symbolů, ale běžně je neužíváme /.



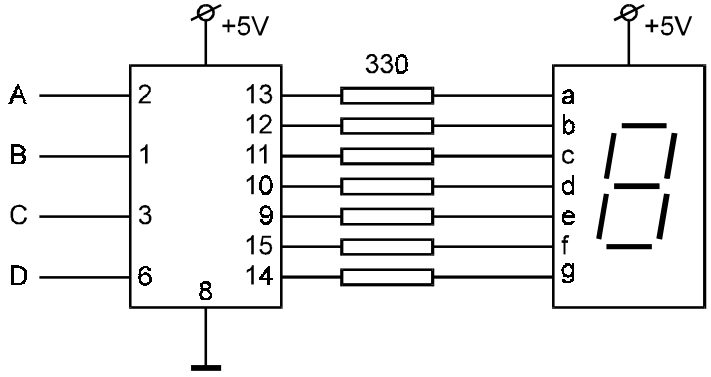
Obr. 7-18

Označíme-li jednotlivé dílčí segmenty displeje písmeny a, b, c, d, e, f, g , můžeme si na tomto případu ukázat užití logických výrazů pro tvorbu logických obvodů. Vezměme si např. segment e a vidíme, že tento segment musí svítit / tzn. je ve stavu logické 1 - čili e / v situaci, kdy mají být zobrazována čísla 0, 2, 6, 8. Proto tedy můžeme vyslovit logický soud, že naše e bude rovné jednotce, pokud bude zobrazováno číslo 0 nebo 2 nebo 6 nebo 8. Čísla dekadická 0, 2, 6, 8 máme v kódu BCD. Chceme-li realizovat základní funkce pomocí hradel AND, znamená to, že výsledkem musí být logická 1, a to bude pouze v tom případě, když budeme uvažovat $0_{10} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$. Podobně číslo $2_{10} = \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D}$; $6_{10} = \overline{A} \cdot B \cdot C \cdot \overline{D}$; $8_{10} = A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}$. Pak můžeme psát výslednou rovnici

$$e = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot \overline{D} + A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \qquad (7.5)$$

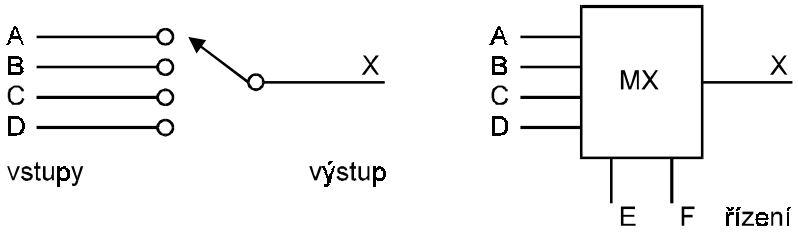
a tuto rovnici mohu realizovat pomocí poměrně složitého obvodu obsahujícího čtyři invertory, čtyři čtyřvstupová hradla AND a jedno čtyřvstupové hradlo OR .

Patrně by se vyplatilo tuto rovnici upravit pomocí zákonů Booleovy algebry nebo grafickými metodami pomocí Veitchových diagramů a dostali bychom podstatně jednodušší rovnici $e = \overline{B} \cdot \overline{D} + C \cdot \overline{D}$ a i obvod obsahující pouze dvě dvouvstupová hradla AND a jedno dvouvstupové hradlo OR a dva invertory. Řešením se více zabývat nebudeme ; chtěli jsme si pouze naznačit problém. Požadovaný převodník se totiž běžně vyrábí jako integrovaný obvod - např. 7446. Zapojení máme na Obr. 7-19 .



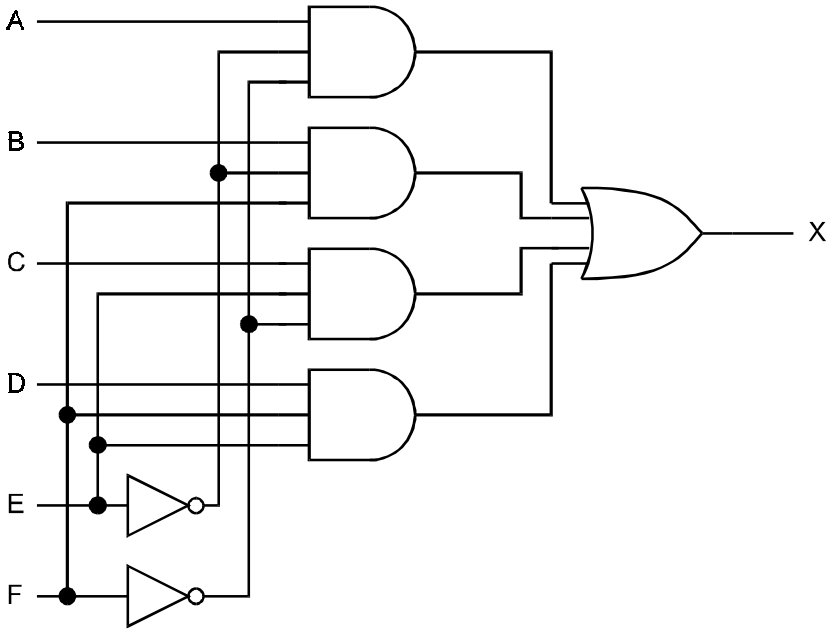
Obr. 7-19

Dalším důležitým obvodem je multiplexer. Je to vlastně přepínač řízený adresovacími vstupy, který vybere jeden z několika vstupů a připojí ho tak k výstupu / viz Obr. 7-20 /.



Obr. 7-20

Vnitřní uspořádání může být podle Obr. 7-21 .

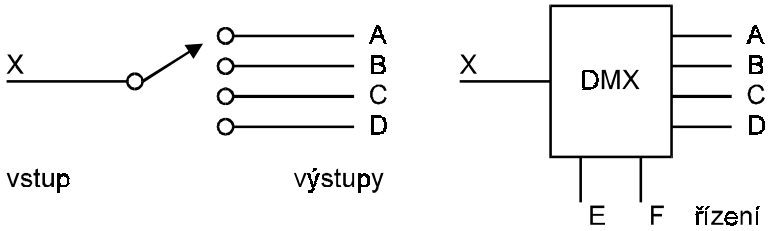


Obr. 7-21

a pravdivostní tabulka bude tato

Vybavený vstup	E	F
A	0	0
B	0	1
C	1	0
D	1	1

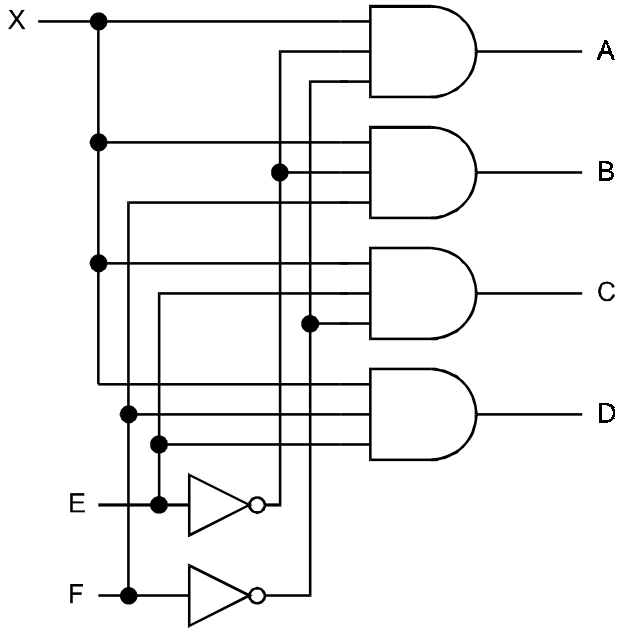
Příkladem vyráběných typů multiplexerů mohou být 74153, což je dvojitý 4:1 multiplexer a třeba 74150, což je jednoduchý 16:1 multiplexer. Multiplexery můžeme zapojovat do víceúrovňových struktur a vytvářet tak třeba z osmivstupových multiplexerů kupříkladu multiplexery 64 - vstupové.
Obrácenou funkcí zastává demultiplexer, který na základě adresy přiřadí jeden vstup několika výstupům - Obr. 7-22 .



Obr. 7-22

Vybavený výstup	E	F
A	0	0
B	0	1
C	1	0
D	1	1

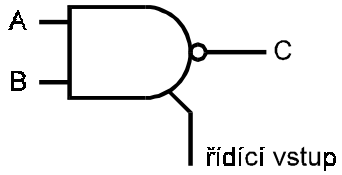
Vnitřní uspořádání může být provedeno dle Obr. 7-23 .



Obr. 7-23

Velká řada demultiplexerů je vyráběna ve tvaru integrovaných obvodů, např. 74155 (1:4) nebo 74151 (4:16) atd.

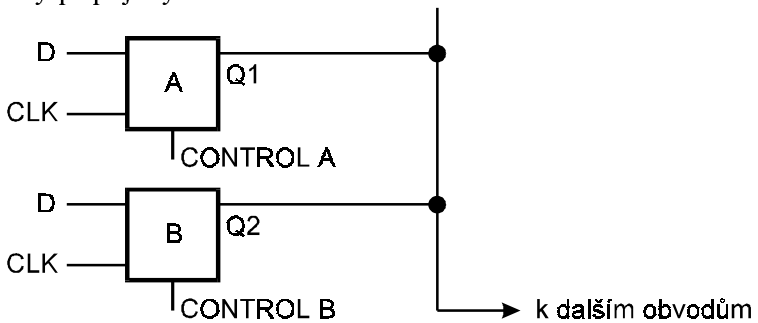
Zatím jsme mluvili o jednoduchých kombinačních obvodech, tedy obvodech ve kterých okamžitý stav výstupu závisí pouze na okamžitém stavu jeho vstupů a není vůbec závislý na předešlém stavu, to znamená, že tyto obvody nemají „paměť“.
Současně je asi jasné, že v rozsáhlých logických soustavách bude zapotřebí velkého množství spojů mezi jednotlivými logickými obvody. Řešením je vytvoření několika spojů - základních, po kterých bychom přenášeli data mezi jednotlivými obvody a k těmto spojům / sběrnicím - BUSům / bychom ve vhodném okamžiku připojovali obvody, které právě v tomto okamžiku mají spolu komunikovat. Proto musíme upravit naše obvody / některé / tak, aby byly schopné takto fungovat.
V dříve popisovaných obvodech jsme měli na výstupu jenom dva stavy - třeba HIGH, LOW. Musíme tedy doplnit tyto obvody řídícím vstupem, který umožní zachování původních dvou stavů / HIGH, LOW / v okamžiku vybavení, ale také umožní vlastně odpojení příslušného obvodu od vedení / BUSu / a tedy to, že stavy tohoto obvodu nebudou ovlivňovány ději na sběrnici / BUSu / a současně i to, že stav sběrnice neovlivní energetické poměry připojených, ale vlastně „odpojených“ obvodů.
To, o čem je zde řeč, se nazývá „třístavová logika“ / TSL /, která má kromě svých původních vstupů ještě vstup „CONTROL“, na němž závisí její činnost podle Obr. 7-24 .



Obr. 7-24

CONTROL - řídící vstup	Výstup C
LOW (Enable)	jako NAND
HIGH (Disable)	Na vysoké impedanci - „plave“

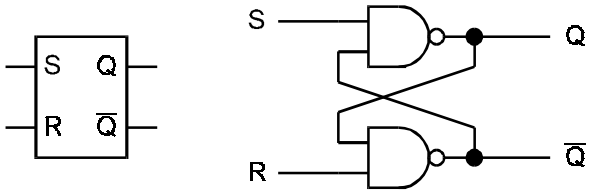
Potom budou obvody připojeny takto - Obr. 7-25 .



Obr. 7-25

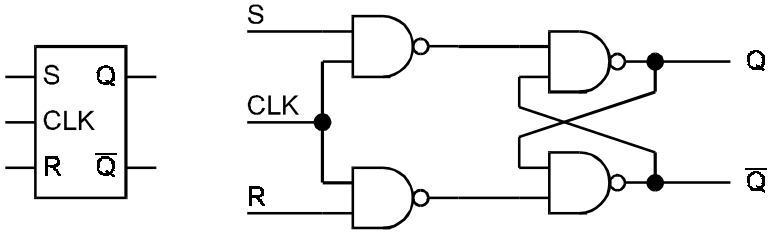
Zabývejme se nyní druhou skupinou logických obvodů, které se od předcházejících liší tím, že jejich výstupní stav závisí nejenom na aktuálním stavu jejich vstupů, ale i na stavu v němž se nacházely v minulosti. Oblast těchto obvodů se někdy nazývá „sekvenční logika“. Jedná se o klopné obvody různého uspořádání. Většinu z nich jsme však již poznali v kapitole o generaci. Proto zde uveďme pouze přehled základních typů. V kapitole o generaci byla u příslušných obvodů u vedena též logická rovnice a pravdivostní tabulka - tyto „předčasně“ uvedené popisy můžeme nyní lépe chápat a ověřit jejich pravdivost na základě principů Booleovy algebry.

Obvod R - S / nebo S - R / ; uveďme si původní obvod a jeho realizaci - Obr. 7-26 .



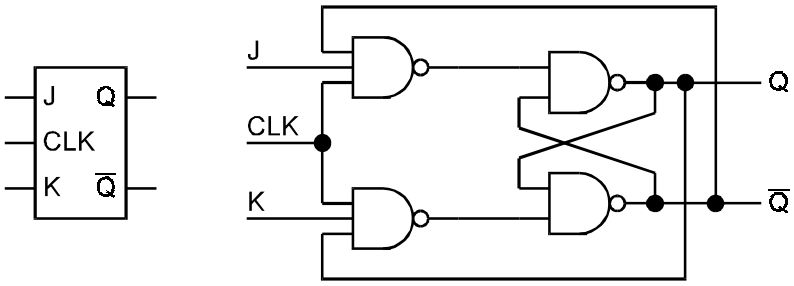
Obr. 7-26

V rozsáhlejších logických soustavách může docházet ke zpoždění reakce / viz úvahy o přechodových charakteristikách a reakcích generačních obvodů /, proto je nutné zavést do těchto složitějších obvodů jistý řád, který znemožní kumulaci zpožděných reakcí. Proto je vhodné doplnit náš uvažovaný obvod řídícím - hodinovým vstupem, který zajistí součinnost řady obvodů ve správném okamžiku - to je tzv. signál CLK / clock /. Proto tedy upravený obvod R - S bude vypadat takto - Obr. 7-27 .



Obr. 7-27

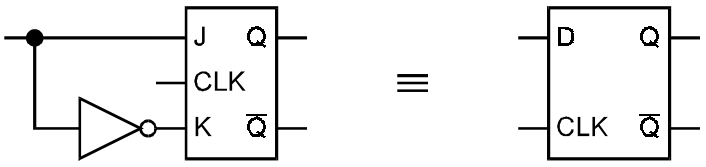
Viděli jsme v kapitole o generaci, že tyto obvody se však vyznačují neurčitým stavem - toto si v řadě případů nemůžeme dovolit, protože by zřejmě celá soustava obsahující takové obvody přestala fungovat nebo by se chovala nesmyslně. Proto byl vytvořen ještě další obvod, který tento hazardní stav nemá ; je to obvod typu J - K / viz Obr. 7-28 /.



Obr. 7-28

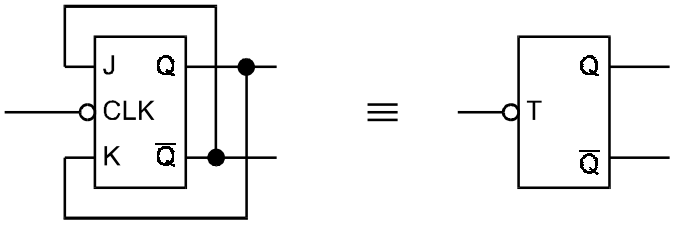
V integrované formě jsou tyto obvody vyráběny pod označením 7470 ; 7472 (TTL) - liší se způsobem spouštění.

Zapojíme-li mezi vstupy J - K obvodu invertor a obvod budeme řídit hodinovým vstupem /CLK/, dostaneme datový obvod neboli obvod D , který je schopen uchovávat data až do příchodu nového hodinového impulsu / pokud ovšem aplikujeme delší hodinový impuls, je obvod „průchodný“ - proto se užívají i složitější obvody typu D, které překlápějí pouze na hranu hodinového impulsu / - viz Obr. 7-29 .



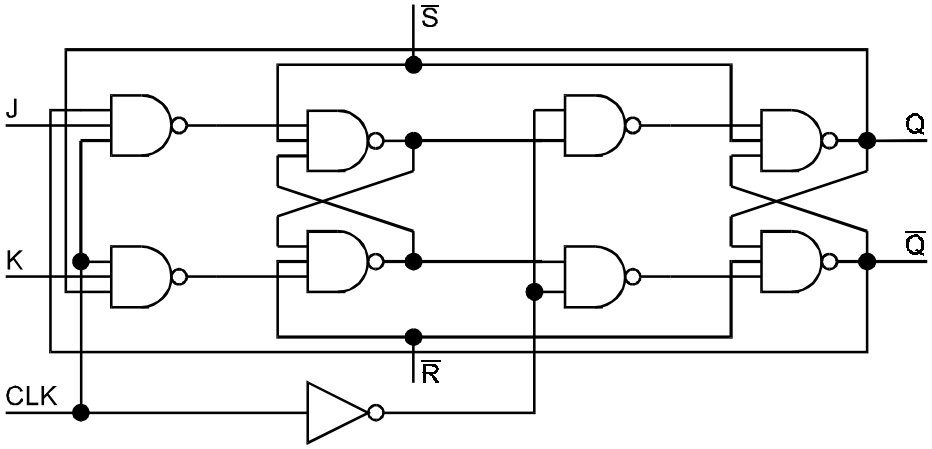
Obr. 7-29

Pro některé účely / čítače / potřebujeme obvod, který bude překlápět při každém hodinovém impulsu. Můžeme ho vytvořit na základě obvodu J - K takto - Obr. 7-30 .



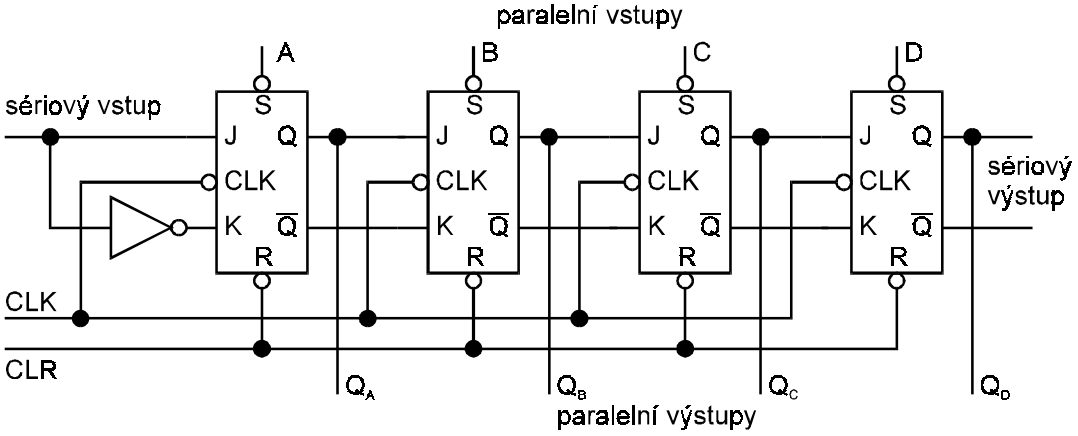
Obr. 7-30

a můžeme přemýšlet o tom, co znamená bublinka u vstupu CLK .
Posledním klopným obvodem , se kterým se potřebujeme seznámit, je obvod JK - MASTER - SLAVE. Jeho schéma je na Obr. 7-31 .



Obr. 7-31

Představitelem tohoto obvodu v integrovaném tvaru je 7072 . Tento klopný obvod je již poměrně složitý - na druhé straně však zaručuje bezpečné působení, protože napřed musí překlopit „master - pán“ a potom teprve překlápí „slave - otrok“. Podrobnější působení tohoto obvodu je však již mimo rámec tohoto skripta.
Klopné obvody se nejvíce užívají v tzv. registrech a čítačích. Uveďme si nejprve krátce problematiku registrů. Typickým představitelem může být 74194, což je čtyřbitový univerzální registr s nulováním. Struktura tohoto obvodu je již poměrně složitá, protože musí umožnit vkládání dat paralelním i sériovým způsobem, vybavení dat simultánně na paralelním výstupu, vybavení dat sériovým způsobem, posouvání dat vlevo nebo vpravo a resetování celého registru. Kromě složitého kombinačního obvodu pro řízení tohoto obvodu obsahuje čtyři klopné obvody MASTER - SLAVE zapojené podle Obr. 7-32 .



Obr. 7-32

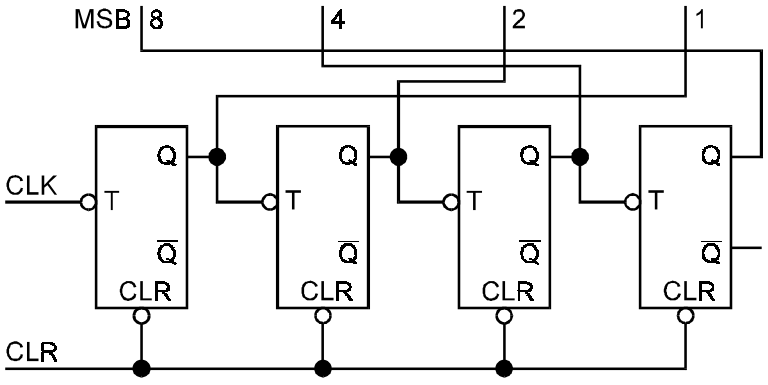
K čemu bychom mohli takový registr užít, plyne z této úvahy. Představme si, že chceme násobit dvě čísla a uvažujme násobení $5_{10} \cdot 7_{10} = 35_{10}$. Pro digitální obvody převedeme dekadická čísla na binární a dostaneme $5_{10} = 101_2$ a $7_{10} = 111_2$. Násobení proběhne tímto způsobem :

$$
\begin{array}{r}
101 \\
\times\ 111 \\
\hline
101 \\
101 \\
\hline
1111 \\
101 \\
\hline
100011
\end{array}
$$

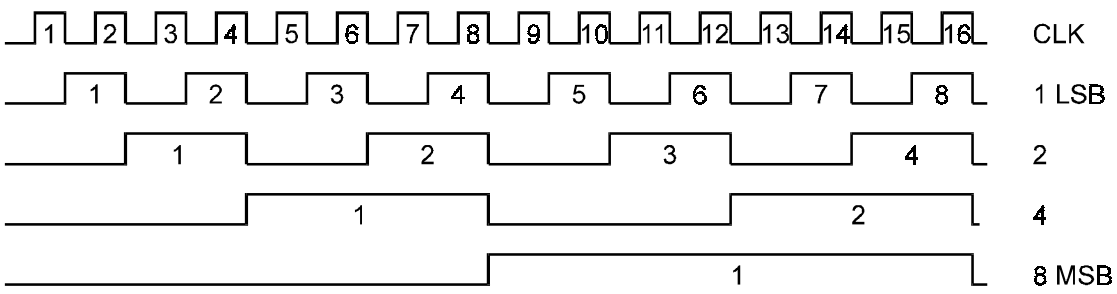
Postup je následující - násobenec je uložen do registru, v dalších krocích je násobení řízeno čísly násobitele a probíhá tak, že se postupně vybírají čísla / bity / násobitele. Je-li daný bit 1, pak se násobenec přičítá k příslušně posunutému dílčímu součinu. Je-li daný bit 0, provádí se pouze posun.

Řada různých registrů se vyrábí jako integrované obvody i jako součást obvodů vyššího stupně integrace a s délkou slova 4, 8, 16, 32 bitů.

Dalším důležitým stavebním prvkem digitálních obvodů jsou čítače. Nejjednodušším čítačem je zřejmě prostý binární čítač / binary up counter /, který užívá několika klopných obvodů v kaskádě. Čítač je vlastně registr, který dokáže přičíst nebo odečíst 1 k stavu, který je v registru obsažen. Můžeme ho vytvořit z obvodů D , ale nejčastěji z obvodů J - K pro jejich výhodné vlastnosti. Nejjednodušší je tzv. asynchronní čítač - to je prostý čítač, který není řízen hodinovými impulsy, to znamená, že čítá vstupní impulsy jak přicházejí v čase. Jednoduchý binární čítač tedy zapojíme podle Obr. 7-33 .



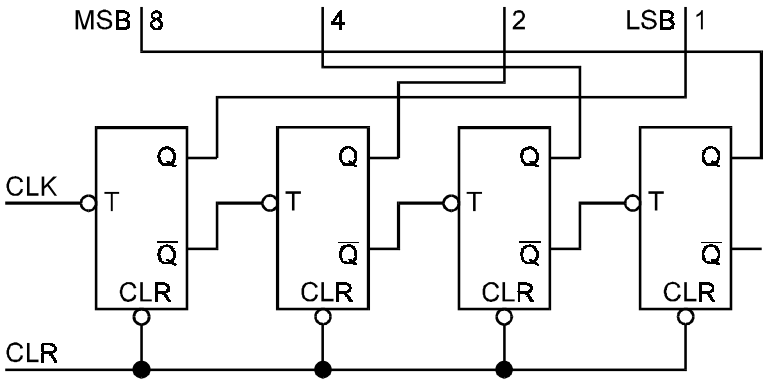
Obr. 7-33



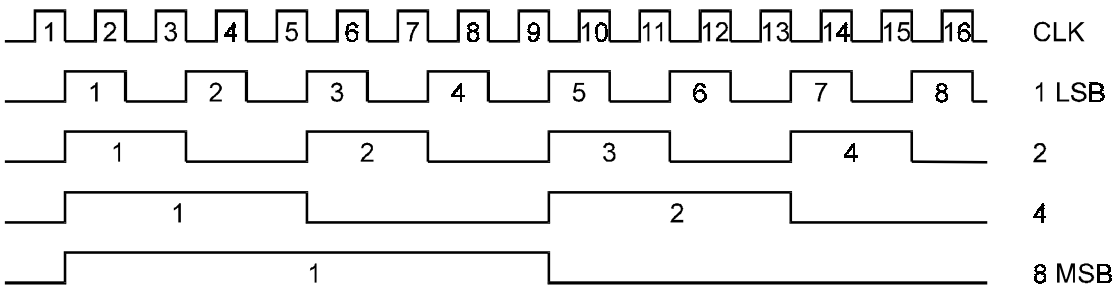
Podobný typ čítače jenom zdokonalený o řízení je k dispozici jako integrovaný obvod 7493. V řadě aplikací však potřebujeme čítač, který bude počítat přišlé impulsy tak, že je vlastně odečítá a samozřejmě jsou realizovány čítače, které mohou čítat v obou směrech.

Typickým použitím binárního čítače jsou hodinkové problémy a nejrůznější řídící obvody.

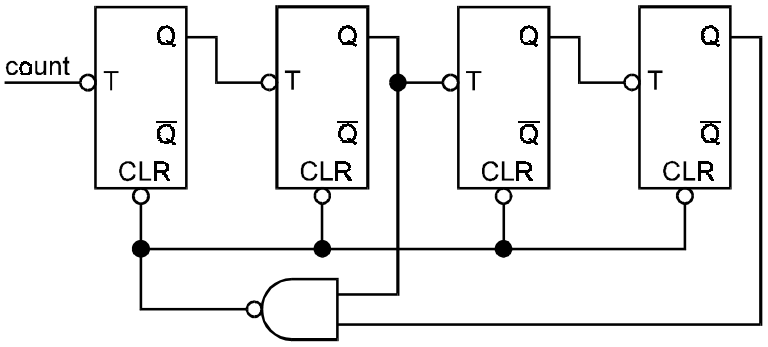
Časové průběhy vratného čítače a jeho schéma máme na Obr. 7-34 .



Obr. 7-34



Ne vždy však potřebujeme čítač, který počítá v dvojkové soustavě. Z předchozího vidíme, že čítač sestavený ze čtyř obvodů počítá do 15 a pak obnoví původní stav. Podobně kdybychom použili pouze tři obvody, obnovil by se stav po příchodu osmého impulsu. V řadě aplikací potřebujeme čítat do deseti tak jako v naší desítkové soustavě. Je zřejmé, že musíme užít čtyř klopných obvodů, ale jejich stav se musí obnovovat po příchodu desátého impulsu. Pro vytvoření takového čítače také zvaného MOD - 10 musíme užít přídavných kombinačních obvodů. Možné řešení je např. na Obr. 7-35 .

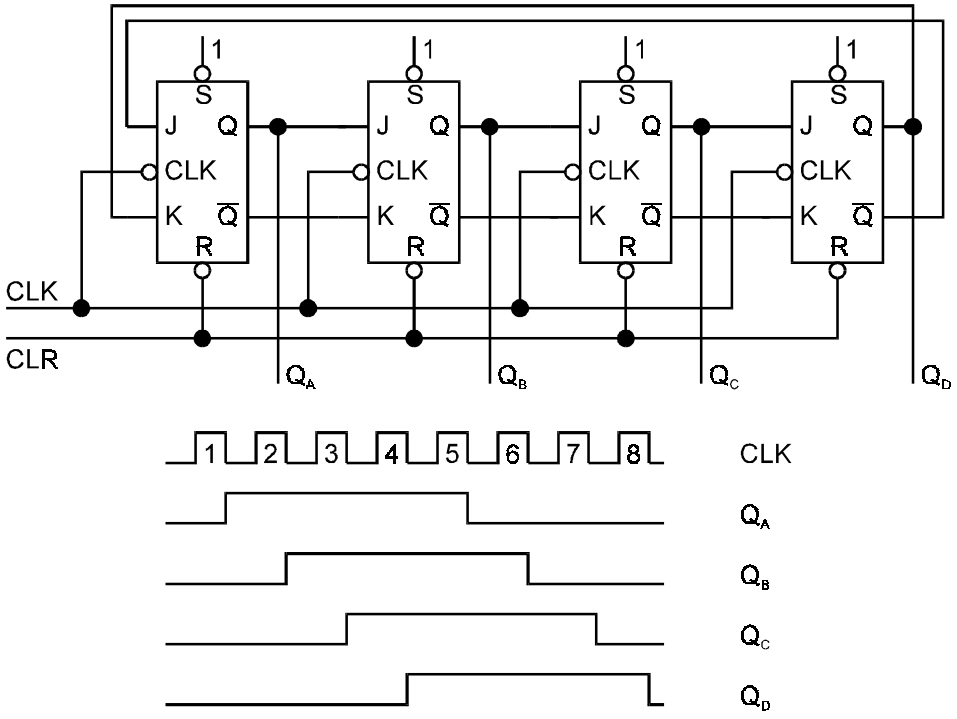


Obr. 7-35

Desítkový čítač se opět vyrábí jako integrovaný obvod 7490A, který je ovšem zapojen odlišně od Obr. 7-35 . Podobně bychom mohli za pomocí hradel vytvořit čítač MOD - n .

Čítače, které jsme zatím uvedli, jsou asynchronní, tj. jejich činnost probíhá jenom v závislosti na vstupních impulsech. Často je však nutné v soustavách užít složitější čítač, který bude řízen také vstupními impulsy, ale a to je důležité, bude měnit svoje stavy v přesně stanovených časech - to je tzv. synchronní čítač. Má tu výhodu, že se v těchto čítačích nekumuluje zpoždění jednotlivých klopných obvodů - v řadě případů tato kumulace nevadí / hodinky /, ale také

v mnoha aplikacích / počítač / hraje důležitou roli. Synchronní čítač je tedy řízen vstupními impulsy, ale mění svoje stavy v okamžiku příchodu (hrany) hodinového impulsu.
Uvedené typy čítačů měly výhodu v tom, že užívaly nejmenší počet klopných obvodů, ale mají i své nevýhody související s dekódováním jejich stavů. Může v nich tak současně docházet ke změnám stavu několika obvodů a v důsledku různých zpoždění může být dekódován falešný stav. Tuto nevýhodu nemají tzv. Johnsonovy čítače, které používají více klopných obvodů než u čítačů, které jsme zatím poznali, ale jejich stavy se mění pouze v jednom bitu. Čítač by se nejdříve zaplňoval jednotkovými bity a po naplnění znovu se začíná vyprazdňovat. Takový čítač můžeme realizovat třeba s obvody J - K tak, jak je nakresleno na Obr. 7-36 . Tyto čítače nacházejí největší použití pro světelné efekty, pro vytváření vícefázových průběhů atd.



Obr. 7-36

Kromě těchto Johnsonových čítačů se užívají i tzv. kruhové čítače, kdy výstup je spojen zpět se vstupem. Obvod velmi připomíná posuvný registr, protože po příchodu je impuls odpovídající jednotkovému stavu posune o jeden bit. Použití nachází v různých světelných efektech, běžících reklamách atd.

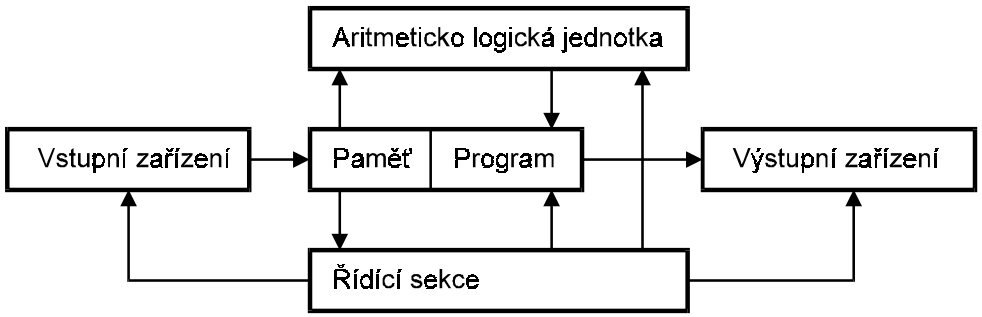
V dalších úvahách o počítačích uvidíme, že architektura většiny dnešních počítačů se stále ještě v podstatě zakládá na koncepci, kterou stanovil Johan von Neumann, kdy program a data ukládáme do operační paměti, která uchovává program i vstupní a výstupní data ve formě binárních čísel.
U dnešních systémů se používají výlučně polovodičové paměti. Tyto paměti můžeme rozdělit na paměti, kde do příslušného místa určeného binární adresou můžeme buď data zapsat nebo je přečíst. Takové paměti označujeme zkratkou RAM / Random Access Memory /. Druhou skupinou jsou paměti, kde zápis nelze již měnit, a proto z těchto pamětí můžeme pouze „číst“. Obvykle je nazýváme ROM / Read Only Memory /. Ještě jeden aspekt je nutné uvést. Ve většině případů u pamětí typu RAM se po vypnutí napájení ztrácí zaznamenaná data ; u pamětí typu ROM tato data zůstávají zachována. V pamětech ROM uchováváme konstanty nebo mikroprogramy - většinou jsou data v nich uložená přímo naprogramována výrobcem.

Kromě těchto pamětí ROM, ve kterých jsou data trvale uložena, existují paměti PROM, EPROM a EEPROM. Jsou to programovatelné paměti ROM, a data jsou zapisována přepálením tavných spojek nebo využitím lavinového nebo tunelového efektu ve strukturách MOS.
Paměti typu RAM můžeme dělit na paměti statické a dynamické. Jako statická paměť slouží pro jeden bit bistabilní klopný obvod, jak jsme ho poznali v úvahách o generaci, i když ne v tak jednoduché formě. Potřebujeme totiž další tranzistory pro zápis a čtení, čímž se taková paměťová buňka dosti komplikuje. Snaha po zvýšení kapacity pamětí vedla k požadavku snížení počtu tranzistorů a byly vytvořeny tzv. paměti dynamické, které uchovávají informaci jako náboj na kapacitoru. Protože vlivem nedokonalostí dielektrika dochází k vybíjení této kapacity, musí být informace po jisté době obnovena. Moderní rychlé počítače však za tuto dobu vykonávají stovky a tisíce operací. Tyto dynamické paměti mají kapacitu běžně 1 Mbit ale i více. Tyto paměti můžeme různým způsobem spojovat a ukládat do nich „slova“ s potřebným počtem bitů.

7.1 Mikropočítače

7.1.1 Základní informace

Základem každého počítačového systému je tzv. mikroprocesor, což je centrální procesorová jednotka / CPU / , která interpretuje a provádí tzv. instrukce, zahrnuje v sobě aritmeticko logickou jednotku / ALU /, která je schopná vykonávat základní aritmetické a logické operace / jako například operace AND a OR/ a je vyroben monolitickou technologií jako samostatný integrovaný obvod. Jestliže je mikroprocesor doplněn dalšími obvody jako jsou paměti či rozhraní / interface /, tvoří formu, které se říká mikropočítač - mohou tu ale mít též místo vstupní zařízení / klávesnice, elektronická čidla, média pro uchování dat, atd. / a výstupní zařízení / tiskárna, monitor, média pro uchování dat, atd. /. Abychom však pochopili činnost těchto složitých obvodů, podívejme se nejprve na to, jak by asi vypadal a pracoval jednoduchý počítačový systém. Na Obr. 7-37 je znázorněn základní blokový diagram takovéhoto systému.



Obr. 7-37

Počítač je řízen programem, který obsahuje sadu instrukcí / kterým přísluší operační kódy /, uložených uvnitř paměti v sekvencích a ten „říká“ počítači, co má vlastně dělat. Data používaná v instrukcích / jako operandy / poukazují na místo v paměti, kde jsou uloženy datové struktury a nazývají se operandové adresy. Pro snadnější pochopení operačních kódů se zavádí jejich mnemonický ekvivalent / konverzi mnemonických kódů do strojového jazyka operačních kódů provádí programovací jazyk assembler /.
Ukažme si na jednoduchém příkladě / A + B = C / toto : přečtěme data A ze vstupního zařízení, přičtěme k nim data B uložená v paměti, výsledek / tedy součet / zaznamenejme opět do paměti jako data C, kopii výsledku vytiskneme na tiskárně a program ukončeme. Počítač

postupně prochází každou paměťovou lokací dle příslušného programu a provádí příslušné operace dané operačními kódy. Jestliže tedy takovýto program na počítači spustíme, stane se následující činnost / viz tabulka / :

Paměťová lokace	Operační kód instrukce	Operandová adresa	Mnemonický kód
/ místo v paměti /	/ co má počítač dělat /	/ kde najdeme data /	
0000	02	-	RIN
0001	50	0020	STA A
0004	10	0020	LDA A
0007	20	0021	ADD B
000A	50	0022	STA C
000D	70	0022	PRT C
0010	99	-	HLT
.			
0020	A (05)	-	-
0021	B (03)	-	-
0022	C (08)	-	-

1/ V první paměťové lokaci / 0000 / nám instrukce / operační kód 02 ; mnemonický kód RIN / říká, abychom ze vstupního zařízení přečetli data / např. 05 / do tzv. akumulátoru neboli hlavního paměťového registru v ALU našeho ukázkového počítače.

2/ Na druhé paměťové lokaci / začínající adresou 0001 / je uložena instrukce, která ke své funkci ovšem již potřebuje argument / tedy operandovou adresu - v našem případě 0020 / a která nám říká, abychom data z akumulátoru / 05 / uložili do paměti na lokaci 0020.

3/ Další buňky paměti / tedy lokace 0004 / obsahují instrukci s operandovou adresou o mnemonickém kódu LDA A - tedy vyčisti / vynuluj / akumulátor a načti do něho obsah paměťové buňky A / tedy obsah adresy 0020, což je 05 /.

4/ Obsah dalších buněk představuje instrukci ADD B / kód 20 ; argument 21 00 /, a tato instrukce nám říká, abychom přečetli data B / třeba 03 / na lokaci 0021 a přičetli je k datům, které obsahuje akumulátor.

5/ Další instrukce je podobná instrukci uvedené v bodě 2/ s tím rozdílem, že výsledek / obsah akumulátoru : 05 + 03 = 08 / ukládáme na adresu příslušející proměnné C / 0022 /.

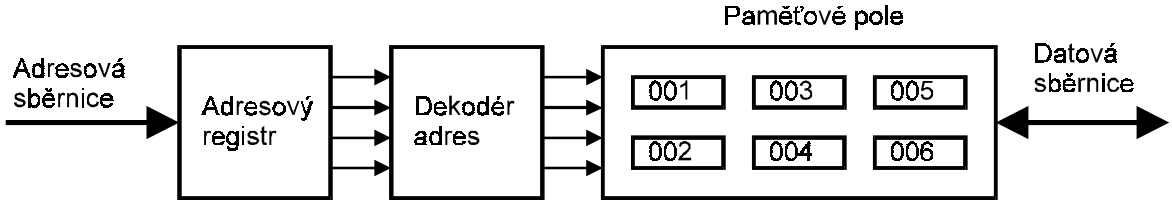
6/ Další instrukce o mnemonickém kódu PRT C provede pak kopii obsahu buňky C na výstupní zařízení / tiskárnu /.

7/ Poslední paměťová buňka, na níž je uložen náš krátký program, má jednobytovou instrukci HLT / halt / a ta říká počítači, aby ukončil svoji činnost.

7.1.2 Funkce hlavních elementů počítače

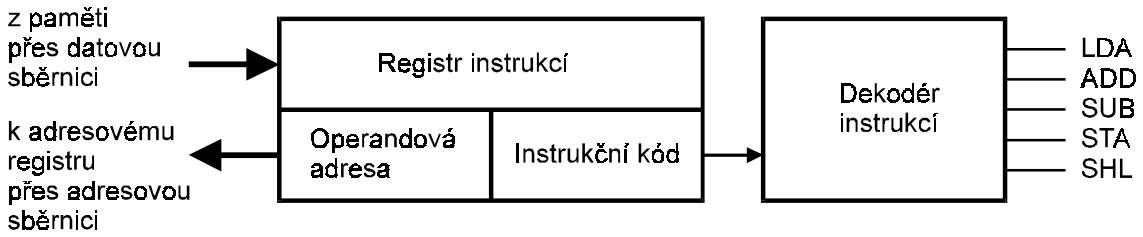
Uveďme si nyní klíčové prvky počítače nezbytných pro interpretaci a spuštění instrukcí tvořících program :

Je to jednak programový čítač / čítač instrukcí /, který obsahuje adresu další instrukce, která je má být provedena. Výstup tohoto čítače ovládá adresový registr paměti prostřednictvím adresové sběrnice. Během běhu programu impulsy inkrementují / zvětšují / obsah programového čítače tak, aby bylo zajištěno vykonání další instrukce. Pro správný start programu je ovšem nutné nastavení počáteční adresy tohoto čítače / jeho obsah může být změněn programem /.



Obr. 7-38

Dále je to adresový registr, který obsahuje adresu neboli lokalizaci buňky paměťového pole, ze které jsou data čtena nebo kam jsou zapisována. Obsah adresového registru je dekódován dekodérem adres, který vybere patřičnou buňku paměťového pole. Zápis a čtení dat probíhá prostřednictvím datové sběrnice / DATA BUS /. Vstup adresového registru je řízen z adresové sběrnice / ADDRESS BUS /, která stejně jako datová sběrnice může být připojena k čítači instrukcí, instrukčnímu registru, ALU nebo vstupně výstupnímu zařízení / I/O device /.



Obr. 7-39

Instrukční registr je použit k uchování obsahu kódovaných instrukcí spouštěných počítačem. Vstup instrukcí z paměti do registru se děje prostřednictvím datové sběrnice. Operační kód instrukce je poslán na dekodér instrukcí, pomocí kterého jsou pak nastaveny řídící obvody potřebné k vykonání instrukcí v rozličných částech počítače. Operandová adresa instrukce je zaslána na adresový registr prostřednictvím adresové sběrnice, a tím je umožněno čtení dat z paměti, které mají být zpracovány touto instrukcí.
Naposledy uveďme snad ještě akumulátor, což je nejdůležitější registr v počítači. Je využíván v mnoha operacích a výsledky některých instrukcí jsou obvykle ukládány právě do něho.

7.1.3 Instrukční cyklus a časování

Instrukce jsou prováděny počítačem pomocí řady řídících impulsů. Hlavní oscilátor, který se nazývá „hodiny“, vyrábí impulsy, které se přivádí na časovací generátor. Tento obvod vytváří z hodinových impulsů na jeho vstupu řadu výstupních signálů s odlišnými časovými průběhy.
Instrukční cyklus se skládá ze dvou částí : části, kdy je instrukce čtena z paměti / F - fetch / a části, kdy je vykonána / E - execute /. Doba vykonání operace bude zpravidla různá / závisí na typu instrukce /, a proto zde bude vyžadována celá řada časovacích průběhů. Instrukční cyklus se bude opakovat tak dlouho / program poběží /, dokud nepřijde instrukce HALT.
Uveďme si nyní příklad takového instrukčního cyklu / časování je uvedeno na Obr. 7-40 /.
Impulsy čtecího strojového cyklu na pěti výstupech / 1 až 5 /:

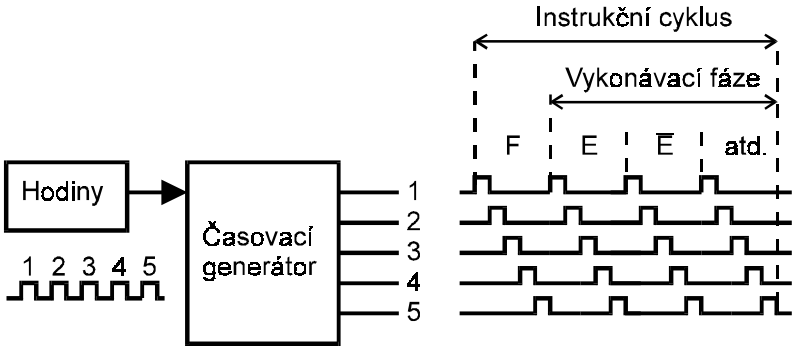
F_0 - Přenos obsahu programového čítače do paměti adresového registru

F_1 - Přenos instrukce z paměti do instrukčního registru

F_2 - Přenos operačního kódu do dekodéru instrukce a operandové adresy do adresového registru

F_3 - Přenos dat z paměti na dané určení

F_4 - Zvětšení obsahu programového čítače



Obr. 7-40

Impulsy vykonávacího strojového cyklu na výstupech 1 až 5 :

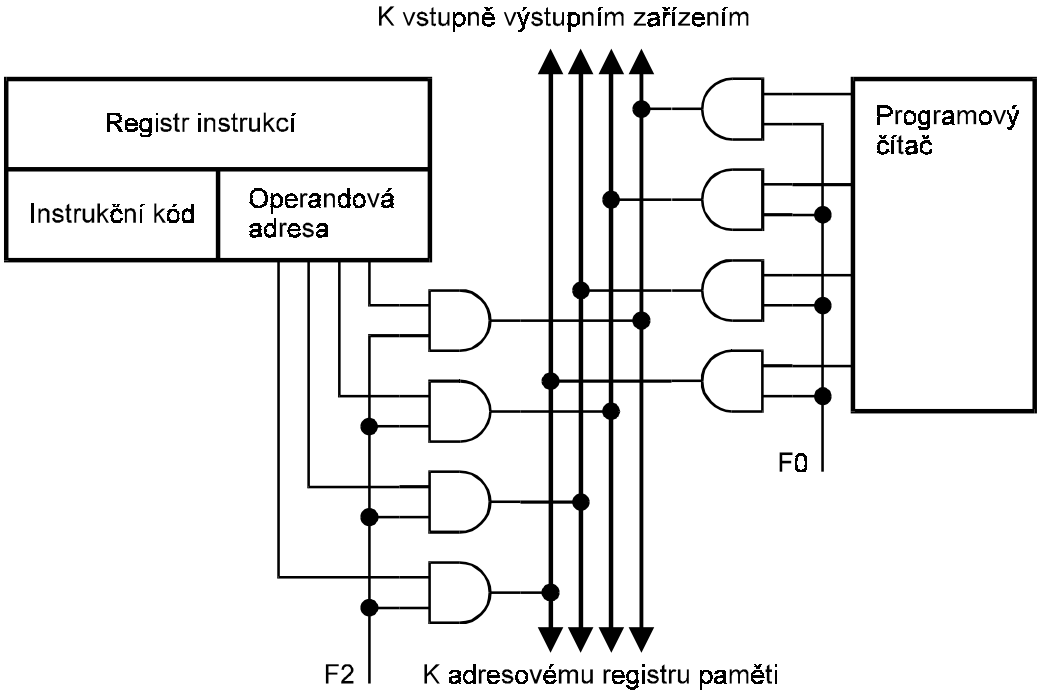
E_0 - Nahrání obsahu určité paměťové lokace do daného registru / např. B /

E_2 - Přičtení obsahu registru do akumulátoru

E_4 - Záznam obsahu akumulátoru do paměti

7.1.4 Systémové sběrnice

Nejrůznější části počítače jsou navzájem propojeny tzv. sběrnicemi / BUSy /. Existují tři základní typy - adresová, datová a řídící sběrnice.

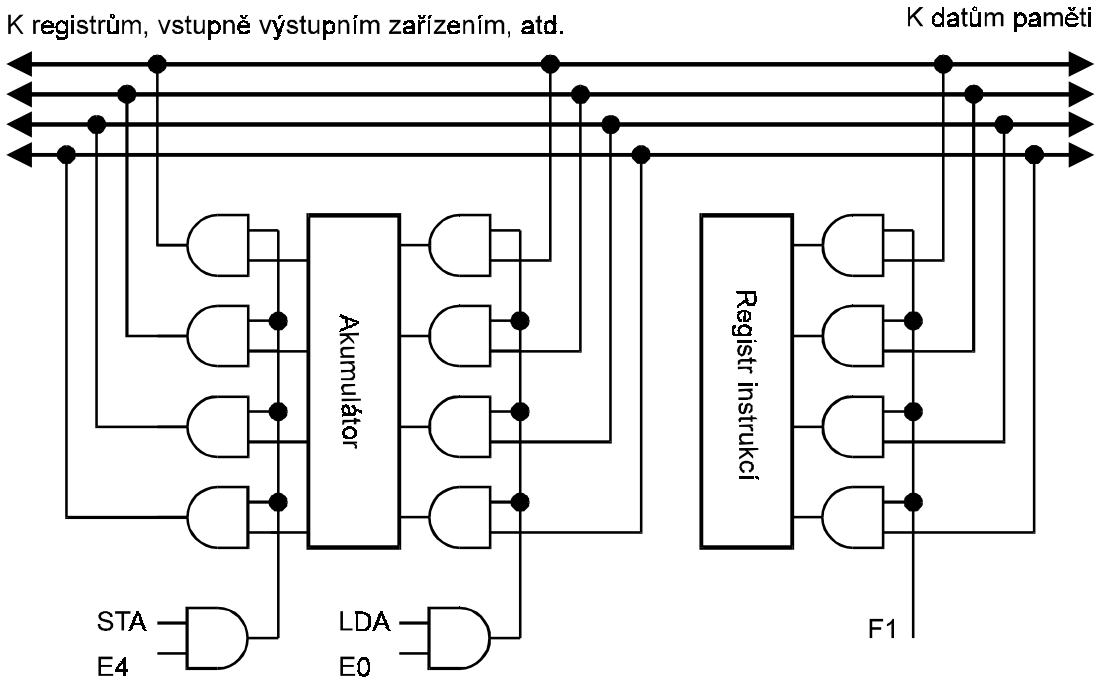


Obr. 7-41

Adresová sběrnice / viz Obr. 7-41 / je umožňuje přenos informace z takových obvodů jako je programový čítač nebo instrukční registr / sekce operandové adresy / do adresového registru paměti nebo vstupně výstupních zařízení / I / O devices /.

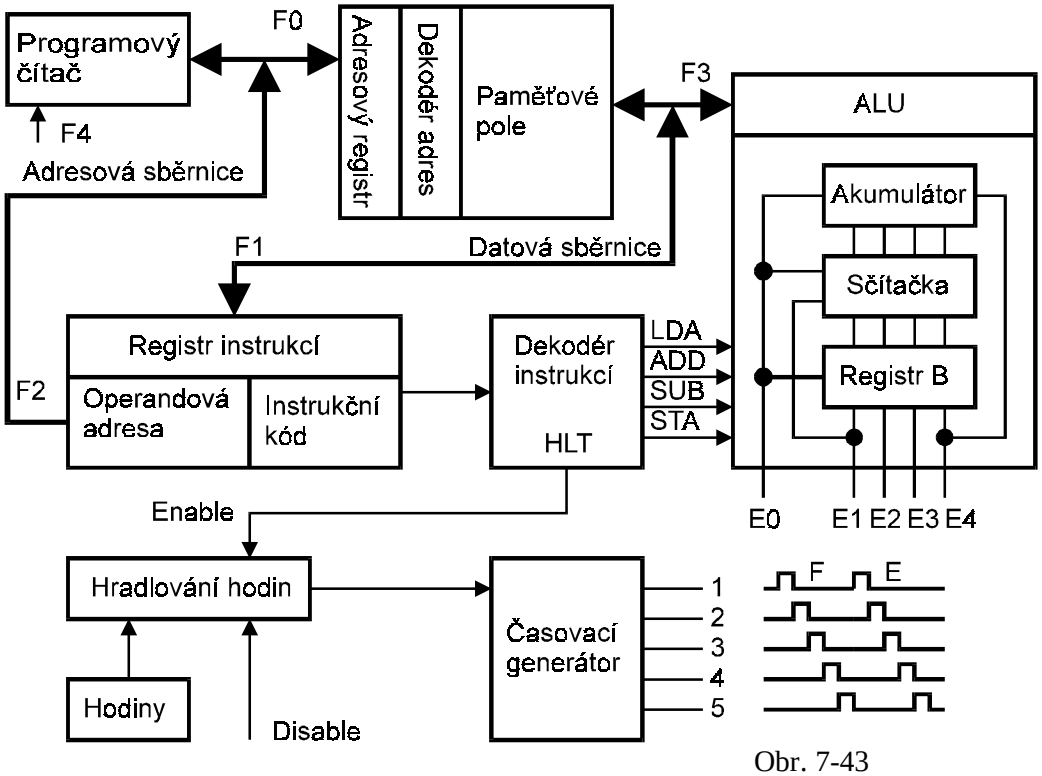
Datová sběrnice / viz Obr. 7-42 / je obousměrná a pomocí ní probíhá přenos informace z paměti do instrukčního registru / během impulsu F_1 / či akumulátoru / během impulsu E_0, kdy se např. provádí instrukce LDA - clear and load the contents of the specified memory to the accumulator /. Výstup akumulátoru může zaslat informaci prostřednictvím této sběrnice do

paměti během impulsu E_4 v době, kdy se např. provádí instrukce STA - store the accumulator. Datová sběrnice může spojovat též další registry počítače nebo vstupně výstupní zařízení.
Řídící sběrnice umožňuje jednoduchým způsobem vykonávat instrukce v počítači pomocí třístavové logiky, o které již byla řeč dříve / zamezuje interakci binárních informací, které jsou na danou sběrnici právě vysílány /.



Obr. 7-42

7.1.5 Základní činnost jednoduchého počítače



Obr. 7-43

Udělejme si nyní na základě našich úvah shrnutí, jak jsou instrukce interpretovány a vykonávány různými prvky počítače / viz Obr. 7-43 / v závislosti na časovacích impulsech. Časovací impulsy z časovacího generátoru inicializují časovací sekvence různých délek a tyto sekvence jsou užity k řízení ALU, registrů, sběrnic, pamětí, popřípadě mají další speciální funkce.

Vraťme se k našemu programu A + B = C, který je uložen v paměti nastavené uživatelem na místo počínající adresou 0000. Je-li zpracovávána např. instrukce LDA , do časovacího generátoru se přivedou hodinové impulsy a následují tyto sekvence :

1/ po příchodu impulsu F_0 je obsah programového čítače přenesen adresovou sběrnicí do adresového registru

2/ další impuls F_1 pošle data /100020 / z adresy 0004 datovou sběrnicí do instrukčního registru

3/ impuls F_2 způsobí to, že operandová adresa / 0020 / je zaslána prostřednictvím adresové sběrnice do adresového registru a operační kód / 10 / je zaslán dekodéru instrukcí, aby se nastavily nezbytné podmínky pro operaci LDA - clear and load to accumulator

4/ po příchodu impulsu F_3 se data / A / přenesou na datovou sběrnici z adresy 0020

5/ vlivem impulsu F_4 se obsah programového čítače zvětší , takže ukazuje nyní na další adresu / 0007 /, která obsahuje následující instrukci

Těchto pět bodů představovalo kompletní čtecí část instrukčního cyklu. Dekodér instrukcí tedy nastavil aritmeticko logickou jednotku / ALU / k vykonání příslušné instrukce / tedy LDA /. Následující strojový cyklus provede její vykonání, tedy :

1/ po příchodu impulsu E_0 se data / A / přítomná na datové sběrnici přenesou do akumulátoru

2/ další impuls E_1 nic nezpůsobí, neboť není pro provedení instrukce LDA nutný

3/ vlivem impulsu E_2 se data / A / v akumulátoru pošlou na sčítačku a hned poté zpět do akumulátoru

4/ impulsy E_3 a E_4 opět nejsou nutné pro tuto instrukci

Analogicky proběhnou další instrukční cykly pro instrukce ADD a STA. Po zachycení instrukce HLT provede impuls na vodiči Enable / vybavení / přerušení příchozích hodinových impulsů do časovacího generátoru a tím se počítač zastaví.

7.1.6 Mikroprocesor a jeho funkce

Vnitřní architektura mikroprocesoru se skládá z prvků schopných vykonávat aritmetické, řídící a logické operace. Je vyroben monolitickou technologií na čipu o rozměru řádově cm^2. Může také obsahovat samostatnou paměťovou oblast. Existuje rozsáhlý soubor mikroprocesorů a obvodů, které společně mohou vytvořit mikropočítač. Mikroprocesory se v zásadě rozlišují podle datové sběrnice na 8 - bitové / Intel 8080A, Zilog Z 80, Motorola 6802 /, 16 - bitové / Intel 8086 /, 32 - bitové / Intel Pentium / až 64 - bitové. Vzhledem k omezeným možnostem našich skript si stručně popíšeme mikroprocesor Intel 8080A . Tento mikroprocesor je umístěn v pouzdře s 40 vývody, které zahrnují 16 bitovou adresovou sběrnici a 8 bitovou obousměrnou třístavovou sběrnici. Může adresovat 65 536 míst v paměti i 256 vstupních a výstupních bran, má šest časovacích a řídících výstupů, čtyři řídící vstupy, čtyři vstupy napájení a dva hodinové vstupy.

Mikroprocesor obsahuje tyto funkční bloky : pole registrů a adresovacích logických obvodů, aritmeticko logickou jednotku, registr instrukcí, řídící obvody a tzv. budiče sběrnic. Pole registrů obsahuje paměť RAM organizovanou do šesti 8 bitových registrů uspořádaných do dvojic, čítač instrukcí, ukazatel zásobníku a dvojici pomocných registrů. Důležitou částí této jednotky je čítač instrukcí, který uchovává adresu instrukce programu uložené v paměti a

automaticky zvyšuje / inkrementuje / svoji hodnotu při každém výběru instrukce. Ukazatel zásobníku / stack pointer / uchovává adresu následujícího místa v zásobníku, který se nachází v hlavní paměti. / Zásobník - stack se např. používá při používání přerušení, funkcí a procedur v programech. Je-li hlavní program přerušen, stav programu je uložen do zásobníku na LIFO základě - last-in, first-out. Obsah stavového registru, akumulátoru a čítač instrukcí je tedy uložen do zásobníku a hned poté, co je obsloužen požadavek přerušení, obsah těchto registrů je obnoven ze správného místa zásobníku a hlavní program může pokračovat v operacích. / K univerzálním registrům B,C, … lze funkčně přiřadit i akumulátor A, který obsahuje výsledek libovolné aritmetické nebo logické operace prováděné v aritmetické jednotce. Ta vykonává logické operace, operace posuvů a všechny aritmetické operace s 8 - bitovými daty. Bezprostředními zdroji dat pro aritmetickou jednotku jsou akumulátor, pomocné registry a tzv. indikátory přenosu, které jsou obsaženy v nezávislém registru. / Indikátory tvoří skupinu nezávislých klopných obvodů - stavových bitů či vlajek, které tvoří podmínky pro výsledek předchozí instrukce. Tyto vlajky indikují např. negativní výsledek předchozí operace, přetečení - overflow, nulový výsledek, atd. Mikroprocesor vyhodnocuje tento registr indikátorů a může na tomto základě modifikovat běh programu - např. instrukce podmíněného skoku : if then ./ Výsledek operací se buď přenáší na vnitřní sběrnici nebo do akumulátoru.
Zásadou činnosti mikroprocesoru je provádění instrukcí během instrukčních cyklů, tj. tedy časových intervalů potřebných např. k výběru a vykonání instrukce. Během výběru je instrukce přečtena z paměti a její první slabika je uložena do registru instrukce. Každý instrukční cyklus obsahuje jeden až pět strojových cyklů neboli taktů. Každý strojový cyklus se skládá ze tří až pěti dob. Doba je nejmenší jednotka aktivity mikroprocesoru a je definována jako časový interval mezi dvěma následujícími hodinovými impulsy. Po připojení napájecích napětí je schopen okamžitě činnosti, ale obsah čítače instrukcí, ukazatel zásobníku a ostatní registry jsou nastaveny náhodně a další činnost není definována. Proto se po zapnutí přivádí signál RESET, který uvedené části mikroprocesoru nuluje. Tento signál však neovlivní indikátory ani pracovní registry - ty je nutné nastavit programem / např. inicializací systému z paměti ROM /.
Každému mikroprocesoru je přiřazen soubor instrukcí, které umožňují provádění předepsaných operací. Soubor instrukcí MHB 8080A např. obsahuje 78 instrukcí, které můžeme rozdělit do pěti skupin :
a/ instrukce přesunů - instrukce pro přesun mezi registry
b/ aritmetické instrukce
c/ logické instrukce - AND, OR, komparace / porovnávání / atd.
d/ instrukce pro větvení programů
e/ řídící instrukce
Další podrobnější popis je mimo rámec těchto skript a případný zájemce o danou problematiku jistě nalezne celou řadu speciální literatury.

7.1.7 Mikropočítač a jeho programování

Mikropočítače umožňují rozšíření číslicové techniky díky nízkým nákladům nejen do oblastí technických aplikací, ale i do spotřební elektroniky. Klíčovým prvkem je zde mikroprocesor. K významným vlastnostem mikropočítače patří možnost programování. Sestavený program - sled instrukcí - se uloží do paměti - nejčastěji ROM, EPROM - a mikroprocesor program realizuje tím, že vybírá instrukce z paměti, vykonává je a předává data přes vstupně výstupní brány na výstup mikropočítače. Základní mikroprocesorový systém mikropočítače tedy obsahuje mikroprocesor / CPU - central processor unit /, paměť, systém vstupů / výstupů a systém sběrnic. Mikroprocesor lze ještě rozšířit o obvody generátoru hodinových signálů a

budiče sběrnice - ten kromě výkonového posílení datové sběrnice vytváří řídící logické signály jako jsou např. čtení z paměti, zápis do paměti, atd . Dalšími obvody mikropočítače mohou být programovatelné obvody velké integrace, které obsahují několik řídících registrů pro zavedení řídících slov. Tímto způsobem je možné realizovat sériový synchronní či asynchronní a paralelní přenos dat, styk s klávesnicí, tiskárnou, zobrazovací jednotkou či magnetickými disky. Dále je možné pomocí paralelního rozhranní použití A/D a D/A převodníků.

Programování mikropočítače je jedním z nejnáročnějších kroků pro vytváření systému mikropočítače. Nejjednodušší programové vybavení mají jednodeskové mikropočítače, které mohou pracovat v asembleru / jazyku symbolických instrukcí /, který je velmi blízký strojovému kódu. Symbolické instrukce jsou většinou uváděny v šestnáctkovém / hexadecimálním / kódu - znaky 0 až 9, A, B, C, D, E, F - např. $4F_{16} = 01001111_2$.

Například symbol LDA v asembleru / $3A_{16} = 00111010_2$ / znamená naplnění akumulátoru A obsahem paměťového místa, jehož adresa je obsažena v operandu této instrukce / v příkladu není uvedena /. Použitím asembleru lze naprogramovat mikropočítač, který obsahuje základní program / tzv. monitor /, zajišťující operace mezi základní jednotkou a periferními zařízeními / klávesnice, zobrazovací jednotka apod. /.

Tento způsob programování je vhodný pro méně rozsáhlé programy, protože takto není možné program kontrolovat. Lépe je takovýto program pro mikropočítač sestavit v mikropočítačovém vývojovém systému / umístěném na výkonnějším počítači /, kde se odladí a pomocí zařízení pro programování pamětí EPROM se do těchto pamětí přímo zapíše. Vývojový systém tedy zahrnuje většinou tyto programy : asembler / nebo jazyk C /, kompilátor / překladač z vyššího jazyka do strojového kódu /, editor / upravující program /, loader / zaváděcí program /, debuger / ladící program umožňující krokování daného programu / a simulátor.

Při vývoji programu zapíšeme zdrojový program / např. v asembleru či jazyku C / přes klávesnici do vývojového systému pomocí editoru, který řídí vstup, umožňuje opravy a ukládá zdrojový program. Zdrojový kód se přeloží pomocí kompilátoru do cílového programu / strojového kódu. Zaváděcím programem jej uložíme do hlavní operační paměti mikropočítače a pro odstranění případných chyb provedeme kontrolu ladícím programem. Po odladění můžeme program uložit do programovatelných pamětí, jak již bylo řečeno.