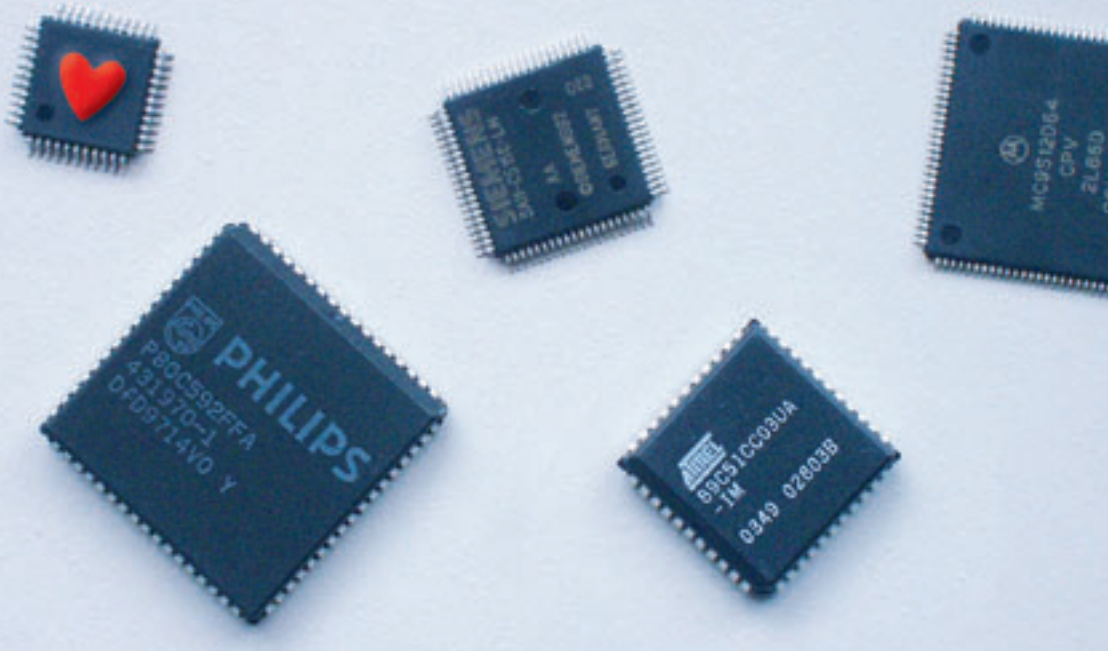
Mikroprocesory a mikropočítače

obecné principy konstrukce
současných mikroprocesorů a mikropočítačů



Jiří Pinker

TECHNICKÁ LITERATURA
BEN

Je mojí milou povinností poděkovat panu Ing. Františku Kostkovi, CSc., za pečlivé přečtení rukopisu a za cenné připomínky. Obsahu i formě knihy to velmi prospělo.

Jiří Pinker
MIKROPROCESORY A MIKROPOČÍTAČE

Lektor František Kostka

Jiří Pinker: Mikroprocesory a mikropočítače
BEN – technická literatura, Praha 2004
1. dotisk 1. vydání
ISBN 978-80-7300-110-0 (tištěná kniha)
ISBN 978-80-7300-353-1 (elektronická kniha v PDF)

426410

OBSAH

426410

426410

SLOVNÍK ANGLICKÝCH TERMÍNŮ A ZKRATEK

anglicky	česky	kapitola (prvý výskyt)
acceptance filter	příjmový filtr	8
accumulator	střadač	3
acknowledge	potvrdit	8
ACR Acceptance Code Register	registr příjmového kódu	8
ALU Arithmetic-Logic Unit	aritmeticko-logická jednotka	3
AMR Acceptance Mask Register	registr příjmové masky	8
auto scan	postupné převádění skupiny vstupních kanálů	8
auto scan continuous	opakované převádění skupiny vstupních kanálů	8
auxiliary carry	pomocný přenos	3
bank switch	přepínač bank (registrů)	3
bidirectional	dvojsměrný	8
bit error	chyba v bitu	8
bit stuffing	vsouvání bitů	8
BIU Bus Interface Unit	instrukční jednotka	3
bottom	dno (zásobníku)	3
broadcasting	vysílání výzvy, určené všem	8
burst	skupina, shluk impulzů	8
bus cycle	sběrnicový cyklus	5
CAN Controller Area Network	viz vysvětlení v kapitole 8	8
capture	zachytit	8
carry	přenos	3
CAS Column Address Select	výběr adresy sloupce (u dynamické paměti)	7
CCR Condition Code Register	registr příznaků	3
CISC Complex Instruction Set Computer	počítač se složitým souborem instrukcí	3
compare	porovnat	8
controller	řadič	3
CSMA/CD+AMP Carrier Sense, Multiple Access/Collision Detection + Arbitration on Message Priority	viz vysvětlení v kapitole 8	8
destination address	adresa cíle dat	4
direct address	přímá adresa	4

426410

anglicky	česky	kapitola (prvý výskyt)
DMA Direct Memory Access	přímý přístup do paměti	7
EA External Access	vnější přístup	5
embedded computer	zabudovaný, vložený počítač	2
EU Execution Unit	výkonná jednotka	3
FIFO First In - First Out	paměť fronty	8
flags	příznakové bity	3
GPR General Purpose Register	univerzální registr	5
GPT General Purpose Timer	univerzální čítač/časovač	8
half-carry	poloviční přenos	3
handshake	korespondenční provoz	8
IE Interrupt Enable	povolení přerušení	6
IIC, I^2C Inter-Integrated Circuit Bus	viz vysvětlení v kapitole 8	8
immediate	bezprostřední (data)	4
IMR Interrupt Mask Register	registr masky přerušení	6
indexed address	indexová adresa	4
indirect address	nepřímá adresa	4
instruction cycle	instrukční cyklus	3
INT Interrupt	přerušení (programu)	3
INTA Interrupt Acknowledge	potvrzení přerušení	6
interrupt controller	řadič přerušení	6
interrupt latency	latence přerušení	6
IRQ Interrupt Request	požadavek na přerušení	6
IRR Interrupt Request Register	registr požadavků na přerušení	6
ISP	programování přímo v aplikaci zpravidla pomocí sériového kanálu	
ISR In-Service Register	registr obsluhovaných požadavků	6
left justified	zarovnaný zleva	8
LIFO Last In – First Out	zásobníková paměť	3
LSB Least Significant Bit	nejnižší, nejméně významný bit	4
MAC Multiply And Accumulate	vynásobení a přičtení	3
master	hlavní (stanice)	8
MC Machine Cycle	strojový cyklus	3
memory map	paměťová mapa	5
microcontroller	mikrořadič, mikrokontrolér	2
misalignment	nesprávné zarovnání slov	5
mnemonic code	mnemotechnická instrukce	4
MSB Most Significant Bit	nejvyšší, nejvýznamnější bit	4
nested	do sebe vložené (podprogramy)	6

anglicky	česky	kapitola (prvý výskyt)
nibble	půlbyte	4
NMI Non-Maskable Interrupt	nemaskovatelné přerušení	6
NRZ Non-Return to Zero	bez návratu k nule	9
op-code	operační kód, operační znak, instrukční kód	3
OTP One-Time Programmable	jednou programovatelná (paměť)	7
overflow	přetečení	3
overrun error	ztráta dat	8
page	stránka	7
parity	parita	3
PC Program Counter	čítač instrukcí	3
pipeline	zřetězená struktura	3
PLL Phase Locked Loop	fázový závěs	5
port	brána	2
PSW Program Status Word	stavové slovo programu	3
PWM Pulse-Width Modulation	impulzní šířková modulace	8
quasi-bidirectional	kvazi-dvousměrný, neúplně dvousměrný	8
RAS Row Address Select	výběr adresy řádku (u dynamické paměti)	7
relative address	relativní adresa	4
reload	znovu nastavit	8
request	požadavek	8
reset	nulování	3
retriggerable	znovuspustitelný	5
right justified	zarovnaný zprava	8
RISC Reduced Instruction Set Computer	počítač se zjednodušeným souborem instrukcí	3
RS 232 C Recommended Standard	(doporučená) norma RS 232 C	
RTR Remote Transmission Request	požadavek na přenos z jiné stanice	8
SFR Special Function Registers	registry speciálních funkcí	5
shadow ROM	stínová ROM	7
sign	znaménko (čísla)	3
SIM System Integration Module	modul pro integraci systému (vzájemnou spolupráci jeho částí)	5
single channel continuous conversion	opakovaný převod jednoho vstupního kanálu	8
single channel conversion	převod jednoho vstupního kanálu	8
SJW Synchronization Jump Width	sychronizační skok	8

anglicky	česky	kapitola (prvý výskyt)
slave	podřízený (stanice)	8
SOF Start of Frame	začátek rámce	8
source address	adresa zdroje dat	4
SP Stack Pointer	ukazatel zásobníkové paměti	3
SPI Serial Peripheral Interface	sériové periferní rozhraní	8
SWI Software Interrupt	softwarové přerušení (instrukce)	6
system controller	systémový řadič	5
TAR Temporary Address Register	pomocný adresový registr	3
TDR Temporary Data Register	pomocný datový registr	3
token passing	předání pověření	8
top	vrchol (zásobníku)	3
UART Universal Asynchronous Receiver-Transmitter	univerzální asynchronní přijímač-vysílač	8
VLSI Very Large Scale Integration	velmi velká integrace	2
wait-state generator	generátor čekacích taktů	5
WDT Watch-Dog Timer	diagnostický časovač	5
zero	nulovost (výsledku, obsahu)	3

426410

1 ÚVOD

Vliv elektroniky je patrný v mnoha oblastech života. Postupně pronikla i do takových zařízení, o kterých by to byl nikdo nepředpokládal. To se týká kanceláří, vývojových pracovišť, strojů, vozidel, telekomunikací, a dokonce i zařízení pro domácnost a hraček. Získaly se nové vlastnosti, snazší ovladatelnost, vyšší účinnost, vyšší spolehlivost a bezpečnost.

Úžasně rychlý rozvoj elektronických systémů nastal díky pokrokům v technologii výroby součástek – zvláště integrovaných obvodů, ale i součástek výkonových a optoelektronických. Při pohledu zpět je však zřejmé, že zdaleka největší vliv měla miniaturizace počítače. Od rozměru sálů, pak velkých a malých skříní se nakonec dospělo k jedinému integrovanému obvodu v pouzdru o ploše řádově čtverečního centimetru. Současně s tím došlo i k jeho neuvěřitelnému zlevnění. To teprve umožnilo vestavění počítačů do takových zařízení, která sama mají malý rozměr – do měřicích přístrojů, mobilních telefonů, televizorů, kardiostimulátorů a mnoha dalších, jejichž prostý výčet by zaplnil několik stran a stejně by nebyl definitivní.

Ve světě existuje několik desítek výrobců integrovaných obvodů určených pro realizaci mikropočítačů. O těch nejčastěji používaných byla publikována řada knih, které vycházejí z firemní literatury, někdy v téměř doslovném překladu. Dostatek informací o jednotlivých obvodech lze získat též z Internetu.

Proto cílem této publikace nebyl výklad, zaměřený pouze na jeden typ obvodu, nýbrž zobecnění základních vlastnosti mikropočítačů. Vychází se přitom z průřezu produkce hlavních světových výrobců z posledních let a trendů vývoje. Nejedná se o učebnici programování a proto jednotlivé typické instrukce jsou vysvětleny z hlediska pochopení vnitřních operací procesoru a k tomu potřebných obvodů. Důraz je kladen na mikropočítače jednočipové, na jejich vlastnosti, rozšíření o vnější obvody a zásady použití. Personální počítače jsou zmíněny jen okrajově. Jsou vysvětleny návaznosti vnitřních obvodů počítače na obvody vnější a některá doporučení k jejich konstrukci. Mikropočítač je tak chápán nikoliv jako izolovaná součástka, nýbrž jako organická část většího systému, do kterého je zabudován.

Po prostudování této knihy by čtenář měl být seznámen s obecnou problematikou mikropočítačů a jejich periferních obvodů v takové míře, aby byl schopen v krátké době začít používat kterýkoliv konkrétní typ, samozřejmě s využitím firemní dokumentace. V textu jsou častá upozornění na důležité údaje, které je někdy nutno ve firemní dokumentaci pracně hledat, ačkoliv bez nich nelze sestavit funkční program. Text je doplněn seznamem anglických termínů a zkratek a seznamem grafických symbolů, používaných v obrázcích.

Kniha je určena studentům technických fakult se zaměřením na elektroniku, automatizaci a výpočetní techniku. Samozřejmě může posloužit i všem, kteří mají

zájem o mikroprocesorovou techniku. Předpokládá se základní znalost číslicových systémů.

Tato publikace vznikla za finančního přispění MŠMT v rámci projektu výzkumu a vývoje LN00B084.

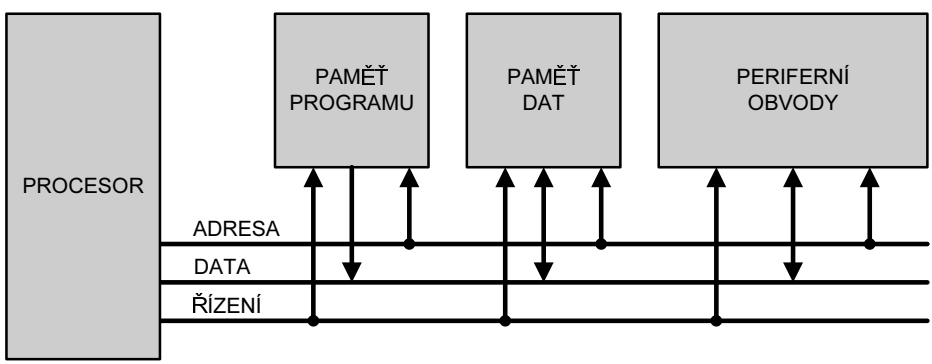
Autor

426410

2 ZÁKLADNÍ ČÁSTI A FUNKCE POČÍTAČE

Počítač můžeme funkčně rozdělit na několik základních částí. Je to procesor, paměť programu, paměť dat a periferní obvody. Periferní obvody tvoří velmi rozmanitou skupinu a jejich skladba je závislá na aplikaci počítače. Vždy však jsou přítomny alespoň vstupní a výstupní obvody, které zprostředkují spojení počítače s okolím. Zvláště v případě využití v automatizaci se počítač vybavuje velmi rozsáhlým souborem speciálních jednotek, jako jsou převodníky, čítače a časovače, výkonové výstupy, galvanicky oddělené vstupy a výstupy atd.

Vzájemné spojení jednotek je v počítači zásadně zprostředkováno soustavou sběrnic (viz *obr. 2.1*). Sběrnice umožňují stavebnicovou koncepci, rozšiřování počítače o další jednotky, a to vše beze změn ve vnitřním zapojení jednotek. Negativní stránkou sběrnice je to, že v každém okamžiku na ni může být připojen jen jeden zdroj dat. Nelze tak např. současně předávat data ze dvou zdrojů ke dvěma příjemcům. Činnost sběrnice je v každém okamžiku řízena jen jednou z jednotek – zpravidla je to procesor, ale v některých případech může dočasně přebírat řízení i jiná jednotka.



Obr. 2.1 Zjednodušené schéma počítače

Datová sběrnice slouží k předávání dat a její šířka (tj. počet vodičů) je zpravidla celým násobkem osmi (tj. jednoho byte). Jednotka, připojená na sběrnici, může být zdrojem dat (pak se z ní čte), příjemcem dat (pak se do ní zapisuje), nebo

426410

střídavě obojím. Jednotka, která je zdrojem dat, se na datovou sběrnici zásadně připojuje prostřednictvím třístavových členů.

Adresová sběrnice je nutná pro adresování paměti (případně i jiných adresovatelných obvodů) a pro rozlišování mezi jednotkami, připojenými na datovou sběrnici. Šířka adresové sběrnice určuje maximální počet adres. U osmibitových počítačů má adresová sběrnice šířku nejčastěji 16 bitů, u šestnáctibitových počítačů bývá minimálně 20bitová.

Čtení, zápis a další aktivity jednotek jsou řízeny signály řídicí sběrnice. Většina řídicích signálů je generována procesorem, ale některé mohou být generovány i ostatními jednotkami, které tak mohou částečně ovlivňovat činnost procesoru – takovéto signály jsou pak na řídicí sběrnici připojeny zpravidla přes členy s otevřeným kolektorem. Řídicích signálů bývá obecně větší počet a podoba řídicí sběrnice je silně závislá na celkové architektuře počítače.

Stručně popíšeme funkci jednotlivých bloků z *obr. 2.1*. Procesor řídí činnost celého počítače. Zajišťuje správné provádění instrukcí uložených v paměti programu, zpracovává data v paměti, řídí tok dat ze vstupních obvodů do počítače a jejich zpracování, řídí tok dat z počítače ven přes výstupní obvody. Podle počtu bitů zpracovávaných dat se procesor označuje jako osmibitový, šestnáctibitový, atd. Je třeba poznamenat, že počet bitů procesoru nemusí být shodný se šířkou datové sběrnice. Tak např. některý 16bitový procesor může využívat jen 8bitovou datovou sběrnici. Vnitřní datové slovo (16 bitů, tj. 2 byte) pak musí samozřejmě být přenášeno na dvakrát. Takovéto řešení je jednodušší z hlediska obvodů i plošného spoje, činnost počítače však bude pomalejší.

Paměť programů obsahuje instrukce, jejichž postupným prováděním je realizována požadovaná činnost počítače. Dále často obsahuje různé konstanty a neměnné tabulky, používané v programu. Někdy je program pro danou aplikaci neměnný a pak bývá bezpečně uložen v paměti ROM (EPROM, EEPROM, FLASH). Jindy se naopak programy potřebují často měnit. To je případ počítače pro vědecko-technické výpočty. Pak musí být programová paměť typu RAM. Je ovšem nezbytné vyřešit otázku, jak spustit program po zapnutí mikropočítače, kdy paměť RAM má zcela náhodný obsah. Proto se počítač i v těchto případech vybavuje malou programovou pamětí ROM. Po náběhu napájení program začíná zde a vyvolá čtení z velkokapacitní diskové paměti. Její obsah (část operačního systému) se přesune do rozsáhlejší programové paměti RAM. Tento program pak přebírá řízení počítače – komunikaci s operátorem, přesunování dalších programů z diskové paměti, jejich spouštění, atd.

Paměť dat zajišťuje dočasné uložení dat, získaných ze vstupních obvodů, uložení mezivýsledků výpočtů apod. Je zásadně typu RAM. Pokud i programová paměť je typu RAM, mohou být případně obě realizovány jedinou společnou paměťovou jednotkou.

Vstupní a výstupní obvody (I/O – Input/Output) umožňují počítači komunikovat s vnějším prostředím. Zpravidla obsahují větší počet **bran** (angl. **port**), tj. připojovacích míst, rozlišených adresově. Brány mohou být paralelní nebo sériové. Při

426410

paralelním přenosu se čte nebo zapisuje najednou celá skupina signálů jako vícebitové slovo (nejčastěji 8bitové). Při sériovém přenosu se data přenášejí postupně bit po bitu. Sériový přenos je mnohem pomalejší než paralelní, ale je úspornější co se týče počtu signálových vodičů.

Výše popsaná struktura platí obecně pro všechny počítače. Pokroky v technologii integrovaných obvodů umožnily zmenšení rozměrů a koncentraci mnoha funkcí do jednoho integrovaného obvodu VLSI. Vznikl tak pojem **mikropočítače** a **mikroprocesoru**. Je nutné zdůraznit, že předpona „mikro“ se vztahuje k fyzickým rozměrům obvodů a neznamená omezení funkce. Právě naopak, vysoká integrace umožnila vývoj velmi složitých architektur s vysokým výpočetním výkonem a velkou variabilitou funkcí. Soustředění obvodů na jednom čipu dovoluje zkrátit spoje a tím i zpoždění signálu.

S postupující integrací bylo možné sdružit všechny obvody mikropočítače do jediného integrovaného obvodu. Vznikla tak řada **jednočipových mikropočítačů**. Zvláště v této skupině počítačů probíhá nejintenzivnější vývoj. Zvětšuje se kapacita paměti programu i paměti dat, rozšiřuje se sestava periferních obvodů. Jednočipový mikropočítač, vhodný pro využití v řízení, je v anglické literatuře označován jako „microcontroller“, česky mikrokontrolér.

Počítače pronikly do všech oblastí života a jsou součástí měřicích přístrojů, vozidel, audiovizuálních přístrojů, mobilních telefonů, atd. Počítače v těchto aplikacích jsou v anglické literatuře označovány jako „**embedded computers**“, v českém překladu **zabudované** nebo **vložené** počítače.

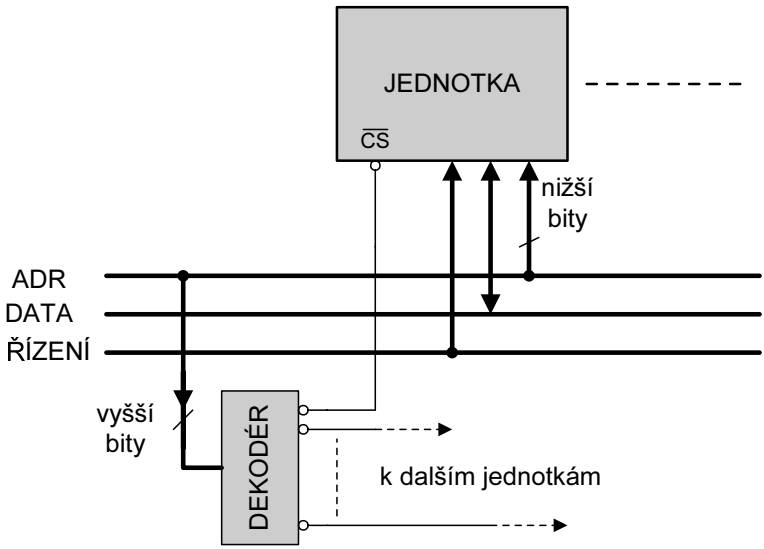
Vývoj nových typů mikrokontrolérů využívá technologii ASIC, založenou na sdružování a sestavování masek pro výrobu integrovaných obvodů (masky slouží v procesu fotolitografie). Dílčí masky odpovídají osvědčeným, ověřeným jednotkám počítače. Podle potřeby se pak sestaví **jádro** počítače (procesor a k němu přilehlé obvody), paměti a několik variant sestavy periferních obvodů – vše integrováno na jednom čipu. Vznikají tak **rodiny mikropočítačů**, které mají společné jádro.

Na rozdíl od rychlého vývoje sestav periferních obvodů a zvyšování kapacity vnitřní paměti je vývoj nových procesorů podstatně konzervativnější. Souvisí to s nutností kompatibility (tj. slučitelnosti) programů po dlouhou dobu. Finanční prostředky, investované do vývoje programů, jsou obrovské a žádný výrobce integrovaných obvodů tuto skutečnost nemůže ignorovat. Modernizace procesoru se děje nejčastěji přidáváním několika málo instrukcí, zvýšením rychlosti, doplněním o specializované obvody. Kompatibilita programů je tak zaručena směrem nahoru (programy pro starší verzi jsou plně použitelné i pro verzi novou), ale ne nutně směrem dolů.

426410

2.1 SBĚRNICOVÉ CYKLY

Jak bylo výše řečeno, přenosy dat v počítači probíhají po sběrnicích. Obecný pohled na jednotku, připojenou na sběrnice, dává *obr. 2.2*.



Obr. 2.2 Připojení jednotky na sběrnice

Nejvyšší bity adresy jsou vedeny do adresového dekodéru, jehož výstupy (v kódu 1 z N) vybírají jednu z jednotek a povolují její činnost. Pokud je jednotkou samotný integrovaný obvod, bude se zřejmě jednat o jeho vstup $\overline{CS}$. Pro jednoduchost budou výběrové signály na výstupech dekodéru takto nazvány i v obecném případě. V jednotce jsou využity zbylé (nižší) bity adresy pro adresování uvnitř obvodů v rámci této jednotky (např. paměti).

Datová sběrnice je obecně dvojsměrná a jednotka, připojená na sběrnici, může být jen zdrojem dat (pak se z ní jen čte, např. paměť ROM), příjemcem dat (pak se do ní jen zapisuje, např. výstupní obvody), nebo střídavě obojím (např. paměť RAM). Směr čtení/zápis se rozlišuje vždy z pohledu od procesoru.

Čtení i zápis jsou řízeny a přesně časovány prostřednictvím signálů řídicí sběrnice. Zatím je označíme jako $\overline{RD}$ a $\overline{WR}$. Aktivním stavem je zpravidla „L“ (low, nízká úroveň) proto jsou v negaci. Řídicích signálů však může být větší počet – např. vstupní a výstupní obvody mohou mít své řídicí signály jiné než paměť.

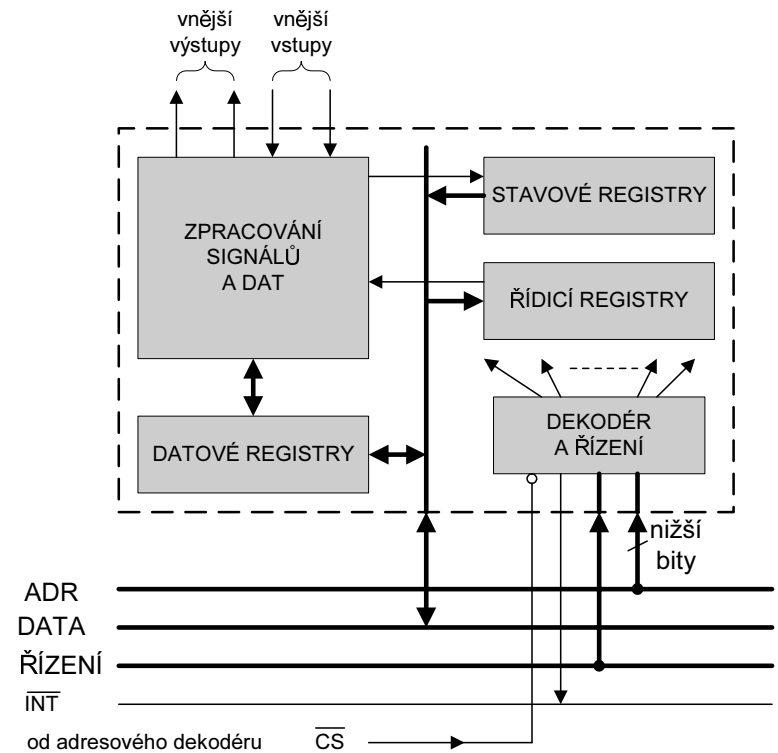
Posloupnost činností, během kterých dojde právě jednou ke čtení nebo zápisu prostřednictvím sběrnic, se nazývá **sběrnicový cyklus** (angl. **bus cycle**). Základními sběrnicovými cykly jsou cyklus čtení z paměti a cyklus zápisu do paměti. Vedle toho mohou existovat i další sběrnicové cykly, jako např. čtení ze vstupních obvodů, zápis do výstupních obvodů, aj. Časování signálů může být v těchto cyklech odlišné.

Pokud má být rychlost procesoru i dalších obvodů počítače využita na maximum, musí být časování velmi přesné. To je zajištěno synchronizací procesoru generátorem hodinových impulzů, řízeným krystalem.

2.2 PERIFERNÍ OBVODY

Periferní obvody tvoří velmi rozmanitou skupinu obvodů, zvláště u počítačů využitých pro řízení. I přes jejich rozmanitost však lze nalézt obecně platné vlastnosti. Velmi často je několik periferních obvodů sdruženo (integrováno) do jedné periferní jednotky – např. několik paralelních bran, několik sériových bran, několik čítačů a časovačů, vícekanálový převodník A/Č, apod.

Zjednodušené schéma periferní jednotky ukazuje *obr. 2.3.*



Obr. 2.3 Zjednodušené schéma periferní jednotky

Jádrem periferní jednotky jsou obvody pro zpracování signálů. To jsou např. čítače, převodníky, nebo třeba jen jednoduché výkonové oddělovací členy. Ze strany datové sběrnice s nimi komunikuje procesor, z druhé strany pak okolí

426410

počítače. Jelikož v jednotce bývá sdruženo několik periferních obvodů, jsou data ukládána do několika **datových registrů**.

Velmi často je nutné činnost těchto obvodů nějak řídit – nastavit cesty signálů, dělicí poměry čítačů, různé vnitřní vazby, apod. K tomu slouží **skupina řídicích registrů**. Dále je třeba zjišťovat stav těchto obvodů – např. přetečení čítače, skončení A/Č převodu, přijetí znaku v sériovém přenosu, apod. K tomu slouží **skupina stavových registrů**. Procesor může prostřednictvím datové sběrnice zapisovat do řídicích registrů a číst stavové registry. U jednočipových mikropočítačů jsou řídicí a stavové registry seskupeny v jedné oblasti vnitřní paměti.

Jednotka může vyžadovat obsluhu též prostřednictvím přerušení programu. K tomu slouží jeden nebo více signálů $\overline{\text{INT}}$. Podnětem k vyvolání přerušení je dokončení požadované operace (dokončení A/Č převodu, přijetí znaku v sériovém přenosu, atd.) nebo případné hlášení chyby (zvláště v komunikačních obvodech).

Do jednotky je vedeno několik (nejnižších) adresových bitů pro rozlišení mezi různými datovými, řídicími a stavovými registry. Uvnitř jednotky zřejmě musí existovat ještě lokální adresový dekodér.

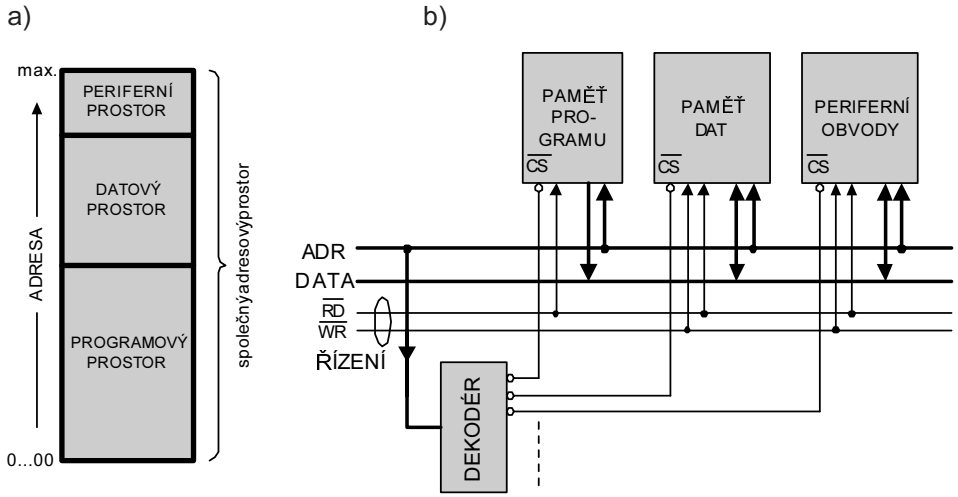
Signál $\overline{\text{CS}}$ (pokud vůbec existuje) blokuje vždy jen komunikaci s jednotkou ze strany **sběrnic**, nikdy **neblokuje** činnost vnitřního jádra, ani vnějších vstupních a výstupních signálů, ani vyvolání přerušení.

2.3 ADRESOVÉ PROSTORY

Adresový prostor je množina adres, na které lze dosáhnout jedním typem sběrnicového cyklu. Tak může existovat programový prostor, datový prostor, vstupní/výstupní prostor, atd. Existuje několik uspořádání adresových prostorů, která úzce souvisejí s architekturou počítače, se složením řídicí sběrnice, a též s instrukčním souborem procesoru.

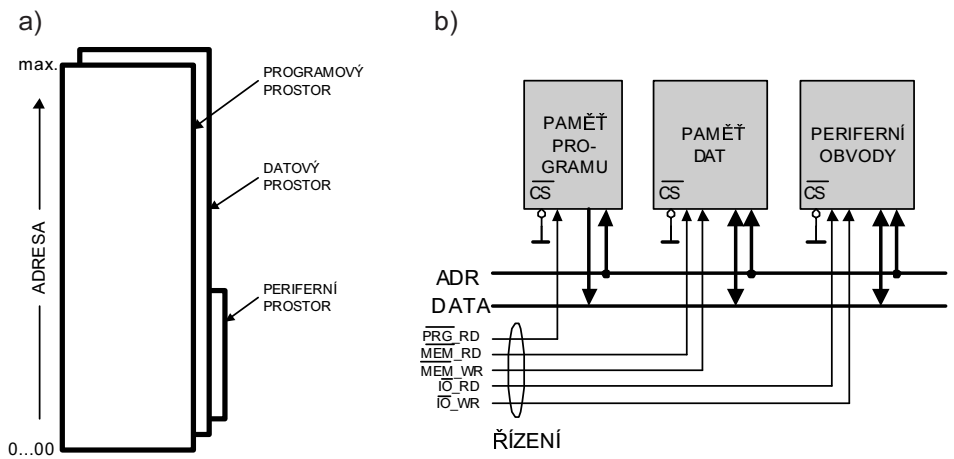
Adresové prostory lze velmi snadno znázornit, jak ukazuje *obr. 2.4a*. Jedná se v tomto případě o **počítač s jediným adresovým prostorem**, do kterého jsou vloženy dílčí prostory – programový, datový a periferní. Adresy rostou směrem zdola nahoru (v některých publikacích je zvolen opačný směr). Co se týče obvodového řešení, je zahrnutí všech obvodů do jednoho prostoru docíleno jedním adresovým dekodérem, který aktivuje vždy jednu skupinu obvodů (např. paměť ROM, paměť RAM, periferní obvody) a společným rozvodem řídicích signálů $\overline{\text{RD}}$ a $\overline{\text{WR}}$ do všech obvodů (s výjimkou do ROM, kde zřejmě signál $\overline{\text{WR}}$ pro řízení zápisu nemá smysl). Existují tedy zásadně jen dva sběrnicové cykly – čtení a zápis. Obvodové uspořádání ukazuje *obr. 2.4b*. K výběru obvodů jsou využity nejvyšší bity adresy. Tato architektura se nazývá **von Neumannova**. Výhodou je jednoduché řízení obvodů počítače.

426410



Obr. 2.4 *Počítač s jedním adresovým prostorem:*
a) obsazení adresového prostoru b) obvodové uspořádání

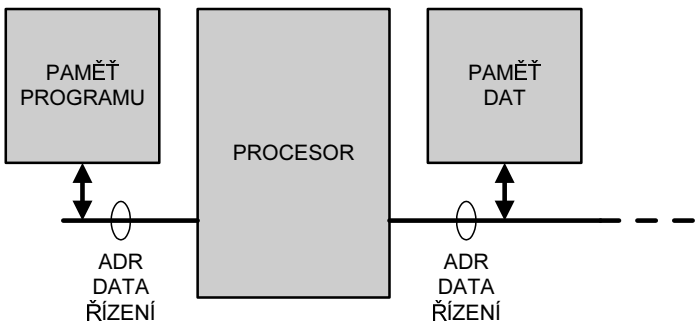
Jinou možnost ukazuje *obr. 2.5a*. V počítači existuje **několik adresových prostorů** (např. programový, datový, periferní), ne nutně stejně rozsáhlých. V několika různých adresových prostorech mohou existovat stejné adresy. K rozlišení prostorů tedy nemůže sloužit adresa, ale **typ sběrnicových cyklů**. Jsou to zásadně tyto: čtení instrukce, čtení dat, zápis dat, čtení z periferních obvodů, zápis do periferních obvodů, případně některé další. Větší počet sběrnicových cyklů



Obr. 2.5 *Počítač s několika adresovými prostory:*
a) obsazení adresového prostoru b) obvodové uspořádání

vyžaduje také větší počet **řídicích signálů** – např. tak, jak je naznačeno v *obr. 2.5b*. Uvedená soustava řídicích signálů není jedinou, existují i jiné – detaily jsou v dalších kapitolách. Rozlišení cyklu čtení instrukce a čtení dat nečiní potíže, neboť se zřejmě jedná o úplně jiné operace procesoru. Jinak je tomu s rozlišením cyklu čtení dat a čtení z periferních obvodů, či rozlišením zápisu dat a zápisu do periferních obvodů. Toto rozlišení je možné jen tak, že bude existovat jiný soubor instrukcí pro práci s datovou pamětí a jiný pro práci s periferními obvody. V závislosti na typu instrukce pak procesor zvolí patřičný typ sběrnicového cyklu. Výhodou je větší celkový počet adres.

Tato architektura může být dále rozvíjena pro realizaci velmi rychlých počítačů. Sběrnice je vždy omezujícím prvkem co do rychlosti, neboť nedovoluje současné přenosy ze dvou nebo více zdrojů dat (zkraty na výstupech třístavových budičů sběrnice). Procesor by sice mohl současně zapisovat data (jako výsledek právě ukončené instrukce) a současně s tím číst novou instrukci, architektura s jedinou soustavou sběrnic to však nedovoluje. Proto byla zavedena architektura se dvěma soustavami sběrnic. Na jednu je připojena paměť programu, na druhou paměť dat (případně další obvody) – viz *obr. 2.6*.



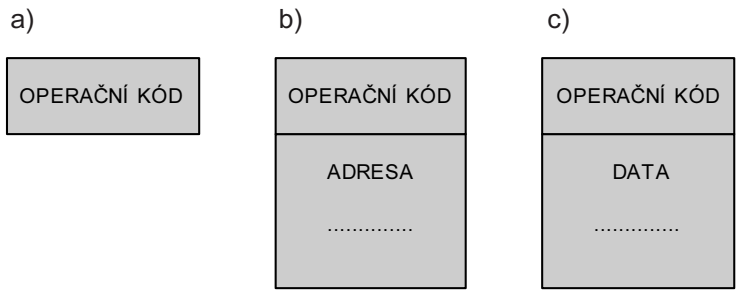
Obr. 2.6 Počítač s dvěma soustavami sběrnic

Popsaná architektura je známá pod názvem **harvardská** a je důsledně aplikována zvláště u signálových procesorů.

3 PROCESOR

3.1 ZÁKLADNÍ POJMY

Úkolem procesoru je provádění instrukcí. Instrukce jsou nejmenší jednotky, ze kterých je složen program. Základní části instrukce jsou znázorněny na *obr. 3.1*. Ve všech případech je první částí **operační kód** (angl. op-code, též operační znak, instrukční kód). Ten přesně definuje další činnost při provádění instrukce.



Obr. 3.1 Základní části instrukce

Některé instrukce (viz případ na *obr. 3.1a* obsahují jen operační kód. V těchto případech jsou prováděny jen vnitřní operace v procesoru – např. „*inkrementuj obsah vnitřního datového registru*“ apod.

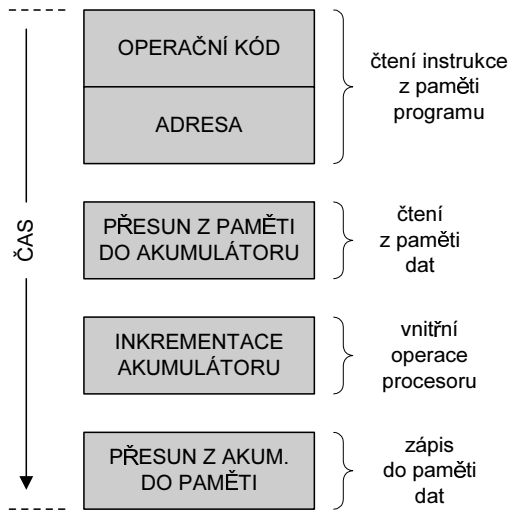
V druhém případě (*obr. 3.1b*) obsahuje instrukce ještě **adresu operandu**. Operandy jsou uloženy v datové paměti. Příkladem mohou být instrukce typu „*přesuň obsah vnitřního datového registru do datové paměti na adresu ...*“ apod.

Ve třetím případě (*obr. 3.1c*) obsahuje instrukce ještě **přímá data**. Jedná se vždy o konstanty, které jsou součástí programu. Příkladem může být instrukce „*naplň vnitřní datový registr číslem ...*“.

Existují i složitější instrukce obsahující dvě nebo tři adresy, adresu i data apod. – záleží vždy na instrukčním souboru procesoru. Instrukce mají tedy obecně různou délku.

Celý průběh instrukce se skládá ze dvou fází – přečtení instrukce a provedení instrukce. **Přečtení instrukce** znamená vyvolání čtecích cyklů z paměti programu při postupně rostoucích adresách tak, aby se postupně přečetly všechny části instrukce. Počet čtecích cyklů odpovídá délce instrukce. **Adresa instrukce** odpovídá adrese, na které je uložen operační kód. **Provedení instrukce** znamená vykonání vnitřních operací v procesoru včetně získání potřebných operandů

z datové paměti (jsou-li operandy v instrukci definovány) a následné uložení výsledků do datové paměti (je-li to v instrukci vyžadováno). Jako příklad může sloužit instrukce „*inkrementuj obsah paměti na adrese ...*“, jejíž provádění je následující – viz *obr. 3.2*.



Obr. 3.2 Příklad činnosti při provádění instrukce

Z výše uvedeného již vyplývají některé potřebné obvody v procesoru. Předně je to **čítač instrukcí** PC (angl. Program Counter), který slouží k adresování paměti programu. Čítač je postupně inkrementován a program je tak postupně čten. Dále je to **instrukční registr** IR, do kterého je uložen operační kód. Na základě obsahu IR je dále řízen průběh provádění instrukce. Má-li instrukce adresovou část, je tato uložena do **pomocného adresového registru** TAR (angl. Temporary Address Register), který bude během následujícího provádění instrukce dodávat adresu pro paměť dat. Pokud instrukce obsahuje datovou část, je tato uložena do **pomocného datového registru** TDR, kde bude k dispozici během provádění instrukce. Tyto (a mnohé další) pomocné registry jsou uživateli zpravidla nedostupné a neviditelné. Vnitřní obvody si předávají data po **vnitřní datové sběrnici**.

Další registry, využívané pro tvorbu adresy, jsou **indexregistry**. Některé jednoduché procesory je nemají, ale většinou jsou alespoň dva. Obsah indexregistru může být v některých instrukcích přičten k obsahu adresové části instrukce a tento součet je teprve použit jako adresa pro paměť. Indexová adresace je výhodná pro práci s tabulkami, řetězci a poli.

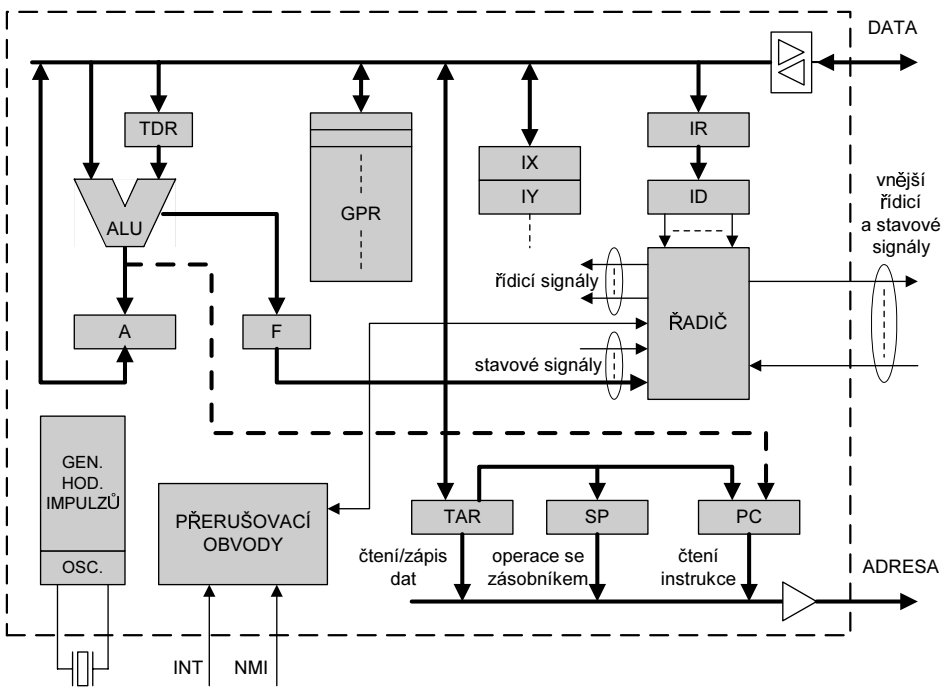
Operace s daty (aritmetické a logické operace) jsou prováděny v **aritmeticko-logické jednotce** ALU (angl. Arithmetic-Logic Unit). Výsledek ve tvaru binárního

čísla je uložen do **střadače** (též **akumulátoru**) A (angl. Accumulator). To však není jediný výsledek – cenná může být i informace o nulovosti přenosu z nejvyššího bitu, přetečení rozsahu čísla atd. Všechny tyto informace jsou ve své podstatě jednobitové. Tyto **příznakové bity** (angl. Flags) jsou uloženy do **registru příznaků** F (též CCR – z angl. Condition Code Register). Jednotlivé bity mohou sloužit např. jako podmínky pro větvení programu.

Pro krátkodobé uložení dat slouží několik **datových registrů** (GPR – angl. General Purpose Register), neboli registrů **zápisníkové paměti**. Jelikož jsou uvnitř procesoru, je přístup k nim rychlejší než k vnější datové paměti. Tyto registry zpravidla plní i jiné úkoly – např. některé mohou sloužit k adresaci datové paměti, k odpočítávání počtu opakování programových smyček atd.

Řízení vnitřních obvodů – nastavení cest dat, zápisy do registrů, řízení operací ALU atd. – provádí **řadič** (angl. Controller). Jedná se vždy o velmi složitý sekvenční obvod, řešený buď jako pevně zapojená logika, nebo jako mikroprogramový automat, v jehož řídicí paměti (paměti mikroprogramů – vždy typu ROM) je naprogramována celá řada mikroprogramů pro různé typy instrukcí. Vždy jeden mikroprogram řídí činnost obvodů procesoru po dobu jedné instrukce.

Jednotlivé posloupnosti řídicích signálů, generované řadičem, jsou vyvolávány **instrukčním dekodérem** ID, který na základě obsahu IR rozliší typ instrukce a tomu přiřadí patřičný počáteční stav sekvenčního obvodu řadiče (či nastaví počáteční adresu mikroprogramu v paměti mikroprogramů).



Obr. 3.3 Základní obvody procesoru

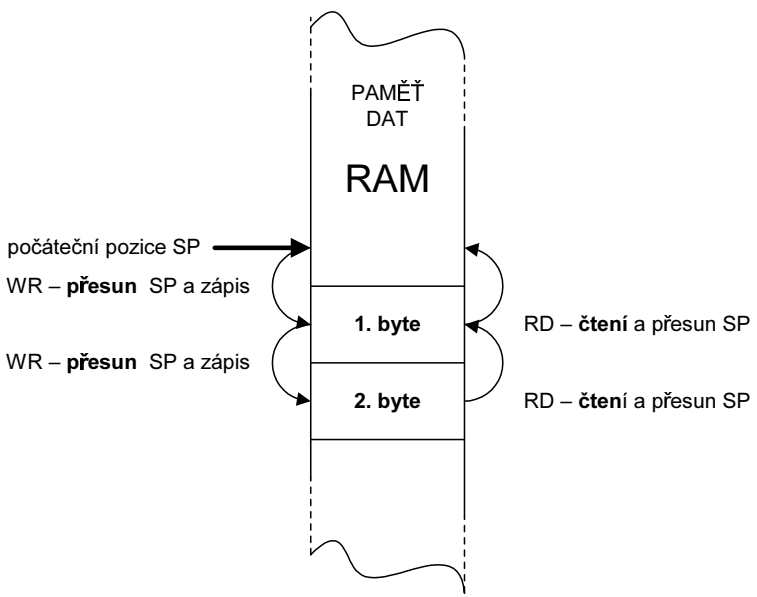
426410

Koncepce řadiče na základě mikroprogramového automatu umožňuje konstruktérovi procesoru snadnou modifikaci instrukčního souboru pouhou změnou obsahu paměti mikroprogramů bez dalších zásahů do architektury procesoru. Naopak řadič, navržený jako pevně zapojená sekvenční logika, umožňuje rychlejší činnost.

Řadič inkrementuje obsah PC tak, aby byly postupně čteny všechny části instrukce a aby byla na konci čtení instrukce k dispozici již adresa následující instrukce. Tak jsou čteny a zpracovávány instrukce s postupně rostoucí adresou v paměti programu. Výjimku tvoří instrukce skoku a volání podprogramů. Ty mají svoji adresovou část, která se přečte a dočasně uloží do TAR. Obsah TAR se pak přesune do PC.

Některé instrukce jsou **podmíněné** (např. podmíněné skoky) v tom smyslu, že jsou provedeny jen při splnění jisté podmínky. Jako podmínky slouží stavy jednotlivých bitů registru F. Příznakové bity jsou zavedeny do řadiče, kde modifikují průběh mikroprogramu. Výběr podmínky je dán tvarem operačního kódu. Tak např. existuje podmíněný skok při splnění podmínky nulovosti výsledku předcházející operace ALU, atd. Při nesplnění podmínky se čte instrukce z následující adresy v programové paměti.

Ukazatel zásobníkové paměti SP (angl. Stack Pointer) slouží pro adresování zásobníku (Stack). Zásobník zaujímá část datové paměti. Při operacích se zásobníkem se v programu neudává adresa. Jedná se o paměť typu LIFO (z angl. Last In – First Out). Adresa je uložena v SP, který je řadičem inkrementován nebo dekrementován, takže zásobník při vkládání dat postupuje na jednu stranu a při



Obr. 3.4 Adresace při postupném zápisu a čtení dvou slabik

vybírání dat na druhou stranu – rozhodující pro růst zásobníku je postup při zápisu. *Obr. 3.4* ukazuje architekturu procesoru, při které zásobník **postupuje dolů**, směrem k nižším adresám. Je třeba upozornit, že u některých procesorů zásobník naopak **postupuje nahoru**. Bez ohledu na směr postupu zásobníku je jeho **vrcholem** (angl. top) nazývána adresa, na kterou ukazuje SP a **dnem** (angl. bottom) adresa, na kterou byla uložena první data.

U moderních procesorů se při zápisu do zásobníku SP nejprve posune a teprve pak je jeho obsah použit jako adresa pro zápis do RAM. Při čtení ze zásobníku je adresa dána obsahem SP, který se teprve po čtení posune. SP tak ukazuje na poslední data uložená do zásobníku.

Ukazatel zásobníku lze přednastavit instrukcí „*naplň SP číslem ...*“ a tím lze založit zásobník v požadované oblasti datové paměti.

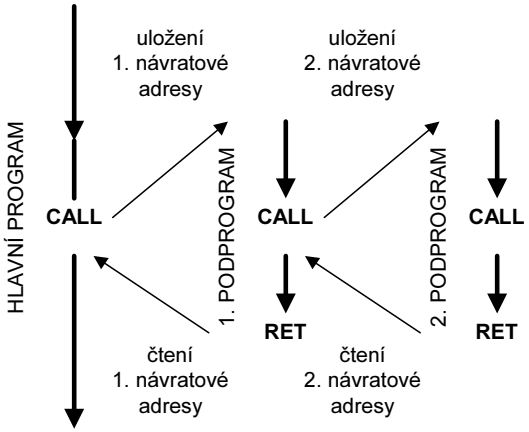
Výše uvedená činnost zásobníku je typická pro instrukce „*ulož data do zásobníku*“ a „*vyber data ze zásobníku*“. Zásobník má však ještě druhou důležitou funkci a to zapamatování adres návratu. K tomu dochází automaticky při volání podprogramů instrukcí „*volej podprogram na adrese ....*“. Po přečtení instrukce je její adresová část dočasně uložena v TAR a PC je inkrementován, takže ukazuje na následující instrukci. Obsah PC je vložen do zásobníku a pak je do PC přesunut obsah TAR. Program tím pokračuje na nové adrese (mimo původní pořadí). Podprogram je zakončen instrukcí „*návrat z podprogramu*“. Tato instrukce způsobí automatické vybrání obsahu zásobníku a jeho přesun do PC. Program proto dále pokračuje tam, odkud byl vytržen instrukcí volání podprogramu. Tento mechanizmus zaručuje, že podprogram lze volat z libovolných míst programu a vždy je zaručen správný návrat. Adresa zapsaná do zásobníku se běžně nazývá „**návratová adresa**“. Přesně vzato je to ale adresa **pokračování** v původním programu. Uchování návratové adresy a její opětné zavedení je u všech procesorů zcela automatické a pro programátora neviditelné.

Je samozřejmé, že před prvním odskokem na podprogram instrukcí „*volej podprogram na adrese ....*“ musí být definován začátek zásobníku, tj. musí být naplněn ukazatel SP vhodným číslem.

Funkce zásobníku umožňuje vložení podprogramů do podprogramů v prakticky libovolném počtu úrovní. Přitom jsou do zásobníku postupně ukládány návratové adresy a při návratech z podprogramů jsou postupně ze zásobníku vybírány. To přesně odpovídá činnosti paměti LIFO – viz *obr. 3.5.*

Používání zásobníku není bez jistého rizika. Jelikož zásobník je vytvořen v datové paměti, může dojít k jeho kolizi s datovou oblastí. **Hloubka zásobníku** (tj. počet vložených údajů) nemusí být předem přesně známa, jak je tomu např. při využívání převzatých knihoven podprogramů bez dostatečné dokumentace, při rekurentních programech, při podmínečných voláních podprogramů. Příliš „narostlý“ zásobník může přepsat data, ale horší případ nastane, když naopak data přepíší obsah zásobníku. Tím jsou ztraceny návratové adresy a zhroucení celého programu je nevyhnutelným důsledkem.

426410



Obr. 3.5 Ukládání adres při vložených podprogramech

Velmi důležitou součástí procesoru jsou obvody pro zpracování **přerušení** programu (angl. Interrupt). Přerušení je vyvoláno vnitřními obvody procesoru při výjimečných stavech (např. dělení nulou, chyba při čtení instrukce, apod.) nebo též vnějšími signály a vyvolá odskok na podprogram. Rozdíl mezi vyvoláním podprogramu instrukcí „*volání podprogramu*“ a vyvoláním podprogramu přerušením spočívá v tom, že k přerušení může dojít kdykoli. Jedná se tedy o okamžitou změnu činnosti počítače jako reakci na neočekávaný vnější podnět. Adresa návratu z podprogramu je v obou případech uchována v zásobníku. Přerušení může být **maskovatelné** nebo **nemaskovatelné**. Procesor je pak vybaven zvláštními vstupy pro maskovatelné přerušení (INT) a pro nemaskovatelné přerušení (NMI, angl. Non-Maskable Interrupt). U maskovatelných přerušení může být podnět blokován v závislosti na stavu vnitřního maskovacího (blokovacího) klopného obvodu. Blokování lze programově nastavovat či nulovat, kromě toho se v některých stavech procesoru přerušení blokuje automaticky. Nemaskovatelné přerušení nelze blokovat a má přednost před přerušením maskovatelným.

K počítači je zpravidla připojen větší počet zařízení, která čas od času vyžadují obsluhu prostřednictvím přerušení. Každému z nich přísluší patřičný obslužný podprogram. Počítač musí identifikovat zdroj požadavku na přerušení, rozhodnout o prioritě při případném konfliktu (tj. při současném výskytu více požadavků), určit adresu obslužného podprogramu a provést odskok. Celou tuto činnost nevykonává procesor samostatně, nýbrž zpravidla využívá vnější obvody pro zpracování přerušení (řadič přerušení). Nemaskovatelné přerušení vyvolává odskok na jedinou pevně stanovenou adresu bez spoluúčasti dalších obvodů, reakce je tak rychlejší. Návrat z podprogramu je záležitostí programátora, který zakončí obslužný podprogram instrukcí „*návrat z interruptového podprogramu*“.

Obdobně jako v případě volání podprogramu, i při přerušení musí již být předem správně nastaven začátek zásobníku, tj. naplněn ukazatel SP – do té doby musí být přerušení zakázána (blokována).

V podprogramu se většinou pracuje s registry zápisníkové paměti, ve kterých se tím přepíší data, uložená do nich v programu, ze kterého se odskočilo do podprogramu. Po návratu tak může dojít k chybě. To platí zvláště v případě přerušení, které obecně může nastat kdykoliv, bez zásahu a bez vědomí programátora. Na začátku podprogramu je proto nutné obsah pracovních registrů „uklidit" instrukcemi „*ulož data do zásobníku*" a na konci podprogramu pak zpětně obnovit instrukcemi „*vyber data ze zásobníku*". Tento **úklid** a **návrat** představuje časové ztráty, ale bez změny architektury procesoru je nevyhnutelný.

Pro omezení časových ztrát byla proto vyvinuta architektura se zdvojenou nebo i vícenásobnou zápisníkovou pamětí – tzv. **registrové banky**. V každém okamžiku je využívána jen jedna banka. Mohou však být přepínány prostřednictvím jednoho nebo více bitů registru, nazvaného **přepínač bank** (angl. bank switch). Zásahem do tohoto registru lze přepnout registrovou banku a pracovat s další skupinou registrů, aniž by obsah předchozí skupiny mohl být porušen. Ve vhodném okamžiku lze přepnout zpět. To se využívá např. v podprogramech (zvláště interruptových), kdy v podprogramu se využívá jiná banka, než ve volajícím programu. Obsah registrů, využívaných ve volajícím programu, tak je po návratu dále k dispozici. Celé přepnutí je záležitostí jen jedné krátké instrukce.

Registr příznaků CCR a další bity, důležité pro řízení činnosti procesoru, jako je např. bit pro blokování přerušení, přepínač bank registrů apod., jsou seskupeny do **stavového slova programu** PSW (angl. Program Status Word). PSW lze jako jeden celek ukládat do zásobníku na začátku podprogramu (zvláště interruptového) a na konci podprogramu je lze opět ze zásobníku obnovit.

Vstup **nulování** (angl. Reset) nastavuje počáteční podmínky pro zahájení činnosti procesoru po náběhu napájení, kdy vnitřní obvody (ze značné části jsou to sekvenční obvody) se dostaly do náhodného stavu. Nejdůležitější akcí je nastavení počátečního stavu čítače instrukcí – tím je dán počátek programu v programové paměti. Blokují se případná přerušení a jiné abnormální stavy a podle typu procesoru se nastavuje počáteční stav ještě dalších obvodů.

Pro synchronizaci vnitřních obvodů procesoru slouží **hodinové impulzy**. Hodinové impulzy procesoru jsou často využity i v jiných obvodech počítače. Pro zvýšení výkonnosti je žádoucí pracovat s jejich co nejvyšším kmitočtem. Vždy je proto používán krystalový generátor (oscilátor), neboť přesně definovaný kmitočet umožňuje držet se horní meze rozsahu povoleného výrobcem. Generátor je často integrován v procesoru. Krystalový rezonátor je vždy vnější.

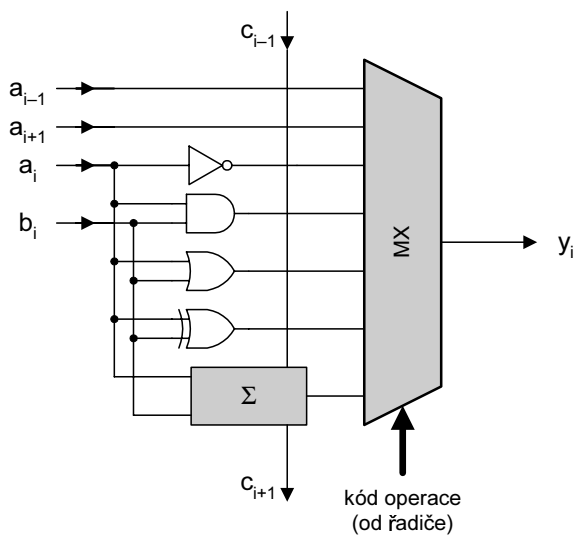
Souhrn všech činností mikroprocesoru, potřebných pro provedení jedné instrukce, se nazývá „**instrukční cyklus**" (angl. Instruction Cycle). Instrukční cyklus lze rozložit na dílčí úseky, nazývané „**strojové cykly**" (angl. Machine Cycle). V každém strojovém cyklu se právě jednou provede čtení nebo zápis (z/do paměti, vstupů a výstupů či jiných obvodů). Existuje několik typů strojových cyklů – např. čtení operačního kódu, zápis do paměti atd. Vždy první strojový cyklus v instrukč-

ním cyklu je čtení operačního kódu. Další strojové cykly jsou pak různé podle instrukce. Strojové cykly se skládají z „**taktů**“, daných periodou vnitřních hodinových impulzů procesoru.

Časování signálů vnitřních sběrnic procesoru se může lišit od časování signálů vnějších sběrnic, po kterých procesor komunikuje s ostatními obvody počítače. Je tak definován „**vnější sběrnicový cyklus**“. Ten je podstatný pro návrh obvodů počítače, neboť vnitřní sběrnice procesoru, které mnohdy tvoří značně složitý systém, nejsou uživateli dostupné.

3.2 ARITMETICKO-LOGICKÁ JEDNOTKA

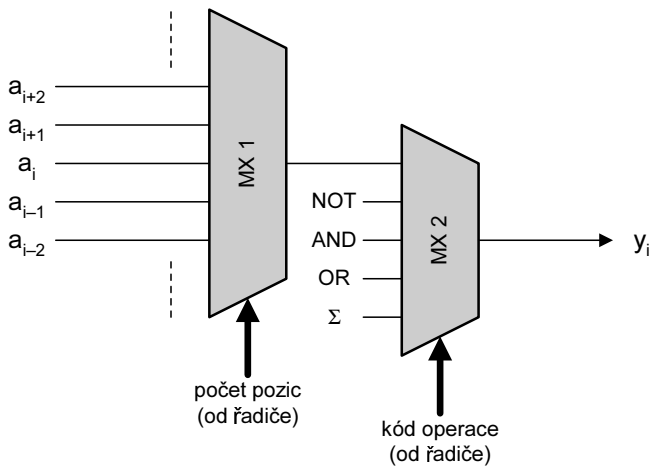
Hlavní aritmetické operace, prováděné ALU, jsou součet, rozdíl, změna znaménka, násobení a dělení. Hlavními logickými operacemi jsou AND, OR, EX-OR, negace, posuny a rotace vlevo či vpravo. Operandy jsou dodávány z datových registrů (ze zápisníkové paměti) nebo z vnější paměti, výsledky jsou uloženy opět do některého z datových registrů nebo do paměti. K výsledkům operace patří i nastavení jednotlivých bitů příznakového registru. Typ (kód) operace je zadáván řadičem. Na *obr. 3.6* je znázorněna velmi jednoduchá ALU pro 1 bit slova. Obvody pro další bity jsou shodné. Operace jsou zde omezeny – chybí rozdíl, násobení, dělení a změna znaménka. Typ operace je zadáván jako kód, kterým je prostřednictvím multiplexeru vybrán jeden z výsledků dílčích obvodů. Pro posuny vlevo či vpravo jsou do obvodů pro bit y_i zavedeny i sousední bity a_{i+1} (posun vpravo) a a_{i-1} (posun vlevo).



Obr. 3.6 Část jednoduché aritmeticko-logické jednotky pro 1 bit

Zřejmě se jedná o ryze kombinační obvody. Rozšířením o násobení a dělení se složitost ALU významně zvětšuje. Násobení dvou slov ve všech bitech současně vyžaduje obvody velmi složité, které mají opodstatnění jen u procesorů s extrémními nároky na rychlost, jako jsou signálové procesory. Druhou krajností je násobení bit po bitu s posuny jednoho operandu – to je příliš pomalé. V běžném případě se volí kompromis a násobička je konstruována na násobení skupiny bitů (půlbyte či byte), takže celé slovo je vynásobeno jen v několika málo krocích. Operace se tak provádí v několika krocích a příslušné obvody jsou již sekvenční. Totéž platí o operaci dělení.

Posuny vlevo a vpravo o větší počet pozic mohou být prováděny postupně, v několika shodných instrukcích posunu o 1 místo – v některých případech tak však může vzniknout významná časová ztráta a proto některé procesory mají možnost provádět takovéto posuny v jedné instrukci. K tomu je nutná dosažitelnost i vzdálenějších bitů, jak ukazuje *obr. 3.7*.



Obr. 3.7 Obvody pro posun vlevo či vpravo o větší počet kroků

U mnoha procesorů je ALU doplněna ještě dalšími obvody. Především slouží pro aritmetické operace s čísly **s plovoucí řádovou čárkou** (čísla typu „float“). Toto řešení, ačkoliv podstatně zvětšuje složitost ALU, vede k daleko rychlejší činnosti počítače proti běžnějšímu programovému řešení pomocí knihovny aritmetických funkcí v plovoucí čárce. Využívá se zvláště u signálových procesorů.

Existují i jiné doplňky ALU, které vyplývají ze snahy výrobce vylepšit dlouhodobě osvědčený procesor a prodloužit mu tak život. Tak se např. můžeme setkat s **jednotkou MAC** (z angl. multiply and accumulate), která provádí operace násobení dvou čísel, na která ukazují dva adresovací registry, a přičtení výsledku k obsahu speciálního akumulátoru. To je často zapotřebí v úlohách číslicového zpracování signálu. Pro dosažení dostatečného dynamického rozsahu má akumulátor v MAC velmi dlouhé slovo (např. 36 bitů).

426410

3.3 PŘÍZNAKOVÉ BITY

Příznakové bity jsou dalšími výstupy ALU. Ačkoliv jejich počet, označení a význam se může u různých procesorů lišit, některé vlastnosti jsou obecně platné. Některé bity jsou uloženy do příznakového registru, jsou zapamatovány a mohou být použity později. V tomto případě bity vyjadřují výsledek poslední operace, která na ně měla vliv. Zápisový impulz do registru vydává řadič jen v některých instrukcích. U některých procesorů jsou však některé bity vydávány přímo, mimo registr. Pak je jejich stav závislý na okamžité situaci na výstupu akumulátoru. Nejsou tedy zapamatovány. To může být příčinou neúspěchu při nepozorném přepisování programu v assembleru jednoho procesoru do jiného.

Kromě automatického ovlivňování v průběhu instrukce lze do registru příznaků F zasáhnout též přímo (programově) a záměrně tak ovlivnit stav příznakových bitů. U některých procesorů k tomu existují speciální instrukce, u jiných je F přístupný stejně jako datové registry.

Bit C (též CY) – přenos (angl. carry) z nejvyššího bitu. Vždy pamatován. Mění stav při aritmetických operacích, operacích porovnání a některých typech posunů a rotací.

Bit Z – nulovost (angl. zero). U některých procesorů je zapamatován jako výsledek aritmeticko-logické operace, u jiných se vztahuje na momentální obsah akumulátoru. Po instrukcích přesunů dat do A tedy v prvém případě není bit Z ovlivněn, ve druhém ano.

Bit S – znaménko čísla (angl. sign), **též N** (angl. negative). Je nastaven na 1, byl-li znaménkový bit výsledku 1 (tj. záporné číslo) – jinak je vynulován. U některých procesorů je zapamatován, u jiných není vůbec realizován a test znaménka je nahrazen instrukcí „*testuj bit ...*“.

Bit V (též OV) – přetečení (angl. overflow). Vždy pamatován. Mění stav při přetečení čísla se znaménkem (nejvyšší bit slova) po operaci součtu, rozdílu, obrácení znaménka, násobení a dělení. Operandy i výsledek se předpokládají ve dvojkovém doplňku. Kladný výsledek nemůže být větší než 7FF...F, vyšší hodnota by způsobila změnu znaménkového bitu. Záporný výsledek nemůže být menší než 800...0 a nižší hodnota by opět způsobila změnu znaménkového bitu. U operace dělení je takto zpravidla signalizováno dělení nulou.

Bit H (též AC) – poloviční přenos (angl. half-carry) nebo pomocný přenos (angl. auxiliary carry). Vždy pamatován. Jedná se o přenos z bitu D_3 do D_4 při operaci součtu. Je využíván následnou instrukcí dekadické korekce akumulátoru, pokud sčítaná čísla byla v BCD kódu.

Bit P – parita (angl. parity) vyjadřuje paritu výsledku (počet jedniček). Při liché paritě je P = 1 při lichém počtu jedniček, u sudé parity je P = 1 při sudém počtu jedniček. Je třeba zjistit, jakou paritu bit vyjadřuje. Paritní bit mají jen některé procesory. Kontrola paritou slouží hlavně jako diagnostický prostředek, je však značně nespolehlivá.

Uvedené příznakové bity mohou být doplněny ještě dalšími, méně běžnými, podle architektury procesoru.

426410

3.4 DALŠÍ VYUŽITÍ ALU

V mnoha instrukcích je adresa v datové paměti tvořena jako součet adresové části instrukce a obsahu dalšího registru (např. indexová adresace). Dále adresa následující instrukce v programové paměti je dána jako adresa současně prováděné instrukce plus délka této instrukce – při proměnlivé délce instrukcí tedy zřejmě nestačí jen prostá inkrementace PC. Existují též instrukce s relativní adresací (relativní skoky), kdy adresa následující instrukce je tvořena jako momentální obsah PC plus adresová část skokové instrukce. Je tedy zřejmé, že pro adresaci v datové paměti i v programové paměti je nutné provádět operaci součtu dvou čísel. K tomu účelu lze využít již existující ALU. Je to běžně používané řešení, které má výhodu obvodové úspornosti. Nevýhodou je ale prodloužení instrukčního cyklu, neboť jedna jediná ALU nemůže všechny výše uvedené operace (a navíc ještě aritmetické a logické operace v příslušných instrukcích) provádět **současně**.

Při vyšších nárocích na rychlost jsou proto procesory vybaveny několika sčítačkami vedle univerzální ALU. Např. jedna sčítačka je spojena s PC, druhá vytváří adresu pro datovou paměť.

V algoritmech číslicového zpracování signálu (i některých dalších) je třeba opakovaně zpracovávat pole dat. Adresa v datové paměti se v nejjednodušším případě postupně inkrementuje (zvyšuje) o jedničku, ale častěji ve větších krocích a v aritmetice *modulo M*. Při extrémních nárocích na rychlost provádění těchto algoritmů je procesor (signálový procesor) vybaven speciální **adresovací jednotkou** – generátorem adres. Ta přebírá úlohu automatické generace adres, kterou by jinak bylo nutno řešit programově a tudíž mnohem pomaleji.

3.5 PROSTŘEDKY PRO ZRYCHLENÍ ČINNOSTI PROCESORU

První a samozřejmou možností je zvyšování hodinového kmitočtu. Závisí na pokrocích v technologii výroby integrovaných obvodů. V tomto směru se zatím stále postupuje. Druhou možností je zvětšování délky slova. Ve srovnání s ALU s osmibitovými daty bude ALU se šestnáctibitovými daty znamenat dvojnásobnou rychlost práce (měřeno v počtu bitů za sekundu), u 32bitových a 64bitových slov je zrychlení samozřejmě ještě větší. Zvětšování délky slova ale znamená vyšší cenu pouzder integrovaných obvodů (větší počet vývodů datové sběrnice), vyšší složitost a vyšší cenu plošného spoje. Třetí možností jsou zásahy do architektury procesoru.

Jednoduchá architektura, tak jak ji ilustruje *obr. 3.3*, bude zpracovávat instrukci „*sečti obsah proměnné ..., indexová adresace, s obsahem akumulátoru*“, přičemž se předpokládá, že výsledek zůstává v akumulátoru. Posloupnost činností během instrukčního cyklu ukazuje *obr. 3.8*. Program je v tomto případě uložen v paměti ROM, data v paměti RAM. Procesor je vybaven indexregistrem a adresa pro RAM je generována jako součet adresové části instrukce a obsahu indexregistru.

čtení OP kódu z ROM	dekódování	čtení adr. operandu z ROM	generace adr. pro RAM	čtení operandu z RAM	provedení arit./log. operace	další instrukce

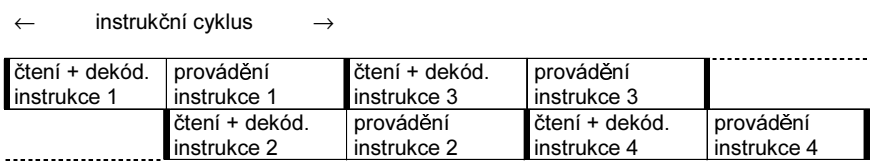
Obr. 3.8 Posloupnost činností jednoduchého procesoru

Využití vnějších sběrnic je vyznačeno tlustou čárou. Na *obr. 3.8* se pravidelně střídá činnost s nečinností, v jiných instrukcích však může obraz vypadat jinak. Jedině čtení operačního kódu a jeho dekódování se shoduje u všech instrukcí, další činnost a tedy i využití sběrnic se může lišit podle typu instrukce.

Z toho je zřejmé, že:

- Vnější sběrnice jsou využity jen po část instrukčního cyklu, v podstatě nepravidelně.
- Během čtení operačního kódu a jeho dekódování je nečinná ALU, naopak během provádění instrukce nepracuje dekodér instrukcí.

Zrychlení činnosti je možné tak, že rozdělíme instrukční cyklus na dvě fáze: fázi čtení + dekódování, a fázi provádění. Umožníme tak čtení a dekódování další instrukce již během provádění instrukce předcházející, jak ukazuje *obr. 3.9*.



Obr. 3.9 Překrývání dvou fází instrukcí

V ideálním případě se tak zřejmě dosáhne dvojnásobného zrychlení instrukčního cyklu. Procesor v každém okamžiku pracuje se dvěma instrukcemi. Je však nutné zajistit oddělení obou stupňů oddělovacími registry – zápisem do registru se výsledek činnosti prvého stupně přesune do druhého stupně, a prvý stupeň se tak uvolní pro novou instrukci. Zcela zřejmě se jedná o zřetězenou strukturu – „**pipeline**“, známou z teorie číslicových systémů. Princip zřetězení lze aplikovat ještě dále a vyrovnávacími registry rozdělit procesor na větší počet úseků, čímž se efekt zřetězené struktury projeví ještě markantněji. Procesor tak zpracovává vždy několik instrukcí, vzájemně posunutých. Rozdělení na dílčí úseky může vypadat následovně:

- Čtení OP kódu z programové paměti.
- Dekódování instrukce.
- Výpočet adres operandů.
- Čtení operandů z datové paměti.
- Činnost ALU, která sama může být zřetězena. Zvláště je-li vybavena pro operace v pohyblivé čárce, které jsou vždy prováděny v několika krocích.
- Zápis výsledků do datové paměti.

426410

Další úpravou je rozdělení procesoru na dvě samostatné jednotky – instrukční jednotku a výkonnou jednotku. **Instrukční jednotka** (angl. Bus Interface Unit, **BIU**) čte instrukce z programové paměti. Instrukce dekóduje a čte operandy z datové paměti. Používá k tomu svůj vlastní generátor adres. Dekódované instrukce (výstup z instrukčního dekodéru) a operandy předává výkonné jednotce. **Výkonná jednotka** (angl. Execution Unit, **EU**) má svůj řadič a ALU. Pokud výsledek operace EU má být uložen do datové paměti, děje se tak opět prostřednictvím BIU.

Mezi BIU a EU je vložena paměť typu FIFO (paměť fronty), která vyrovnává krátkodobé rozdíly mezi rychlostmi práce obou jednotek. V této vyrovnávací paměti je uloženo několik dekódovaných instrukcí včetně operandů. Díky tomu nemusí jedna jednotka čekat na druhou. BIU plně využívá možnosti vnějších sběrnic a zaplňuje FIFO v případě, že EU právě pracuje na instrukcích s dlouhou dobou zpracování (např. násobení, dělení). EU má ve FIFO rezervu několika instrukcí v případě, že právě prováděné instrukce potřebují jen krátkou dobu na zpracování (např. vnitřní přesuny mezi registry) a BIU nemůže dodávat nové instrukce tak rychle. Vnější sběrnicové cykly neodpovídají právě vykonávané instrukci v EU. Pro účely ladění programů může však být tato informace o tom, která instrukce je právě vykonávána, nezbytná. Některé procesory proto mají zvláštní výstupy, na kterých vydávají informaci o stavu zaplnění FIFO.

Skutečnost, že zřetězený procesor pracuje na několika instrukcích současně, však přináší i komplikace. Zřetězená struktura představuje zrychlení jen při plynulém dodávání dat na vstup. Narušení plynulosti nastává v instrukci skoků nebo volání podprogramů, kdy za touto instrukcí jsou v řetězci (a ve FIFO) již načteny a případně částečně zpracovány další instrukce, které bezprostředně následovaly v programové paměti. Provedením skoku se dostáváme do jiné oblasti programové paměti, kde následují samozřejmě zcela jiné instrukce. FIFO je nutno z BIU naplnit novými instrukcemi. Zvláště obtížná je situace u podmíněných skoků, kdy nelze předvídat, kterou ze dvou větví bude program pokračovat. Existují sice hardwarové i softwarové prostředky, kterými lze v některých případech frontu instrukcí zachovat (zdvojené FIFO, každé je plněno jednou alternativní sekvencí instrukcí a k přepnutí dojde po vyhodnocení podmínek skoku) a rozpracované instrukce dokončit (metoda „zpožděného skoku“), ale obecně platí, že časté skoky a volání snižují účinnost zřetězeného zpracování.

Ke konfliktům může docházet i v datech. Je to v případě, že instrukce 1 je právě dokončována a její výsledek má být uložen do paměti. Současně je ale načítán operand instrukce 2, který je shodou okolností na stejné adrese v paměti. Při sekvenčním zpracování instrukcí, kdy data, zpracovaná instrukcí 1 a uložená do paměti, jsou v instrukci 2 z paměti přečtena a dále zpracována, nevzniká žádný problém, ale u zřetězeného zpracování ano. Přečtení dat v instrukci 2 (což následuje ihned po přečtení OP kódu) totiž může předběhnout uložení dat do paměti v instrukci 1. Konflikt lze řešit na úrovni software při překladu – překladač mezi dvě konfliktní instrukce vloží potřebný počet prázdných instrukcí (NOP), čímž vznikne potřebný časový odstup. Lepší řešení spočívá v hardwarové úpra-

vě procesoru o obvody, detekující v instrukčním dekodéru konfliktní skupiny instrukcí (na základně shodnosti adres operandů). Načítání operandů pro instrukci 2 je automaticky zpožděno až za okamžik zápisu výsledku instrukce 1.

3.6 PROCESORY TYPU CISC A RISC

Procesory typu CISC jsou základem počítačů CISC (angl. Complex Instruction Set Computer) – počítačů se složitým souborem instrukcí. Procesory typu RISC jsou základem počítačů RISC (angl. Reduced Instruction Set Computer) – počítačů se zjednodušeným souborem instrukcí. Počítače RISC jsou dosud zvláštní, méně obvyklou kategorií proti běžnějším počítačům CISC. Soubory instrukcí moderních procesorů jsou velmi rozsáhlé a mnohé z nich realizují operace tak složité, že by je bylo možné nahradit celým úsekem programu, sestaveného z jednoduchých instrukcí. Často jsou tyto instrukce „šity na míru“ pro snadnou kompilaci z jazyka C. Důsledkem je ale velmi složitý řadič procesoru, řešený vždy jako mikroprogramový automat. Celý procesor je pomalejší.

Naopak u procesoru RISC je soubor instrukcí zúžen na pečlivě vybrané jednoduché, často používané instrukce. Tím je usnadněna účinná optimalizace obvodů procesoru s cílem maximálního zrychlení činnosti. Řadič je konstruován jako klasický sekvenční obvod s důrazem na maximální rychlost. Program, sestavený z jednoduchých instrukcí RISC, je samozřejmě delší než program, sestavený z instrukcí CISC – celková doba na jeho provedení je však kratší vzhledem k rychlosti provádění jednoduchých instrukcí. Pro účinnost této metody je však nutné pozměnit architekturu procesoru.

Procesory RISC se vyznačují těmito vlastnostmi:

- Instrukce mají jednotnou délku (v počtu byte) a jsou prováděny zpravidla v jednom strojním cyklu. Tím je zvýšena účinnost zřetězeného zpracování. Výjimku tvoří instrukce, pracující s vnější datovou pamětí.
- Typické jsou instrukce tříoperandové – např. „*sečti obsah registru 1 s obsahem registru 2 a výsledek ulož do registru 3*“. Adresa registrů však nesmí být dlouhá, aby se neprodlužovala instrukce.
- Většina instrukcí pracuje s vnitřními registry. Ty jsou dosažitelné mnohem rychleji, než vnější paměť. K adresování registrů postačí malý počet bitů adresy, proto i tříoperandové instrukce mohou být krátké.
- Výsledek operace ALU není ukládán do akumulátoru, ale do libovolného vnitřního registru. Tím odpadají časté přesuny dat z a do akumulátoru.
- Je běžné přepínání bank registrů.
- Důsledně je uplatněno zřetězené zpracování.
- Kompilátor z vyššího jazyka je složitý, optimalizuje rychlost a může za tím účelem i měnit pořadí instrukcí.

426410

4 ZÁKLADNÍ TYPY INSTRUKCÍ

4.1 ADRESACE V INSTRUKCÍCH

Adresa slouží k určení **zdroje** dat (angl. source address) nebo **cíle** dat (angl. destination address). Nejčastější způsoby adresace jsou následující:

- **Bezprostřední data** (angl. immediate). Data jsou obsažena již v instrukci, která v tomto případě obsahuje datovou část. Tento způsob adresace je uváděn jen pro úplnost, žádná adresa vlastně není zapotřebí.
- **Přímá adresa** (angl. direct). Adresa je obsažena v instrukci, která v tomto případě obsahuje adresovou část.
- **Nepřímá adresa** (angl. indirect). V instrukci není uvedena adresa, ale zdroj adresy. Během provádění instrukce je tento zdroj přečten a tak je teprve získána adresa. Zdrojem adresy může být buď některý z vnitřních registrů procesoru, nebo paměť. Jedná se tedy o nepřímou adresaci prostřednictvím registru nebo prostřednictvím paměti. Při nepřímé adresaci registrem je definice registru součástí instrukčního kódu (počet možných registrů je malý), takže instrukce je krátká. Rovněž získání adresy proběhne rychle, neboť registr je součástí procesoru a může rovnou sloužit jako zdroj adresy. Naopak při nepřímé adresaci prostřednictvím paměti musí být v instrukci uvedena nepřímá adresa v plné délce. Provádění instrukce je pomalejší, přibývá jeden cyklus (čtení adresy z paměti). Výhodou však je to, že počet možných zdrojů nepřímé adresy je prakticky neomezený, na rozdíl od počtu registrů při prvém způsobu. Nepřímá adresa je často využívána ve vyšších programovacích jazycích (typ *pointer* v C).
- **Relativní adresa** (angl. relative). Instrukce obsahuje adresovou část, tzv. **offset**, jejíž obsah se přičte k obsahu čítače instrukcí. Využívá se nikoliv ke čtení nebo zápisu dat, ale k získání adresy následující instrukce v některých instrukcích skoků nebo smyček typu „*skoč o offset dále od momentální polohy čítače instrukcí*“. Offset je vždy chápán jako číslo se znaménkem, takže skoky v programu jsou možné vpřed i vzad. Offset bývá kratší než přímá adresa (typicky 1 slabika), takže relativní adresace je úsporná z hlediska délky instrukce. Vzdálenost v programu, kterou lze skokem překlenout, je však omezená.
- **Indexová adresa** (angl. indexed). K adresové části instrukce je přičten obsah indexregistru a tento součet je teprve použit jako adresa pro data

(přímá indexová adresace), nebo jako adresa pro zdroj adresy (nepřímá indexová adresace). Indexregistry jsou nejčastěji dva a jsou součástí procesoru. Indexová adresace je výhodná pro práci s tabulkami (pole v jazyce C).

Pro realizaci některých algoritmů (např. číslicového zpracování signálů) je nutné mnohokrát probíhat programovou smyčkou a zpracovávat data s postupně se měnícími adresami. Obdobně je nutné postupně zvyšovat adresy při přesunech bloků dat z jedné oblasti paměti do druhé. V těchto případech se využije možnost **autoinkrementace** či autodekrementace adresy. Adresace je přitom nepřímá prostřednictvím registru. Na konci provádění instrukce je automaticky inkrementován (dekrementován) obsah adresovacího registru. Některé instrukce umožňují i inkrementaci (dekrementaci) dvou registrů, takže jsou automaticky posunovány ukazatele na adresu zdroje a cíle při přesunech bloků, nebo ukazatel do pole vzorků a ukazatel do pole koeficientů (typické např. pro diskrétní Fourierovu transformaci). U některých procesorů (signálových procesorů) může být velikost inkrementu (dekrementu) volitelná a adresu lze generovat v aritmetice „modulo M“. To je zvláště důležité pro rychlé provádění algoritmů číslicového zpracování signálů. Absolutní adresa, vypočítaná v průběhu instrukce z adresy uložené v adresové části instrukce, se nazývá **efektivní adresa**.

Všechny výše uvedené způsoby adresace nejsou samozřejmě dostupné u každého procesoru. Dokonalejší (zvláště 16 a vícebitové) procesory je však obsahují zcela běžně.

Pro některé programové úlohy je nutné znát **pořadí ukládání** jednotlivých slabik v paměti. Týká se adres a dlouhých proměnných. Existují dvě možnosti: nejnižší slabika na nejnižší adrese (pořadí **LO-HI**, tzv. „**Little Endian**“), a nejvyšší slabika na nejnižší adrese (pořadí **HI-LO**, tzv, „**Big Endian**“). Pořadí se liší i u různých procesorů téhož výrobce a je to důležitá informace, kterou je nutno v dokumentaci někdy pracně hledat. Pořadí bitů ve slabice je vždy takové, že nejvyšší bit ve slabice **MSB** (angl. Most Significant Bit) je na pozici D_7, nejnižší bit **LSB** (angl. Least Significant Bit) je na pozici D_0.

4.2 TYPY INSTRUKCÍ A JEJICH PROVÁDĚNÍ

Soubory instrukcí různých procesorů se sice vzájemně liší, přesto však lze nalézt společné vlastnosti, podle kterých mohou instrukce být zařazeny do několika hlavních skupin. Pro programování v assembleru je nutné znát přesnou syntaxi jednotlivých instrukcí i činnost počítače v nich. Detaily lze nalézt ve firemní literatuře. I **symbolické názvy** instrukcí, též „mnemotechnické instrukce“ (angl. **mnemonic code**) se liší u různých výrobců. V této kapitole nebude popisován soubor instrukcí žádného konkrétního typu procesoru, nýbrž bude podán přehled základních instrukcí. Instrukce zde uváděné jsou **fiktivní**, i když bylo snahou vybrat názvy blízké názvům skutečným a nejčastějším, či nejvýstižnějším.

Operandy v instrukcích jsou v dalším textu značeny následovně:

x, y	přímá adresa, nepřímá adresa prostřednictvím paměti, nepřímá adresa prostřednictvím registru, označení registru, přímý operand
reladdr	relativní adresa (zpravidla krátká)
r	označení registru
inaddr	adresa vstupního obvodu
outaddr	adresa výstupního obvodu

Uvedené značení operandů je velmi hrubé a vhodné jen pro popis typů instrukcí. Pro zjištění všech možností a omezení u konkrétního typu procesoru je nutné se seznámit s dokumentací výrobce.

Instrukce přesunů dat. Přesuny jsou možné mezi vnitřními registry procesoru, mezi pamětí a vnitřním registrem a mezi dvěma adresami v paměti. Obsah cíle dat se přepíše novými daty, obsah zdroje dat se nemění.

Typické instrukce přesunů:

```
MOV x,y          ;univerzální
LDr x            ;z paměti do registru
STr x            ;z registru do paměti
Tr1r2            ;z registru do registru – mění se obsah
                 ;cílového registru
EXr1r2           ;vzájemná záměna obsahu – mění se obsahy
                 ;obou míst!
```

Směr přenosu je dán konvencí při psaní instrukcí, tzv. „směr šipky“ – zpravidla na prvém místě je označení cíle dat a na druhém místě zdroje dat (ale pozor, u některých procesorů může u některých instrukcí být směr opačný – pečlivé prostudování konkrétního seznamu instrukcí je vždy nutné).

Instrukce aritmeticko-logické. Jsou prováděny v akumulátoru, kde je pak rovněž uložen výsledek. Pokud instrukce pracuje se dvěma operandy, musí zpravidla být jeden již předem uložen v akumulátoru a druhý je určen v instrukci. Některé procesory však umožňují pracovat s operandy v registrech zápisníkové paměti a směrovat výsledek rovněž do nich, nebo do paměti. To může významně zrychlit program, neboť je tak odstraněna nutnost častého přesunování operandů mezi akumulátorem a registry či pamětí.

Typické aritmetické instrukce:

```
ADD x            ;operand se přičte k obsahu akumulátoru
ADDC x           ;totéž, navíc se přičte obsah bitu C
SUB x            ;operand se odečte od obsahu akumulátoru
```


426410

```
SUBB x          ;totéž, navíc se odečte obsah bitu C
                ;(záporný přenos, "borrow")
```

Jeden operand je předem uložen v akumulátoru. Instrukce se započítáním přenosu jsou využívány hlavně při práci s dlouhými proměnnými o větším počtu slabik, kdy součet (rozdíl) je prováděn postupně (po slabikách). Některé procesory mají pro jednoduchost jen instrukce ADDC a SUBB, takže při operacích bez započítání přenosu se musí předem vynulovat bit C.

```
NEG x           ;obrácení znaménka (negace všech bitů +1)
INC x           ;inkrementace obsahu (přičtení 1)
DEC x           ;dekrementace obsahu (odečtení 1)
CLR x           ;vynulování obsahu

DAA             ;dekadická korekce akumulátoru
```

Používá se po předcházející operaci součtu dvou čísel v BCD kódu (Binary Coded Decimal). Každá dekadická číslice je kódována čtveřicí bitů, „půlbyte“ (angl. nibble), takže v jedné slabice jsou dvě dekadické číslice (tzv. kompaktní tvar BCD). V binárním kódu jsou čtveřice bitů vyjádřeny jako hexadecimální (šestnáctkové) číslice s hodnotami 0...9, A, B, C, D, E, F. Binární sčítačka ovšem sčítá BCD čísla jako by byla binární a tím vzniká chyba. Tu je třeba korigovat. Pokud součet v rámci čtveřice bitů nepřesáhne 9, je vše v pořádku. Je-li součet v rozsahu 10 až 15 (AH až FH), mělo by dojít k přenosu do vyššího dekadického řádu (tj. do vyšší čtveřice bitů). U binárních čísel však ještě k přenosu nedojde, rozsah čtveřice bitů je 0...15 (0...FH). Až při výsledku 16 a výše dojde k přenosu, což je signalizováno nastavením příznakového bitu H (half-carry). Korekce je provedena přičtením čísla 6 kdykoliv součet v rámci dolní čtveřice bitů přesáhne 9 nebo když příznakový bit H = 1. Totéž se provede v horní čtveřici bitů, jen místo bitu H se pracuje s bitem C. Některé procesory mají ještě instrukci pro korekci po rozdílu dvou BCD čísel. Průběh je obdobný, šestka se však odečítá.

```
MUL             ;násobení čísel bez znaménka
MULS            ;násobení čísel se znaménky
```

U násobení bývá předem přesunut jeden operand do akumulátoru a druhý do jiného registru, pro tuto instrukci pevně předurčeného. Výsledek (dvojnásobné délky) je pak v obou těchto registrech.

```
MAC x           ;součin a akumulace (angl. multiply and
                ;accumulate),
                ;výsledek akumuluj na adrese x
```

Pro mnoho algoritmů číslicového zpracování dat, statistických výpočtů apod. je nutné opakovaně násobit dvě čísla (např. vzorky signálu a váhové koeficienty z tabulky) a dílčí součiny stále přičítat. Přitom nároky na rychlost zpracování bývají značné. Novější procesory proto byly vybaveny instrukcí MAC. Akumulátor

426410

v tomto případě bývá jiný než je registr A v ALU – je to buď další, specializovaný akumulátor, dvojice registrů zápisníkové paměti, nebo proměnná dvojnásobné délky v paměti. Dvojnásobná délka je nutná k zabránění přetečení při mnohonásobném opakování MAC. Ani dvojnásobná délka akumulátoru však není mnohdy dostatečná a specializovaný akumulátor má např. 36 bitů (rezerva 4 bity nad 4 slabiky). Adresování v instrukci MAC je typicky pomocí indexregistrů (musí se měnit adresy obou operandů) nebo jiných registrů, předurčených pro tuto operaci. Výsledek je u některých procesorů směrován jedině do speciálního akumulátoru a pak MAC nemá adresovou část. Důležitá pro rychlé opakování MAC ve smyčce je tvorba nových adres operandů. Pokud nejsou adresy vytvářeny automaticky, je jejich programová tvorba zpravidla příliš pomalá a účinnost MAC je podstatně snížena.

```
DIV             ;dělení čísel bez znaménka
DIVS            ;dělení čísel se znaménky
```

U dělení bývá umístěn dělenec v akumulátoru, dělitel v jiném registru, pro tuto instrukci pevně předurčeném, výsledný celočíselný podíl zůstává v akumulátoru, a zbytek po dělení v registru, předurčeném pro dělitele.

Typické logické instrukce:

```
AND x           ;logický součin operandu s obsahem
                ;akumulátoru
OR x            ;logický součet operandu s obsahem
                ;akumulátoru
XOR x           ;exkluzivní logický součet operandu
                ;s obsahem akumulátoru
NOT x           ;též COM – negace (komplement) všech
                ;bitů operandu
```

Mezi aritmeticko-logické instrukce se zpravidla počítají i instrukce porovnání. Vždy se provede odečtení obou operandů. Výsledek však není nikam ukládán, ale projeví se jen nastavením patřičných příznakových bitů, podle kterých lze pak větvit program.

```
CMP x,y         ;porovnání, druhý operand se odečte
                ;od prvého, ovlivní příznakové bity C, Z,
                ;S, V
```

U některých procesorů je první operand implicitně v akumulátoru, pak má instrukce tvar

```
CMP x           ;porovnání, operand se odečte od obsahu
                ;akumulátoru, ovlivní příznakové bity C,
                ;Z, S, V
```


426410

Instrukce pro logické operace s jednotlivými bity:

Operace s jednotlivými bity jsou velmi užitečné zvláště v úlohách řízení. Jako jednobitové vstupní proměnné vystupují např. signály ze spínačů, výstupními proměnnými jsou signály pro ovládání akčních členů (zapnout-vypnout), apod. Též jsou intenzívně využívány při ovládání periferních obvodů počítače.

Adresování bitů je možné dvojím způsobem: některé procesory mají malou oblast paměti adresovatelnou po jednotlivých bitech, jiné umožňují zadat adresu ve dvou složkách jako *adresa_slova . pořadí_bitu*. Tak v prvém případě např. *bitaddr = 19* (dekadicky) odpovídá ve druhém případě *bitaddr = 2 . 3*:

```
                pořadí bitu:   7  6  5  4  3  2  1  0
                               ________________________
addr. byte:     0    bitaddr   7  6  5  4  3  2  1  0
                1              15 14 13 12 11 10  9  8
                2              23 22 21 20 19 18 17 16
                3              31 30 29 28 27 26 25 24
                4              ......................
```

```
    BSET bitaddr              ;nastavení bitu na 1
    BCLR bitaddr              ;vynulování bitu
    BCPL bitaddr              ;negace bitu
    BMOV bitaddr1,bitaddr2    ;přesun bitu
    BAND bitaddr1,bitaddr2    ;logický součin 2 bitů
    BOR  bitaddr1,bitaddr2    ;logický součet 2 bitů
    BXOR bitaddr1,bitaddr2    ;exkluzivní logický součet 2 bitů
    BCMP bitaddr1,bitaddr2    ;porovnání 2 bitů
```

U některých procesorů jsou možnosti výběru operandů omezeny – jedním operandem je pak vždy příznakový bit C, výsledek je opět v C.

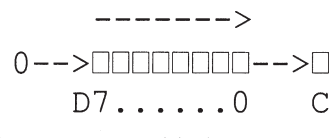
Instrukce posunů a rotací. Všechny bity operandu jsou posunuty vlevo či vpravo o počet míst, která jsou dána v instrukci. U některých procesorů však tento parametr neexistuje a posun je vždy o jedno místo.

Typické instrukce posunů:

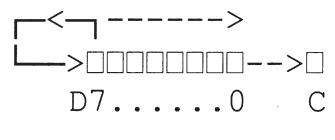
```
    SHL x,n         ;logický posun vlevo o n míst,
                    ;nejvyšší bit přesunut do příznakového
                    ;bitu C, nejnižší bit vynulován
                         <-------
                    □<--□□□□□□□□<--0
                    C   D7......0

    SHR x,n         ;logický posun vpravo o n míst
```


426410



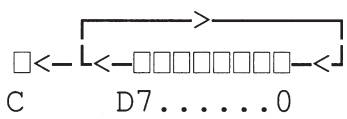
```
ASHR x,n        ;aritmetický posun vpravo o n míst,
                ;nejvyšší bit se ale nemění
```



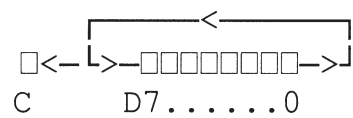
Při aritmetickém posunu je operandem číslo se znaménkem, nejvyšší bit je znaménkový. Zůstává zachován a jeho hodnota se šíří doprava (nuly u kladného čísla, jedničky u záporného). Kdyby se číslo se znaménkem podrobilo **logickému** posunu vpravo, byl by výsledek vždy kladný (nula do nejvyššího bitu), což by u záporného čísla bylo špatně.

Typické instrukce rotací:

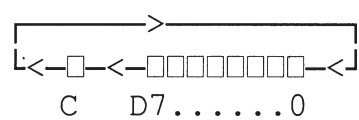
```
ROL x           ;rotace vlevo, nejvyšší bit přesunut
                ;do nejnižšího bitu a současně
                ;do příznakového bitu C
```



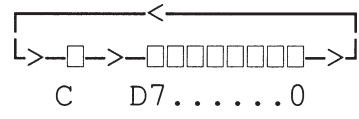
```
ROR x           ;rotace vpravo, nejnižší bit přesunut
                ;do nejvyššího bitu a současně
                ;do příznakového bitu C
```



```
RCL x           ;rotace vlevo přes příznakový bit C
```



```
RCR x           ;rotace vpravo přes příznakový bit C
```



426410

Instrukce pro práci se zásobníkem. Zásobník je adresován registrem SP. Ten je při zásobníkových operacích automaticky inkrementován či dekrementován. Navíc je však možné s ním pracovat též jako s registrem zápisníkové paměti, i když ne v plném rozsahu. Lze mezi ním, ostatními registry a pamětí přesunovat data, inkrementovat a dekrementovat jeho obsah, porovnávat obsah. Na začátku programu je nutné nastavit SP a tím definovat oblast zásobníku. Běžné zásobníkové operace jsou:

Typické zásobníkové instrukce:

```
PUSH r          ;přesun obsahu registru do zásobníku,
                ;SP se před zápisem posune
POP r           ;přesun obsahu zásobníku do registru,
                ;SP se po čtení posune
CSP x           ;komparace SP s operandem
```

Některé procesory mohou ukládat do zásobníku (a vybírat z něj) i **obsah paměti**.

Instrukce pro práci s indexregistry. Jsou možné přesuny mezi indexregistry a jinými registry (včetně SP) i pamětí, naplnění indexregistrů daty, komparace, přičtení obsahu akumulátoru, inkrementace a dekrementace.

Instrukce skoků. Efektivní adresa se přesune do PC a program pokračuje na nové adrese. Ve skokových instrukcích se využívá přímá adresa, nepřímá adresa, indexová adresa a relativní adresa. U některých jednodušších procesorů jsou tyto možnosti omezeny, nejčastěji na přímou a relativní adresu. Nepřímá (zvláště indexová) adresa však dává možnost velmi zajímavých programových struktur, jako je např. „rozeskakovací tabulka“. V adresové části instrukce je adresa začátku tabulky, v indexregistru pořadí v tabulce, a efektivní adresa tedy ukazuje do tabulky na místo, kde je teprve adresa pokračování programu. Všechny informace o různých pokračováních programu jsou tak soustředěny na jednom místě. Je-li tabulka v RAM, lze její obsah kdykoliv změnit, místo aby se musely měnit adresy skoků, rozmístěné na mnoha různých místech programu.

Skoky mohou být nepodmíněné nebo podmíněné. Podmínkou pak jsou jednotlivé příznakové bity a jejich kombinace, nebo stav jednobitové proměnné. Adresa v podmíněných skocích je zpravidla relativní. U procesorů Intel bývá instrukce pojmenována jako J ... (jump = skoč), u procesorů Motorola jako BRcc (branch = rozvětvi).

Typické skokové instrukce:

```
JMP x           ;nepodmíněný skok na adresu "x"
Jcc x           ;podmíněný skok, podmínka "cc"
                ;z příznaků
Jcc bitaddr,x   ;podmíněný skok, podmínka "cc"
                ;podle 1bitové prom.
Bcc x           ;alternativní jméno k Jcc
```

Symbolické názvy podmínek „cc“ podle příznakových bitů a jejich kombinací jsou vysvětleny v dalším textu.

Instrukce smyček umožňují opakované provádění programových smyček (cyklů) s předvoleným počtem opakování. Jako počítadlo průchodů smyčkou figuruje některý z registrů (někdy je pevně dán jen jediný), který je na začátku instrukce automaticky dekrementován (některé procesory umožňují alternativně i inkrementaci) a není-li pak jeho obsah nulový, provede se relativní skok. Je-li jeho obsah nulový, skok se již neprovede a přejde se na další instrukci. Má-li se jednat o smyčku, má praktický význam hlavně skok zpět v programu.

Typické instrukce smyčky:

```
LOOP reladdr   ;registr-počítadlo pevně dán
DJNZ r,reladdr ;registr-počítadlo volitelný
```

Instrukce volání podprogramů a návratu. Do zásobníku se automaticky uloží adresa návratu (přesněji adresa pokračování v původním programu). Efektivní adresa se přesune do PC a program pokračuje na nové adrese. Na rozdíl od skoku, kdy se neukládá adresa návratu, je z podprogramu možný návrat ihned za instrukci, která podprogram volala (viz RET). V adresové části instrukce může být přímá, nepřímá i relativní adresa. Jednodušší procesory se omezují jen na absolutní adresu.

Typické instrukce volání podprogramů:

```
CALL x          ;volání podprogramu
JSR x           ; "jump to subroutine"
                ;alternativní název – neplést se  skokem (JMP)!
BSR x           ; "branch to subroutine"
                ;alternativní název – neplést se
                ;skokem (BR)!
```

Při návratu se automaticky ze zásobníku přesune zpět do PC adresa návratu.

```
RET             ;návrat z podprogramu
RETI            ;návrat z interruptového podprogramu
```

Návrat z interruptového podprogramu se liší od návratu z obyčejného podprogramu (vyvolaného CALL). U některých procesorů se totiž při vyvolání přerušení ukládá automaticky do zásobníku nejen adresa návratu, ale i další informace (např. stav registru PSW, CCR, atd.). Při návratu se pak musí ze zásobníku vybrat i tyto informace, takže RETI je zcela neslučitelná s RET, která vybírá jen adresu návratu. Dále je nutné návrat signalizovat řadiči přerušení, který při ukončení interruptového podprogramu musí posunout hladinu priority.

Instrukce vstupu a výstupu. Jako specializované instrukce existují jen u některých procesorů. Umožňují adresování vstupních a výstupních (I/O) obvodů (případně i jiných periferních obvodů) odděleně od paměti. Vyvolají patřičný vstupní či výstupní sběrnicový cyklus, ve kterém jsou aktivovány I/O obvody.

426410

Typické vstupní a výstupní instrukce:

```
IN inaddr       ;přesun ze vstupní brány do A
OUT outaddr     ;přesun z A do výstupní brány
```

Rozsah adres inaddr, outaddr je vždy podstatně menší než v případě instrukcí přesunu z paměti. Většina procesorů ale pracuje s I/O obvody jako s pamětí (jsou „mapovány do paměti“) a pak tento typ instrukcí není vůbec realizován.

Instrukce pro řízení procesoru. Jsou velmi rozmanité, neboť úzce souvisejí s architekturou procesoru a počítače. Zde bude uvedeno jen několik instrukcí, které se běžně vyskytují.

```
NOP             ;prázdná operace
```

Procesor nevykoná žádnou činnost. Jen PC je inkrementován, takže program postoupí na další instrukci. Ačkoliv zdánlivě je tato instrukce zbytečná, je naopak využívána velmi často. Jedna nebo více NOP může sloužit jako časová výplň pro přesné doladění časování programu, jejich skupina může vyplnit úsek v programové paměti, kde se program teprve v budoucnu doplní, může nahradit některé instrukce při ladění programu, atd.

```
STI             ;povolení přerušení
CLI             ;blokování přerušení
```

Instrukce se vztahují k maskovatelným přerušením. Nastaví nebo nulují řídicí bit přerušení, který je součástí stavového slova programu (PSW), a který povoluje či blokuje činnost řadiče přerušení. U mnoha procesorů je však PSW umístěno v poli řídicích a stavových registrů, se kterými se pracuje jako s datovou, bitově adresovatelnou pamětí. Výše uvedené instrukce pak neexistují.

```
IDLE            ;též WAIT, snížený příkon, oscilátor běží
STOP            ;též PWRDN, SLEEP,... minimální příkon,
                ;oscilátor zastaven
```

Obě instrukce zastavují procesor, který je hlavním konzumentem proudu. V prvém případě však není zastaven generátor hodinových impulzů, takže zůstávají v činnosti ostatní obvody počítače (to se týká zvláště jednočipových počítačů). Stav IDLE může být ukončen přerušením (bylo-li však povoleno), jiným vnitřním signálem počítače (časovačem) nebo signálem RESET. Ve druhém případě (STOP) je zastaven i generátor hodinových impulzů, takže příkon je minimalizován. Ukončení je možné jen signálem RESET.

Instrukce pro ladění programů. Umožní nastavení režimu krokování po instrukcích, zavedení bodů zastavení, přečtení a naplnění registrů či oblasti paměti, atd. Jsou velmi závislé na architektuře procesoru.

Další speciální instrukce. Předcházející stručný výčet typických instrukcí nemohl zahrnout všechny speciální instrukce, které se vyskytují třeba jen u jednoho typu procesoru od jednoho výrobce. Jsou mnohdy velmi komplikované a mohou nahradit celé úseky programů pro statistické výpočty, zpracování signálů, fuzzy řízení, apod.

4.3 VYUŽITÍ PŘÍZNAKOVÝCH BITŮ V INSTRUKCÍCH PODMÍNĚNÝCH SKOKŮ

Příznakové bity jsou využívány hlavně v instrukcích podmíněných skoků, které umožňují větvení programů. Jako podmínka pro provedení skoku mohou sloužit jednotlivé příznakové bity i jejich kombinace. Lze tak vyjádřit mnoho různých podmínek. K nastavení příznakových bitů slouží nejčastěji instrukce porovnání (CMP) nebo též aritmetické instrukce a některé instrukce posunů a rotací. Zdaleka ne všechny procesory však mají úplnou sestavu níže uvedených instrukcí.

Nejjednodušší jsou podmínky, dané stavem jen jednoho z příznakových bitů.

Typické instrukce podmíněných skoků podle jednotlivých příznakových bitů:

Symbolický název	*Branch if ....*	*Podmínky*
B**CC**	Carry Clear	C = 0
B**CS**	Carry Set	C = 1
B**VC**	Overflow Clear	V = 0
B**VS**	Overflow Set	V = 1
B**MI**	Minus	N = 1
B**PL**	Plus	N = 0

V případě práce s čísly lze po předcházejícím porovnání (CMP) zjistit vzájemné relace **dvou čísel**. Podmínka skoku se vztahuje k prvému operandu instrukce CMP (případně implicitně k obsahu akumulátoru) v porovnání s druhým operandem. Jedná-li se o čísla **bez znaménkového bitu** („unsigned integer“), jsou využívány bity C a Z:

Typické instrukce podmíněných skoků podle porovnání čísel bez znaménka:

Symbolický název	*Branch if ....*	*Podmínky*
B**EQ**	Equal	Z = 1
B**NE**	Not Equal	Z = 0
B**HI**	Higher	C + Z = 0
B**LO**	Lower	C = 1
B**HS**	Higher or Same	C = 0
B**LS**	Lower or Same	C + Z = 1

Je třeba si uvědomit, že stav příznakových bitů po CMP je výsledkem odečtení dvou čísel. Byla-li shodná (*Branch if Equal*), je výsledek nulový a tudíž Z = 1. Bylo-li prvé číslo větší než druhé, nemůže vzniknout přenos do bitu C a ani nemohl být výsledek nulový, tudíž C + Z = 0. Podobně i v ostatních případech.

426410

V případě čísel **se znaménkovým bitem** je nejvyšší bit znaménkový a bude třeba vzít do úvahy i bit přetečení V (symbol ⊕ značí operaci EX-OR či neekvivalenci):

Typické instrukce podmíněných skoků podle porovnání čísel se znaménkem:

Symbolický název	*Branch if ....*	*Podmínky*
B**GT**	Greater Than	Z + (N ⊕ V) = 0
B**LT**	Less Than	N ⊕ V = 1
B**GE**	Greater or Equal	N ⊕ V = 0
B**LE**	Less or Equal	Z + (N ⊕ V) = 1

5 OBVODY POČÍTAČE

V této kapitole budou popsány obvody pro řízení počítače. Především se to týká řízení jednotek, připojených na sběrnice. Budou zdůrazněny rozdíly mezi počítačem, složeným z několika integrovaných obvodů (procesor a další) a počítačem na jediném integrovaném obvodu. V prvém případě se bude jednat o počítač **vícečipový** nebo též jednodeskový, ve druhém o **jednočipový** mikropočítač. U vícečipového počítače jsou všechny jednotky, nutné pro jeho činnost, vnější vzhledem k procesoru. U jednočipové verze jsou již na čipu integrovány a vnějším jednotkám se pokud možno vyhýbáme, tak aby byl celý systém realizován minimálním počtem integrovaných obvodů. Někdy je však **rozšíření** o vnější jednotky nutné, pokud rozsah nebo skladba vnitřních jednotek pro danou aplikaci nestačí. Konstruktéři jednočipových mikropočítačů vždy musí šetřit s počtem vývodů pouzdra (ten značně ovlivňuje cenu) a proto jsou nuceni využívat některé vývody pro dvě či více funkcí. Vývody mají **alternativní význam**. Příkladem mohou být vývody vstupních a výstupních obvodů, které se alternativně využívají k rozšíření jako vývody vnějších sběrnic. Ty pak již samozřejmě nemohou sloužit pro původní funkci. Nedostatek paralelních bran bývá citelným důsledkem rozšiřování o vnější obvody.

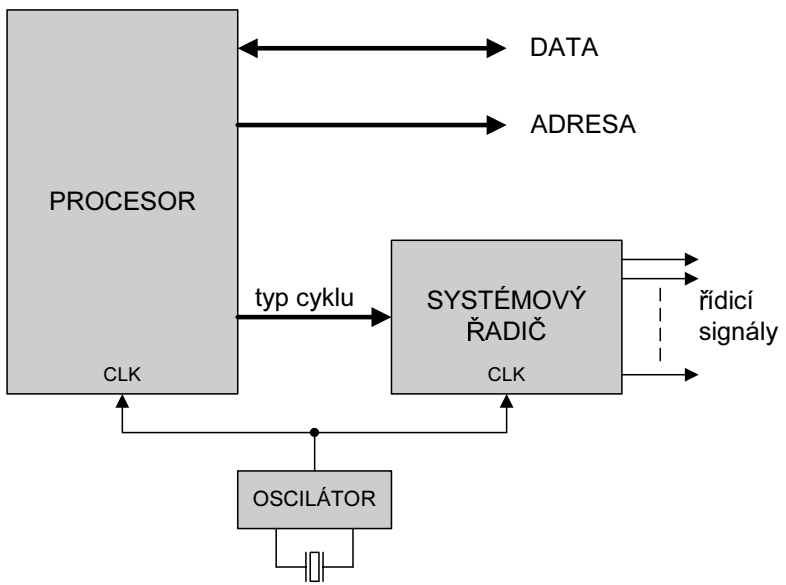
Je zřejmé, že jednočipové řešení bude mít jak společné vlastnosti s řešením vícečipovým, tak vlastnosti velmi specifické. U vícečipového řešení je snadno možné na plošném spoji různě **propojovat** a zpracovávat vstupy a výstupy jednotlivých integrovaných obvodů a tím realizovat jejich různé funkce. U jednočipového řešení jsou všechny vnitřní jednotky **nedostupné**. Jejich nejrůznější vzájemné vazby musí již předem existovat a programátor si z nich vybere. K výběru slouží soustava **řídicích registrů**, která se může rozrůst do velmi značných rozměrů (i přes stovku). Program pro takový jednočipový mikropočítač pak na začátku, ještě před vlastním aplikačním programem, obsahuje **inicializační část** pro naprogramování řídicích registrů. Inicializace má zcela klíčovou důležitost a může výrazně ovlivnit výpočetní výkon celého systému.

Ke spojení procesoru s ostatními obvody počítače slouží soustava sběrnic. Základní uspořádání a jednoduché časové diagramy byly uvedeny již v kapitole 2. Vnější sběrnice jsou vyvedeny z procesoru, je-li samostatným integrovaným obvodem, nebo z jednočipového mikropočítače, je-li nutné jeho vnitřní obvody rozšířit. Znalost signálů vnějších sběrnic a jejich časování je nutná pro návrh systému vnějších obvodů. Posloupnost činností, během kterých dojde právě jednou ke čtení nebo zápisu prostřednictvím (vnějších) sběrnic, se nazývá (vnější) **sběrnicový cyklus** (angl. bus cycle). Základními sběrnicovými cykly jsou cyklus čtení z paměti

a cyklus zápisu do paměti. Vedle toho mohou existovat i další sběrnicové cykly, jako např. čtení ze vstupních obvodů, zápis do výstupních obvodů, aj. Časování signálů může být v těchto cyklech odlišné.

5.1 SYSTÉMOVÝ ŘADIČ

Budeme se blíže věnovat hlavně signálům řídicí sběrnice. V nejjednodušším případě jsou tyto signály generovány přímo procesorem, u dokonalejších architektur jsou generovány obvodem, zvaným **systémový řadič** (angl. system controller, též SIM – system integration module). Jiná situace nastává u počítače vícečipového a jiná u jednočipového. V prvém případě může konstruktér systému využít řídicí signály procesoru buď přímo, nebo přes vnější systémový řadič, jehož funkci může navrhnout v souladu se zamýšlenou architekturou počítače. Pro spolupráci se systémovým řadičem je procesor vybaven několika výstupními signály (např. třemi), kterými je na začátku každého sběrnicového cyklu definován jeho **typ** – např. čtení z paměti, zápis do paměti, čtení ze vstupních obvodů, atd. Systémový řadič pak samostatně generuje řídicí signály podle typu cyklu. Pro jejich přesné časování jsou do systémového řadiče zavedeny hodinové impulzy procesoru – viz *obr. 5.1.*

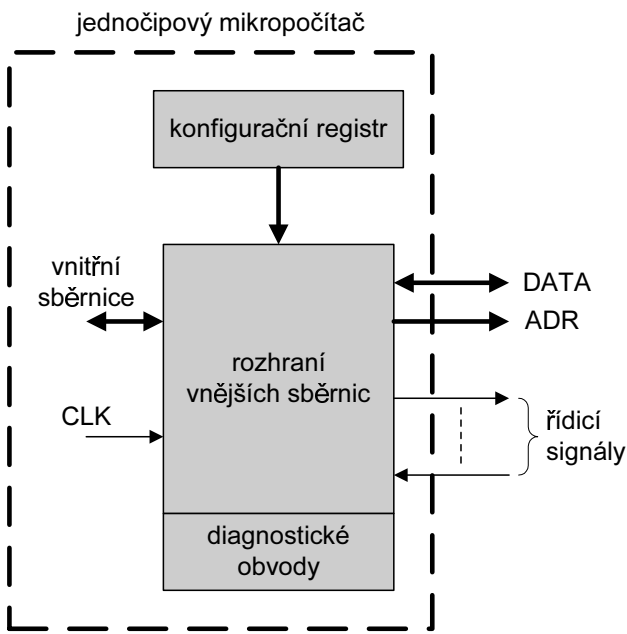


Obr. 5.1 Vnější systémový řadič

426410

U jednočipového počítače existuje systém **vnitřních sběrnic** a též vnitřní systémový řadič. Obojí je optimalizováno z hlediska rychlosti a do řízení vnitřních sběrnic nelze zasahovat. Pro případné rozšíření o vnější obvody je však počítáno s vývody adresové, datové a řídicí sběrnice. Aby příliš nenarůstal počet vývodů pouzdra, jsou pro vnější sběrnice využity vývody již jednou obsazené – typicky jsou to vývody paralelních bran.

Zvláště u 16bitových mikropočítačů umožňuje systémový řadič činnost vnějších sběrnic v několika různých režimech a s různým časováním signálů. Lze přenášet slova 16bitová nebo 8bitová, adresovou a datovou sběrnici lze sdružit do jedné multiplexní sběrnice, lze měnit časování jednotlivých signálů, atd. Režim činnosti vnějších sběrnic lze měnit prostřednictvím naprogramování systémového řadiče, který je k tomu účelu vybaven řídicím registrem, nazývaným **konfigurační registr**, též MODE registr.



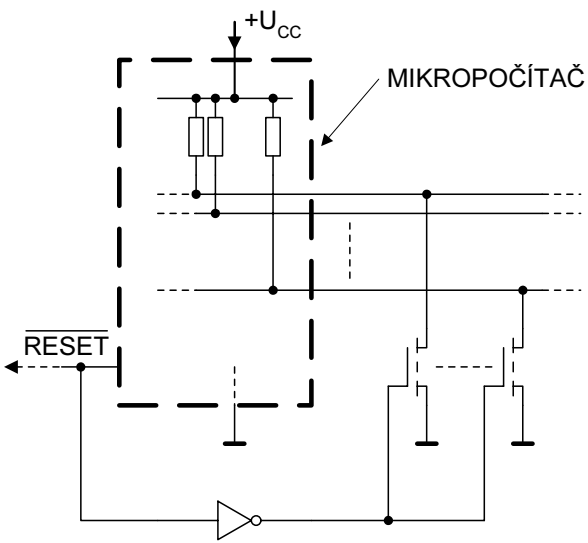
Obr. 5.2 Vnitřní systémový řadič u jednočipového mikropočítače

Konfigurační data musí být uložena do konfiguračního registru **při náběhu napájení** počítače nebo bezprostředně po něm. V případě jednočipového mikropočítače je absolutně nutná informace o rozšíření o vnější obvody, zvláště o programovou paměť – ta může být vnitřní (na čipu) nebo vnější. Stejně důležité jsou i údaje o šířce vnějších sběrnic (zvláště datové) a o jejich řízení a časování.

426410

Bez těchto informací se počítač nemůže rozběhnout a proto jsou příslušné bity konfiguračního registru nastavovány hardwarovými prostředky během nulování počítače. Druhá část bitů konfiguračního registru se týká takových signálů a činností počítače, které mohou být definovány až po začátku programu, a registr tedy může být částečně naplněn až samotným programem – typicky se to týká např. prostředků pro ladění programů. Zápis do konfiguračního registru je vždy **znesnadněn** a obsah registru je chráněn před náhodným přepsáním. Ochrana je řešena tak, že přístup do registru je možný jen po stanovený (a malý) počet hodinových impulzů procesoru po skončení signálu *Reset*, nebo po každém vynulování je zápis do něj povolen jen jednou a případné další zápisy jsou blokovány (čtení blokováno není).

Hardwarové prostředky pro určení konfigurace mohou u jednoduchých architektur být velmi omezené a jednoduché. Jeden vývod pouzdra je vyhrazen pro konfigurační vstup, na který je přivedeno napětí logické 0 nebo 1. Bývá označen jako EA (angl. External Access) nebo MP/MC (Microprocessor/Microcomputer s vnější/vnitřní pamětí). U dokonalejších architektur by byl zapotřebí větší počet takovýchto vývodů pouzdra, což je zřejmě nevhodné (s vývody pouzdra se vždy šetří). Proto jsou pro **vstup konfigurační informace** využívány vývody jiných signálů. Po dobu aktivního signálu *Reset* mikropočítač od těchto vývodů odpojí své vnitřní obvody a čte na nich stav. Pomocné vnější obvody pak musí během aktivního signálu *Reset* na tyto vývody vnutit stav 0 nebo 1. Po skončení *Reset* naopak musí být tyto pomocné vnější obvody odpojeny, tak aby nebyla rušena řádná činnost vývodů při běhu programu. Jednoduché uspořádání ukazuje *obr. 5.3*.

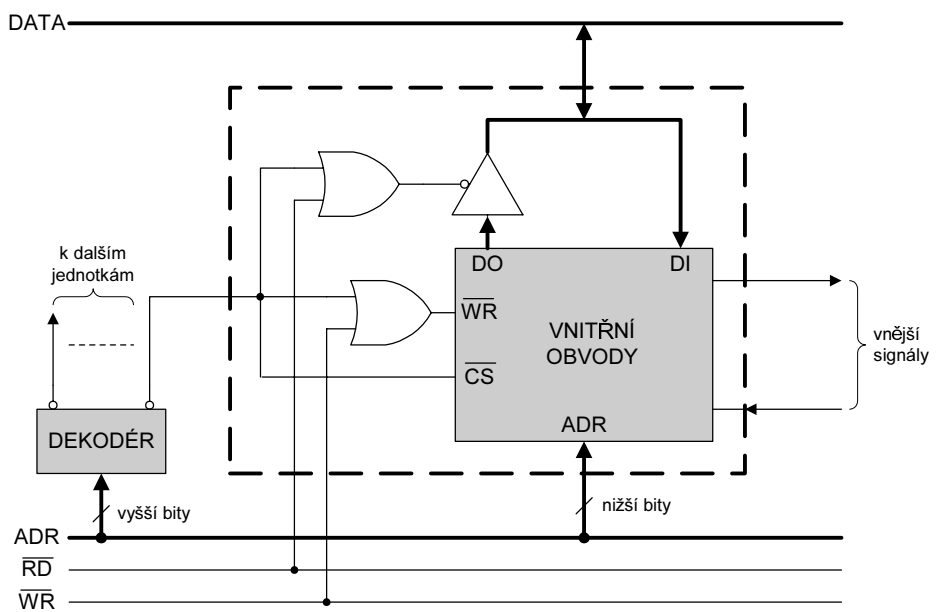


Obr. 5.3 Vstup konfigurační informace během nulování

Stav 0 je zajištěn uzemněním patřičného vývodu tranzistorem, stav 1 odporem od napájecího napětí – ten je zpravidla již realizován na čipu. Přítomnost či nepřítomnost uzemňujícího tranzistoru tedy definuje stav 0 či 1 patřičného bitu konfigurační informace. V praxi lze místo jednotlivých tranzistorů využít obvody s otevřenými kolektory nebo obvody GAL s třístavovými výstupy.

5.2 VNĚJŠÍ SBĚRNICE A ŘÍDICÍ SIGNÁLY

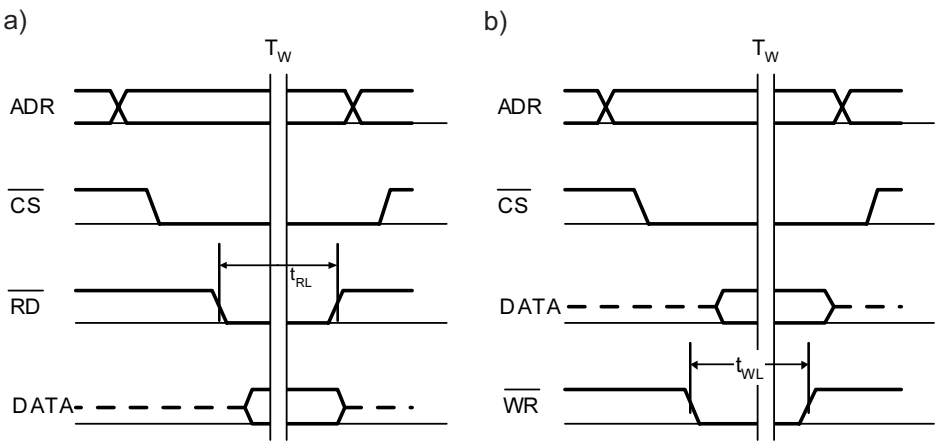
Na datovou sběrnici jsou vnější jednotky vždy připojeny **přes třístavové členy**, které musí být řízeny tak, aby v žádném okamžiku nemohlo dojít k současnému připojení dvou nebo více jednotek, což se nazývá **konflikt na sběrnici**. Přitom vždy dojde k chybě ve čtených datech, ale může dojít i ke zničení třístavových členů. Bezkonfliktní výběr vždy jen jedné jednotky zajišťují výběrové signály $\overline{CS}$, generované **adresovým dekodérem**. Jako u většiny kombinačních obvodů, i u adresového dekodéru existují **hazardní stavy**, které se projevují jako falešné, krátké impulzy na výstupech. Proto signály z dekodérů nesmí nikdy samy spouštět činnost jednotek, nýbrž jen vybranou jednotku odblokují (aktivují), zatím co vlastní výkonný povel je časován **jinými** řídicími signály. Obecné uspořádání ukazuje *obr. 5.4.*



Obr. 5.4 Řízení jednotky připojené na sběrnice

Vlastní vnitřní obvody jednotky (v obdélníku uvnitř) jsou obecně zdrojem i příjemcem dat z datové sběrnice. Jednotka je vybrána jedním výstupem adresového dekodéru. Při čtení dat je tak blokován či povolen čtecí impulz, který vyvolá rychlé připojení třístavového členu. Při zápisu pak je obdobně blokován či povolen průchod zápisového impulzu do vnitřních obvodů. Výběrový signál $\overline{CS}$ může, ale nemusí být zaveden do vnitřních obvodů. Je-li do nich zaveden, může jednotky, které nejsou vybrány, uvádět do **úsporného režimu** se sníženým příkonem (typické u pamětí). Návrat do normálního režimu u vybrané jednotky vyžaduje jistou dobu – ta je zajištěna **předstihem** $\overline{\mathbf{CS}}$ před čtecím či zápisovým impulzem. Jednotka může zpracovávat vnější signály. Tato činnost není ovlivněna výběrovým signálem $\overline{CS}$.

Nejjednodušší typ sběrnicového cyklu využívá pouze dva řídicí signály, $\overline{RD}$ a $\overline{WR}$. Průběh čtení ukazuje *obr. 5.5a*. Počátkem je okamžik, kdy procesor vydá adresu, ze které bude číst nová data. Čtecí impulz $\overline{RD}$ je vydán, až když jsou vnitřní obvody jednotky uvedeny do pohotového stavu signálem $\overline{CS}$ = 0, který je generován dekodérem adres. Výstupy dekodéru se mění se zpožděním po změně adresy. K výstupu dat na datovou sběrnici dochází jen s malým zpožděním po změně $\overline{RD}$. Platná data na sběrnici pak přebírá procesor. Mimo aktivní stav čtecího povelu ($\overline{RD}$ = 0) je sběrnice ve vysokoimpedančním stavu. Adresa zůstává konstantní až do konce čtecího povelu. Může se stát, že jednotka, ze které se čte, má příliš velké zpoždění ve výstupu dat, a že data ještě nejsou ustálena v době, kdy je vzorkuje procesor. Dojde tak k chybě ve čtení dat. Situaci může vyřešit vložení tzv. čekacích taktů T_W (WAIT), na místě vyznačeném v obrázku dvojicí čar. Prodlouží se doba platné adresy a čtecí impulz. Jednotka má k dispozici delší čas na vydání platných dat. Časování signálů je odvozeno od hodinových impulzů procesoru (na obrázku nejsou znázorněny). Jsou jimi časovány i čekací takty.

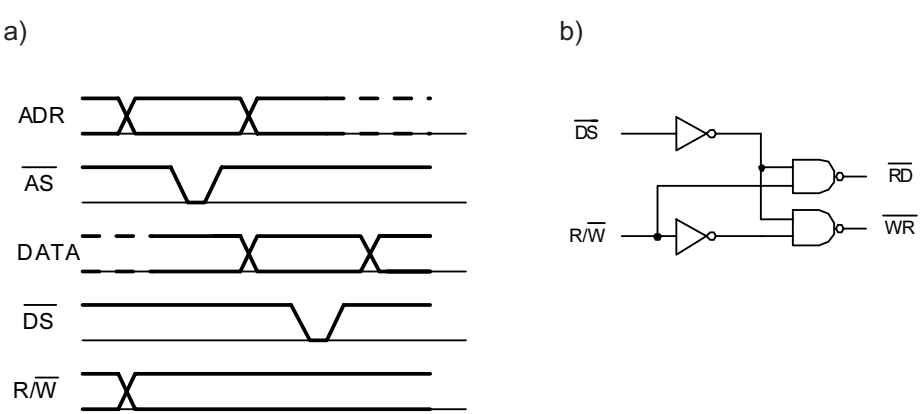


Obr. 5.5 Základní sběrnicové cykly: a) cyklus čtení; b) cyklus zápisu

Průběh zápisu ukazuje *obr. 5.5b*. Procesor vydá novou adresu a s jistým zpožděním pak i data. Do jednotky, která byla vybrána signálem $\overline{CS}$, jsou data zapsána impulzem $\overline{WR}$. I zde se může stát, že jednotka je příliš pomalá a doby, znázorněné v obrázku, jsou příliš krátké. Dojde tak k chybě v zápisu dat. Situaci opět může vyřešit vložení čekacích taktů T_W na místě, vyznačeném v obrázku dvojicí čar.

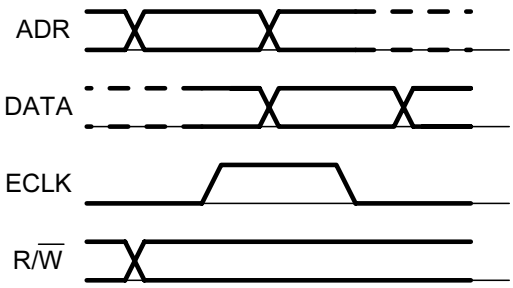
U různých počítačů existují **různé soustavy řídicích signálů**, mnohdy tradičně dodržované jednotlivými výrobci a úzce související s architekturou počítače. V předchozím textu se jednalo o velmi jednoduchý systém dvou řídicích signálů $\overline{RD}$ a $\overline{WR}$, vhodný pro architekturu s jediným adresovým prostorem. Zmíníme se ještě o několika dalších soustavách. Výčet zdaleka nebude vyčerpávající.

- Soustava $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, $\overline{IOW}$, $\overline{INTA}$ (Memory Read, Memory Write, Input-Output Read, Input-Output Write, Interrupt Acknowledge). Signály jsou časovány přibližně podle *obr. 5.5*, pro periferní obvody jsou zvláštní signály, $\overline{INTA}$ slouží pro spolupráci procesoru s řadičem přerušení – viz další kapitoly.
- Soustava $\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$ (Read, Write, Program Store Enable). Prvé dva signály řídí čtení a zápis dat, $\overline{PSEN}$ slouží výhradně pro čtení z vnější paměti programu, je aktivní jen při čtení instrukce. Vnější adresový prostor je tak rozdělen na dva prostory se stejnými adresami – programový a datový.
- Soustava $\overline{RD}$, $\overline{WR}$, MEM-IO. Význam prvých dvou signálů je jako v *obr. 5.5*, třetí rozlišuje mezi pamětí a periferními obvody. Mění se na začátku cyklu, přibližně v době, kdy se mění adresa.
- Soustava $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$. Impulz $\overline{AS}$ (Address Strobe) potvrzuje platnost adresy, impulz $\overline{DS}$ (data strobe) platnost dat. Signál R/$\overline{W}$ rozlišuje cyklus čtení (stav 1) od zápisu (stav 0) a mění se na začátku cyklu. Časování ukazuje *obr. 5.6a* pro případ zápisového cyklu. Signály této soustavy lze snadno konvertovat na signály soustavy $\overline{RD}$, $\overline{WR}$ – viz *obr. 5.6b*.



Obr. 5.6 *Sběrnicový cyklus s potvrzeními platnosti adresy a dat*

- Soustava ECLK, R/$\overline{W}$. Sběrnicové hodiny ECLK nahrazují impulzy $\overline{AS}$ (náběžná hrana ECLK) a $\overline{DS}$ (doběžná hrana ECLK). Význam a časování R/$\overline{W}$ je obdobné jako v předchozím případě – viz *obr. 5.7*.

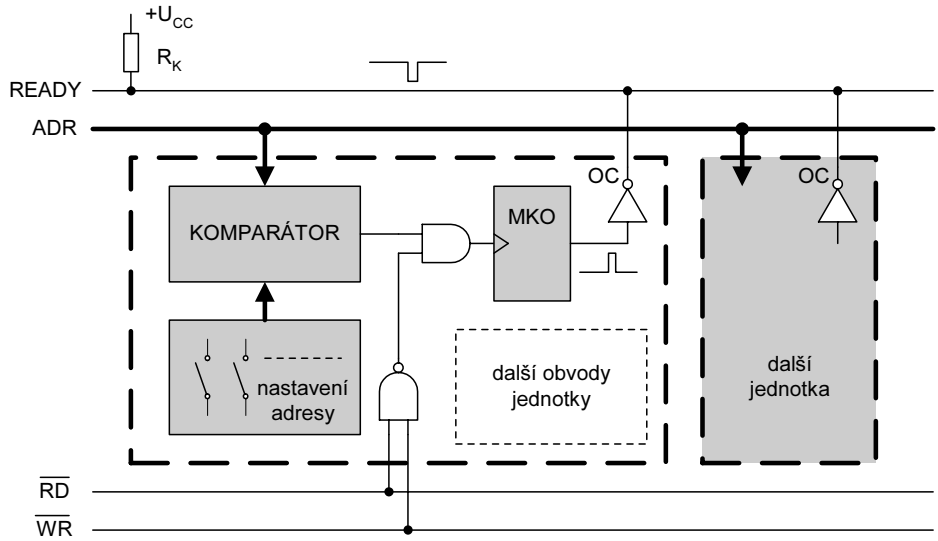


Obr. 5.7 Sběrnicový cyklus se sběrnicovými hodinami

Vkládání čekacích taktů je jednoduchou metodou, jak zařídit spolupráci pomalých obvodů s rychlým procesorem. Musí se samozřejmě jednat o ojedinělé cykly, aby prodlužování cyklů bylo v průměru málo významné. Typicky se jedná o periferní obvody, s nimiž procesor pracuje relativně zřídka ve srovnání s pamětí programu a dat. Paměti nemají sběrnicový cyklus zpomalovat. Pokud ano, je nejjednodušším řešením celkové zpomalení hodinových impulzů procesoru. Vkládání čekacích taktů je vyvoláno signálem na vstupu procesoru, často označeným jako READY. Signál READY je v procesoru vzorkován prostřednictvím vnitřních hodinových impulzů a proto je možno vkládat jen **celý počet čekacích taktů** o celkové délce rovné násobku periody vzorkování.

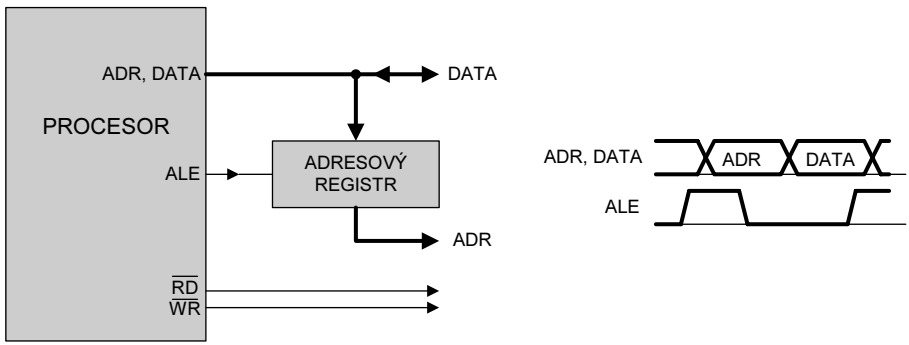
Jsou dva způsoby generace tohoto signálu. V prvém případě je vytvářen v centrálním obvodu, tzv. **generátor čekacích taktů** (angl. Wait-State Generator), na základě dekódování adresy. Podle příslušnosti do jedné z několika oblastí adres je vybrán příslušný registr, ve kterém je předem na začátku programu zapsán údaj o délce impulzu READY, tj. počty čekacích taktů. Registrů je několik a tak lze odstupňovat časování sběrnicových cyklů pro různé periferní obvody. Druhý způsob generace READY spočívá v decentralizaci časovacích obvodů a jejich integraci do jednotek, připojených na sběrnice. Princip ukazuje *obr. 5.8*.

Signál READY vyvolá vkládání T_W při stavu logické nuly, což dovoluje jeho generaci **paralelně zapojenými** obvody s otevřenými kolektory, z nichž kterýkoliv může na společný vodič vnutit stav 0. Stav 1 je zajištěn společným kolektorovým odporem R_K (někdy je již vytvořen na čipu procesoru). Součástí jednotky je adresový komparátor, který zjišťuje, zda adresa z adresové sběrnice spadá do oblasti adres, která patří jednotce. Pokud adresa jednotce patří, je začátkem čtecího ($\overline{RD}$) nebo zápisového ($\overline{WR}$) impulzu spuštěn **časovací obvod** (monostabilní klopný obvod MKO) a vytvořen impulz READY. Zadání oblasti adres je pevně dáno, např. miniaturními mechanickými spínači, nebo může být celá funkce realizována programovatelným obvodem GAL, případně i jinými obvodovými prostředky.



Obr. 5.8 Decentralizované řízení sběrnicového cyklu

Multiplexní sběrnice je velmi často využívána pro úsporu počtu vývodů pouzdra. U jednočipových počítačů, kde pro vyvedení vnějších sběrnic jsou využity původní vývody bran, je úspora ještě významnější. V prvé části sběrnicového cyklu je po adresové-datové sběrnici vydávána adresa, ve druhé části jsou přenášena data. Adresa musí být zachycena ve **vnějším adresovém registru**, aby byla vnějším obvodům k dispozici po celý cyklus. Zápisový impulz pro registr je jedním z řídicích signálů vnější sběrnice. Ve výše popsaných soustavách řídicích signálů to např. mohl být impulz $\overline{AS}$, často je však označován jako ALE (Address Latch Enable). Zápis do registru zpomaluje sběrnicový cyklus – to je daň za úsporu vývodů. Celé uspořádání a časování ukazuje *obr. 5.9*.



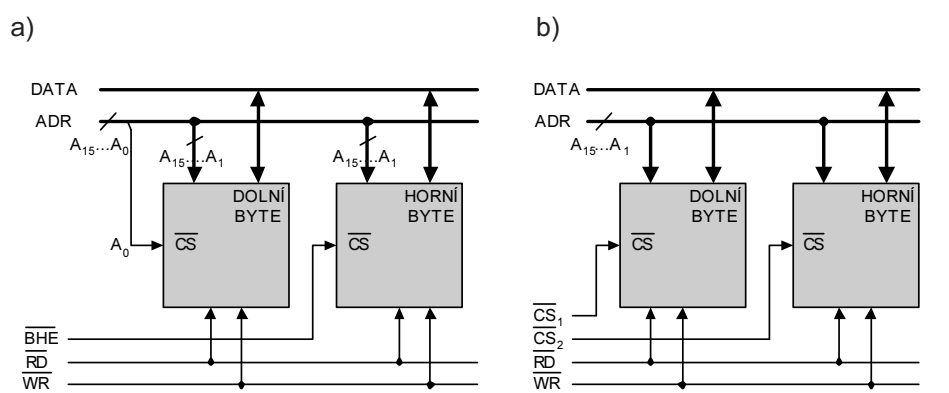
Obr. 5.9 Multiplexní sběrnice a časování

Často je datová sběrnice užší než adresová (8 bitů proti 16 bitům). Pak jen 8 bitů je multiplexováno jako adresa/data, zatím co zbylých 8 bitů má trvale význam adresy. V tom případě jsou vždy multiplexovány nižší bity adresy – předpokládá se, že nejvyšší bity jsou zavedeny do adresového dekodéru, který vytváří výběrové signály. Zpoždění dekodéru takto zhruba kompenzuje zpoždění adresového registru.

V případě široké datové sběrnice (16 a více bitů) mohou vnější jednotky pracovat jak s daty o plné šířce, tak také s daty 8bitovými (to je dokonce mnohem častější). Osmibitové obvody – zvláště paměti – mohou být připojeny tak, že jeden je připojen na dolních 8 bitů (tj. dolní slabika), druhý na horních 8 bitů dat (tj. horní slabika). Lze tak přenášet **horní slabiku, dolní slabiku**, nebo obě současně, tj. **celé slovo**. Některé instrukce takovouto volbu umožňují. K připojení „horní-dolní-oba“ slouží výběrové vstupy $\overline{CS}$. Jeden způsob používá adresový bit A_0 (tj. nejnižší) a přídavný signál $\overline{BHE}$ (byte high enable) takto:

A_0	$\overline{BHE}$	funkce
0	0	slovo
0	1	dolní slabika
1	0	horní slabika
1	1	nepoužito

Uspořádání ukazuje *obr. 5.10a*. Všimněme si bitu A_0, který je zaveden jen do jednoho obvodu. Paměť je adresována po slabikách, při postupném zvyšování adresy by se pravidelně střídala dolní-horní slabika podle druhé a třetí řádky tabulky. Druhá možnost je na *obr. 5.10b* – adresový bit A_0 není vůbec použit a k rozlišení slabik slouží výběrové signály $\overline{CS_1}$ a $\overline{CS_2}$. Při přenosu slova je vždy vhodné, aby byly obě slabiky přenášeny současně. To ale znamená, že slovo musí začínat na sudé adrese ($A_0 = 0$). Pokud je adresa slova posunuta o 1, musí



Obr. 5.10 Adresování slabik a slov: a) signály A_0 a $\overline{BHE}$; b) výběrovými signály

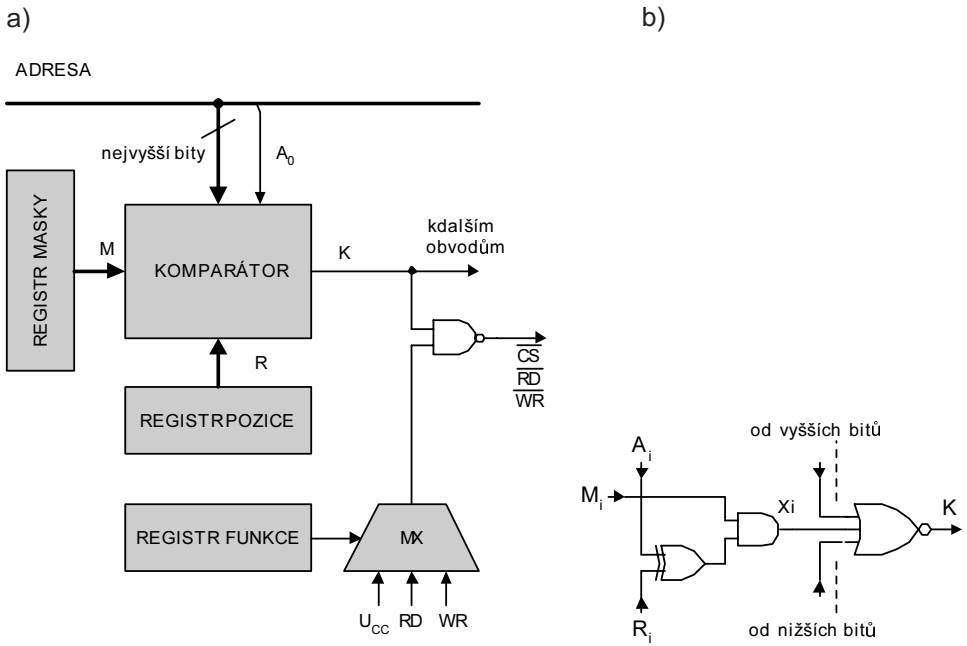
být slovo přenášeno postupně (ve dvou cyklech). Jedná se o **nesprávné zarovnání slov** (angl. misalignment), které zpomaluje činnost počítače. Překladače tuto skutečnost respektují a umožňují vhodně posunout adresy slov.

Z úsporných důvodů lze dokonce systematicky pracovat jen s 8bitovými daty i u 16bitové sběrnice – vyšší bity datové sběrnice nejsou vůbec využity. To sice zpomalí celý systém, neboť pak bude zapotřebí dvojnásobný počet sběrnicových cyklů, ale u malých systémů se zjednoduší plošný spoj a zlevní se obvody. Systémový řadič dostává informaci o šířce sběrnice jako součást **konfiguračních dat** během nulování počítače.

5.3 ADRESOVÉ DEKODÉRY A VÝBĚROVÁ LOGIKA

Každé jednotce, připojené na datovou sběrnici, je přidělená jistá oblast v adresovém prostoru. Nejjednodušším řešením je adresový dekodér, který na svých výstupech generuje výběrové signály pro jednotky. Podstatným nedostatkem ale je to, že dekodér dělí adresový prostor na oblasti o shodné velikosti. To se zpravidla nehodí, neboť v počítači s von Neumannovou architekturou zaujímá největší prostor paměť programu, pak paměť dat a velmi malý prostor periferní obvody. Místo dekodéru lze navrhnout jiné kombinační obvody, což je dobré řešení uvnitř jednočipového mikropočítače. Při návrhu vnějších obvodů je lépe využít jednoduché programovatelné logické obvody (GAL) – vše je v jednom pouzdru a jsou možné pozdější změny. Bez ohledu na způsob realizace budeme obvody, které vytvářejí výběrové signály, nazývat **výběrovými obvody**.

Výběrové obvody mohou být s výhodou založeny na **komparátorech s maskováním**. Princip ukazuje *obr. 5.11a*, pohled dovnitř komparátoru ukazuje *obr. 5.11b*. Komparátor srovnává stav na jednotlivých bitech adresy se stejnými bity v **registru pozice**, který udává pozici oblasti v adresovém prostoru. **Registr masky** určuje, které bity adresy jsou v komparátoru ignorovány (tj. nezáleží na nich). Podle schématu na *obr. 5.11b* je zřejmé, že bit adresy je maskován při stavu 0 v registru masky. V bodu X_i je stav 0 při shodě A_i a R_i (člen EX-OR realizuje funkci neekvivalence), při shodě ve všech bitech je na všech X_i stav 0 a na výstupu komparátoru je pak stav 1. Stav 0 na M_i vnutí stav 0 do bodu X_i bez ohledu na situaci na vstupech EX-OR. Do komparátoru jsou zavedeny jen vyšší bity adresy a bit A_0. Počtem nejvyšších nemaskovaných bitů je dán rozsah nejmenší rozlišitelné oblasti adres, maskou může být tato oblast zvětšována. Nejvyšší bity, pokud nejsou maskovány, určují počátek oblasti, při které komparátor dává na výstupu stav 1. Po negaci je získán výběrový signál. Manipulací s jednotlivými bity v registru pozice a registru masky lze **posunovat oblast** adres, ve kterých je výběrový signál aktivní, a též **měnit velikost** této oblasti.



Obr. 5.11 Adresový komparátor: a) použití pro generaci výběrových signálů; b) vnitřní zapojení

Jako příklad může sloužit komparátor, který zpracovává nejvyšších 6 bitů šestnáctibitové adresy (tedy A_{15} ... A_{10}), registr pozice a registr masky má též šest bitů. Obsahy registrů jsou např. takovéto:

R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}
0	1	–	–	–	–

M_{15}	M_{14}	M_{13}	M_{12}	M_{11}	M_{10}
1	1	0	0	0	0

Bity A_{13} až A_{10} jsou maskovány, proto na R_{13} až R_{10} nezáleží (vyznačeno jako –). Oblast adres začíná od **01**00 0000 0000 0000, tj. 4000 H. Jelikož bity R_{13} až R_{10} jsou maskovány a bity R_9 až R_0 nejsou do komparátoru vedeny vůbec, je rozsah oblasti 14 bitů, tj. 16 K adres (1 K = 1024).

Možnost programování pozice a rozsahu oblasti adres je velmi výhodná zvláště u jednočipových mikropočítačů. U některých je na čipu (v rámci systémového řadiče) integrováno několik adresových komparátorů a přilehlých registrů, takže z pouzdra je vyvedeno několik výběrových signálů. Univerzalita může být ještě zvýšena vhodným **časováním výstupů**, jak ukazuje *obr. 5.11*. Výstup komparátoru je logicky vynásoben se čtecím impulzem, zápisovým impulzem, nebo s konstantou 1. Výstupem je pak čtecí impulz platný jen v jisté oblasti adres, zápisový impulz platný jen v jisté oblasti adres, nebo výběrový impulz ($\overline{CS}$) – viz časování těchto impulzů

v *obr. 5.5.* Takto generované čtecí a zápisové impulzy mohou zrychlit činnost celého počítače a ušetřit vnější obvody. Způsob časování je volen a časovací signály přepínány pomocí multiplexeru na základě obsahu registru funkce.

Dalším úkolem komparátoru může být rozlišení sudé a liché adresy – využití výběrových signálů s touto vlastností bylo popsáno v předchozím textu. Kromě nejvyšších bitů adresy je do komparátoru zaveden ještě bit nejnižší (A_0), registr pozice a registr masky má též o jeden bit víc (R_0 a M_0). Není-li nejnižší bit zamaskován, bude záležet na stavu R_0. Je-li $R_0 = 0$, bude výběrový signál aktivní při sudé adrese ($A_0 = 0$), je-li $R_0 = 1$, bude aktivní při liché adrese.

Po náběhu napájení je však obsah registrů neznámý, což by kromě jiného znemožnilo i čtení programu. Proto jeden z výběrových signálů je pevně určen **pro paměť programu** (značen např. $\overline{\text{CSBOOT}}$). Registry, příslušné tomuto výstupu, jsou při nulování počítače automaticky přednastaveny tak, aby se program mohl rozběhnout. Registr pozice je přednastaven tak, aby oblast adres pro paměť programu zahrnovala adresu, na kterou je při nulování přednastaven čítač instrukcí, velikost oblasti je nastavena na maximum, atd. Na začátku programu pak jsou vhodně naprogramovány i registry pro ostatní výstupy.

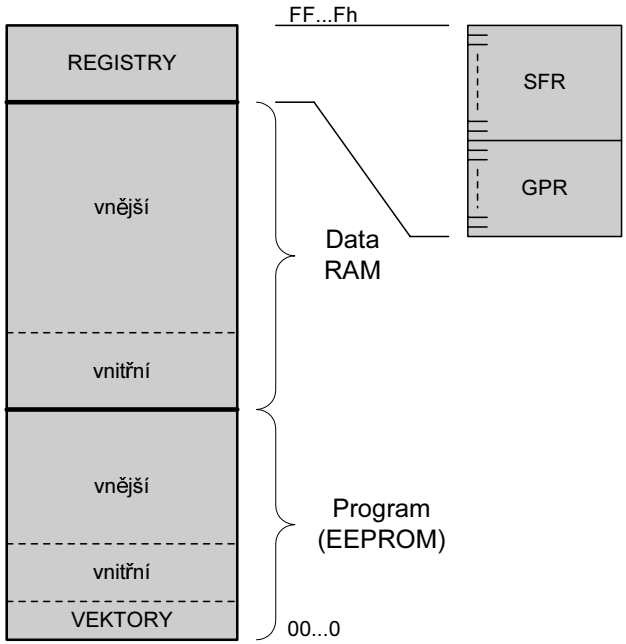
Celý blok komparátorů a doplňkových obvodů této koncepce (angl. chip-select logic) je součástí některých jednočipových mikropočítačů s dobře propracovanou architekturou – jejich činnost a programování se může částečně lišit od toho, co zde bylo popsáno, princip však je obdobný. Využití adresových komparátorů je však daleko širší. Výběrové obvody mohou být decentralizovány, jednotlivé komparátory mohou být součástí vnějších jednotek, připojených na sběrnice. Takovéto jednotky se budou „samy“ aktivovat jen v příslušných oblastech adres. Místo registrů pozice a masky lze použít miniaturní spínače.

Výstupy výběrových obvodů mohou mít ještě další využití. Mohou být zavedeny **do generátoru čekacích taktů**, který podle oblasti adres a vybavovací doby jednotky, umístěné do této oblasti, vloží příslušný počet čekacích taktů do sběrnicového cyklu. Další využití je **v diagnostických obvodech**, které kromě jiného kontrolují správnost adresy.

5.4 PAMĚŤOVÁ MAPA

Grafické znázornění pozice adresovatelných obvodů v adresovém prostoru se nazývá „paměťová mapa“ (angl. memory map). Název není zcela přesný, neboť zdaleka ne všechny adresovatelné obvody jsou pamětí. Paměťová mapa je jednou z hlavních informací o počítači a zvláště o počítači jednočipovém, který má velmi bohatý soubor periferních obvodů a řídicích a stavových registrů. Příklad paměťové mapy u jednočipového mikropočítače s jedním adresovým prostorem je na *obr. 5.12*. Obrázek se nevztahuje k žádnému konkrétnímu typu.

Oblast vektorů na dně **oblasti programu** obsahuje v tomto příkladě adresy, které procesor za zvláštních podmínek používá jako adresy skoků nebo podprogramů. Je to případ „reset-vektorů“ při nulování počítače (viz další text), nebo



Obr. 5.12 Paměťová mapa

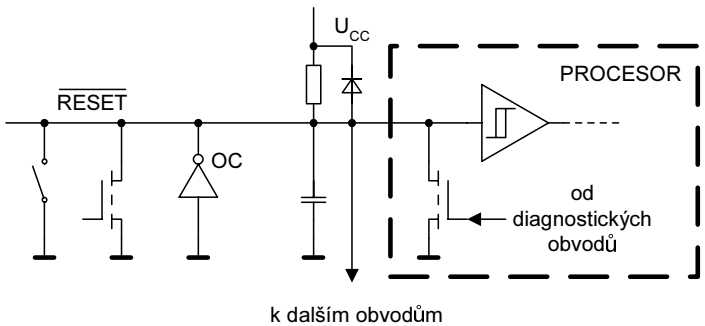
vektorů přerušení (viz příslušné kapitoly, popisující přerušení). Částí oblasti dat je vnitřní RAM. **Oblast registrů** (viz detailní pohled) obsahuje jednak řídicí a stavové registry periferních obvodů i procesoru a systémového řadiče (pole SFR – angl. Special Function Registers), jednak univerzální datové registry procesoru (pole GPR – angl. General Purpose Registers). U vyspělých architektur lze jak oblast vnitřní RAM, tak oblast GPR **přemisťovat**. Přemístění GPR má smysl při podprogramech (zvláště při přerušení) namísto úschovy obsahu pracovních registrů. Takto může vzniknout téměř neomezený počet registrových bank. Přemístění vnitřní datové paměti i GPR má dále využití v operačních systémech reálného času, kdy počítač pracuje na několika úlohách. Každá pak může mít svoji datovou oblast a banku registrů. U počítače s několika adresovými prostory existuje samozřejmě též několik paměťových map.

Po náběhu napájení jsou během nulování počítače přemístitelné oblasti nastaveny **do definovaných poloh** a po rozběhu programu může být jejich pozice změněna dle potřeby.

Je zřejmé, že v adresovém prostoru mohou existovat prázdné oblasti, které nepatří žádným obvodům. Mohou být rezervovány pro budoucí zvětšení kapacity vnitřní paměti, další periferní obvody, atd.

5.5 NULOVÁNÍ POČÍTAČE

Po náběhu napájení nebo po některých abnormálních stavech počítače je nutné nastavit do definovaného stavu jeho řídicí registry a sekvenční logiku. K nulování procesoru slouží jeho vstup $\overline{\text{RESET}}$ (negovaný tvar je častější než nenegovaný). I některé další obvody počítače jsou takovýmto vstupem vybaveny. **Zdroje nulovacího signálu** jsou v podstatě tři: zásah operátora (tlačítko), kontrolní obvody napájení, vnitřní obvody počítače. Negovaný tvar signálu umožňuje jednoduché zapojení obvodů, které jej generují – uzemňující tlačítko, tranzistor, obvod s otevřeným kolektorem. Automatickou generaci nulovacího impulzu lze snadno realizovat kondenzátorem – viz *obr. 5.13*.

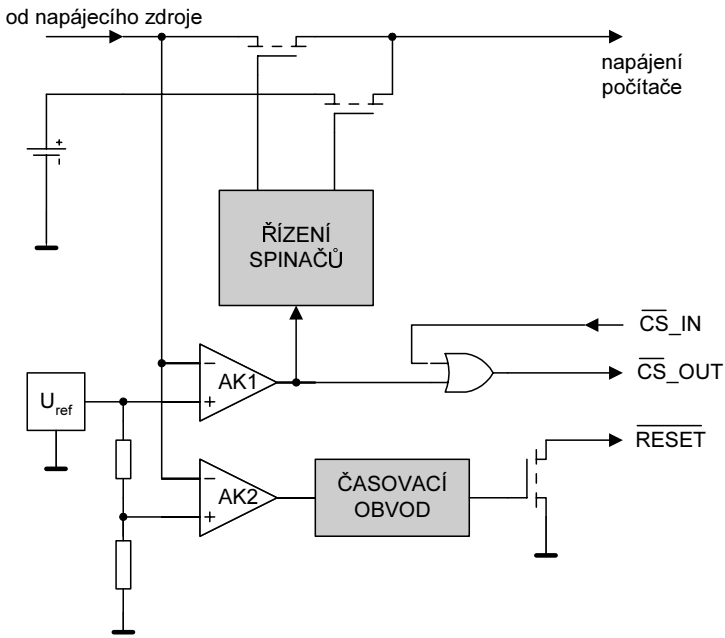


Obr. 5.13 Zdroje signálu pro nulování

Při náběhu napájení se kondenzátor nabíjí přes rezistor pomalu a po jistou dobu, dostatečnou pro vynulování, se na něm udržuje napětí menší než rozhodovací úroveň vstupu. Účelem diody je umožnit rychlé vybití kondenzátoru i při krátkodobém výpadku napájení. Vstupní obvod signálu $\overline{\text{RESET}}$ má vždy **hysterezní** charakter, aby se zabránilo případnému rozkmitání obvodu při pomalých změnách na vstupu. Nulovací impulz může být generován i **uvnitř** procesoru. Typickým zdrojem jsou diagnostické obvody. Vnitřní nulovací signál je spojen s vývodem $\overline{\text{RESET}}$, aby byl použitelný i pro ostatní obvody počítače stejně jako nulovací signál generovaný vnějšími obvody.

Kontrolní obvody napájení jsou zdokonalenou verzí jednoduchého RC členu z *obr. 5.13*. Kontrolují překročení tolerance napájecího napětí v úzkých mezích (5 až 10 %), generují signál $\overline{\text{RESET}}$, přepínají napájení na záložní baterii, a často mají ještě další funkce. Ukázka takového obvodu je na *obr. 5.14*. Obvod typicky obsahuje dva analogové komparátory. Při snížení napětí zdroje na dolní toleranční mez se překlopí analogový komparátor AK1 a přeruší přívod výběrového signálu $\overline{\text{CS}}$_OUT do paměti dat (uvede jej do stavu trvalé 1). Současně se napájení přepne na záložní baterii. Tím jsou ochráněna data v paměti ještě dříve, než napájecí napětí poklesne natolik, že procesor začne fungovat nespolehlivě a než může případně zničit data. Při dalším poklesu je komparátorem AK2 vynulován proce-

426410



Obr. 5.14 Kontrolní obvod napájení

sor. Nulovací signál je časovacím obvodem prodloužen ještě po náběhu napájení, takže existuje v patřičné délce i při krátkodobém výpadku napájení.

Délka nulovacího impulzu není libovolná, je specifikována výrobcem. Rozlišuje se případ, kdy nabíhá napájecí napětí od případu, kdy nulování bylo vyvoláno vnitřními obvody procesoru při nepřerušeném napětí. V prvém případě musí být nulovací impulz podstatně delší, neboť oscilátor v generátoru hodinových impulzů potřebuje relativně dlouhou dobu k **ustálení kmitočtu** – řádově desítky ms. Ve druhém případě postačí krátký impulz o délce jen několika period hodinových impulzů procesoru.

Nulování má vliv na řadu vnitřních obvodů procesoru i dalších částí počítače. Je zajištěn takový stav řídicích registrů a dalších sekvenčních obvodů, který umožní rozběh programu. Pak teprve jsou řídicí registry programem definitivně nastaveny.

U všech počítačů:

- **Čítač instrukcí** je nastaven na počáteční adresu, která se u různých procesorů liší. U některých procesorů je to **nula**, u jiných adresa blízko **horního kraje** adresového prostoru (tak, aby tam bylo možné umístit skokovou instrukci), a u některých procesorů je na stanoveném místě programové paměti uložena **počáteční adresa programu** (tzv. reset-vector) – ta je během nulování přesunuta do čítače instrukcí.
- Jsou **blokovány** obvody pro **přerušení**.
- Periferní obvody též vyžadují uvedení do definovaného počátečního stavu. To se zvláště týká **dvojsměrných vstupních a výstupních obvodů**, u kterých je jako počáteční stav vždy nastavován směr **„vstup“**, aby nemohlo dojít ke konfliktu s vnějšími **zdroji** signálu.

Jen u některých počítačů:

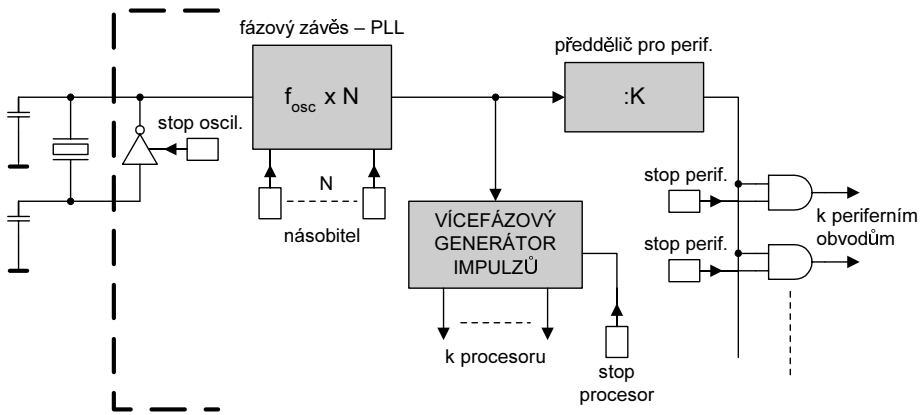
- **Ukazatel zásobníkové paměti** (registr SP) je nastaven na počáteční adresu zásobníku. U mnohých počítačů však není vůbec jeho počáteční obsah definován a pak musí být nastaven až programem. Do té doby nelze volat podprogram nebo povolit přerušení.
- **Generátor čekacích taktů** je nastaven na maximum pro případ pomalé programové paměti. Po rozběhu programu lze počet čekacích taktů podle situace snížit.
- **Výběrové obvody** pro $\overline{\text{CSBOOT}}$ jsou nastaveny tak, že oblast programové paměti souhlasí s počáteční adresou v čítači instrukcí.
- **Diagnostické obvody** jsou uvedeny do počátečního stavu (tj. „počítač bez poruchy“).
- Jednočipové mikropočítače obsahují velký počet **řídicích registrů** pro periferní obvody. Jejich stav po nulování je vždy popsán ve firemní literatuře.

Datové registry procesoru ani **datová paměť** není nulováním nijak ovlivněna (nula jako číslo totiž nemá žádnou prioritu) a jejich počáteční stav musí být zajištěn až samotným programem.

426410

5.6 GENERACE A VNITŘNÍ ROZVOD HODINOVÝCH IMPULZŮ

Generátor hodinových impulzů je vždy řízen krystalovým rezonátorem. Obvody oscilátoru jsou součástí procesoru nebo jednočipového mikropočítače, rezonátor je vnější. Používá se Pierceovo zapojení oscilátoru, vyžadující dva vnější kondenzátory – viz *obr. 5.15*. Jejich kapacita je doporučena ve firemní literatuře, je třeba však použít též doporučený rezonátor. Při jiných rezonátorech je někdy nutné pozměnit i hodnoty kondenzátorů, tak aby po zapnutí napájecího zdroje oscilátor nabíhal naprosto spolehlivě.



Obr. 5.15 Generace a rozvod hodinových impulzů

Moderní řešení zdroje hodinových impulzů využívá fázový závěs PLL (angl. Phase Locked Loop) ve funkci násobiče kmitočtu. Násobící činitel je dán obsahem řídicího registru, při nulování počítače je kmitočet nastaven na minimum. Vzhledem k násobení postačí rezonátor s nižším kmitočtem. Často se používají levné a malé „hodinkové“ rezonátory s rezonančním kmitočtem 32,768 kHz. Oscilátor s nízkým kmitočtem má menší rušivé vyzařování a menší příkon.

Hodinové impulzy vstupují do procesoru, kde určují základní **takty**. Strojový cyklus se skládá z několika taktů – jejich počet závisí na architektuře procesoru. Prostřednictvím dvou bitů v řídicích registrech procesoru lze zablokovat hodinové impulzy v procesoru nebo zablokovat přímo oscilátor (viz instrukce IDLE a STOP v kapitole 4). V tomto i v dalších obrázcích značí symbol □ → jeden bit v některém z řídicích registrů.

U jednočipových mikropočítačů jsou dále hodinové impulzy vedeny **k periferním obvodům** na čipu. Periferní obvody většinou potřebují hodinové impulzy o nižším kmitočtu a proto mají své **předděliče** s programovatelným dělicím po-

426410

měrem. U mnoha jednočipových mikropočítačů lze vstup hodinových impulzů do periferního obvodu blokovat jedním bitem v řídicím registru obvodu a tím obvod vyřadit z činnosti, není-li pro danou aplikaci potřebný.

Vyřazování obvodů z činnosti tím, že se zablokují hodinové impulzy, je prostředkem k minimalizaci příkonu. Všechny procesory i jednočipové mikropočítače jsou vyráběny technologií CMOS, jejíž vlastností je zanedbatelný příkon ve statickém stavu a přibližně lineárně rostoucí příkon s kmitočtem hodinových impulzů. Zablokování hodinových impulzů tedy slouží stejně dobře jako odpojení napájení, ale obvod zůstane v definovaném vnitřním stavu a po opětovném dodání hodinových impulzů pokračuje v původní činnosti. Po odpojení napájecího napětí a jeho opětovném přivedení však obvod bude v nedefinovaném stavu a je nutné jeho vynulování. Dalším prostředkem ke snížení příkonu je snížení kmitočtu hodinových impulzů pro procesor. Ten je totiž největším spotřebičem a opět jeho příkon je zhruba úměrný kmitočtu. Manipulace s kmitočtem generátoru je možná díky existenci fázového závěsu. Kmitočet lze nastavit jen tak vysoký, jak je třeba pro danou aplikaci (ne vždy je nutný maximální výpočetní výkon, úspora energie může být důležitější).

5.7 DIAGNOSTICKÉ PROSTŘEDKY POČÍTAČE

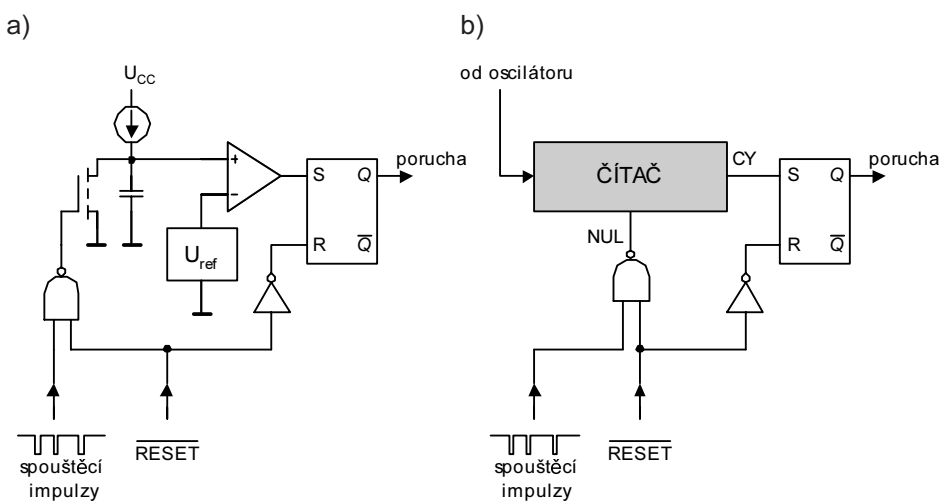
Diagnostika počítače představuje velmi rozsáhlou problematiku a týká se jak programů, tak i obvodů. Ani jeden z těchto prostředků není sám o sobě dokonalý a je nutné je vhodně kombinovat. Softwarové diagnostické prostředky jsou velmi pružné, mají však jednu podstatnou nevýhodu – při zhroucení programu nefungují. Podstatná důležitost hardwarových prostředků je proto zřejmá.

Zmíníme se o nejčastějších kontrolních obvodech. Ty lze u vícečipového počítače doplnit jako vnější obvody k procesoru. U jednočipového mikropočítače je kontrola vnitřních obvodů záležitostí konstruktéra integrovaného obvodu. U moderních architektur je na ně pamatováno.

Velmi častým kontrolním obvodem je **diagnostický časovač** WDT (angl. Watch-Dog Timer). Jedná se o znovuspustitelný (angl. retriggerable) monostabilní klopný obvod nebo jeho funkční ekvivalent, který je programově spouštěn. Do programu jsou vloženy instrukce pro jeho spuštění tak často, aby se stále udržoval v dočasném stavu. Při poruše ve vykonávání programu lze s velkou pravděpodobností očekávat, že dojde k výpadku ve spouštění WDT a obvod se překlopí zpět do stabilního stavu. Tím je generováno hlášení o poruše. Existují zásadně dvě verze tohoto obvodu: asynchronní a synchronní – viz *obr. 5.16*. **Asynchronní verze** na *obr. 5.16a* je založena na analogovém principu, kdy kondenzátor je nabíjen ze zdroje proudu a při dosažení jistého napětí (tj. po jistém čase) se překlopí komparátor. Spínacím tranzistorem je kondenzátor čas od času vybit a tudíž napětí na něm při dostatečné častém vybíjení nikdy komparátor nepřeklopí. Při nu-

lování počítače se vybije kondenzátor a výstup se uvede do neaktivního stavu, takže při náběhu programu je dostatek času na první obsluhu WDT. Obvod vyžaduje kondenzátor o velké kapacitě, kterou nelze realizovat v integrované technologii. Tento WDT je tedy vždy koncipován jako vnější obvod.

Synchronní verze WDT – viz *obr. 5.16b* – využívá oscilátor a za ním následující čítač se vstupem nulování. Pokud čítač počítá nahoru a je dostatečně často programově nulován, nedopočítá nikdy do své plné kapacity. Pokud dojde k chybě v provádění programu a ten neobsluhuje WDT dostatečně často, dojde k přenosu z nejvyššího bitu čítače a je tak signalizována porucha. Jako u asynchronní verze, i zde je během nulování počítače WDT blokován a jeho čítač je vynulován. Tento typ WDT může být součástí jak vnějších obvodů (např. kontrolních obvodů napájení), tak může být i uvnitř procesoru či jednočipového mikropočítače. Pokud je uvnitř, nemá svůj oscilátor a využívá hodinové impulzy procesoru. V tomto uspořádání však nelze rozpoznat selhání zdroje hodinových impulzů – čítač totiž přestane čítat a pak nikdy nedojde k přenosu. Proto při umístění WDT uvnitř jsou diagnostické obvody doplněny ještě o **kontrolní obvod oscilátoru**. Jeho výstup je logicky sečten s výstupem WDT.



Obr. 5.16 Diagnostický časovač: a) asynchronní verze; b) synchronní verze

Výstupem WDT je zpravidla generován nulovací impulz počítače a tím může být celý program nastartován znova od začátku. Toto řešení není ideální, neboť při trvalé poruše v paměti programu dojde k zásahu WDT znova (periodicky), což musí být u zařízení s vyššími nároky na spolehlivost vyloučeno. U vyspělejších architektur jednočipových mikropočítačů je proto aktivita každého zdroje nulova-

cího signálu **identifikována**, a to zápisem do příslušného bitu v jednom ze stavových registrů procesoru, nebo je patřičně modifikována počáteční adresa programu – typicky existuje **několik reset-vektorů**. Při náběhu se pak program větví a v případě nulování od WDT lze provést jinou akci, než při normálním náběhu. Pokud je WDT ve vnějších obvodech, nemusí jeho výstup způsobit nulování, nýbrž může být zaveden do speciálních havarijních obvodů.

Ke spouštění WDT ve vnějších obvodech lze využít jeden bit jedné výstupní brány, nebo může být adresován prostřednictvím adresového dekodéru na vnějších sběrnicích. Pokud je uvnitř jednočipového mikropočítače, zachází se s ním jako s registrem, který má svoji adresu v prostoru periferních obvodů – jen zápis do něj je záměrně zkomplikován, aby se snížila pravděpodobnost jeho falešného spouštění při „bloudění" programu v poruše. Zpravidla je do registru WDT nutné zapsat předepsané číslo, nebo se musí zápis bezprostředně po sobě dvakrát opakovat, apod.

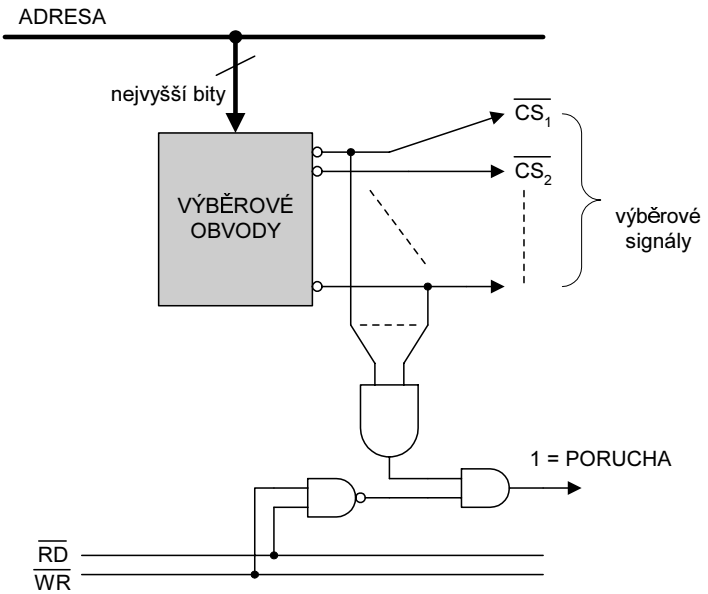
V programu je nutné **rozmístit instrukce s obsluhou WDT** tak, aby nebyl překročen jeho nastavený čas (typicky desítky ms). Musí být proto vloženy jak v hlavní větvi programu, tak v podprogramech. Je nutné zvážit dobu provádění programových smyček (zvláště při iteracích) a eventuálně vložit obsluhu WDT i do nich. Při ladění programů je nutné WDT blokovat – to je u jednočipových mikropočítačů možné jedním bitem registru WDT.

Diagnostická účinnost WDT není absolutní. Mezi výskytem poruchy a vyvoláním zásahu WDT může uplynout dosti **dlouhá doba**, za kterou může vadně fungující počítač způsobit řadu chybných akcí. Proto jsou využívány ještě další kontrolní obvody.

Kontrola správnosti adresy zjišťuje, zda omylem nebyla vydána adresa, která nepatří žádné z existujících jednotek počítače. Na datovou sběrnici se pak v okamžiku čtecího impulzu nepřipojí žádný obvod a data, která jsou čtena, závisí na impedančním zakončení sběrnice. V každém případě jsou to data nesprávná, ale program o tom nemá žádnou informaci. Pokud se jedná o zápis, budou data ztracena, neboť se nikam nezapsala. Kontrolní obvody jsou založeny na zpracování výběrových signálů z adresového dekodéru (případně dokonalejších výběrových obvodů). Jestliže při čtecím nebo zápisovém sběrnicovém cyklu není žádný výběrový signál aktivní, zřejmě byla vydána adresa neexistující jednotky a je generováno hlášení o poruše – viz *obr. 5.17*.

Kontrola správnosti adresy nepostihuje případ, že adresovaná jednotka sice existuje, ale nepracuje v důsledku své vnitřní poruchy. Důležité jednotky proto mohou být vybaveny svými vnitřními diagnostickými obvody, které na konci sběrnicového cyklu potvrzují správné provedení operace. **Potvrzovací signály** z jednotek jsou přes obvody s otevřenými kolektory přivedeny na společnou potvrzovací sběrnici, zavedenou do centrálních kontrolních obvodů. Ty pak kromě kontroly správnosti adresu ještě kontrolují včasné potvrzení operace.

426410



Obr. 5.17 Kontrola správnosti adresy

Jiným případem je **kontrola správnosti čtené instrukce**, přesněji operačního kódu. Neexistující kód hlásí dekodér instrukcí, který je vnitřní částí procesoru. Mezi chybou při čtení dat a při čtení instrukce je podstatný rozdíl: zatím co u dat nelze na poruchu usuzovat z jejich hodnoty (všechna čísla jsou možná), u instrukce existuje jen omezený počet operačních kódů (co je mimo tuto množinu, je chybou). Tyto kontrolní obvody musí být zabudovány přímo v dekodéru instrukcí a nelze je realizovat dodatečně jako vnější jednotku.

6 PŘERUŠENÍ PROGRAMU

Přerušení je metoda, kterou počítač reaguje na asynchronní, neočekávanou událost a vyvolá její obsluhu. Události mohou být omezeny na vnitřní obvody procesoru nebo jednočipového mikropočítače, nebo mohou být vnější vzhledem k počítači. Takovou událostí, která nastává v periferních obvodech, může např. být doběhnutí časovače, změna na vstupech paralelní brány, příjem znaku v sériovém přijímači, dokončený převod v A/Č převodníku a mnoho jiných. Události, které mají svůj původ uvnitř procesoru, jsou typicky vyvolávány diagnostickými obvody a jsou zvláštním případem událostí. Někteří výrobci je označují jako „traps“ (pasti) na rozdíl od ostatních událostí v periferních či vnějších obvodech, které jsou pak označovány jako „interrupts“ (přerušení). U jiných výrobců je používáno označení „exceptions“ (výjimky) pro obě tyto kategorie událostí.

Událost vyvolá přerušení dosud běžícího programu a odskok na obslužný podprogram. Po jeho dokončení se počítač opět vrací do přerušeného programu. V případě periferních obvodů je obsluha přerušení poměrně jednoduchá a spočívá v přečtení příslušného datového registru a přesunu dat do datové paměti nebo opačně, v přesunu dat z paměti do datového registru periferního obvodu. Často je přitom nutné ošetřit i řídicí registry periferních obvodů tak, aby byla umožněna jejich další činnost. Událost může být také podnětem pro výstupní operaci, např. změnu stavu na výstupech paralelní brány.

Při práci v reálném čase je třeba, aby vstupní i výstupní operace probíhaly v přísně stanovených časech s malými tolerancemi. Zde je důležitým parametrem **latence přerušení** (angl. interrupt latency), což je doba od výskytu události až do začátku její obsluhy.

Výskyt události je signalizován požadavkem na přerušení IRQ (angl. Interrupt Request), a to buď hladinou nebo hranou. Hladinový požadavek je aktivní **po celou dobu trvání** stanoveného stavu – často je to logická 0 – pak tedy je značen jako $\overline{\text{IRQ}}$. Hranový požadavek existuje jen krátkodobě a musí být zachycen v obvodu citlivém na změnu vstupního stavu, typicky v klopném obvodu.

6.1 ŘADIČ PŘERUŠENÍ

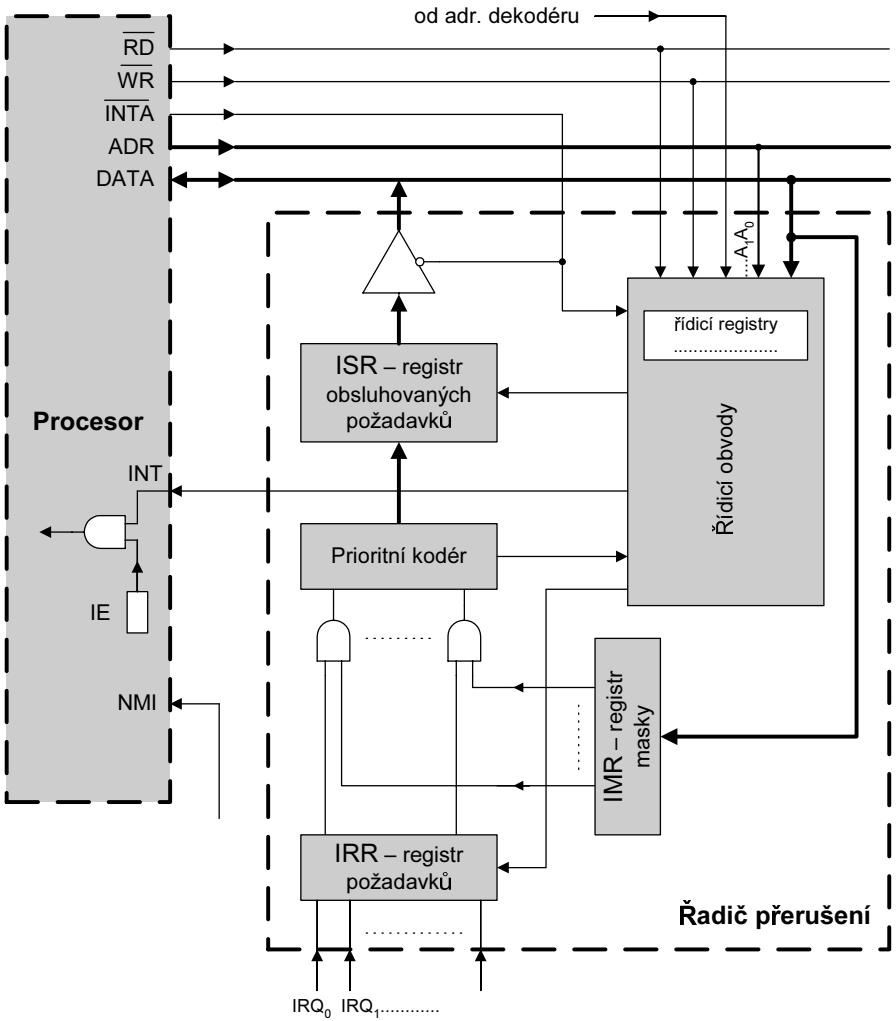
Požadavků na přerušení je prakticky vždy větší počet. K jejich zpracování pak slouží **řadič přerušení** (angl. interrupt controller). Řadič úzce spolupracuje s procesorem, ale není jeho součástí. U vícečipové koncepce je vnějším obvodem, nebo je integrován s dalšími vnějšími obvody.

426410

Úloha řadiče je následující:

- Registrovat aktivní požadavky na přerušení.
- V případě více než jednoho aktivního požadavku je zařadit dle priority.
- Dodat procesoru informaci o tom, který požadavek byl vybrán, tj. identifikovat zdroj požadavku.
- Informovat procesor o tom, že existuje alespoň jeden aktivní požadavek.

Obr. 6.1 ukazuje hlavní části řadiče přerušení, řešeného jako vnější obvod k procesoru.



Obr. 6.1 Hlavní části vnějšího řadiče přerušení

Registr požadavků na přerušení IRR (angl. Interrupt Request Register) uchovává vstupní požadavky IRQ po dobu nutnou k jejich zpracování navazujícími obvody. Klopné obvody IRR lze naprogramovat tak, že reagují buď na hladinu, nebo na hranu. U některých řadičů lze volit náběžnou nebo doběžnou hranu, někdy i obě – požadavek na obsluhu je pak vyvolán každou změnou stavu IRQ.

Jednotlivé výstupy IRR mohou být blokovány jednotlivými bity registru masky IMR (angl. Interrupt Mask Register), takže jen ty požadavky, které jsou povoleny, postupují dále. Maskování požadavků může být průběžně měněno během provádění programu. Prioritní kodér vybírá aktivní požadavek s nejvyšší prioritou a přiřazuje mu binární číslo – IRQ_0 přiřadí 000, IRQ_1 přiřadí 001 atd. Současně je vytvářen signál INT, kterým je informován procesor o existenci požadavku na přerušení. Procesor pak zahájí komunikaci s řadičem přerušení jako první akci pro zpracování přerušení. Požadavek INT může být maskován **bitem globální masky přerušení** – v *obr. 6.1* je reprezentován symbolem □→ a je označen jako IE (angl. Interrupt Enable – povolení přerušení). U samostatných procesorů je tento bit zpravidla součástí stavového registru PSW a ovládá se zvláštními instrukcemi „povol přerušení" a „zakaž přerušení". Polarita je u různých procesorů různá – u některých se přerušení povoluje stavem 1, u jiných stavem 0. Při nulování procesoru (Reset) se přerušení vždy automaticky zakazuje.

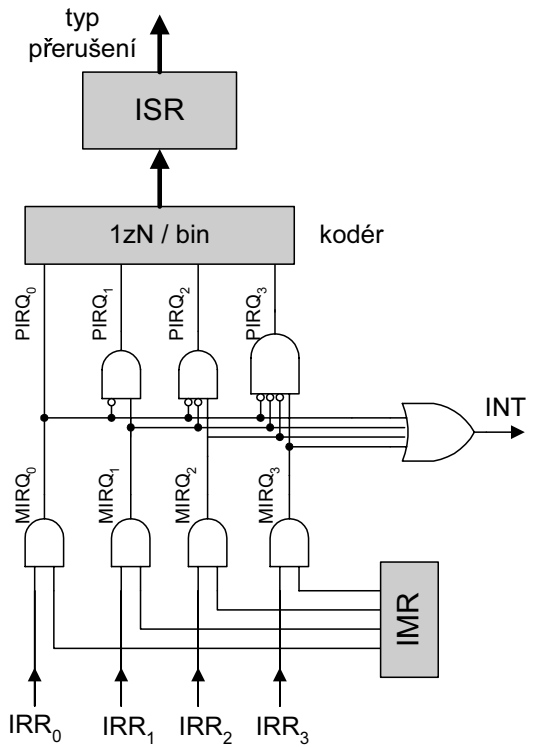
Obsah registru obsluhovaných požadavků ISR (angl. In-Service Register) udává typ přerušení. Je čteno ve zvláštním sběrnicovém cyklu „čtení typu přerušení". Na *obr. 6.1* je tento cyklus doprovázen čtecím impulzem na zvláštním vodiči řídicí sběrnice $\overline{\text{INTA}}$ (angl. Interrupt Acknowledge), u jiných procesorů mohou být k tomuto účelu využity jiné řídicí signály nebo jejich kombinace – v každém případě se jedná o speciální čtecí cyklus, nezaměnitelný s jinými čtecími cykly (z paměti apod.).

Do řídicích registrů řadiče a do registru masky zapisuje procesor obdobně jako do ostatních periferních obvodů, které mají vyhrazenu část adresového prostoru. Obsahem řídicích registrů je určen režim činnosti řadiče (způsob vyhodnocení priority, způsob zachycení požadavků IRQ, apod.). Vzhledem k existenci několika registrů je do řadiče zavedena i adresa o rozsahu jednoho nebo více nejnižších bitů.

Téměř každý procesor je vybaven ještě vstupem pro nemaskovatelné přerušení NMI (angl. Non-Maskable Interrupt), který má absolutní prioritu a není veden přes řadič. Má kratší latenci než maskovatelné přerušení a nevyžaduje spolupráci procesoru s řadičem. Je využíván pro havarijní případy (zablokování programu, selhání obvodů, atd.).

Způsobů vyhodnocení priority je několik. Nejčastější (a nejjednodušší) je **priorita podle připojení** nebo též **pevná priorita**. Zpravidla vstup IRQ_0 má nejvyšší prioritu, IRQ_1 druhou nejvyšší, atd. Obvody takovéhoto prioritního kodéru s maskováním ukazuje *obr. 6.2*.

Požadavky zachycené v registru IRR jsou logicky vynásobeny s výstupy maskovacího registru IMR. Je-li stav příslušného bitu IMR nula, požadavek se neuplatní. Prioritní obvod propustí požadavek $MIRQ_0$ vždy, $MIRQ_1$ jen je-li $MIRQ_0$



Obr. 6.2 Prioritní obvod s pevně přiřazenou prioritou

neaktivní (tj. 0), $MIRQ_2$ jen jsou-li $MIRQ_0$ i $MIRQ_1$ neaktivní, atd. Při libovolné kombinaci aktivních signálů MIRQ (stav 1) je aktivní jen jediný signál PIRQ. K dalšímu zpracování se proto může použít běžný kodér, převádějící kód 1 z N na kód binární. Pokud je alespoň jeden ze signálů MIRQ aktivní, je v součtovém členu vytvořen signál INT, kterým je informován procesor.

Dalším způsobem vyhodnocení priority (i když výjimečným) je **cyklická priorita**. Při ní se s každým vykonaným obslužným podprogramem přesunuje maximální priorita o 1 místo doprava, takže bude na pozici IRQ_1, druhá nejvyšší na IRQ_2 atd., až na IRQ_0 bude nejnižší. Při dalším přerušení se opět posune, atd. Cyklická priorita je využívána výjimečně – je vhodná tam, kdy obsluhované požadavky mají stejnou důležitost a vyskytují se stejně často. Při pevně přiřazené prioritě by totiž požadavek na IRQ_0 získával obsluhu přednostně a na ostatní – ačkoliv stejně důležité – by nedošla řada.

Dokonalejší řadiče umožňují každému požadavku přiřadit **libovolnou prioritu** s jediným omezením: žádné 2 požadavky nesmějí mít priority shodné. K nastavení priorit je každý vstup opatřen individuálním registrem priority. Priority lze během činnosti programu měnit. Obvodové řešení vyžaduje vřazení přepínače (např. jako matice spínačů MOS ve funkci multiplexeru) mezi maskované signály a vstupy prioritního kodéru.

Mezi libovolným přiřazením priority a pevnou prioritou leží metoda zařazení prorit do úrovní, typicky dvou. Řadič je v tom případě vybaven **registrem úrovně priority**, kde každý požadavek IRQ má svůj bit, určující zařazení požadavku do úrovně priority H (vyšší)nebo L (nižší). V rámci téže úrovně stále platí pevná priorita podle připojení, ale požadavky ve vyšší úrovni mají vždy přednost. Tak např. při zařazení do úrovně dle *obr. 6.3* má požadavek IRQ_7 vyšší prioritu než IRQ_0, ale v rámci vyšší úrovně má až nejnižší prioritu. Nejvyšší prioritu ze všech bude mít požadavek IRO_3 a nejnižší IRQ_5.

úroveň H				IRQ_3	IRQ_4		IRQ_6	IRQ_7
úroveň L	IRQ_0	IRQ_1	IRQ_2			IRQ_5		
	← vyšší ..... **priorita** ..... nižší →							

Obr. 6.3 Zařazení požadavků do dvou úrovní priority přerušení

6.2 ČINNOST PROCESORU A ŘADIČE PŘI PŘERUŠENÍ

Při výskytu aktivního požadavku, který není maskován příslušným bitem v registru IMR, je signálem INT = 1 upozorněn procesor. Byl-li bit globální masky nastaven do stavu „povoleno“, je zahájena obsluha přerušení. Nejprve je dokončena momentálně prováděná instrukce. Výjimkou jsou některé instrukce, jejichž dokončení by trvalo příliš dlouho. Typicky jsou to u některých procesorů cyklicky probíhající instrukce pro zpracování bloků dat – takovéto instrukce lze přerušit a po návratu dokončit. Pak je do zásobníku uložena návratová adresa. U některých procesorů je dále uloženo stavové slovo PSW a někdy i další důležité registry (akumulátor, indexregistry …) – to je ale spíše výjimečné. U většiny procesorů se o uložení těchto registrů musí postarat programátor.

Dále procesor vyvolá cyklus čtení typu přerušení z řadiče přerušení. Přitom je v řadiči nejprve zapsán typ přerušení do registru ISR, a pak je vynulován ten klopný obvod IRR, kterým bylo vyvoláno přerušení. To je však typické jen u klopných obvodů naprogramovaných na reakci na **hranu** IRQ. Klopné obvody řízené **hladinou** IRQ se nevynulují, pokud požadavek IRQ trvá. Číslo, vyjadřující typ přerušení, zůstává zafixováno v ISR. Procesor přečte typ přerušení a přiřadí mu počáteční adresu obslužného podprogramu.

Přiřazení může být u některých procesorů pevné, kdy každému typu přerušení je pevně přiřazena jiná počáteční adresa obslužného podprogramu v programové paměti. Procesor volá podprogram na této adrese. Začátky podprogramů jsou pro úsporu místa poměrně blízko sebe (několik slabik) a do mezery mezi nimi se v podstatě vejde jen instrukce skoku na další pokračování podprogramu. Čas-

tějším případem je interpretace typu přerušení jako **ukazatele do tabulky vektorů**. Tabulka je uložena v programové paměti a jsou v ní těsně za sebou seřazeny počáteční adresy podprogramů. Typ přerušení je vynásoben konstantou tak, aby jako ukazatel vždy ukazoval na některou počáteční adresu. Tak např. u počítače s pamětí adresovanou po slabikách a s 16bitovou adresou (tj. 2 slabiky) je typ přerušení vynásoben dvěma. Kdyby byla adresa 4slabičná, byl by vynásoben čtyřmi. Při vyvolání obslužného podprogramu procesor dosadí adresu z tabulky vektorů. Je-li tabulka umístěna v paměti RAM, jak je tomu např. u personálních počítačů, mohou být adresy obslužných podprogramů dynamicky měněny, tj. během běhu programu. U počítačů určených pro řízení (vestavěné počítače) je program i tabulka typicky v paměti ROM a nelze je měnit. Některé procesory proto mají zvláštní registr pro určení počátku tabulky, jehož obsah je sečten s ukazatelem. V paměti ROM tak může existovat několik předem připravených tabulek a podle potřeby je vybrána jedna z nich.

Současně s vyvoláním podprogramu procesor nastaví globální masku přerušení na stav „zakázáno“ a tím je znemožněno vyvolání dalšího přerušení během již běžícího obslužného podprogramu. Pokud je existence **do sebe vložených** (angl. Nested) obslužných podprogramů skutečně nutná, může být globální maska programově nastavena na „povoleno“. Dříve však, hned na začátku podprogramu, je nutné uložit do zásobníku registr PSW a další datové registry – ne nutně všechny, ale alespoň ty, jejichž obsah bude v podprogramu ovlivněn. Jak již bylo zmíněno, některé procesory tuto činnost vykonávají při přerušení automaticky. Automatický zákaz přerušení na začátku podprogramu zabraňuje opakovaným přerušením v případě, že přerušení bylo vyvoláno klopným obvodem IRR nastaveným na hladinovou reakci. Ten totiž nelze automaticky nulovat, požadavek trvá a vyvolá nové přerušení ještě dříve, než může obslužný podprogram prostřednictvím vnějších obvodů (v připojeném zařízení) zrušit odpovídají IRQ. To by se dělo znova a znova, zásobník by neomezeně narůstal a program nakonec havaruje.

Na konci podprogramu musí být vrácen původní obsah do těch registrů, jejichž obsah byl předem uložen. I tato činnost je u některých procesorů automatická, u jiných se o ni musí postarat programátor. Těsně před návratem z podprogramu je zapsán příkaz do řadiče přerušení, kterým se určuje priorita pro příští přerušení (např. posunuje se maximum u cyklické priority). Stav ISR, typ přerušení a jeho priorita zůstávají během obslužného podprogramu neměnné, i když případně vznikají nové požadavky IRQ. V registru IRR jsou sice zachyceny, ale řídicí obvody řadiče vytvoří signál INT jen v případě, že nový požadavek má vyšší prioritu než požadavek současně obsluhovaný. Požadavky nižší nebo shodné priority nevytvoří INT. Přerušení běžícího podprogramu je tedy možné, ale jen při odblokování globální masky a při požadavku s **vyšší prioritou**. Činnost procesoru a řadiče přerušení je pak ve vloženém přerušení obdobná, jak bylo popsáno.

Obslužný program je zakončen instrukcí návratu z přerušení (RETI, IRET, apod.), která vyvolá čtení návratové adresy ze zásobníku a její přesun do čítače instrukcí. Současně se globální maska přerušení nastaví na „povoleno“. U **některých**

procesorů je ještě ze zásobníku obnoven **obsah PSW**, případně dalších registrů. V tomto stavu se procesor vrací do přerušeného programu.

Vedle přerušení vyvolaných prostřednictvím obvodů řadiče je u většiny procesorů možno vyvolat přerušení i programově, tzv. **softwarové přerušení**. U vícečipových počítačů k tomu účelu existuje speciální instrukce *SWI n* (nebo jinak pojmenovaná), kde *n* označuje číslo vektoru v tabulce. Lze tak simulovat libovolné hardwarové přerušení. Někdy parametr *n* chybí a SWI je přiřazen jen jediný vektor v tabulce. Instrukce *SWI n* má nejkratší možnou délku, tak aby mohla v paměti programu nahradit libovolnou jinou instrukci. Typické použití je při ladění programu, kdy SWI je vložena na místa bodů zastavení („breakpoint“) a vyvolá obslužný podprogram, který na monitoru zviditelní stav procesoru. To je ovšem možné jen v programu uloženém v paměti RAM. Jiné využití může být u programů v paměti ROM, kdy instrukcí SWI je vyplněn veškerý nevyužitý prostor v ROM – v případě poruchy programu, kdy je omylem čtena instrukce mimo program, je pak přečtena SWI a provedena vhodná operace k nápravě chyby. Opravný program má přitom k dispozici informaci o adrese, na které došlo k chybě – ta je totiž uložena v zásobníku při provedení SWI jako návratová adresa.

6.3 PŘERUŠENÍ PROGRAMU U JEDNOČIPOVÝCH MIKROPOČÍTAČŮ

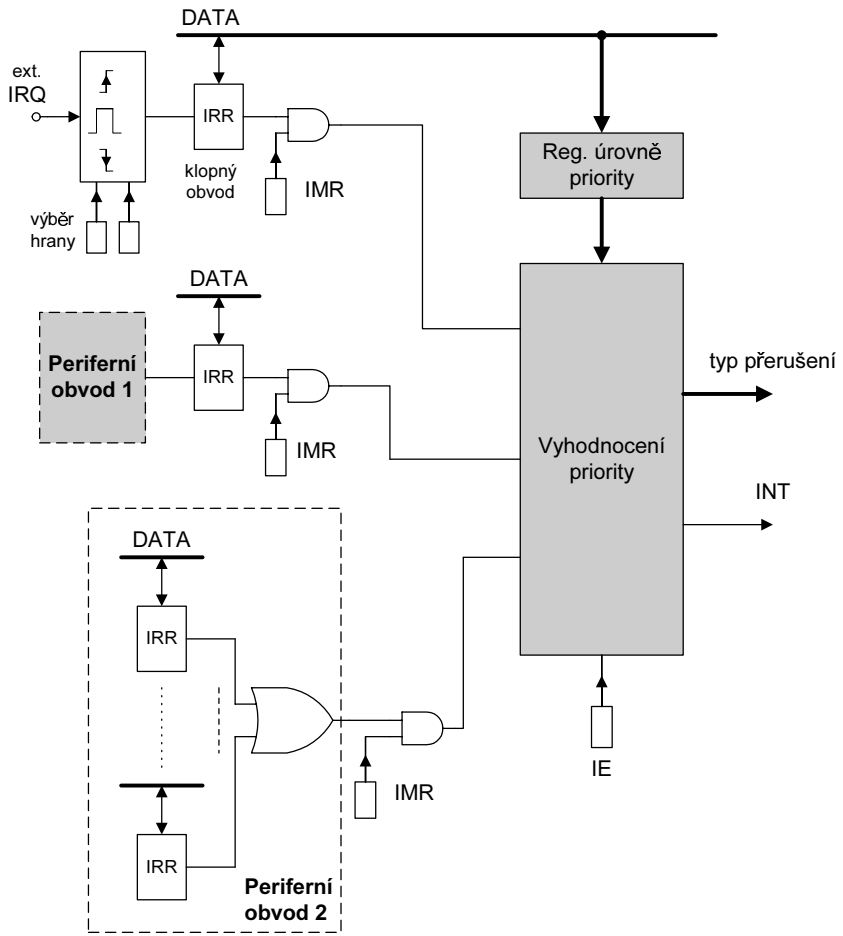
Integrace všech obvodů na jednom čipu vede i v případě přerušení k rozdílům vzhledem k počítačům vícečipovým. Obvody pro zpracování přerušení jsou přímo propojeny s periferními obvody. Procesor má přístup ke všem obvodům řadiče přerušení bez nutnosti komunikace přes vnější sběrnici. To se projevuje v uspořádání bitů v registrech řadiče i v detailech jeho činnosti. Většina požadavků na přerušení je vnitřních, od periferních obvodů a od diagnostických obvodů. Alespoň jeden maskovatelný požadavek je však vyhrazen pro vnější obvody a je vyveden z pouzdra. Velmi často je z pouzdra vyveden i NMI.

Jednotlivé klopné obvody IRR jsou přístupné pro čtení i zápis. To dává možnost **programově testovat** jejich stav a větvit podle toho program. Tak lze vyvolat obslužný program, aniž by se využíval mechanizmus přerušení – reakce na požadavek je však mnohem pomalejší, neboť k ní může dojít jen v předem stanovených bodech programu, kde jsou testovány klopné obvody IRR, zatím co reakce na přerušení by byla bezprostřední a nezávisela by na momentálním vývoji programu.

Zápis do klopných obvodů IRR umožňuje **vyvolat přerušení** stejně, jako by příslušný požadavek skutečně existoval. Tím se dosáhne stejného efektu jako instrukcí SWI, která pak u takovéto architektury vůbec nemusí existovat. Tato programová simulace požadavků IRQ se využívá při ladění programů. Klopný obvod IRR lze též programově vynulovat a zdroj přerušení tak zrušit (což není

totéž, jako maskování). To je zapotřebí v obslužném programu, pokud procesor patřičný klopný obvod sám automaticky nenuluje.

Jednotlivé bity IRR i masky IMR mohou být různě uspořádány. Jejich seskupení se u různých výrobců liší, též jejich symbolické názvy mohou být jiné než IRR a IMR. Na *obr. 6.4* je příklad typického uspořádání obvodů pro zpracování přerušení. V tomto i v dalších obrázcích značí symbol □ → jeden bit v některém z řídicích registrů.



Obr. 6.4 Typické obvody vnitřního řadiče přerušení

U vstupu vnějšího požadavku lze nastavit typ reakce – hladinovou či hranovou, na náběžnou či doběžnou hranu. Výstupní signál klopného obvodu IRR je blokován jedním bitem registru masky (IMR). Vnitřní periferní obvod 1 dává impulzní

požadavek, a proto jeho klopný obvod reaguje na hranu. Periferní obvod 2 představuje složitý systém, jehož každá složka generuje svůj požadavek na přerušení. Pro omezení počtu vektorů v tabulce jsou všechny požadavky logicky sečteny a z celého periferního obvodu pak vychází jen jeden signál, maskovatelný jako celek. Identifikace zdroje přerušení je dokončena až v obslužném podprogramu tak, že se přečte stav klopných obvodů v periferním obvodu 2 a zjistí se, který generoval požadavek. Priorita je velmi často pevně přidělena, ale některé požadavky lze přesunout do vyšší úrovně. Do všech klopných obvodů lze zasahovat z vnitřní datové sběrnice. Je zřejmé, že se dobře uplatní instrukce pro práci s jednotlivými bity.

Seskupení klopných obvodů je v podstatě dvojí:

1. Klopné obvody požadavků jsou seskupeny do jednoho řídicího registru, klopné obvody masky do druhého a přiřazení úrovní priority do třetího řídicího registru. Jsou zásadně umístěny v adresovém prostoru řídicích a stavových registrů (SFR). Tím se z hlediska programování blíží podoba vnitřního řadiče podobě řadiče vnějšího.
2. Klopný obvod IRR požadavku z periferního obvodu a klopný obvod jeho masky IMR jsou začleněny do řídicího registru daného periferního obvodu. Ostatní bity řídicího registru ovládají funkce periferního obvodu. Přerušení je považováno za jednu z funkcí. Registry jsou rovněž v prostoru SFR. Programování a ovládání přerušení je v tomto případě součástí ovládání periferního obvodu. Celé uspořádání je velmi variabilní podle typu periferního obvodu a též podle typu počítače.

U periferních obvodů s jedním vnitřním zdrojem požadavku je klopný obvod, který způsobil přerušení, **automaticky vynulován** při vyvolání obslužného podprogramu – výjimkou je vnější požadavek v případě, že byl nastaven na **hladinovou** reakci. Ale u periferních obvodů s několika vnitřními zdroji požadavku spojenými logickým součtem (viz *obr. 6.4*) je teprve v obslužném podprogramu možné zdroj požadavku identifikovat a programově vynulovat.

U některých jednočipových mikropočítačů je součástí stavového slova PSW bit IE a dále několikabitové číslo, vyjadřující prioritu právě obsluhovaného požadavku. Číslo je generováno prioritním obvodem – viz též registr obsluhovaného požadavku (ISR) na *obr. 6.1*. Umístění těchto informací do PSW je výhodné tehdy, když se PSW automaticky ukládá do zásobníku. Při návratu do přerušeného programu se vrací stav priority a globální masky, jaký byl před vstupem do podprogramu. Priorita momentálně vzniklých požadavků je stále srovnávána s prioritou v PSW. Pokud se během obsluhy požadavku objeví další požadavek s vyšší prioritou než má požadavek současně obsluhovaný a bitem IE je přerušení povoleno, je generováno nové přerušení. Nové požadavky stejné nebo nižší priority jsou však ignorovány. Celé toto uspořádání je ekvivalentem registru ISR a řídicí logiky ve vnějším řadiči přerušení. Navíc je zde možnost kdykoliv do priority programově zasáh-

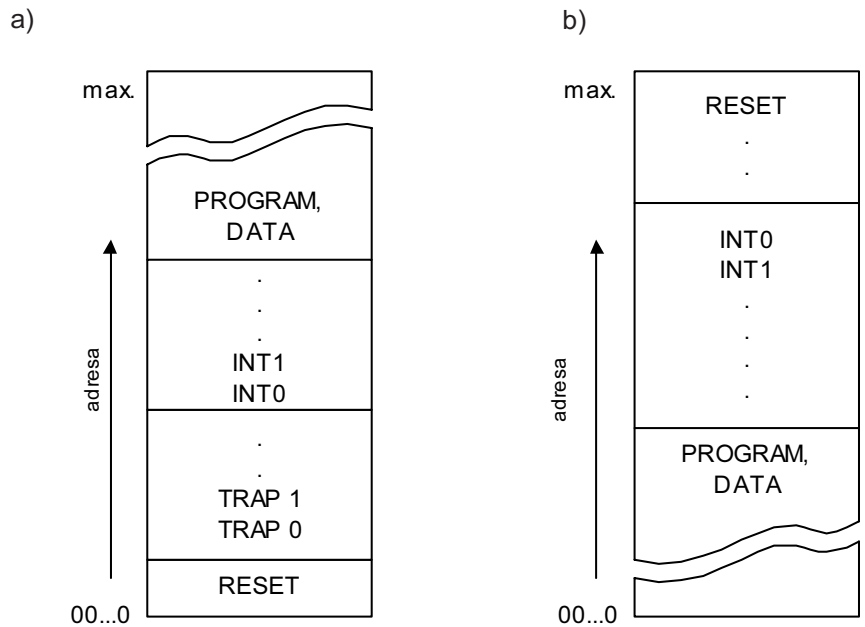
426410

nout. Na *obr. 6.5* je příklad PSW, ve kterém je bit IE a priorita obsluhovaného požadavku.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRIORITA				IE	..... příznakové bity										

Obr. 6.5 Příklad stavového slova programu

Jak bylo popsáno v kapitole 5, nulování počítače může být vyvoláno nejen vnějším signálem (Reset), ale i z řady vnitřních příčin. Aby bylo možné po vynulování přizpůsobit činnost počítače těmto příčinám, jsou některé procesory vybaveny registrem, jehož jednotlivé bity ukazují na zdroje nulovacích povelů. Jiné řešení chápe i nulování jako „výjimku" a jednotlivým příčinám přiřazuje vektory (Reset Vector), takže nový náběh programu je rozvětven od samého začátku. Nulování je však drastický zásah do běžícího programu a některé jeho příčiny nemusí být tak podstatné, aby musel být program spuštěn znovu od začátku. Proto u některých jednočipových mikropočítačů existují **vektory pro výjimky**, které ošetřují abnormální situace (Trap Vector). Činnost procesoru při těchto výjimkách je ob-



Obr. 6.6 Příklady tabulky vektorů: a) na začátku programového adresového prostoru; b) na konci programového adresového prostoru

dobná jako u přerušení, ale s několika odchylkami. Trap vektory mají svoje vlastní pevně dané priority, nesouvisející s vektory přerušení, a nejsou maskovatelné. Výjimky jsou vyvolávány signály z diagnostických obvodů. Mohou přerušit již běžící obslužné podprogramy z maskovatelných přerušení, ale ne naopak. Nemaskovatelné přerušení má však vždy přednost. Příklad tabulky vektorů je na *obr. 6.6*.

6.4 LATENCE PŘERUŠENÍ

Latence přerušení je doba, která uplyne od výskytu požadavku IRQ až do prvé instrukce podprogramu, kterým je daný požadavek obsluhován. Platí následující vztah:

$$T_{I\_max} = T_{inst\_max} + T_{zprac}$$

kde T_{I_max} je doba latence, T_{inst_max} doba na provedení nejdelší instrukce, která nemůže být přerušena a T_{zprac} doba na zpracování přerušení. Během ní je z řadiče čten typ přerušení, přiřazen vektor, z paměti přečten obsah z tabulky vektorů, zakázáno další přerušení, do zásobníku uložena adresa návratu a případně i některé registry. **Některé instrukce lze přerušit**, aby latence nebyla příliš dlouhá. Typicky jsou to instrukce s mnohonásobným cyklických průběhem, jako jsou přesuny bloků dat, hledání v tabulkách apod.

Zatímco T_{zprac} je pro daný počítač konstantní, je T_{inst} proměnná. Jednak jsou instrukce různě dlouhé, jednak požadavek se může objevit kdykoliv během instrukce – ta však musí být dokončena (s výše uvedenými výjimkami). Jediným jistým údajem je proto T_{I_max}. Skutečná doba latence v sobě bude obsahovat složku konstantní a složku **náhodnou**. To je třeba vzít v úvahu u programů s vysokými nároky na přesné časování. Doba latence se u jednočipových mikropočítačů pohybuje v řádu několika mikrosekund.

6.5 ZÁSADY PRO PRÁCI S PŘERUŠENÍM

Při sestavování programu, který vyžaduje přerušení, je třeba nejprve naprogramovat řídicí registry, celý přerušovací systém a zaplnit tabulku vektorů. Při vynulování počítače je vždy bit globální masky nastaven na „zakázáno“, takže po náběhu programu nehrozí nebezpečí nedefinované činnosti při případných žádostech o přerušení. Podle požadované činnosti programu je však třeba zaplnit řídicí registry řadiče, přiřadit priority a zamaskovat či povolit jednotlivé vnitřní zdroje požadavků na přerušení. Dále je třeba nastavit **počátek zásobníku** tím, že se do registru SP uloží počáteční adresa. I když některé procesory při vynulování registr SP automaticky nastavují, nemusí být tato pozice zásobníku optimální a je nutné ji změnit.

Priority přerušení jsou zpravidla přiděleny takto (postupně od nejvyšší NMI k nejnižší A/Č):

1. Nemaskovatelné přerušení (NMI).
2. Přerušení od diagnostických obvodů.
3. Vnější požadavek (EXT IRQ).
4. Časovače.
5. Sériové vstupy/výstupy.
6. Převodníky A/Č.

Priority NMI a diagnostických obvodů nelze měnit, ostatní ano. Výše uvedené přiřazení vyhovuje většině systémů, pracujících v reálném čase, kdy přesné časování má zvláštní význam. Teprve po ošetření všech přerušovacích obvodů lze povolit globálním maskovacím bitem přerušení.

Globální zákaz přerušení je využíván i během činnosti programu. Jedná se o tzv. „kritické úseky programu". Přerušení se zakazuje během zápisů do řídicích registrů řadiče přerušení i do jiných periferních obvodů. Přerušení je třeba zakázat i v těch úsecích programu, které pracují se stejnou oblastí datové paměti, se kterou by pracoval i obslužný podprogram v přerušení. Zabrání se tak nežádoucí situaci, kdy hlavní program zpracovává data a obslužný podprogram je současně mění – např. získává vzorky signálu z A/Č převodníku a plní jimi paměť.

Zákaz přerušení, jakkoliv je někdy nutný, prodlužuje latenci přerušení, která pak je:

$$T_{I\_max} = T_{zak} + T_{zprac},$$

kde T_{zak} je doba, po kterou je přerušení zakázáno. Požadavek se obecně může vyskytnout kdykoliv během této doby, takže v době latence se vyskytuje významná náhodná složka.

Vnější požadavky (EXT, IRQ) mohou být jak hranové, tak hladinové. U hranové reakce je nutné zabránit překmitům na náběžné i doběžné hraně impulzu. K překmitům dochází u signálů z mechanických spínačů vlivem nedokonalosti kontaktů při spínání i rozpínání. Dochází k nim i vlivem špatně impedančně zakončeného dlouhého vedení. U hladinové reakce, kdy požadavek působí jako aktivní po celou dobu trvání impulzu, musí požadavek skončit před návratem z obslužného podprogramu – pokud trvá, bude po návratu vyvolávat znova tentýž obslužný podprogram. Obsluhované zařízení včetně případných doplňkových obvodů musí požadavek ukončit **před koncem** obslužného podprogramu.

Zaplnění tabulky vektorů znamená znát počáteční adresy obslužných podprogramů. Tuto etapu zajišťuje překladač nebo spojovací program (linker). Zaplňuje však jen použité vektory. **Nepoužité vektory** je vhodné zaplnit adresou podprogramu pro diagnostické účely. K jeho vyvolání může dojít jen chybou v programu a v tom případě by měla být provedena vhodná opravná akce a generováno chybové hlášení.

426410

7 PAMĚŤ

Paměť, a zvláště paměť programu, je součástí počítače, se kterou procesor pracuje nejčastěji. Její dynamické parametry musí plně vyhovovat časování signálů procesoru. Do sběrnicového cyklu je sice většinou možné vkládat čekací takty (viz kapitolu 5.2), v případě programové paměti však toto řešení znehodnotí rychlost procesoru a celého počítače. Datová paměť představuje menší problém. Jednak s ní procesor pracuje v průměru méně často než s pamětí programovou a případné vkládání čekacích taktů by proto nepředstavovalo tak podstatné časové ztráty, jednak datové paměti jsou zpravidla rychlejší než paměti programu. Obdobně jako s datovou pamětí, i s periferními obvody se pracuje v cyklu čtení i cyklu zápisu. Často jsou však pomalejší a vyžadují vkládání čekacích taktů.

U některých procesorů s oddělenými adresovými prostory pro paměť programu, paměť dat a periferní obvody jsou vnější sběrnicové cykly vzájemně odlišně časovány. Pro každou z těchto kategorií obvodů je pak nutné provést rozbor časování s jinými dynamickými parametry procesoru.

Výběru vhodných paměťových obvodů z hlediska jejich dynamických vlastností je věnován odstavec 7.2.

7.1 TYPY PAMĚTÍ

Pro realizaci programové paměti lze využít následující varianty:

- **Paměť ROM**, maskou programovaná. Obsah je dán již při výrobě. Typické využití je u některých verzí jednočipových mikropočítačů, vyráběných pro jednu aplikaci ve velkých sériích.
- **Paměť EPROM**, programovaná uživatelem. Paměť lze naprogramovat jen v přístroji – programátoru pamětí. K vymazání obsahu je třeba paměť umístit do mazacího zařízení, produkujícího ultrafialové záření. Pouzdro je vybaveno okénkem z materiálu propouštějícího toto záření. Celá manipulace je zdlouhavá. Paměti EPROM dovolují jen několik desítek přeprogramování a nehodí se pro rozsáhlejší experimentální práci. Má-li se uplatnit možnost přeprogramování, musí být na plošném spoji obvody EPROM uloženy v objímkách. Tím se systém prodražuje a snižuje se jeho spolehlivost.
- **Paměť PROM**, realizovaná jako jednou programovatelná paměť EPROM. Technologicky se jedná o paměť EPROM, pouzdro však nemá okénko a paměť nelze mazat. Obvod je levnější. Odpadá objímka, v plošném spoji je obvod zapájen. Tento typ paměti, často označovaný jako **OTP** (angl. One-Time Programmable), je vhodný pro sériovou výrobu systémů, jejichž program se již nebude měnit.

- **Paměť EEPROM**, kterou lze programovat jak v programátoru, tak i přímo v systému – pak může být zapájena v plošném spoji. Obsah se programuje po jednotlivých adresách, vždy celé slovo. Fyzikální děje jsou shodné pro programování stavu 1 i stavu 0 v jednotlivých buňkách, proto zde neexistuje ekvivalent „mazání“ celého obsahu. Programování je však pomalé (ve srovnání s EPROM nebo FLASH). Paměť dovoluje řádově 10^4 přeprogramování (někteří výrobci uvádějí 10^5).
- **Paměť FLASH**, kterou lze programovat jak v programátoru, tak i přímo v systému. Programuje se po jednotlivých adresách, mazat se však musí celá. Paměti FLASH o velké kapacitě bývají rozděleny na sektory, které lze programovat a mazat jednotlivě. Programování probíhá rychleji než u EEPROM (srovnatelně s EPROM – fyzikální princip programování je shodný), mazání je však pomalé. I tak je naprogramování celé paměti po jejím předchozím vymazání mnohem rychlejší, než je naprogramování celé paměti EEPROM. Paměť dovoluje řádově 10^4 přeprogramování (někteří výrobci uvádějí 10^5). Buňka paměti FLASH je jednodušší, než je buňka paměti EEPROM a lze proto dosáhnout vyšších paměťových kapacit a nižší ceny.
- **Paměť RAM**. Obsah paměti není po náběhu napájení definován, program proto do ní musí být nejprve přesunut buď z některé z výše uvedených permanentních pamětí, nebo z disku, nebo po komunikačním kanálu z jiného zařízení. Naplnění programové paměti z disku je typické pro personální počítače. Program v paměti RAM může sám sebe modifikovat, může modifikovat i tabulky vektorů přerušení. Paměti RAM, a zvláště paměti statické, jsou všeobecně rychlejší, než všechny výše uvedené permanentní paměti. Lze tak dosáhnout i několikanásobného zrychlení činnosti počítače.

Z porovnání vlastností jednotlivých typů programových pamětí vyplývá i jejich nejvýhodnější využití. Pro sériovou výrobu je vhodná paměť OTP. Pro experimentální práci i pro výrobu v malých sériích se hodí paměť FLASH. Paměť EEPROM je vhodná spíše pro zálohování dat, neboť umožňuje zápisy na jednotlivé adresy. Paměť EPROM se stává zastaralou. Paměť RAM umožňuje výrazné zvýšení rychlosti a dává možnost modifikace programu, celý systém je však složitější.

Pro realizaci **datové** paměti lze využít následující varianty:

- **Statická paměť RAM** (SRAM) je založena na matici buněk, tvořených klopnými obvody. Prakticky jediná technologie je v současné době CMOS. Ovládání SRAM je extrémně jednoduché: výběrovým signálem $\overline{CS}$ je paměťový obvod vybrán, impulzem na vstupu $\overline{WR}$ je do paměti zapisováno, signál na vstupu $\overline{OE}$ slouží jako čtecí povel, který připojuje třístavové výstupní členy na datovou sběrnici. Všechny bity adresy jsou po celou dobu čtecího či zápisového cyklu udržovány jako konstantní. Detaily průběhu čtecího cyklu jsou na *obr. 7.2b*, zápisového cyklu na *obr. 7.3b*. Obsah paměti SRAM je stabilní, dokud není přepsán a dokud má paměť napájecí napětí.

- **Dynamická paměť RAM** (DRAM) je založena na matici buněk, tvořených paměťovými kondenzátory a spínači. Paměťový kondenzátor se časem samovolně vybíjí a obsah DRAM se musí pravidelně obnovovat. Ovládání DRAM je složitější, než SRAM. Adresa je multiplexována a je do obvodu přiváděna po stejných vývodech pouzdra ve dvou krocích, řízených signály $\overline{RAS}$ (Row Address Strobe) a $\overline{CAS}$ (Column Address Strobe). Třetí řídicí signál $\overline{WR}$ určuje typ cyklu. Multiplexování adresy, generaci $\overline{RAS}$ a $\overline{CAS}$ a obnovování obsahu provádí řadič dynamické RAM. Je to buď obvod navíc, nebo je integrován s procesorem. Paměťová buňka je u DRAM velmi jednoduchá a proto se u DRAM dosahuje vysokých kapacit v jednom pouzdru, a to při výhodné ceně.

Z porovnání vlastností jednotlivých typů datových pamětí vyplývá i oblast jejich použití. Protože statická paměť je rychlá, jednoduše se ovládá a nepotřebuje speciální řadič, je zpravidla využívána u jednoduchých systémů s menším rozsahem datové paměti. Naopak dynamická paměť je téměř výhradně využívána u personálních počítačů, a to ve funkci jak programové, tak i datové paměti o velké kapacitě.

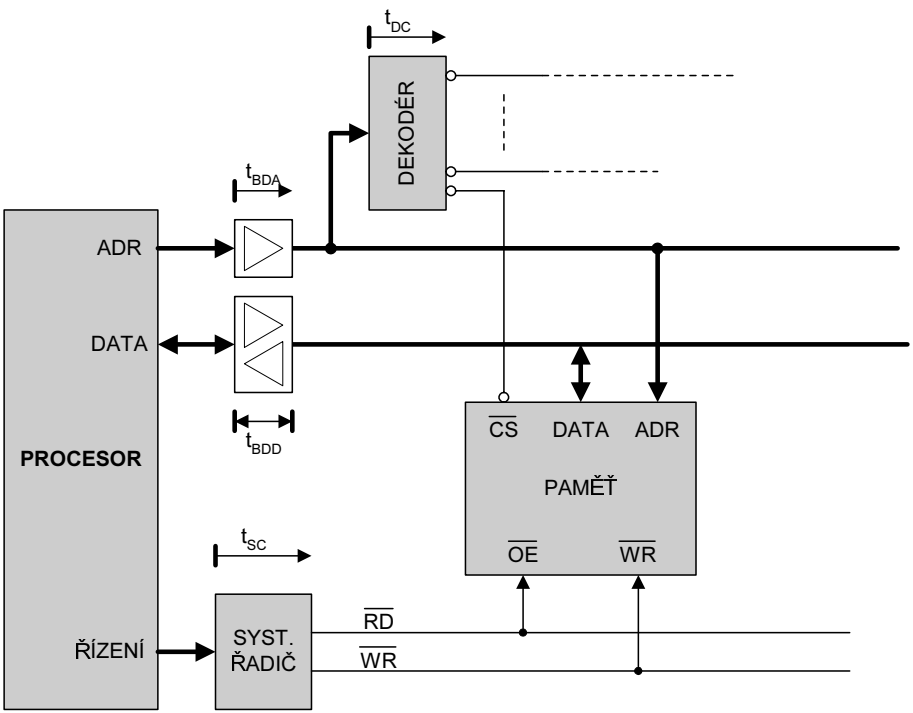
Jednočipové mikropočítače obsahují jak programovou, tak i datovou paměť. U moderních architektur převažuje programová paměť FLASH. Někdy je doplněna ještě pamětí EEPROM menšího rozsahu pro zálohování důležitých dat. Jako paměť dat je využívána výhradně SRAM. Jako externí paměť, rozšiřující vnitřní paměť, je typická paměť rovněž FLASH pro program a SRAM pro data. Paměť DRAM je výjimečná. Pro intenzivní experimentální práci a časté ladění programů je výhodná architektura s vnější pamětí SRAM pro data i pro program. Vnitřní paměť FLASH obsahuje ladicí program, vnější SRAM pak aplikační programy. Ladící program umožňuje komunikaci s personálním počítačem, přesunování programů do paměti SRAM, spouštění programů, jejich krokování, apod. V personálním počítači běží druhá část ladicích programů.

Krajním případem jsou velmi rychlé počítače, zvláště signálové procesory. Program je uložen v rychlé vnitřní paměti SRAM. Tato paměť má nejkratší vybavovací dobu a její umístění na čipu minimalizuje zpoždění ve spojích.

7.2 POŽADAVKY NA DYNAMICKÉ PARAMETRY PAMĚTI

Rozbor časování signálů adresové, datové a řídicí sběrnice mikroprocesoru nebo vývodů sběrnic pro rozšíření jednočipového mikropočítače poskytuje informace pro výběr vhodných paměťových obvodů. Soulad dynamických parametrů paměti s parametry procesoru je **kritický** a jeho nedokonalé dodržení je příčinou nespolehlivé činnosti počítače. Připojení paměti na sběrnice může být velmi jednoduché, jak ukazují *obr. 2.1* a *obr. 2.2* V mnoha případech však je nutné zařadit dodatečné obvody – budiče sběrnice – mezi procesor a sběrnice a mezi

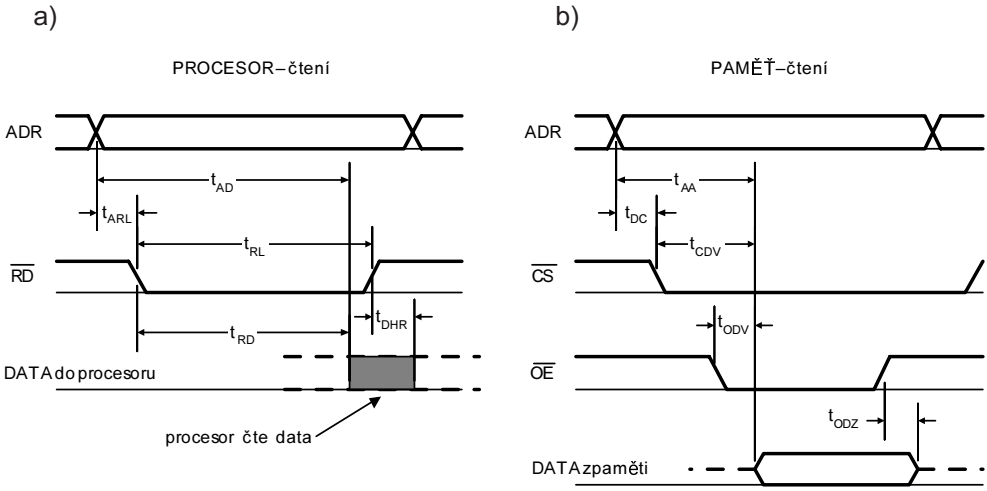
paměť a sběrnice. Je to nutné u rozsáhlejších systémů, kdy sběrnice představuje značnou kapacitní zátěž, na kterou nejsou procesor a paměťové obvody dimenzovány. Předpokládá se řízení paměti signály $\overline{RD}$ a $\overline{WR}$. Řídicí signály procesoru mohou být na tuto soustavu převáděny ve vnějších obvodech (systémový řadič), nebo přinejmenším výkonově posíleny obdobně jako adresové a datové signály. Výsledkem je sestava obvodů, znázorněná na *obr. 7.1.* Šipkami jsou vyznačena jejich jednotlivá zpoždění, která se při přesném rozboru časování musí brát v úvahu. Pokud u jednoduchých systémů budiče chybí, mohou se jejich zpoždění položit rovna nule. Pro vytváření výběrových signálů je podle *obr. 7.1* pro jednoduchost využit dekodér, i když výběrové obvody jsou často realizovány jinak – zde je podstatné jen jejich zpoždění.



Obr. 7.1 Zpoždění mezi procesorem a pamětí

Sběrnicový cyklus čtení:

Cesty signálů a jejich zpoždění jsou vyznačeny na *obr. 7.1*, časový diagram čtecího cyklu procesoru na *obr. 7.2a* a paměti na *obr. 7.2b*. Doba, po kterou musí být data na vstupu procesoru konstantní, je na *obr. 7.2a* zvýrazněna vyšrafováním.



Obr. 7.2 Časování cyklu čten: a) na straně procesoru; b) na straně paměti

Splnění podmínky (7.1) zajišťuje, aby data byla dodána na vstup procesoru včas, se zpožděním maximálně t_{AD} – cesta signálu vede z výstupů adresy na procesoru, přes budiče adresové sběrnice se zpožděním t_{BDA}, na adresové vstupy paměti; po uplynutí vybavovací doby paměti t_{AA} jsou data z paměti zpožděna v budičích datové sběrnice o t_{BDD} a dostanou se na datové vstupy procesoru. Na základě srovnání *obr. 7.2a* a *obr. 7.2b* musí platit:

$$t_{AD} \geq t_{BDA} + t_{AA} + t_{BDD}$$

Pro vybavovací dobu paměti t_{AA} pak platí:

$$t_{AA} \leq t_{AD} - t_{BDA} - t_{BDD} \qquad (7.1)$$

Splnění podmínky (7.2) zajišťuje, aby výběrovým signálem $\overline{CS}$ byla paměť aktivována tak, aby data byla dodána na vstup procesoru včas. Výběrové signály jsou generovány v adresovém dekodéru, který má zpoždění t_{DC}. Na vstupu $\overline{CS}$ paměti tedy dojde ke změně se zpožděním $t_{BDA} + t_{DC}$. Pro celou cestu od adresových výstupů procesoru až po jeho datové vstupy platí:

$$t_{AD} \geq t_{BDA} + t_{DC} + t_{CDV} + t_{BDD}$$

Pro dobu t_{CDV} paměti pak platí:

$$t_{CDV} \leq t_{AD} - t_{DC} - t_{BDA} - t_{BDD} \qquad (7.2)$$

Splnění podmínky (7.3) zajišťuje, aby po aktivaci třístavových výstupů paměti čtecím impulzem $\overline{OE}$ dodala paměť data na vstup procesoru včas. Čtecí impulz procesoru $\overline{RD}$, zpožděný v obvodech systémového řadiče o t_{SC}, je přiveden na vstup $\overline{OE}$ paměti. Po době t_{ODV} jsou data dodána na výstupy paměti a po zpoždění budičů datové sběrnice t_{BDD} se dostávají na datové vstupy procesoru. Platí vztah:

$$t_{RD} \geq t_{SC} + t_{ODV} + t_{BDD}$$

Pro dobu t_{ODV} paměti pak platí:

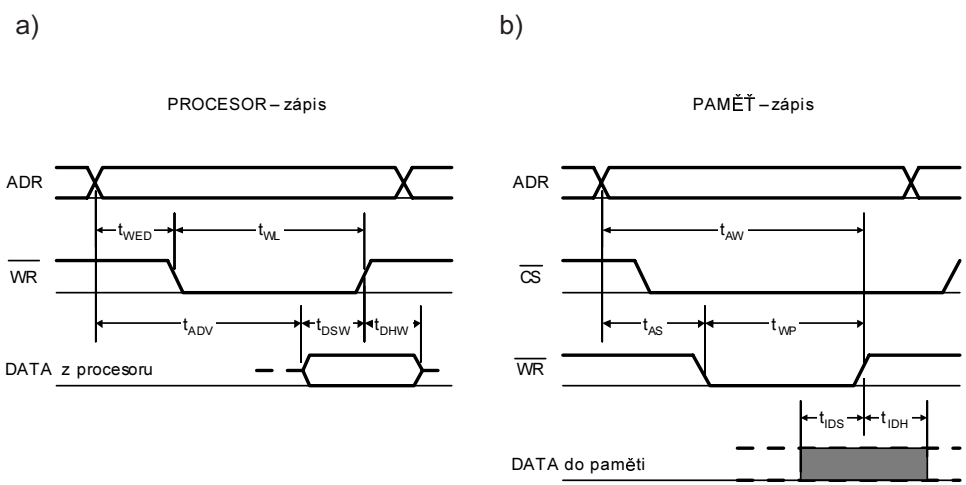
$$t_{ODV} \leq t_{RD} - t_{SC} - t_{BDD} \qquad (7.3)$$

Splnění podmínky (7.4) zajišťuje, aby data na vstupech procesoru trvala s dostatečně dlouhým přesahem po skončení čtecího impulzu $\overline{RD}$. Závěrná hrana $\overline{RD}$ je v systémovém řadiči zpožděna, a tak i čtecí impulz $\overline{OE}$ paměti končí později. Data z paměti, která existují ještě po dobu t_{ODZ}, jsou dále zpožděna v budičích datové sběrnice. Platí:

$$t_{DHR} \leq t_{SC} + t_{ODZ} + t_{BDD}$$

Pro dobu t_{ODZ} paměti pak platí:

$$t_{ODZ} \geq t_{DHR} - t_{SC} - t_{BDD} \qquad (7.4)$$



Obr. 7.3 Časování cyklu zápisu: a) na straně procesoru; b) na straně paměti

Sběrnicový cyklus zápisu:

Cesty signálů a jejich zpoždění jsou vyznačeny na *obr. 7.1*, časový diagram zápisového cyklu procesoru na *obr. 7.3a* a paměti na *obr. 7.3b*. Doba, po kterou musí být data na vstupu paměti konstantní je na *obr. 7.3b* zvýrazněna vyšrafováním.

Splnění podmínky (7.5) zajišťuje, aby adresa na vstupu paměti byla ustálena dostatečně dlouho před začátkem zápisového impulzu na vstupu $\overline{WR}$ paměti. Signál $\overline{WR}$ je procesorem generován s odstupem t_{WED} po změně adresy. Adresa na vstupu paměti je zpožděna o t_{BDA} a tím se odstup zkrátí. Signál $\overline{WR}$ je na cestě z procesoru do paměti zpožděn o t_{SC} a tím se naopak odstup prodlouží. Celkově tedy:

$$t_{AS} \leq t_{WED} + t_{SC} - t_{BDA} \qquad (7.5)$$

Splnění podmínky (7.6) zajišťuje, aby adresa na vstupu paměti trvala dostatečně dlouho před koncem zápisového impulzu na vstupu $\overline{WR}$ paměti. Na straně procesoru končí impulz v době $t_{WED} + t_{WL}$ po změně adresy. Obdobně jako v předchozím případě:

$$t_{AW} \leq t_{WED} + t_{WL} + t_{SC} - t_{BDA} \qquad (7.6)$$

Splnění podmínky (7.7) zajišťuje, aby data na vstupu paměti byla ustálena dostatečně dlouho před koncem zápisového impulzu na vstupu $\overline{WR}$ paměti. Na výstupu procesoru jsou data ustálena s předstihem t_{DSW} před koncem $\overline{WR}$. Na vstupu paměti je tento předstih zkrácen o t_{BDD}, ale současně i prodloužen o t_{SC}. Celkově tedy pro dobu předstihu platí:

$$t_{IDS} \leq t_{DSW} + t_{SC} - t_{BDD} \qquad (7.7)$$

Splnění podmínky (7.8) zajišťuje, aby data na vstupu paměti byla ustálena dostatečně dlouho ještě po konci zápisového impulzu na vstupu $\overline{WR}$ paměti. Na vstupu paměti je konec $\overline{WR}$ zpožděn o t_{SC} a tím je zde přesah dat t_{IDH} zkrácen. Data na vstupu paměti jsou zpožděna o t_{BDD} a to je přesah naopak prodloužen. Celkově tedy pro dobu přesahu platí:

$$t_{IDH} \leq t_{DHW} - t_{SC} + t_{BDD} \qquad (7.8)$$

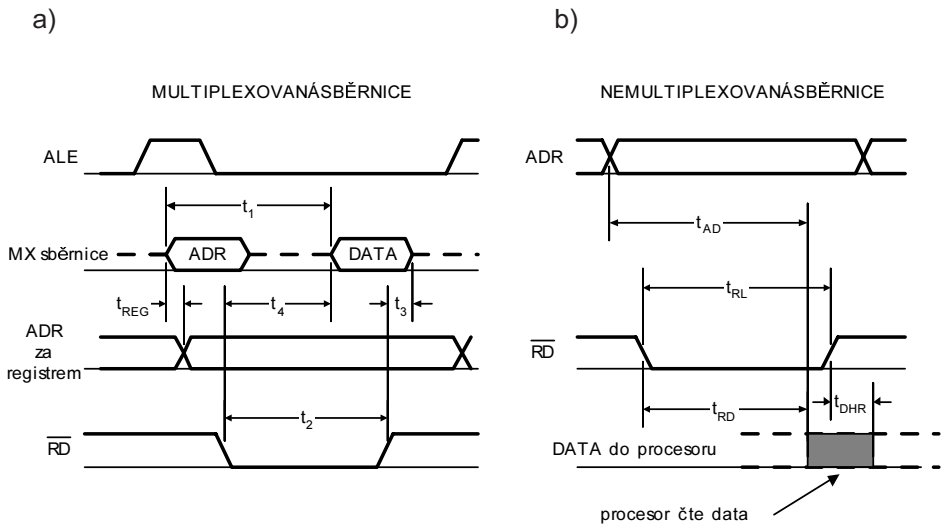
Požadavky na podstatné dynamické parametry paměti jsou shrnuty takto:

$$\begin{aligned}
&t_{AA} \leq t_{AD} - t_{BDA} - t_{BDD}\\
&t_{CDV} \leq t_{AD} - t_{DC} - t_{BDA} - t_{BDD}\\
&t_{ODV} \leq t_{RD} - t_{SC} - t_{BDD}\\
&t_{ODZ} \geq t_{DHR} - t_{SC} - t_{BDD}\\
&t_{AS} \leq t_{WED} + t_{SC} - t_{BDA}\\
&t_{AW} \leq t_{WED} + t_{WL} + t_{SC} - t_{BDA}\\
&t_{IDS} \leq t_{DSW} + t_{SC} - t_{BDD}\\
&t_{IDH} \leq t_{DHW} - t_{SC} + t_{BDD}
\end{aligned}$$

Je zřejmé, že **zpoždění dodatečných obvodů** – budičů sběrnice a obvodů pro úpravu řídicích signálů – ve většině případů **zvyšuje nároky** na dynamické parametry paměti.

Časování čtecího cyklu u **multiplexní sběrnice** ukazuje *obr. 7.4a* ve srovnání s časováním u nemultiplexní sběrnice. Přepočet časů, tak jak jsou běžně definovány, je jednoduchý. Ze srovnání *obr. 7.4a* a *obr. 7.4b* vyplývá:

$$t_{AD} = t_1 - t_{REG}$$
$$t_{RD} = t_4$$
$$t_{RL} = t_2$$
$$t_{DHR} = t_3$$

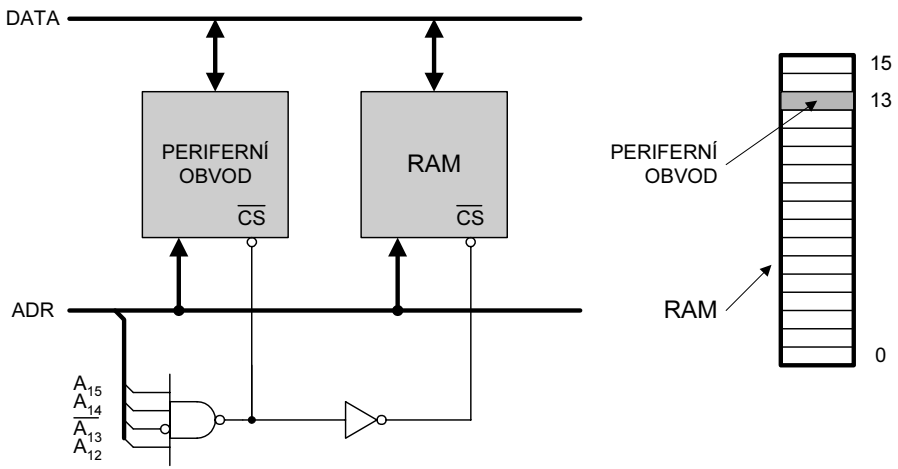


Obr. 7.4 *Přepočet dob z multiplexní sběrnice ne nemultiplexní sběrnici: a) definice dob u multiplexní sběrnice; b) definice dob u nemultiplexní sběrnice*

Doba t_{REG} udává zpoždění od vstupu do výstupu adresového registru (viz též odstavec 5.2. a *obr. 5.9*. Zpravidla se používá registr řízený hladinou hodinového impulzu ALE. Při časování podle *obr. 7.4a*, kdy ALE = 1 již před okamžikem výdeje nové adresy na výstupu procesoru, mohou adresové signály volně procházet registrem a jejich změna se jen s malým zpožděním t_{REG} objeví na výstupu registru. Po ukončení impulzu ALE se již stav na výstupu registru nemůže změnit až do dalšího impulzu ALE.

7.3 PŘEKRÝVÁNÍ PAMĚTI

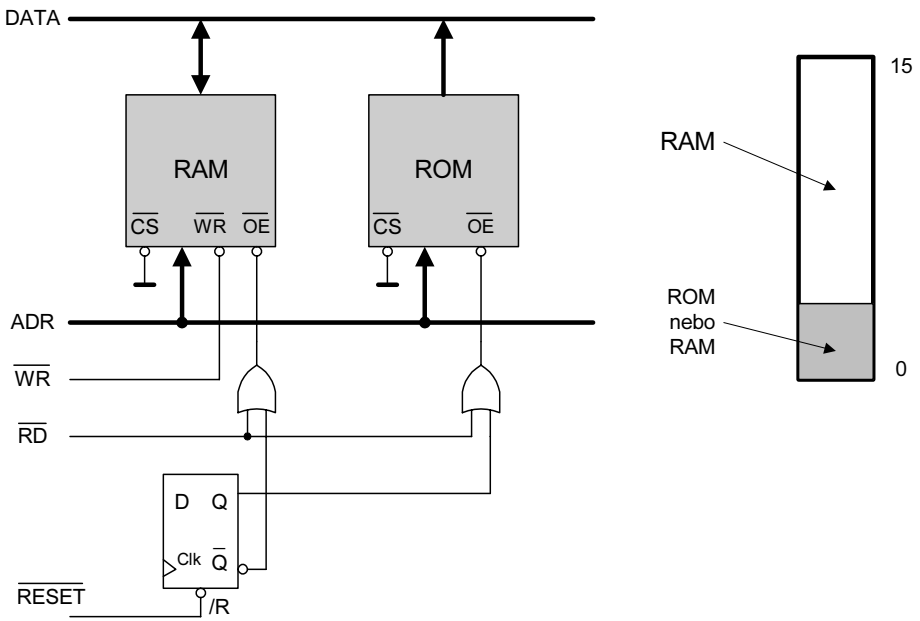
Velmi jednoduchý způsob generace výběrových signálů ukazuje *obr. 7.5*. Téměř celý adresový prostor zaujímá paměť RAM, malou část zabírá soustava periferních obvodů. V této pozici je paměť nedostupná, i když fyzicky existuje – je „překryta“ jinými obvody. Cena za jeden bit paměti klesá s rostoucí hustotou integrace a proto je výhodnější použít obvod co nejvyšší kapacity, i když je částečně nevyužit. Pro generaci výběrových signálů jsou na *obr. 7.5* využity 4 nejvyšší bity adresy, proto adresový prostor je dělen na $2^4 = 16$ shodných úseků. Při stavech těchto bitů $1101_B = 13_D$ jsou aktivovány periferní obvody, při ostatních RAM.



Obr. 7.5 Překrytí paměti jiným obvodem

Jinou variantu překrytí ukazuje *obr. 7.6*. Podle stavu klopného obvodu je čtení možné buď z ROM (při Q = 0), nebo z RAM (při Q = 1). Zápis do RAM je možný vždy. Po vynulování počítače je vždy Q = 0 a proto se čte z ROM. V ní je uložen program, potřebný pro rozběh počítače. Jeho činností je přepis obsahu celé ROM na stejné adresy do RAM. Pak je klopný obvod programově nastaven do stavu 1 a další čtení budou již z RAM. Program je tedy od tohoto okamžiku čten z RAM, což je jednak rychlejší, jednak program může sám sebe modifikovat. Klopný obvod může být součástí některého řídicího registru počítače, nebo u jednočipových mikropočítačů to může být jeden bit výstupní brány. Toto zapojení paměti se nazývá „stínová ROM“ (angl. Shadow ROM) a je běžné např. u personálních počítačů.

426410

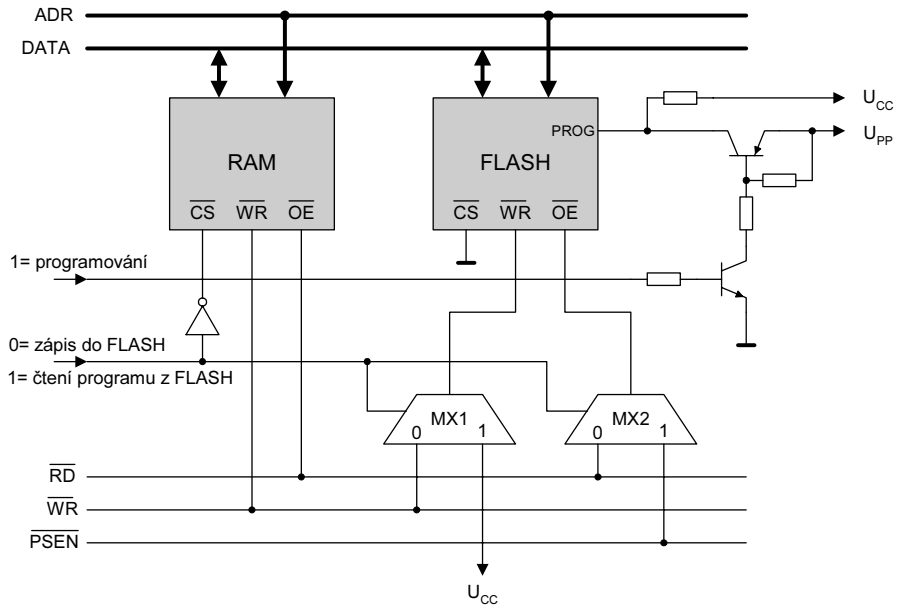


Obr. 7.6 Řízené překrytí ROM a RAM

7.4 PROGRAMOVÁNÍ PAMĚTI FLASH V SYSTÉMU

Paměť FLASH může být jak vnitřní součástí jednočipového mikropočítače, tak i dodatečnou vnější jednotkou, připojenou na sběrnice. Tento případ ukazuje *obr. 7.7*, kde vnější obvody zahrnují jak programovou, tak i datovou paměť. Předpokládá se zde jednočipový mikropočítač s odděleným programovým a datovým prostorem a soustavou řídicích signálů $\overline{PSEN}$, $\overline{RD}$, $\overline{WR}$. Pro jiné architektury a jiné soustavy řídicích signálů by bylo nutné zapojení pozměnit.

Ve stavu „zápis do FLASH“ je datová paměť RAM vyřazena. Oba přepínače (MX1, MX2) jsou přepnuty do pozice „0“ a tím se do FLASH dostává čtecí i zápisový impulz. S FLASH se pak zachází jako s RAM – lze ji mazat, programovat a kontrolovat. Ve stavu „čtení programu z FLASH“ se datová paměť aktivuje. Na vstupu $\overline{WR}$ paměti FLASH je stav „1“ (povoleno čtení) a na vstup $\overline{OE}$ je přiváděn řídicí signál $\overline{PSEN}$, kterým je vyvoláváno čtení programu. Během mazání a programování je třeba na vývod U_{PP} přivést programovací napětí (kolem 12 V), během čtení programu vyhovuje napětí o velikosti U_{CC}. Spínač U_{PP} musí během programování sepnout proud několika desítek mA při malém napěťovém úbytku (tolerance U_{PP} jsou řádově jen desetiny V). K ovládání multiplexerů i spínače U_{PP} lze využít např. dva bity výstupní brány jednočipového mikropočítače.



Obr. 7.7 Příklad připojení paměti FLASH

Algoritmus pro mazání a programování paměti FLASH se částečně liší typ od typu – závazný je předpis výrobce. Velmi hrubě lze postup znázornit takto:

Mazání:

1 Zvýšit napětí na vývodu U_{PP}.

2 Zapsat příkaz „*erase1*“.

3 Zapsat příkaz „*erase2*“.

4 Odpočítat čas na mazání (10 ms).

5 Zapsat příkaz „*erase-verify*“.

6 Odpočítat čas na prodlevu (6 µs).

7 Čtení: je obsah FF_H?

Pokud není FF_H, opakovat mazání od kroku 2.

Pokud je FF_H, zvýšit adresu a pokračovat v kontrole krokem 5.

Pokud je adresa poslední, pokračovat krokem 8.

8 Zapsat příkaz „*reset*“.

9 Snížit napětí na vývodu U_{PP}.

Dva příkazy „*erase*“ bezprostředně po sobě jsou pojistkou proti náhodnému nechtěnému vymazání celého obsahu paměti.

Programování:

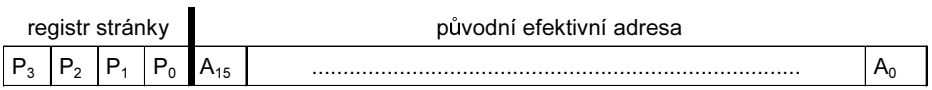
1 Zvýšit napětí na vývodu U_{PP}.
2 Zapsat příkaz „*program*“
3 Zapsat data na příslušnou adresu.
4 Odpočítat čas na programování na této adrese (10 μs).
5 Zapsat příkaz „*program-verify*“.
6 Odpočítat čas na prodlevu (6 μs).
7 Přečíst a zkontrolovat data.
Pokud data nesouhlasí, opakovat se stejnou adresou od kroku 2.
Pokud data souhlasí, zvýšit adresu a pokračovat krokem 2.
Pokud je adresa poslední, pokračovat krokem 8.
8 Zapsat příkaz „*reset*“.
9 Snížit napětí na vývodu U_{PP}.

U pamětí o velké kapacitě je běžné jejich rozdělení na **několik sektorů**. Se sektory lze pracovat jako s nezávislými celky, takže malá změna obsahu paměti zasáhne třeba jen jeden sektor – tím se urychlí mazání a programování. Sektor je určen nejvyššími bity adresy. Je běžné, že jeden sektor, tzv. „**boot sector**“ má zvláštní postavení a jeho mazání je zvláště důkladně chráněno. Slouží k uložení základních programů počítače, včetně programovacího algoritmu. Za normálních okolností není tento sektor mazán, může ale řídit mazání a programování ostatních sektorů.

Obr. 7.7 ukazoval zapojení vnější paměti FLASH do počítače. V případě jednočipového mikropočítače je paměť sice integrována na čipu, postup při jejím mazání a programování je však obdobný postupu pro samostatnou paměť. Příkazy jsou zapisovány do vnitřních řídicích registrů. Uspořádání s „boot sektorem“ je běžné.

7.5 ROZŠÍŘENÍ PAMĚŤOVÉHO PROSTORU

Některé úlohy vyžadují větší rozsah paměti, než odpovídá počtu bitů adresové části instrukce daného procesoru. Rozšíření je možné doplněním hardware procesoru o další prostředky, které generují nejvyšší (přidané) bity adresy. Nejčastější metodou je **stránkování**. K efektivní adrese, vydávané procesorem, se zleva připojí další bity z **registru stránky**. Paměť je tak členěna na jednotlivé **stránky** (angl. „page“). V dokumentaci výrobců se používá i název **segment** nebo **banka**. *Obr. 7.8* ukazuje příklad, kdy původní 16bitová adresa se rozšíří na 20bitovou pomocí 4bitového registru.



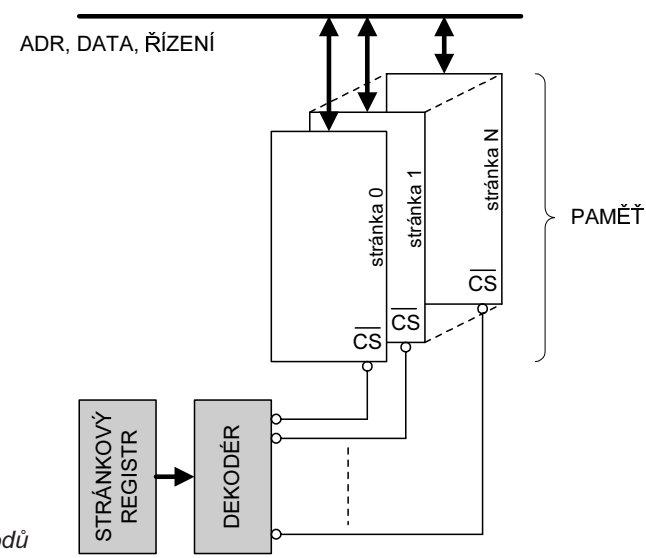
Obr. 7.8 *Zvýšení počtu bitů adresy stránkováním*

U dokonalejších architektur jsou stránkové registry součástí procesoru. Je jich vždy několik. Existuje zvláštní stránkový registr pro program – ten rozšiřuje počet bitů čítače instrukcí. Obdobně může být vyhrazen jeden registr pro zásobníkové operace a jeden pro práci s datovou pamětí. Podle konkrétní architektury jsou stránkové registry dostupné buď speciálními instrukcemi, nebo u jednočipových mikropočítačů mohou být též součástí pole speciálních funkčních registrů SFR a pracuje se pak s nimi jako s datovou pamětí.

Práce s datovou pamětí nepředstavuje problém. Po změně obsahu datového stránkového registru lze kdykoliv použít instrukci přesunu z paměti nebo do paměti – její adresová část bude ukazovat do nové stránky. Jiná situace je u programové paměti. Při postupném čtení instrukcí se časem narazí na konec stránky, tj. na maximální číslo v čítači instrukcí. Při další instrukci se pak obsah čítače změní z maximálního čísla na nulu. U procesorů, vybavených stránkovými registry, je v tom případě **automaticky inkrementován** programový stránkový registr a program pokračuje na další stránce od její nulté adresy.

Stránkování musí umožňovat i skoky a volání podprogramů přes hranice stránky. U procesorů, vybavených stránkovými registry, k tomu slouží speciální instrukce **mezistránkového skoku** a **mezistránkového volání**. Tyto instrukce obsahují dva operandy – adresu v rámci stránky (jako u běžného skoku nebo volání) a navíc i číslo stránky. Toto číslo se přesune do programového stránkového registru. U instrukce mezistránkového volání podprogramu se na rozdíl od běžné instrukce volání automaticky ukládá do zásobníku nejen adresa návratu, ale navíc i původní obsah programového stránkového registru. Speciální instrukce existuje též pro návrat z mezistránkového volání. Na rozdíl od běžné instrukce návratu se v ní obnoví i obsah programového stránkového registru.

Stránkování může být zavedeno i dodatečně pomocí **vnějších obvodů**, jak ukazuje *obr. 7.9*.



Obr. 7.9 *Stránkování s pomocí vnějších obvodů*

426410

Jako stránkový registr může být využita některá z výstupních bran jednočipového mikropočítače. V případě, že počet stránek není větší než 8, mohou být výběrové signály v kódu „1z8“ generovány přímo a odpadne tak i dekodér. Adresace datové paměti je bez problémů – pro změnu stránky stačí patřičně změnit výběrové signály. Jinak je tomu se stránkováním programové paměti. Zde totiž nemáme k dispozici speciální instrukce pro mezistránkové skoky a volání, a ani přechod z jedné stránky na druhou při postupném čtení programu není zajištěn. Pro skok a volání není možné jednoduše za sebe zařadit instrukci pro přepsání stránkového registru a instrukci skoku (volání). Po změně výběrových signálů program pokračuje na nové stránce a instrukce skoku (volání), která byla na právě opuštěné stránce, se tudíž neuplatní.

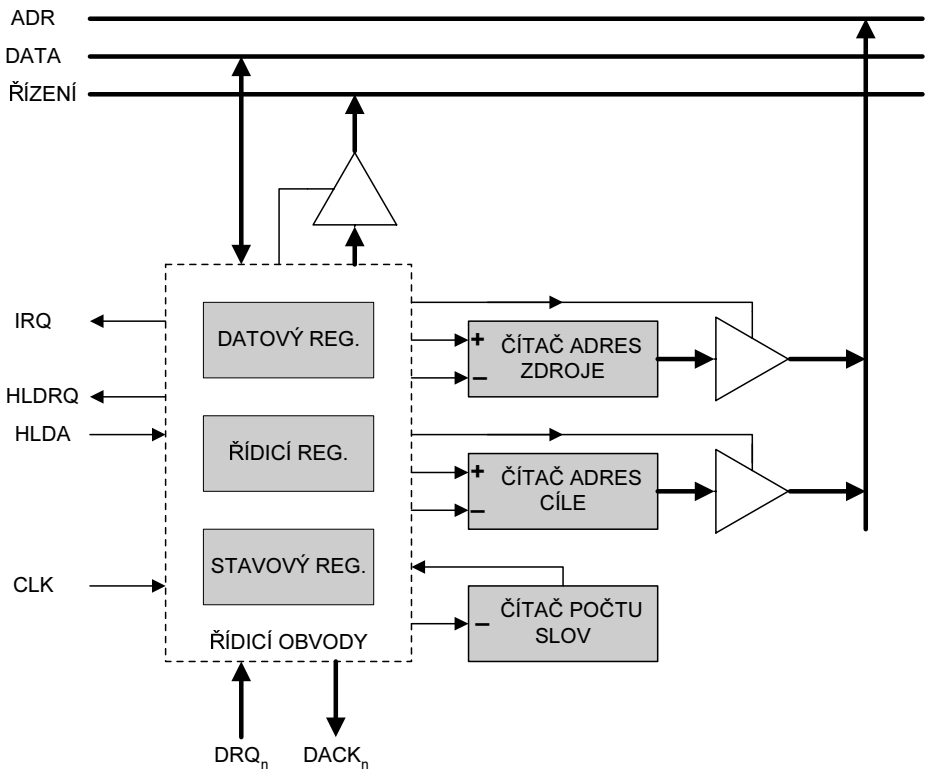
Řešení lze nalézt v software. Na všech stránkách, mezi kterými mají být možné skoky a volání, se na stejných adresách uloží stejné, krátké programové úseky – „přepínací zóna“. Číslo stránky se předem uloží do paměti nebo do pracovního registru, obdobně se uloží i požadovaná adresa v nové stránce. Pak se provede skok na začátek přepínací zóny. Zde se číslo stránky zapíše do stránkového registru (výstupní brány). Program nyní pokračuje na nové stránce. V přepínací zóně se pokračuje instrukcí skoku na nepřímou adresu, která byla uložena předem v paměti nebo v pracovním registru. Nemá-li procesor skokové instrukce s nepřímou adresou, pomůže malý trik: požadovaná adresa se předem uloží instrukcí PUSH do zásobníku a po přepnutí stránky se v přepínací zóně provede instrukce RET. Program tak pokračuje na „adrese návratu“, předem uložené v zásobníku. Dodatečně zavedené stránkování je tedy s malými komplikacemi použitelné.

7.6 KANÁL PŘÍMÉHO PŘÍSTUPU DO PAMĚTI

Kanál přímého přístupu do paměti DMA (angl. Direct Memory Access) umožňuje zrychlené přesuny bloku dat mezi dvěma oblastmi datové paměti nebo mezi datovou pamětí a periferními obvody. Přenosy se přitom dějí bez účasti procesoru. Základní uspořádání řadiče kanálu DMA ukazuje *obr. 7.10*.

Čítač adres zdroje realizuje funkci ukazatele do zdrojové oblasti dat, tj. do oblasti datové paměti, odkud jsou data vybírána. Čítač adres cíle realizuje funkci ukazatele do cílové oblasti dat, tj. do oblasti datové paměti, kam jsou data přesunuta. Čítač počtu přenesených slov počítá dolů a informuje řídicí obvody o dosažení požadovaného počtu přenosů. Všechny čítače jsou přednastavitelné z datové sběrnice.

Pokud byl řadič předem naprogramován prostřednictvím řídicího registru a je připraven, reaguje na požadavek periferního zařízení na přenos DRQ_n (DMA Request) tak, že vyšle do procesoru požadavek na uvolnění sběrnic (v *obr. 7.10* označen jako HLDRQ – Hold Request). Procesor reaguje tak, že dokončí současný sběrnicový cyklus a již nezačíná nový. Vnitřní operace v procesoru ale



Obr. 7.10 Řadič kanálu DMA

mohou pokračovat, dokud nepotřebují přístup na sběrnice – v tom případě se procesor automaticky zastaví. Procesor dále uvede výstupy adresové, datové a řídicí sběrnice do vysokoimpedančního stavu a tento stav potvrdí (v *obr. 7.10* signálem označeným jako HLDA – Hold Acknowledge).

Na základě potvrzení pak řadič kanálu převezme ovládání sběrnic a samostatně s nimi pracuje. Na adresovou sběrnici připojuje čítač adres zdroje nebo cíle a generuje tak adresy. Svými řídicími obvody generuje i řídicí signály a tím přesunuje data mezi obvody, připojenými na sběrnice. Může se jednat o paměť i o datové registry periferních obvodů. Nejprve je proveden sběrnicový cyklus čtení, adresa je získána z čítače adres zdroje. Data jsou dočasně uložena v datovém registru řadiče. Pak je proveden cyklus zápisu, adresa je získána z čítače adres cíle. Přesné časování sběrnicových cyklů je zajištěno synchronizačními impulzy na vstupu CLK řadiče. Po každém přeneseném slovu jsou čítače adres inkrementovány či dekrementovány, takže lze přenést celý blok dat „zdola nahoru“ nebo „shora dolů“. Stav 0 v čítači slov způsobí ukončení přenosu bloku. Řadič uvolní sběrnice a vynuluje požadavek HLDRQ, procesor následně převezme sběrnice a vynuluje potvrzení HLDA. Tím je obnoven normální běh programu.

Ukončení přenosu bloku je procesoru signalizováno požadavkem na přerušení (IRQ). Vyrozumění je potřebné, neboť přenosy v režimu DMA se dějí mimo procesor a v náhodném místě programu, který o proběhlém přenosu „neví“. Přerušením je možné ihned vyvolat podprogram, který zpracuje právě zaplněnou oblast paměti, apod. Informace o stavu kanálu DMA jsou k dispozici i v jeho stavovém registru. Řídicí registr řadiče kanálu umožňuje činnost kanálu zablokovat a opět povolit. To je zapotřebí např. v programu, který periodicky zpracovává pole dat v paměti, přičemž všechna data se musí vztahovat ke stejnému okamžiku. Pokud již program začal toto pole zpracovávat, pak jeho přepsání případným blokovým přenosem DMA kanálu musí být odloženo až do konce jeho zpracování.

Řadiče umožňují volit způsob reakce na požadavek na přenos (DRQ). Jsou možné následující režimy:

- **Blokový**. Hranou DRQ se zastaví procesor a je spuštěn přenos celého bloku, který nelze přerušit. Skončí dopočítáním čítače slov do nuly. Procesor je odpojen po celou dobu.
- **Požadavkový**. Stavem 1 na vstupu DRQ se zastaví procesor, je spuštěn přenos a postupně se přenáší slovo za slovem. Stavem 0 se přenos přeruší a procesor se opět spustí. Všechny čítače v řadiči se zastaví. Stavem 1 na vstupu DRQ se přenos opět obnoví, čítače pokračují od stavu při zastavení. To lze opakovat, dokud čítač slov nedosáhne nuly. Pak je přenos definitivně zastaven.
- **Přerušovaný**. S každou hranou DRQ se přenese jen jedno slovo a čítače se inkrementují (dekrementují). S další hranou se přenese další slovo. Skončí se dopočítáním čítače slov do nuly.

V závislosti na naprogramování řadiče může přenos probíhat mezi dvěma oblastmi paměti, z paměti do periferních obvodů nebo z periferních obvodů do paměti. Je-li zdrojem a cílem dat paměť, jsou inkrementovány či dekrementovány oba čítače adres a jedná se typicky o přenos bloku dat. Procesory s dokonalejší architekturou jsou však již samy vybaveny instrukcemi, vhodnými pro rychlý přenos bloku a tento režim činnosti DMA kanálu u nich nepřináší významnou výhodu.

Při přenosu **z paměti do periferního obvodu** se jedná typicky o přerušovaný režim. Čítač adresy cíle přitom nemění stav, takže data jsou opakovaně směrována do stejného obvodu. Čítač adresy zdroje (tj. ukazatel do paměti) může buď postupovat s každým přeneseným slovem, nebo může stát. V prvém případě to znamená postupný přesun bloku dat do jednoho periferního obvodu – např. celá zpráva, předem připravená v paměti, se postupně vysílá sériovým vysílačem. Ve druhém případě se přenáší obsah stále jedné adresy v paměti do periferního obvodu a je záležitostí programu, aby obsah paměti včas doplňoval. Při přenosu **z periferního obvodu do paměti** naopak čítač adresy zdroje nemění stav a čítač adresy cíle může buď postupovat s každým přeneseným slovem, nebo může stát.

Důležitou vlastností řadiče kanálu DMA je možnost **opakování přenosu**. Bez této možnosti přenos končí při dopočítání čítače přenesených slov do nuly a pro

další činnost musí být všechny čítače znovu přednastaveny. Řadiče s možností opakování přenosu jsou vybaveny registry pro přednastavení čítačů. Při počátečním programování kanálu jsou data pro přednastavení čítačů současně uložena i do těchto registrů, odkud jsou pak vždy při dopočítání počtu slov do nuly automaticky přepsána do čítačů.

Pokud je DMA u počítače použit, je zpravidla kanálů několik. Každý kanál má svoje čítače a řídicí registr. Každý kanál má vstup požadavku na přenos DRQ_n a výstup $DACK_n$, kterým řadič signalizuje ukončení přenosu. Činnost jednotlivých kanálů je vzájemně nezávislá, v každém okamžiku však může pracovat se sběrnicemi jen jeden z nich. Proto jsou požadavky na přenos (DRQ_n) vyhodnocovány **podle priority**, podobně jako je tomu u požadavků na přerušení.

Zvláště účinnou metodou ke zrychlení přenosu mezi periferními obvody a pamětí je **integrace DMA** do jednočipového mikropočítače. Přístupnost vnitřních signálů a společných synchronizačních impulzů umožňuje zkrátit dobu na přenos jednoho slova až na jeden vnitřní sběrnicový cyklus. U některých architektur lze volit, zda výstupy klopných obvodů IRR, zachycujících události v periferních obvodech, budou zavedeny buď do řadiče přerušení, nebo do řadiče DMA. Přenosy se dějí po jednotlivých slovech. Na rozdíl od obsluhy periferních obvodů pomocí přerušení, kdy latence může mít významný negativní vliv, probíhá v režimu DMA přenos dat mezi pamětí a periferním obvodem **zcela bez zásahu programu**. Není nijak omezen ani v již běžícím přerušení, není zapotřebí ukládat návratovou adresu, ani ukládat a pak navracet obsah pracovních registrů. Řadič DMA zastupuje činnost řadiče přerušení i v tom, že současně s přenosem vynuluje záchytný klopný obvod IRR v tom periferním obvodu, který vyvolal přenos. Řadič též může generovat žádost o přerušení v případě dopočítání čítače slov do nuly při přenosu bloku. Obsluha periferních obvodů se tak velmi zrychlí a zautomatizuje.

426410

8 ČÍTAČE A ČASOVAČE

Čítače a časovače patří mezi nejčastější a nejrozsáhlejší součásti počítačů, určených pro práci v reálném čase. Slouží ke zpracování časových funkcí, ke generování signálů s přesným časováním, k počítání vnějších událostí, k synchronizaci programu s vnějšími událostmi, atd. U velmi jednoduchých systémů, se zvláštním důrazem na nízkou cenu a s malými nároky na výpočetní výkon, se uvedené úlohy realizují zčásti programově. Rozsah speciálních obvodů je pak sice malý, ale zápornou stránkou je velmi podstatné snížení výkonnosti počítače, neboť procesor je po většinu času vázán na zpracování časových funkcí.

Tam, kde je zapotřebí plně využít výpočetní výkon procesoru pro řízení procesů, jsou speciální časovací obvody nezbytné. I zde je rozdíl mezi jejich řešením u počítače vícečipového a u počítače jednočipového (mikrokontroléru). V prvém případě jsou časovací obvody vnější, připojené na datovou sběrnici procesoru. Jejich vzájemné vazby a činnost jsou určeny částečně naprogramováním, částečně též propojením vnějšími spoji. V ojedinělých případech se může ukázat jako výhodné navrhnout obvody zcela „šité na míru“ pro danou aplikaci, s využitím běžných obvodů MSI jako jsou čítače, monostabilní klopné obvody, generátory impulzů atd. Lze tak zjednodušit a zkrátit programovou obsluhu časovacích obvodů.

Ve druhém případě, u univerzálního mikrokontroléru, lze samozřejmě také použít časovací obvody jako obvody externí, připojené na vnější sběrnice. Tím se však zvyšuje počet integrovaných obvodů, což je proti filosofii jednočipového systému. Výrobci mikrokontrolérů proto věnují systému vnitřních čítačů a časovačů velkou pozornost. Pro dosažení vysoké universality je celý systém programovatelný, s mnoha možnostmi vzájemných vazeb a režimů činnosti. K jeho ovládání slouží systém řídicích registrů. Konkrétní význam jejich jednotlivých bitů nebo skupin bitů je uveden ve firemní dokumentaci a typ od typu se výrazně liší. V zásadě platí, že režim a vzájemné vazby se během činnosti programu nemění a nastavují se na jeho začátku, v inicializační části. Naopak s některými dalšími řídicími bity se pracuje průběžně – např. povolování a blokování čítání.

Je-li nutná spolupráce s procesorem, děje se prostřednictvím přerušení programu. Systém čítačů a časovačů je rozsáhlý a tomu odpovídá i velký počet požadavků na přerušení. Jelikož požadavky jsou vždy impulzní, jsou zachyceny v klopných obvodech. Teprve výstupní signály těchto klopných obvodů vyvolávají přerušení. Přerušovací klopný obvod lze programově číst a lze jej též programově překlápět. U některých mikrokontrolérů je každý požadavek zaveden do přerušovacích obvodů. U jiných jsou jednotlivé požadavky seskupeny (logicky

sečteny) a z celé jednotky čítačů a časovačů vychází jen jeden sumární požadavek. Přerušovací obvody jsou jednodušší, ale obsluha přerušení se prodlužuje o identifikaci obvodu, který požadavek na přerušení vyvolal. Při identifikaci jsou v obslužném podprogramu čteny stavy požadavkových klopných obvodů. Další činnost v obslužném podprogramu se pak větví podle toho, který klopný obvod je v aktivním stavu (stav 1).

Čítače jsou zpravidla 16bitové, u jednoduchých mikrokontrolérů i 8bitové. Obvody čítačů a časovačů, o kterých pojednává další text, jsou jistým zobecněním obvodů a možností, které lze nalézt u moderních mikrokontrolérů. U jednotlivých konkrétních typů se nevyskytují všechny tyto možnosti současně, u jednodušších a starších typů jsou dokonce značně omezené.

8.1 UNIVERZÁLNÍ ČÍTAČ/ČASOVAČ

Pod názvem univerzální čítač/časovač (GPT – z angl. General Purpose Timer) je míněno takové zapojení čítače a pomocných obvodů, které umožňuje činnost v režimech čítání impulzů, časování, a generace periodických impulzů. Každý z těchto režimů vyžaduje jiné vnitřní vazby, případně i jiné pomocné obvody. K jejich nastavení a ovládání slouží řídicí registry. Mnohé aplikace mikrokontrolérů jsou založeny na měření kmitočtu, periody, fáze apod. Tato měření lze snadno realizovat seskupením dvou nebo více čítačů a časovačů, opět s jejich patřičnými vazbami. V běžném mikrokontroléru je proto vždy několik univerzálních čítačů a časovačů s možností jak nezávislé činnosti, tak činnosti ve skupině. Nejjednodušším seskupením je spojení dvou čítačů do kaskády pro práci s velmi dlouhými časy, nebo pro čítání velkých počtů impulzů. Tyto jednoduché případy nebudou dále rozváděny.

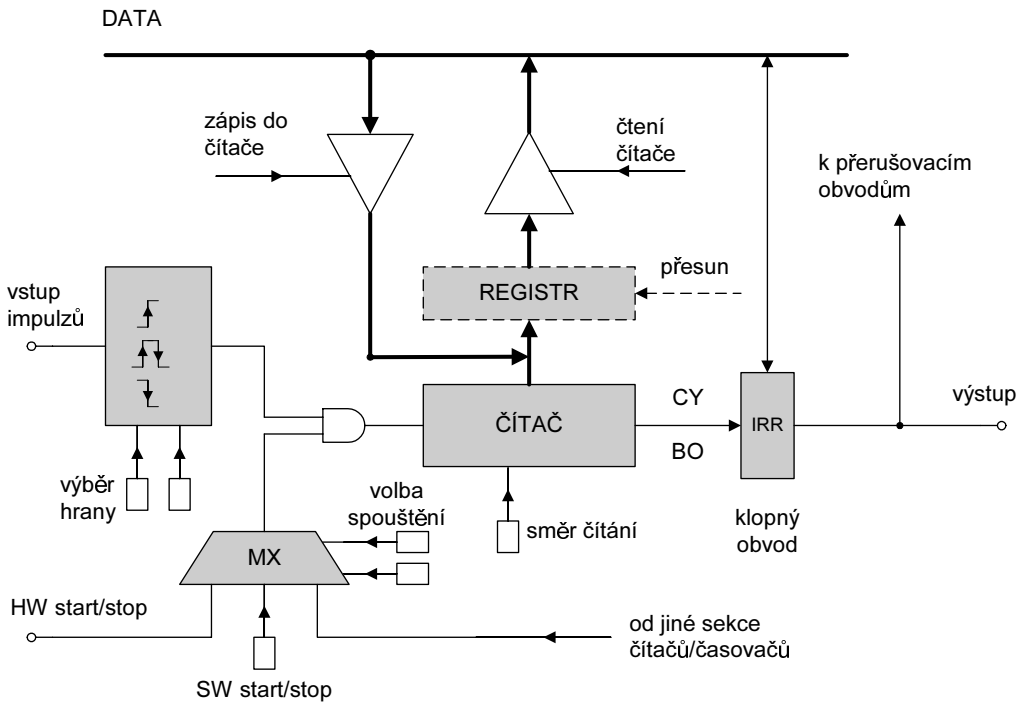
Pomocné obvody a programovatelné vnitřní vazby umožňují **autonomní činnost** GPT a generaci signálů na výstupních branách mikrokontroléru bez nutnosti časté programové obsluhy. Nezávislost na programové obsluze podstatně zpřesňuje zpracování časových funkcí.

8.1.1 Režim čítání impulzů

Nejjednodušší je režim prostého čítání impulzů podle *obr. 8.1*. Na vstup čítače jsou přiváděny impulzy z vnějšího zařízení. Stav čítače lze programově číst, případně do něj zapsat počáteční hodnotu. V cestě vstupních impulzů je detektor hrany. **Aktivní hranu**, na níž čítač mění stav, lze většinou volit jako hranu buď náběžnou, nebo doběžnou, případně i obě. Volitelný je často i směr čítání nahoru nebo dolů. V tomto i v dalších obrázcích značí symbol □ → jeden bit v některém z řídicích registrů jednotky čítačů/časovačů.

Začátek a konec čítání je ovládán buď vnějším signálem, nebo programově, nebo vnitřním signálem od jiné sekce čítačů/časovačů. Multiplexerem MX je volena jedna z těchto možností. Přenosem z nejvyššího bitu čítače je překlápěn

výstupní klopný obvod. Při počítání nahoru se překlápí impulzem kladného přenosu (CY) při přechodu ze stavu čítače 1...11 do 0...00, při počítání dolů se překlápí impulzem záporného přenosu (BO) při přechodu z 0...00 do 1...11. Spojem klopného obvodu IRR s datovou sběrnicí je na *obr. 8.1* symbolicky znázorněna možnost čtení stavu KO IRR i jeho programového ovládání. Výstup klopného obvodu je u externích čítačů/časovačů vyveden z pouzdra. U mikrokontrolérů je zpravidla zdrojem požadavku na přerušení, může však též být vyveden z pouzdra, nebo může být zaveden do další čítačové sekce.

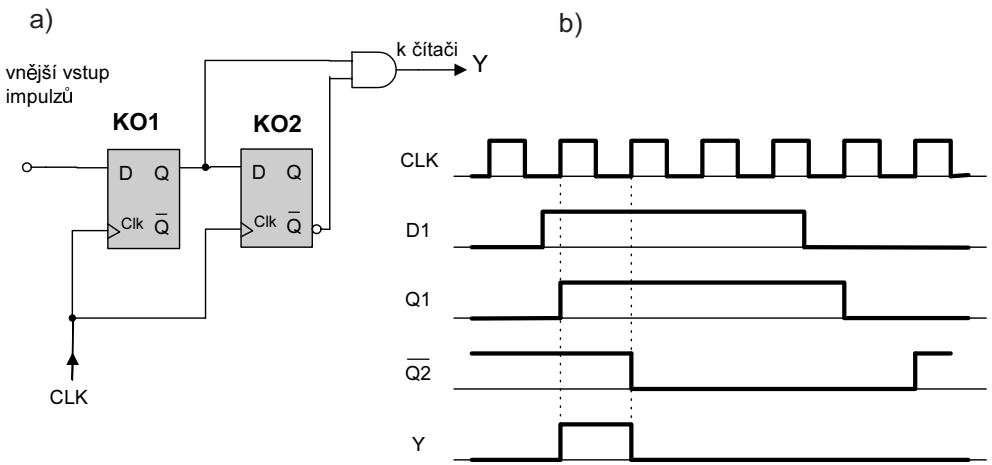


Obr. 8.1 Čítač/časovač v režimu čítání impulzů

Obsah čítače nelze číst kdykoliv. Pokud by došlo ke čtení právě v okamžiku, kdy se mění jeho stav, mohla by být přečtená informace nesprávná. U čítačů v mikrokontroléru se tento problém řeší synchronizací vstupního impulzu s vnitřním strojovým cyklem, takže ke změně stavu dojde vždy mimo čtecí impulz (viz též *obr. 5.5*). K synchronizaci jsou využity vnitřní hodinové impulzy. Princip **synchronního detektoru náběžné hrany** ukazuje *obr. 8.2*.

Detektor se skládá ze dvou klopných obvodů, tvořících posuvný registr. Vytvoření výstupního impulzu jako odezvu obvodu na vstupní impulz ilustruje časový diagram. Je zřejmé, že stav (0 nebo 1) na vstupu D prvého klopného obvodu musí trvat alespoň jeden takt hodinových impulzů CLK, aby byl vždy registrován

(na *obr. 8.2* jsou to takty dva). Perioda vstupních impulzů se tak rovná alespoň dvěma periodám CLK. Hodinové impulzy jsou generovány jednou za strojový cyklus procesoru. Tím lze vysvětlit skutečnost, že ve firemních údajích jsou maximální kmitočty čítaných impulzů překvapivě nízké, ačkoliv se třeba jedná o moderní a rychlé mikrokontroléry.



Obr. 8.2 Synchronní detektor hrany

U **externích čítačů** není takováto synchronizace možná. Využívá se pak pomocný registr na výstupu čítače (v *obr. 8.1* vyznačen čárkovaně), do kterého je obsah čítače přepsán – procesor tedy čte obsah registru, nikoliv obsah čítače. Přepis je vyvolán řídicími obvody jako důsledek předchozího povelu, zapsaného do řídicího registru. Pokud se během přepisu vyskytne vstupní impulz do čítače, je pozdržen. Jinou metodou je programové zastavení čítání před čtením čítače a jeho opětné povolení po přečtení. Pomocný registr sice není zapotřebí, po dobu blokování čítače však může dojít ke ztrátě počítaných impulzů.

Typické využití čítače v programech:

Častou aplikací je např. odpočítávání impulzů z inkrementálního čidla polohy u nějakého mechanizmu. Je požadováno přesunutí o přesnou vzdálenost, danou počtem impulzů z čidla. Předpokládá se, že v inicializační části programu byl nastaven režim čítače/časovače na čítání vnějších impulzů, směr čítání dolů a aktivní hrana náběžná. Obsluha čítače bude spočívat v následujících operacích, soustředěných v podprogramu:

- zastavit čítání (start/stop=0, viz *obr. 8.1*),
- zapsat do čítače požadované číslo,
- povolit přerušení od čítače,
- nastavit proměnnou `citac_v_behu=1`

- povolit čítání (start/stop=1),
- spustit pohon mechanizmu,
- návrat do hlavního programu.

Procesor je dále využit pro jiné aktivity hlavního programu a celá úloha odpočítávání impulzů je záležitostí výhradně periferního systému čítačů/časovačů. Po dopočítání do nuly je vydán požadavek na přerušení. Tím je vyvolán podprogram, který zastaví mechanizmus:

- zastavit pohon mechanizmu,
- zakázat přerušení od čítače,
- zastavit čítání,
- nastavit proměnnou `citac_v_behu=0`
- návrat do hlavního programu.

Proměnná `citac_v_behu` slouží k informaci hlavního programu o momentální situaci, tak aby nový přesun nemohl být zadán ještě před dokončením současně probíhajícího přesunu. Před zadáním nového přesunu ji program testuje.

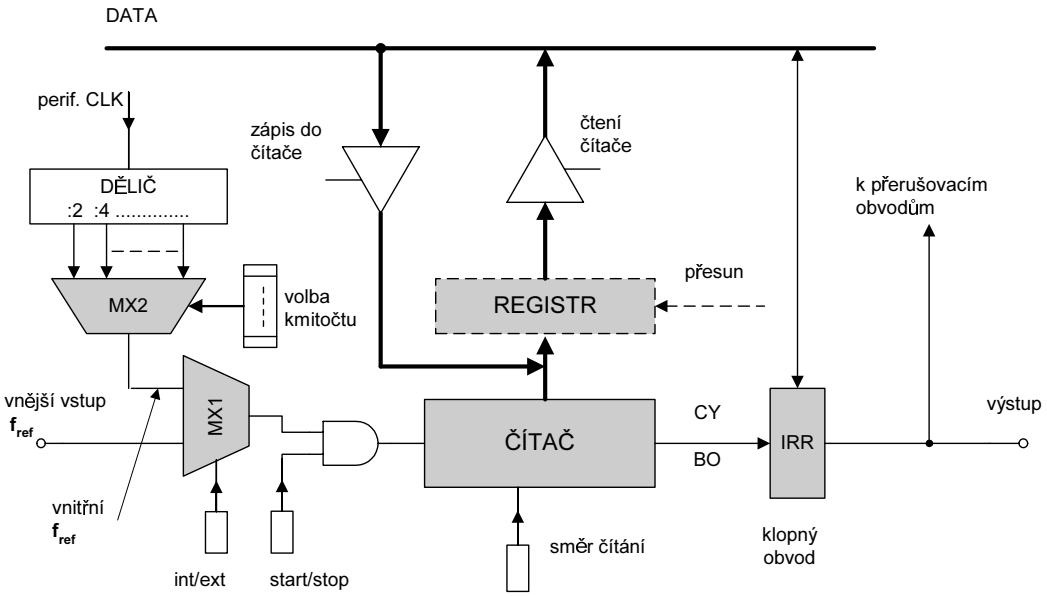
Výše uvedená struktura programů může představovat riziko, neboť pohon mechanizmu je zastaven programově, a to až v obslužném podprogramu přerušení. To sice bylo jistě předem povoleno, nemusí ale mít nejvyšší prioritu. Může nastat situace, kdy zastavení operace je časově posunuto až za ukončení běžících interruptových podprogramů vyšší priority. To je samozřejmě zcela nežádoucí. U rozsáhlého systému autonomních čítačů/časovačů se tak **přiřazení priorit přerušení** stává vážným problémem.

Řešením je úprava obvodů v jednotce čítačů/časovačů tak, aby výstupní signál pro technologické zařízení byl jimi generován bezprostředně, bez spolupráce s programem. Vývod klopného obvodu z pouzdra je naznačen na *obr. 8.1*. U některých typů mikrokontrolérů je tento princip velmi dobře rozpracován a jednotka čítačů/časovačů může **přímo ovládat** vývody paralelních bran. Bohužel, u jiných typů to není možné a pak jejich využitelnost pro řízení je omezená i přes jinak třeba velmi dobré parametry.

Další častou aplikací režimu čítání impulzů je měření kmitočtu nebo otáček. K tomu postačí vynulovat čítač, pak signálem „start/stop“ zapnout čítání na přesně stanovenou dobu, a nakonec přečíst stav čítače. Je ovšem nutné mít k dispozici zdroj přesných časových funkcí. K tomu může sloužit druhý čítač/časovač v režimu časování (viz třetí vstup multiplexeru v *obr. 8.1*), který generuje impulz přesné délky.

8.1.2 Režim časování

Funkce časovače se dosáhne tak, že na vstup čítače se přivede kmitočet z přesného generátoru, zpravidla řízeného krystalem. Zjednodušený princip činnosti čítače/časovače v režimu časovače ukazuje *obr. 8.3*.



Obr. 8.3 Čítač/časovač v režimu časování

Z podobnosti s *obr. 8.1* je zřejmé, že pro tento režim jsou využívány tytéž obvody, jako pro režim čítání impulzů, zde však čítač počítá impulzy přesného referenčního kmitočtu f_{ref}. U některých typů mikrokontrolérů je možné přepnout na vnější zdroj přesného kmitočtu (multiplexer MX1), běžnější však je využití vnitřního rozvodu periferních hodinových impulzů. Ty jsou odvozeny od základního generátoru hodinových impulzů počítače, řízeného krystalem (viz též kapitolu 5.6). I když periferní hodinové impulzy mají nižší kmitočet než hodinové impulzy procesoru, je často i tento kmitočet příliš vysoký pro účely časování. Např. pro ovládání technologických zařízení jsou zapotřebí časové intervaly řádu milisekund až mnoha sekund. Při kmitočtu hodinových impulzů 20 MHz (pro procesor) by to vyžadovalo čítač nejméně 24bitový. Proto je ve většině mikrokontrolérů ještě další dělič kmitočtu pro jednotku čítačů/časovačů. Patřičný dělicí poměr lze vybrat multiplexerem MX2.

Odvození f_{ref} od základního generátoru počítače může být problematické, neboť krystalový generátor nelze plynule přelaďovat. Časovač pak není možno nastavit zcela přesně – je nutné najít nejvhodnější kombinaci dělicích poměrů všech předděličů tak, aby chyba byla přijatelná. Někdy se dokonce musí využít vnější zdroj přesného kmitočtu.

Klopný obvod na výstupu čítače může vyvolat přerušení programu. To souvisí s typickými způsoby práce s časovačem.

Typické využití časovače v programech s přerušením:

Častou úlohou je generování impulzu přesné délky na výstupu počítače. Jako výstup bude sloužit paralelní brána. Předpokládá se, že v inicializační části programu byl nastaven režim čítače/časovače na čítání vnitřních impulzů, byl vybrán patřičný dělicí poměr, byl nastaven směr čítání dolů. Obsluha časovače bude spočívat v následujících operacích, soustředěných v podprogramu:

- zastavit čítání (start/stop=0),
- zapsat do časovače požadované číslo,
- povolit přerušení od časovače,
- nastavit proměnnou `cas_v_behu=1`
- zapsat „1“ do patřičného bitu výstupní brány,
- povolit čítání (start/stop=1),
- návrat do hlavního programu.

Procesor dále pracuje na úkolech hlavního programu a celá úloha časování je záležitostí výhradně časovače. Po dopočítání do nuly je vydán požadavek na přerušení. Tím je vyvolán podprogram, který ukončí impulz:

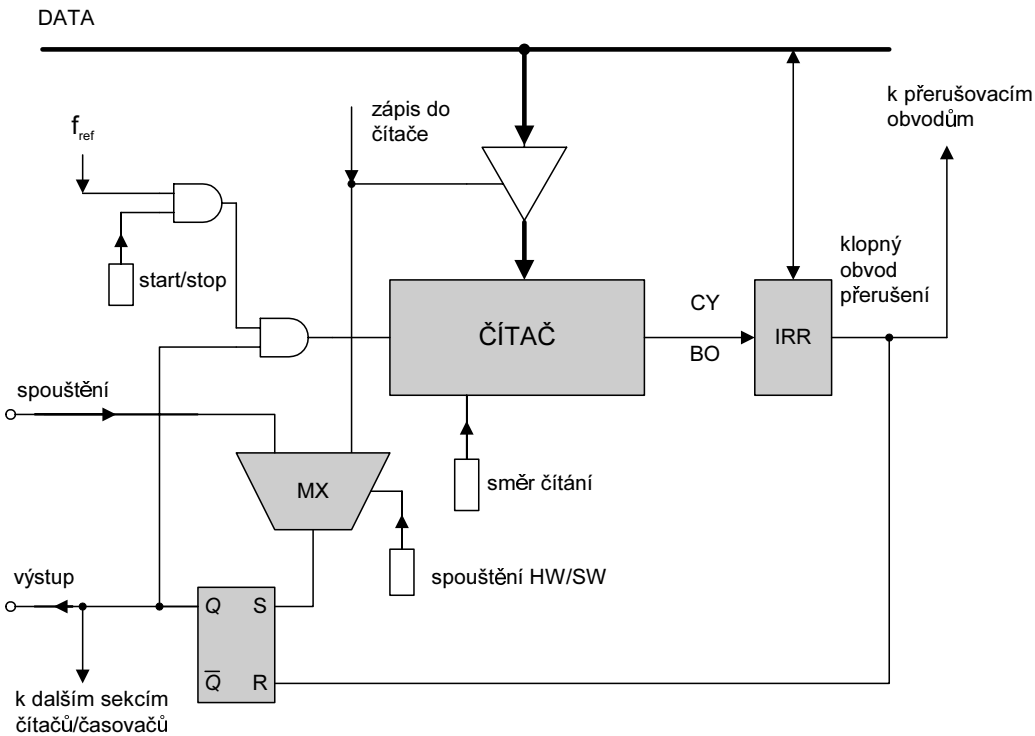
- zapsat „0“ do patřičného bitu výstupní brány,
- zakázat přerušení od časovače,
- zastavit čítání,
- nastavit proměnnou `cas_v_behu=0`
- návrat do hlavního programu.

Proměnná `cas_v_behu` slouží k informaci hlavního programu o momentální situaci, tak aby nová časová funkce nemohla být zadána ještě před dokončením prvé.

Je zřejmé, že doba od spuštění časovače do jeho průchodu 0 je úměrná přednastavenému číslu a je časovačem odměřena velmi přesně. Rozsah času je od jedné periody referenčního kmitočtu až do počtu period, daného kapacitou čítače. Čas může být pokaždé programově zadán libovolně v tomto rozmezí. Vzhledem k tomu, že stav čítače lze číst i nastavit v kterémkoliv okamžiku, je možné s takovýmto časovačem pracovat jako se stopkami – lze je zastavit, číst a porovnávat, vynulovat, nastavit novou hodnotu.

U výše uvedené programové konstrukce je třeba zdůraznit, že odměření času je přesné jen na **výstupu časovače**, bohužel nikoliv na výstupu brány. Její obsluha je vyvolávána přerušením. Výstupní operace tak má svoji latenci (viz kapitolu 6) se složkou konstantní a složkou náhodnou. Dále, pokud přerušení od časovače nemá nejvyšší prioritu, může výstupní operace být časově posunuta až za ukončení běžících interruptových podprogramů vyšší priority. Tím se projeví náhodná složka latence ještě podstatně delší. Obsluha časovače pomocí přerušení je proto přípustná pro dlouhé časy, kde náhodná složka latence hraje relativně malou roli. Starší a jednoduché mikrokontroléry nenabízejí lepší řešení.

Při vyšších nárocích na přesnost časování a při větším počtu nezávisle obsluhovaných časových funkcí je nutné čítače doplnit o další obvody, které mohou samostatně (bez spolupráce s programem) generovat výstupní signály počítače. Princip ukazuje *obr. 8.4* – pro jednoduchost zde nejsou znázorněny obvody, dodávající referenční kmitočet.

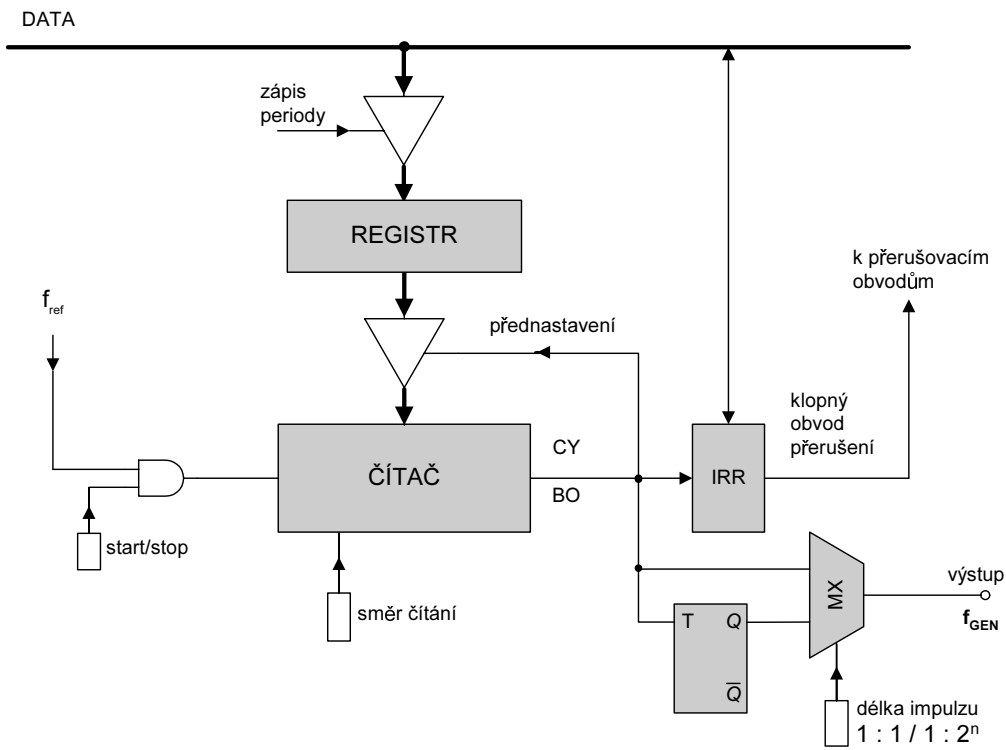


Obr. 8.4 Přímá generace výstupního signálu

Výstupem multiplexeru MX je klopný obvod SR překlopen do stavu „1“ a povolí tak průchod impulzů do čítače. Po dopočítání do nuly je klopný obvod SR překlopen do stavu „0“ a zablokuje vstupní impulzy. Současně je generován i požadavek na přerušení. Signál na výstupu klopného obvodu SR, který je přesně časován, je vyveden z pouzdra. Přerušení tak nemusí být vůbec použito, případně signál IRR slouží jen k informaci hlavního programu o ukončení časové funkce. Multiplexerem lze volit, zda začátek generovaného impulzu je dán okamžikem přednastavení čítače (softwarové spouštění), nebo vnějším impulzem (hardwarové spouštění). Při hardwarovém spouštění se nejprve čítač přednastaví a tím je připraven na budoucí spouštěcí impulz.

8.1.3 Režim generace impulzů

Generátor impulzů vznikne doplněním čítače o pomocný registr, jehož obsahem se periodicky čítač přednastavuje (angl. reload). K přednastavení dojde při počítání nahoru při přechodu ze stavu čítače 1...11 do 0...00, při počítání dolů při přechodu z 0...00 do 1...11. Obsah pomocného registru se zadá jen jednou, při další činnosti generátoru se již nemění. Stabilita kmitočtu je dána kvalitou zdroje impulzů pro čítač. Zjednodušené zapojení čítače/časovače v režimu generace impulzů ukazuje *obr. 8.5*.



Obr. 8.5 Čítač/časovač v režimu generace impulzů

Možnost volby zdroje přesného kmitočtu je obdobná, jako v *obr. 8.3*. O přesném nastavení f_{GEN} platí přiměřeně totéž, co pro časovač. Hlavní generátor hodinových impulzů počítače se všemi předděliči nemusí dávat zcela správný f_{ref} – pak je nutné použít vnější zdroj referenčního kmitočtu, nebo se spokojit s omezenou přesností.

Impulzy na výstupu čítače jsou krátké vzhledem k periodě (jen jeden takt hodinových impulzů). Někdy je zapotřebí generovat impulzy s větší střídou (poměrem impulz/mezera). K tomu postačí zařadit za čítač klopný obvod T, který dělí kmitočet dvěma a dodává signál s poměrem impulz : mezera přesně 1 : 1.

Typické využití čítače pro generaci impulzů:

Předpokládá se, že se jedná o mikrokontrolér. V inicializační části programu byl nastaven režim čítače/časovače na čítání vnitřních impulzů, byl vybrán patřičný dělicí poměr, byl nastaven směr čítání dolů. Začátek a konec generace impulzů je pak ovládán bitem „start/stop“.

Kmitočet impulzů f_{GEN}, generovaných čítačem, je

$$f_{GEN} = f_{PERIF}/(K_{PD} \cdot N_{PR}),$$

kde f_{PERIF} je kmitočet periferních hodinových impulzů počítače, K_{PD} je dělicí poměr předděliče jednotky čítačů/časovačů, N_{PR} je obsah pomocného registru generátoru.

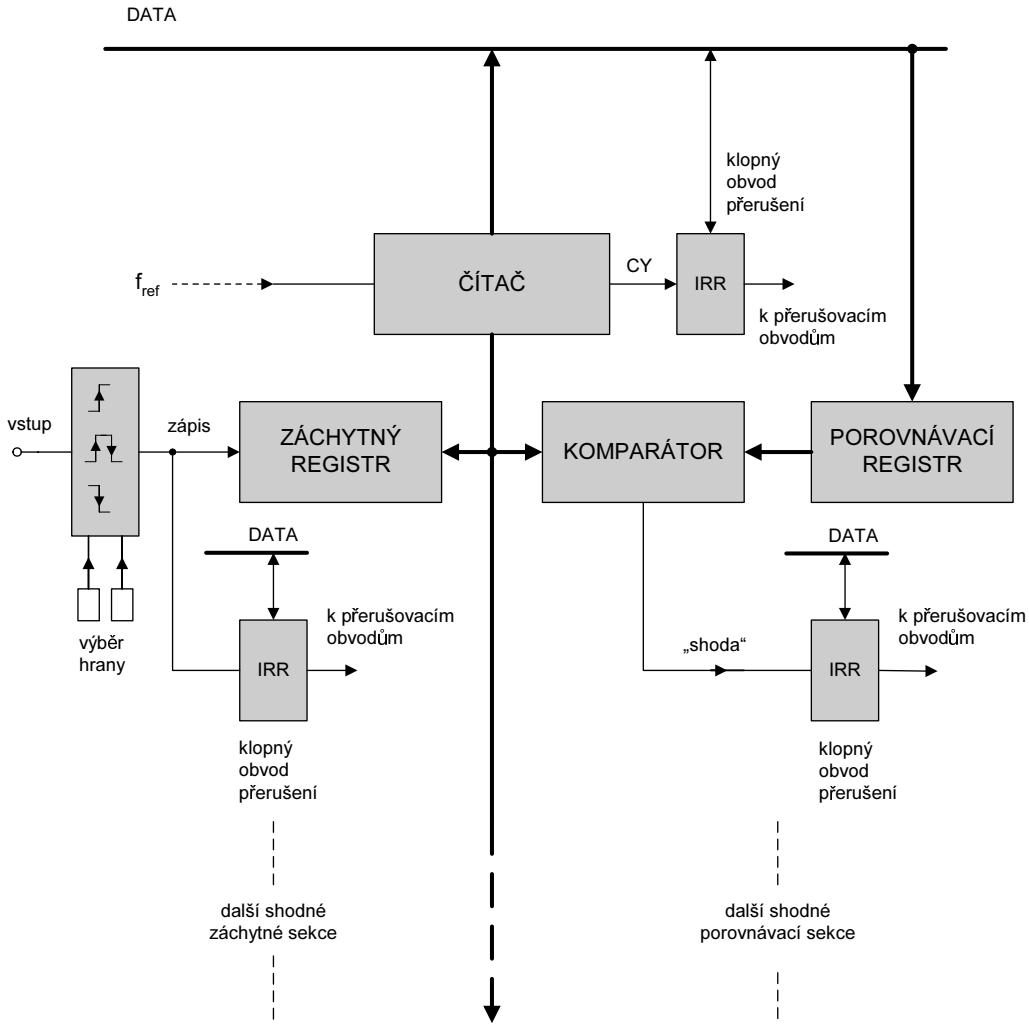
Generované impulzy mohou být vyvedeny na výstup, nebo využity dalšími vnitřními obvody. Typicky to jsou sériové komunikační obvody. Velmi časté je též **periodické vyvolávání přerušení**, potřebné pro práci v reálném čase – tzv. RTI (Real-Time Interrupt). Perioda je typicky nastavena na několik (10) milisekund. V obslužném podprogramu tohoto přerušení je pouze inkrementována časová proměnná, případně několik takovýchto proměnných `cas_1, cas_2, ...` Proměnné jsou typicky deklarovány jako globální. V hlavním programu s nimi lze pracovat jako se stopkami, tj. vynulovat je, nastavit, číst a porovnávat. Tak lze alespoň částečně nahradit rozsáhlou jednotku čítačů/časovačů, která u daného počítače chybí.

8.2 ČASOVACÍ JEDNOTKA CAPCOM

Univerzální čítače a časovače jsou cenným nástrojem pro aplikace v oblasti měření a řízení. Pro jejich plné využití však musí být možné je vzájemně propojovat, přičemž vzájemné vazby musí být mnohostranné a programovatelné. Je řada příkladů jednočipových mikropočítačů, u kterých je jednotka univerzálních čítačů/časovačů velmi dobře přizpůsobena pro některé aplikace, a velmi špatně pro jiné, které konstruktér obvodu zřejmě nepředvídal. To je jedním z důvodů, proč byly časem vyvinuty jednotky, zkratkou nazývané CAPCOM.

Zkratka vznikla z označení dvou činností jednotky: **zachycení stavu** (angl. Capture), a **porovnání stavu** (angl. Compare). Zachycuje se stav volně běžícího čítače a dočasně se přesunuje do záchytného registru. Porovnává se stav čítače s obsahem porovnávacího registru. Záchytných i porovnávacích registrů je několik – např. 8. Princip ukazuje *obr. 8.6*.

Jednotka má svůj dělič hodinových impulzů, případně i vnější vstup. Čítač je zpravidla 16bitový, jednoduchý, bez možnosti změny směru čítání, a bez možnosti zápisu (čtení je možné). Při přechodu ze stavu 11...1 do 00...0 je generován požadavek na přerušení.



Obr. 8.6 Jednotka CAPCOM

Do záchytného registru se přepíše stav čítače při změně stavu na vstupu, tj. působením **vnější události**. Lze volit buď náběžnou hranu, doběžnou hranu, případně obě hrany. Současně se překlopí klopný obvod, kterým je generován požadavek na přerušení. Rozdíl dvou po sobě následujících čtení udává počet vstupních impulzů čítače. Při jejich známé periodě tak lze přesně měřit čas mezi dvěma událostmi.

Stav čítače je v komparátoru porovnáván s obsahem porovnávacího registru a při shodě obou čísel je generován impulz, kterým je překlopen klopný obvod. Ten generuje požadavek na přerušení. Obslužný program zajistí příslušnou akci, např. změnu stavu na výstupní bráně. Impulzem z komparátoru mohou být ovládány i další obvody.

Pro úsporu počtu registrů jsou u některých konstrukcí registry využity dvojnásobně – lze jim přiřadit buď funkci záchytného, nebo porovnávacího registru. Rovněž počet požadavků na přerušení je někdy podstatně redukován.

8.2.1 Záchytná jednotka

Obsah záchytného registru je k dispozici, dokud není dalším vstupním impulzem přepsán. Do té doby může být procesorem přečten. Informaci o novém záchytu dostává procesor prostřednictvím přerušení. Další shodné záchytné sekce jednotky pracují obdobně. Je tak možné měřit časy mezi událostmi na jenom vstupu i na několika různých vstupech jednotky.

Je třeba zdůraznit, že záchytná jednotka ke své činnosti potřebuje programovou podporu, tj. obslužné podprogramy, vyvolávané požadavky na přerušení. I zde je důležité uvažovat latenci přerušení, tak aby obsahy záchytných registrů byly přečteny včas. Bude samozřejmě záležet na aplikaci – tak např. při měření kmitočtu sítě 50 Hz je perioda 20 ms jistě dostatečně dlouhá pro spolehlivé přečtení registrů za každé situace. Naopak časově kritické situace nastávají např. v řídicím počítači pro rychloběžný spalovací motor.

Typické využití záchytné jednotky:

Řídicími bity pro výběr hrany se nejprve v každé použité sekci zvolí aktivní hrana vstupního impulzu. Zde uvedeme jen několik nejběžnějších konfigurací.

K **měření periody vstupních impulzů** postačí jedna sekce. V obslužném podprogramu přerušení se přečte obsah záchytného registru a od něj se odečte „minulý obsah“ záchytného registru. Ten byl uložen v minulé periodě dočasně v paměti počítače. Rozdíl obou čísel udává počet impulzů čítače. Jejich kmitočet je známý (a přesný). Přečtený obsah záchytného registru se pak uloží do paměti jako „minulý obsah“, takže je připraven pro příští měření periody. Jednoduché odečtení dává správný výsledek i tehdy, když mezi současným a minulým čtením čítač přešel z maximální hodnoty (11...1) na nulu. Je ovšem nutné s oběma čísly počítat jako s „unsigned integer“ o počtu bitů shodném s počtem bitů čítače, a nerespektovat případný přenos z nejvyššího bitu součtu. Pro ilustraci uvažujme záchytný registr pouze 8bitový, minulý obsah je např. 233 a současný obsah 253, tedy rozdíl je 20.

V binárním kódu:	1 1 1 1 1 1 0 1	(=253)	... současný obsah
	– 1 1 1 0 1 0 0 1	(=233)	... minulý obsah
	0 0 0 1 0 1 0 0	**(=20)**	... přírůstek

Při přechodu čítače přes nulu je minulý obsah např. 353 a současný obsah 17.

V binárním kódu:	0 0 0 1 0 0 0 1	(=17)	... současný obsah
	– 1 1 1 1 1 1 0 1	(=253)	... minulý obsah
(přenos) <---	**0 0 0 1 0 1 0 0**	**(=20)**	... přírůstek

Přírůstek je opět správný, pokud se přenos z nejvyššího bitu ignoruje.

426410

K **měření délky impulzu** se signál zavede současně do dvou sekcí. U prvé sekce se nastaví jako aktivní hrana náběžná (začátek impulzu) a u druhé sekce hrana doběžná (konec impulzu). Obslužný podprogram je vyvolán druhou sekcí. Obsah prvého záchytného registru se odečte od obsahu druhého registru.

K **měření fázového posuvu** dvou signálů se využijí dvě sekce, každá pro jeden signál. U obou se nastaví stejná aktivní hrana. Signál s fázovým zpožděním vyvolává přerušení. Měří se časový posuv impulzů na základě rozdílu obsahu záchytných registrů. Současně se u jednoho signálu měří i perioda. Fáze se vypočte jako poměr zpoždění ku periodě, vynásobený patřičnou konstantou (např. 360, 2π, apod.).

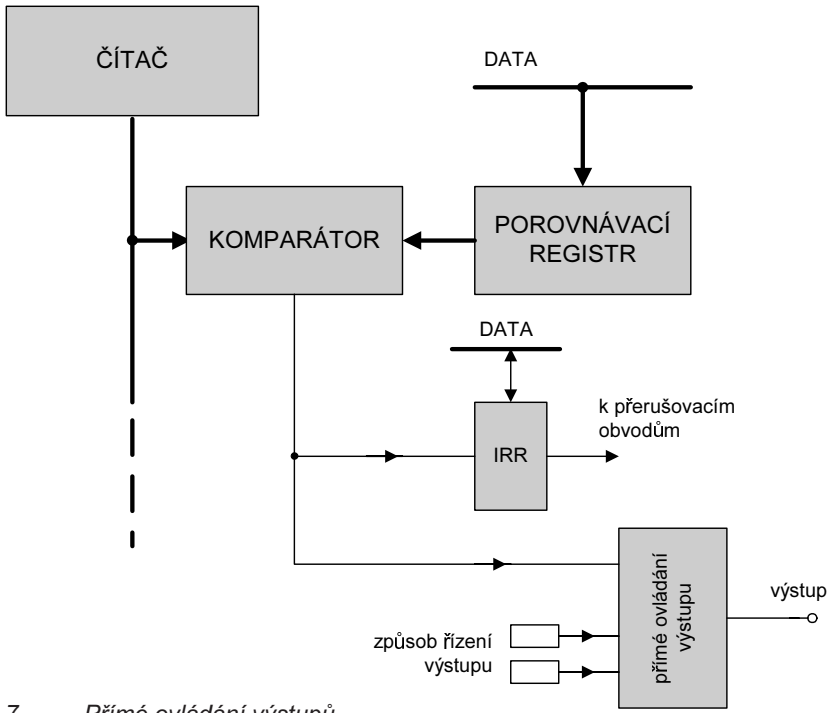
8.2.2 Porovnávací jednotka

Impulz z komparátoru signalizuje momentální shodu obsahů čítače a porovnávacího registru. Okamžik shody má samozřejmě smysl jen ve vztahu k jiným událostem. Těmi může být shoda v dalších sekcích porovnávací jednotky, překlopení čítače ze stavu 11...1 do 00...0, nebo události na vstupech záchytné jednotky. Vhodným naprogramováním porovnávacích registrů lze přesně časově odstupňovat impulzy z několika komparátorů a postupně vyvolávat různé akce, tj. změny stavu na některé výstupní bráně počítače. Do porovnávacích registrů lze samozřejmě zapisovat kdykoliv a tak lze průběžně měnit časování.

Akci lze vyvolat pomocí přerušení. Zde je ovšem velmi významná latence přerušení, zvláště její náhodná složka – konstantní složku lze totiž vykompenzovat vhodným časovým posunutím impulzu z komparátoru. Náhodná složka latence souvisí s počtem požadavků na přerušení a s přidělením priorit. U mikrokontrolérů s rozsáhlým systémem periferních obvodů to může vést k problémům. Proto jsou porovnávací jednotky doplněny o další obvody, které mohou generovat výstupní signály zcela samostatně, bez spolupráce s programem. *Obr. 8.7* ukazuje jednu možnost řešení.

Komparátor je spojen s **obvody přímého ovládání výstupu**. V nich lze zvolit několik typů reakce na impulz z komparátoru – např. stav 0 na výstupu, stav 1 na výstupu, střídání stavu na výstupu. Časování akce je vždy dáno obsahem porovnávacího registru, předem nastaveného. Nastavení je nutné provádět prostřednictvím programu, stejně jako v předchozím případě. Rozdíl je však v tom, že nyní je akce okamžitá, bez jakékoliv latence, a tedy přesněji časovaná.

Tento princip je u některých mikrokontrolérů rozveden ještě dále. Místo ovládání jednoho bitu výstupní brány lze vydat celé výstupní slovo. To se předem uloží do speciálního registru příslušné sekce porovnávací jednotky a impulzem komparátoru je automaticky přesunuto na výstupní vývody. Je tak ulehčena např. generace impulzů ve více fázích.



Obr. 8.7 Přímé ovládání výstupů

Typické využití porovnávací jednotky:

Úkolem bude generace impulzů na výstupu mikrokontroléru. Bude využita jedna porovnávací sekce s plně programovou obsluhou podle *obr. 8.6*. Na začátku se do příslušného porovnávacího registru zapíše nula. Při tomto stavu čítače pak komparátor vyvolá přerušení. V obslužném podprogramu je nastaven stav 1 na požadovaném výstupu a dále je zvýšen obsah porovnávacího registru o číslo K_1. Až čítač dosáhne této zvýšené hodnoty, je vyvoláno další přerušení. Na výstupu je nastaven stav 0 a obsah porovnávacího registru je zvýšen o číslo K_2. Tím je skončena prvá perioda impulzního průběhu na výstupu. V dalších periodách se stále zvyšuje obsah porovnávacího registru střídavě o K_1 a K_2. Jednoduché přičítání konstant dává správný výsledek při přičítání, pokud obě konstanty jsou typu „unsigned integer“ o počtu bitů shodném s počtem bitů porovnávacího registru, a nerespektuje se případný přenos z nejvyššího bitu součtu. Latence přerušení zpožďuje změny na výstupu. Její náhodná složka způsobuje chyby v časování.

Obdobně lze generovat i **impulzy ve více fázích**. Zde je nejlépe použít pro každou fázi jednu sekci porovnávací jednotky. Postup je shodný s postupem pro jednu fázi, jen na začátku jsou do jednotlivých porovnávacích registrů zapsány odlišné, odstupňované hodnoty.

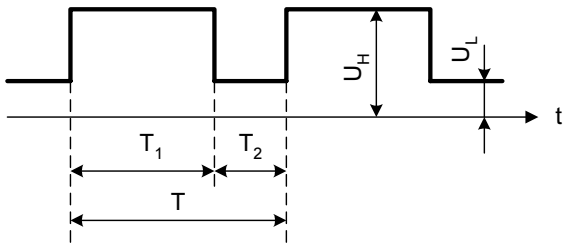
Přesnější časování lze dosáhnout při přímém ovládání výstupů podle *obr. 8.7*.

426410

Programová obsluha je částečně shodná s předchozími případy – i zde se při každém přerušení zvyšuje obsah porovnávacího registru střídavě o K_1 a K_2. Odpadá manipulace s výstupy, neboť jejich ovládání je nyní automatické, impulzem z komparátoru. Vedle zásahu do porovnávacího registru je však třeba v obslužném podprogramu zadat i příští reakci obvodu pro přímé ovládání výstupu, v tomto případě střídavě vydávání nuly a jedničky. Latence přerušení zde ztrácí význam.

8.3 IMPULZNÍ ŠÍŘKOVÝ MODULÁTOR

Tímto modulátorem je vybaven téměř každý mikrokontrolér. Impulzní šířková modulace (PWM – angl. Pulse-Width Modulation) je při své jednoduchosti velmi vhodná pro přenos signálu na větší vzdálenosti v přítomnosti rušení. Často se využívá pro ovládání elektromotorů, elektromagnetů, elektrotepelných zařízení, apod. Průběh signálu na výstupu mikrokontroléru ukazuje *obr. 8.8*.



Obr. 8.8 Signál s impulzní šířkovou modulací

Střední hodnota napětí $U_{stř}$ je dána vztahem:

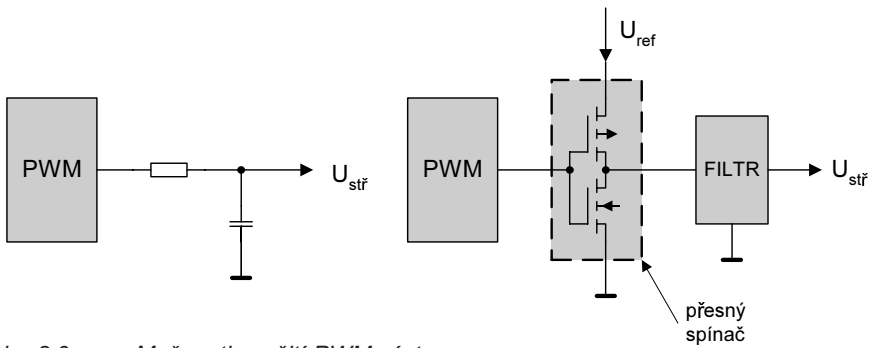
$$U_{stř} = \frac{T_1}{T}(U_H - U_L) + U_L,$$

kde U_H je napětí ve stavu 1 a U_L je napětí ve stavu 0. V ideálním případě, při konstantním a přesně definovaném U_H a nulovém U_L, je střední hodnota napětí závislá jen na **poměru dvou časů**, a ten lze dodržet velmi přesně.

Jako demodulátor lze použít jakýkoliv obvod, který vytvoří střední hodnotu z impulzního průběhu. Stačí i jednoduchý filtr typu dolní propust, např. integrační článek RC. Pokud má ovládané zařízení dostatečnou setrvačnost, není případně ani filtr nutný. Tak je tomu u již zmíněných elektromotorů, elektrotepelných zařízení, atd. V těchto případech postačuje jen výkonový spínač. Filtr či setrvačnost připojeného zařízení je dále účinným prostředkem pro **potlačení rušení**, které může do spoje pronikat kapacitní či induktivní vazbou. Střední hodnota přenášeného signálu přitom není ovlivněna – u obou těchto vazeb je totiž střední hodnota pronikajícího signálu nulová.

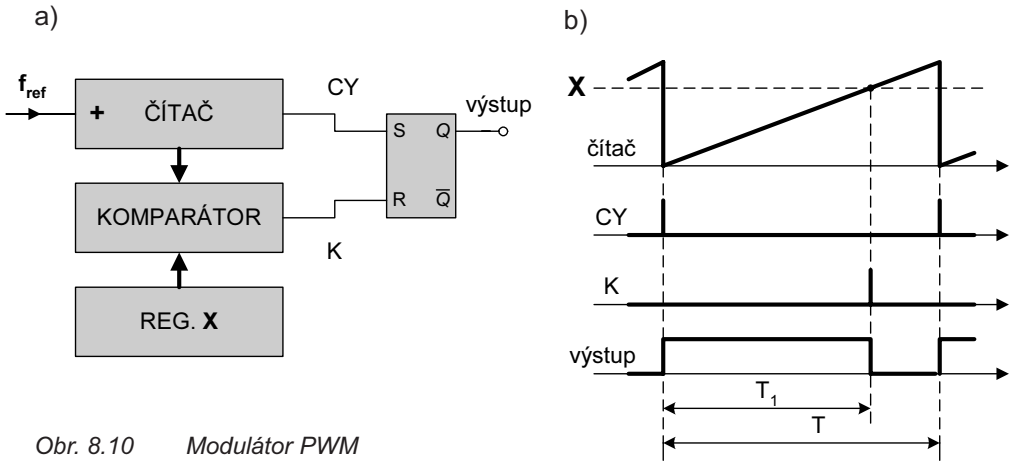
Výstup PWM signálu s navazujícím jednoduchým filtrem lze též využít namísto vnitřního číslicově/analogového převodníku. Tím je vybavena jen malá část mik-

rokontrolérů, zatím co výstup PWM je téměř pravidlem. V praxi je však nutné počítat s realistickými hodnotami U_H a U_L. Mikrokontroléry jsou vesměs vyráběny v technologii CMOS, u které tolerance výstupních napětí dosahují několika procent i při velmi malé zátěži. Při vyšších nárocích na přesnost je pak nutné výstupním signálem řídit **přesný spínač** referenčního napětí. *Obr. 8.9* ukazuje dvě možnosti využití PWM výstupu. V prvém případě se jedná o nejjednodušší způsob s malými nároky na přesnost, ve druhém případě lze prostřednictvím přesného spínače a kvalitního filtru dosáhnout vyšší přesnosti a nezávislosti na výstupním napětí modulátoru. Záměnou přesného napěťového spínače za spínač výkonový lze ovládat vyšší výkon. Spínač na *obr. 8.9* obrací fázi a proto některé PWM modulátory umožňují volit přímý nebo negovaný výstup.



Obr. 8.9 Možnosti využití PWM výstupu

Samotný PWM modulátor je založen na čítači. Čítač počítá nahoru a při přechodu ze stavu 11...1 do 00...0 překlopí klopný obvod do stavu 1. Při shodě čísla v čítači s číslem v registru je naopak klopný obvod vynulován. Zapojení, které je běžné u nejjednodušších mikrokontrolérů, ukazuje *obr. 8.10a*. Průběhy signálů jsou na *obr. 8.10b*. Pilové kmity zde symbolizují zvyšující se číslo v čítači.



Obr. 8.10 Modulátor PWM

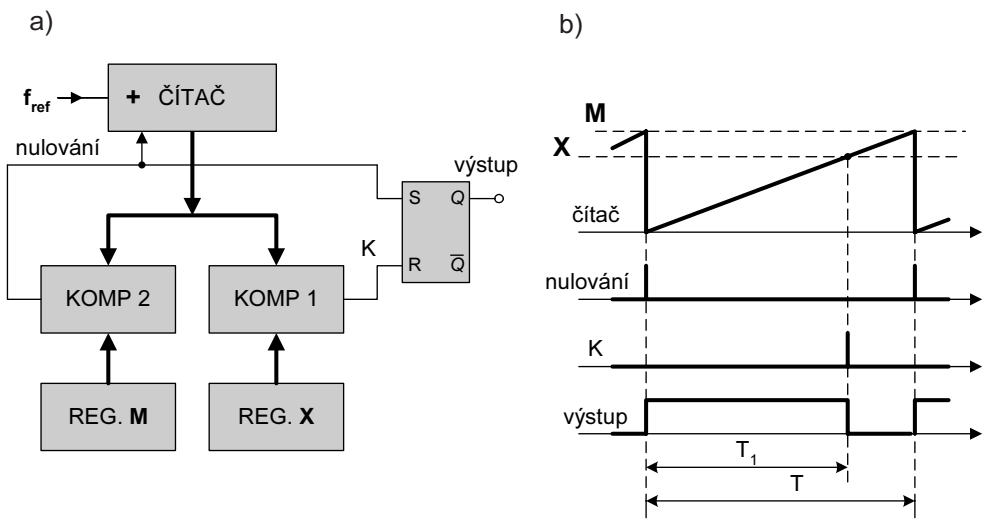
Pro poměr T_1/T platí:

$$\frac{T_1}{T} = \frac{X}{2^N},$$

kde X je číslo v registru a N je počet bitů binárního čítače. Střední hodnota výstupního napětí zřejmě nezávisí na kmitočtu impulzů do čítače f_{ref}. Ten však přesto nelze volit zcela libovolně. Vysoký kmitočet v kombinaci s parazitní kapacitou vnějších spojů deformuje tvar impulzů a komplikuje demodulátor. Nízký kmitočet naopak klade vyšší nároky na filtr a zvyšuje zvlnění na jeho výstupu. Proto lze kmitočet vybírat podobně, jak je znázorněno v *obr. 8.3* a dalších. Navíc bývá možné vhodný kmitočet generovat jinou čítačovou sekcí, nastavenou do režimu generace impulzů.

Relativní chyba v nastavení šířky impulzů je dána počtem stavů čítače – v případě čítače binárního je chyba $1/2^N$. U jednoduchých modulátorů je nejčastěji využíván čítač 8bitový. Chyba 1/256, tj. cca 0,4 % je přijatelná pro uspořádání modulátor-demodulátor bez přesného spínače. Vyšší přesnost s využitím 16bitového čítače je mezi mikrokontroléry spíše výjimkou.

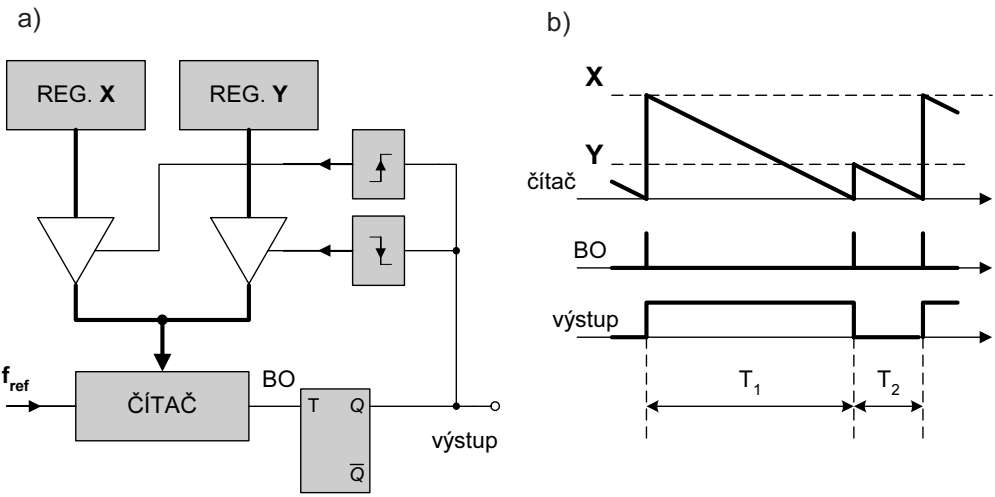
Binární čítač nemusí být v některých případech vhodný. Můžeme např. počítat se šířkou impulzů T_1 vyjádřenou v úhlových stupních, od 0 do 359. Pak tedy T_1 je třeba nastavovat v inkrementech po T/360. Jindy je žádoucí nastavovat T_1 v procentech s inkrementem T/100. Obecně je pak nutné pracovat s počtem stavů čítače (jeho modulem) jiným, než je mocnina dvou. Odpovídající zapojení modulátoru ukazuje *obr. 8.11*. Komparátor č. 1 má stejnou funkci, jako má komparátor v *obr. 8.10*, komparátor č. 2 vynuluje čítač v okamžiku shody čísla v čítači s číslem v registru M.



Obr. 8.11 Modulátor PWM s nastavitelným modulem čítače

426410

Jiné zapojení modulátoru, u kterého lze nezávisle řídit šířku impulzu T_1 a šířku mezery T_2, je na *obr. 8.12*. Čítač v tomto zapojení počítá dolů, při přechodu ze stavu 00...0 do 11...1 je generován záporný přenosový impulz BO a tím překlopí klopný obvod typu „T“. Ten se tedy při jednom průchodu čítače nulou překlopí do stavu 1 a při druhém do stavu 0 – to se stále opakuje. Náběžnou hranou na výstupu klopného obvodu se do čítače přesune obsah registru X, doběžnou hranou obsah registru Y.



Obr. 8.12 PWM modulátor s nastavitelnou šířkou impulzu i mezery

9 VSTUPNÍ A VÝSTUPNÍ OBVODY

Vstupní a výstupní obvody zprostředkují spojení počítače s okolím. Všeobecně je lze rozdělit na obvody paralelní a obvody sériové. Paralelní obvody pracují současně s celou skupinou signálů (nejčastěji s osmi). Sériové obvody pracují s jedním vstupním a jedním výstupním signálem a informaci přenášejí bit po bitu. Ke vstupním obvodům budou v této kapitole přiřazeny i obvody pro vstup analogových signálů.

Vstupní a výstupní obvody tvoří složitý komplex, který je třeba řídit a jehož stav je třeba znát. K řízení slouží soubor řídicích bitů rozmístěných v řídicích registrech. Obdobně ke zjištění stavu slouží soubor stavových bitů rozmístěných ve stavových registrech. V obou případech lze využívat instrukce procesoru pro práci s jednotlivými bity. Způsob rozmístění řídicích a stavových bitů po jednotlivých řídicích a stavových registrech je charakteristický pro danou architekturu a nelze jej zobecnit.

9.1 ŘÍZENÍ VSTUPNÍCH A VÝSTUPNÍCH OBVODŮ

Část řídicích bitů se nastavuje na začátku programu a dále se již jejich stav nemění. Typickým případem je např. volba funkce vývodu pouzdra, modulační rychlost u sériových obvodů, režim A/Č převodníku, apod. Během provozu pak se pracuje s dalšími řídicími a stavovými bity. Lze obracet směr přenosu v paralelních obvodech, blokovat či povolovat příjem v sériovém přijímači, volit kanál a spouštět převod A/Č převodníku, apod. Stavové bity dávají informaci o změně stavu na vstupech, o dokončeném přenosu slova, o chybě přenosu, apod. Existuje několik způsobů řízení vstupních a výstupních operací.

Přímé programové řízení

Před každou operací se vstupními nebo výstupními obvody se zjistí jejich stav. Pokud stav ukazuje na jejich připravenost, operace se provede. V opačném případě se znovu opakovaně testuje stav. Doba na testování však musí být nějak omezena (časovačem nebo počítáním opakovaní), neboť po dobu opakovaného testování je pozastaveno další provádění programu. Mohlo by dokonce dojít k úplnému zablokování programu v důsledku případné trvalé poruchy ve vstupních

a výstupních obvodech nebo ve vnějších zařízeních, spojených s těmito obvody. Při větším počtu současně probíhajících vstupních a výstupních operací lze libovolným způsobem programově stanovovat jejich priority. Přímé programové řízení je jednoduché a velmi pružné. Dochází však ke zpomalování programu.

Vyvolání přerušení

Vstupní a výstupní obvody mohou generovat požadavky na přerušení. U vstupních obvodů je podnětem získání nových dat, jako např. při dokončení převodu A/Č převodníku, dokončení příjmu znaku, změně stavu na vstupech. U výstupních obvodů je podnětem dokončení předchozí operace a připravenost k operaci nové. Přerušením je proveden odskok na patřičný obslužný podprogram. Zpoždění při vyvolání podprogramu (latence přerušení) je zpravidla kratší, než zpoždění při programovém řízení. Při velkém počtu obvodů, obsluhovaných na základě přerušení, však vzniká problém s přidělováním priorit. Požadavek s nízkou prioritou může být obsluhován s velkým zpožděním, pokud se vyskytne během obsluhy požadavků s vyšší prioritou.

Vyvolání přenosu DMA

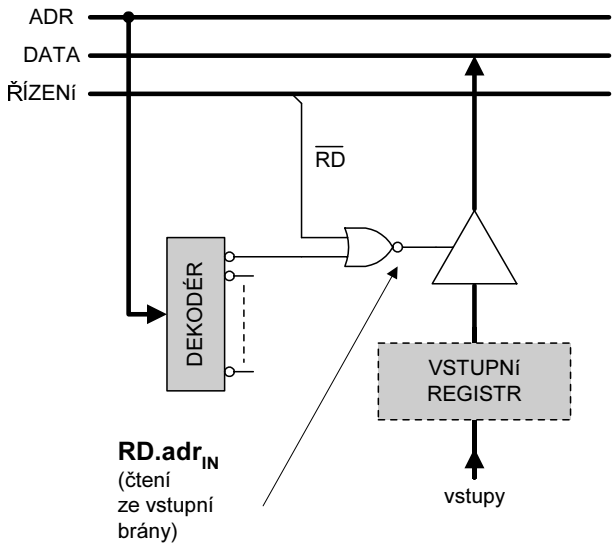
O přenosech v režimu DMA pojednávala kapitola 8. V případě vstupních a výstupních obvodů lze požadavky na obsluhu zavést buď do řadiče přerušení, nebo do řadiče DMA. Při obsluze prostřednictvím DMA odpadá zpoždění, způsobené latencí přerušení. Využití DMA je možné jen u vyspělých architektur jednočipových mikropočítačů.

9.2 PARALELNÍ VSTUPNÍ A VÝSTUPNÍ OBVODY

Paralelní vstupní a výstupní obvody mohou přenášet najednou celé slovo. Připojovací místo na počítači se nazývá „paralelní brána“. Typická je brána osmibitová, méně často šestnáctibitová. Kromě přenosu celého slova umožňují paralelní obvody zpravidla i přenos jednotlivých bitů. Lze tak zpracovávat jednobitové proměnné.

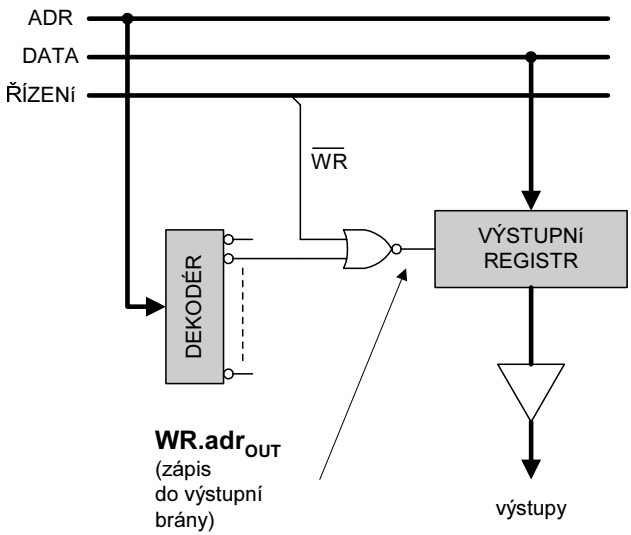
Princip vstupních obvodů, umožňujících čtení vstupního slova jako celku, ukazuje *obr. 9.1*. Vstupní signály jsou zavedeny na datovou sběrnici přes třístavové členy. Okamžik připojení je dán současným výskytem čtecího impulzu $\overline{RD}$ a výběrového signálu z adresového dekodéru. Způsob dekódování adres závisí na architektuře počítače a není z obrázku patrný. Každý vstupní a výstupní obvod má svoji adresu v patřičném adresovém prostoru počítače. Na *obr. 9.1* je čárkovaně vyznačeno možné umístění vstupního registru. Většina paralelních vstupních obvodů tento registr nemá, v některých případech je však nutný. O tom pojednává další text.

Výstupní obvody musí být bezpodmínečně opatřeny registry, neboť data na datové sběrnici jsou vydávána jen pro část taktu sběrnicového cyklu. Výstupní



Obr. 9.1 Jednoduché paralelní vstupní obvody

signály registru jsou před vyvedením z pouzdra výkonově zesíleny. Zápis do registru je dán současným výskytem zápisového impulzu $\overline{WR}$ a výběrového signálu z adresového dekodéru. Princip ukazuje *obr. 9.2.*



Obr. 9.2 Jednoduché paralelní výstupní obvody

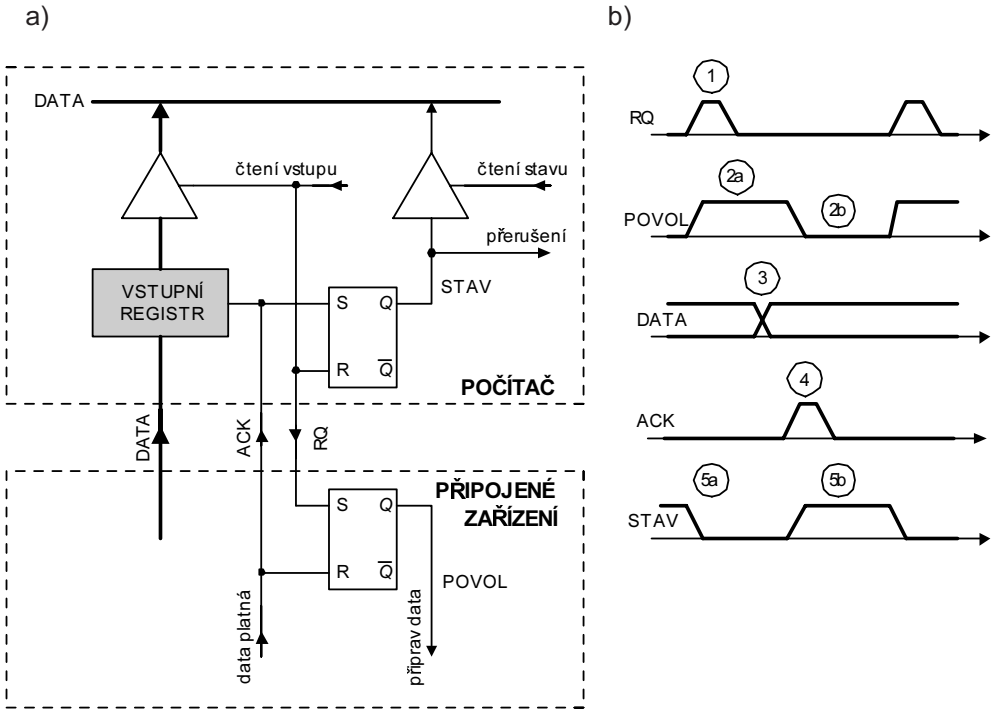
U vstupních obvodů nejsou registry bezpodmínečně nutné, pokud je zaručeno, že vstupní signály budou konstantní pro dobu sběrnicového cyklu čtení. Pokud tato podmínka není splněna, může dojít ke čtení vstupů během jejich změn. To je všeobecně nežádoucí situace. Méně závažný je však případ, kdy vstupní data mají význam vzájemně nezávislých jednobitových informací – např. je čten stav jednotlivých kontaktních spínačů v technologickém zařízení, apod. V tomto případě vznikne nejistota, zda změna stavu daného signálu bude zaregistrována již v současném, nebo až v příštím čtení vstupů. Závažnost této nejistoty bude záležet na celkové periodě programu a nárocích na přesnost časování, ale všeobecně se nejedná o katastrofální chybu. Jiný případ nastává tehdy, kdy vstupní data mají význam vícebitového (8bitového) čísla. Pak při nejistotě v okamžiku změn jednotlivých bitů čísla může být přečteno číslo zcela falešné a chyba může být katastrofální.

Tento problém lze odstranit po doplnění vstupních obvodů o registr, jehož zápisový impulz je vhodně časován. Impulz ale nemůže být generován procesorem tak jako u výstupních obvodů, neboť okamžik změny vstupních signálů není předem známý a nelze jej tedy synchronizovat s činností procesoru. Ani na vstupech vloženého registru totiž nelze připustit změny signálů v okamžiku zápisového impulzu. Celý problém by se tak vlastně jen přesunul z datové sběrnice procesoru na vstupy zmíněného registru.

Vzájemnou synchronizaci počítače s připojeným zařízením řeší doplňkové obvody podle *obr. 9.3a*. Termínem „připojené zařízení“ budeme označovat zařízení zcela obecné, bez ohledu na jeho charakter. Zařízení potvrzuje platnost svých předávaných dat impulzem *ACK* (angl. acknowledge – potvrzení), kterým se data zapíší do registru ve vstupní bráně počítače. Počítač vyžaduje nová data impulzem *RQ* (angl. request – požadavek). Brána je doplněna o pomocný klopný obvod, jehož stav může procesor číst jako jeden z bitů stavových registrů. Výstupem klopného obvodu lze též generovat přerušení. Připojené zařízení je doplněno o pomocný klopný obvod, který povoluje či blokuje vydání nových dat.

Tento způsob vzájemné synchronizace mikropočítače s připojeným zařízením, kdy se používají signály s významem „požadavek – potvrzení“ se nazývá „korespondenční provoz“ (angl. „handshake“). Zabraňuje čtení dat, která se současně mění, opakovanému čtení a zpracování starých dat, a ztrátě dat. K opakovanému čtení dat už jednou zpracovaných nemůže dojít, neboť program je konstruován tak, že vstupní registr přečte jen tehdy, když pomocný klopný obvod v bráně počítače je ve stavu 1. Při čtení dat ze vstupního registru se pomocný klopný obvod brány automaticky nuluje a zůstane tak, dokud nebyla dodána nová data. Ke ztrátě dat nemůže dojít, neboť pomocný klopný obvod v připojeném zařízení blokuje vydání nových dat do té doby, než je vstupní registr brány přečten. Při potvrzení dat je tento klopný obvod automaticky nulován. K vyvolání obsluhy může být využito přerušení programu – požadavek je odvozen od zápisu nových dat do vstupního registru. V obslužném podprogramu je pak při přečtení registru požadavek na přerušení automaticky nulován.

Stejný problém se vzájemnou synchronizací nastává u výstupu dat z počítače. V tomto případě nesmí připojené zařízení číst výstupní data v libovolném oka-



Obr. 9.3 *Vstupní brána s pomocným klopným obvodem: a) schéma zapojení; b) časový diagram – **jednotlivé akce očíslovány:** 1 – počítač přečte data ze vstupního registru a tím generuje signál RQ jako žádost o nová data; 2a – pomocný obvod v připojeném zařízení se překlopí do stavu 1 a tím se povolí generace nových dat; současně se překlopí pomocný klopný obvod v bráně počítače do stavu 0 a tím je programu signalizován zákaz dalšího čtení ze vstupního registru (5a), dokud do něj nebudou zapsána nová data; 3 – zařízení vydá nová data; 4 – zařízení potvrzuje platnost dat signálem ACK; 5b – je překlopen pomocný klopný obvod v bráně počítače do stavu 1 a tím je programu signalizováno povolení dalšího čtení ze vstupního registru; současně se překlopí pomocný klopný obvod v připojeném zařízení do stavu 0 a tím je v zařízení dočasně zablokováno vydání dalších dat (2b)*

mžiku. Princip korespondenčního provozu a potřebné doplňkové obvody lze odvodit z *obr. 9.3* jednoduše po vzájemné záměně bloků PŘIPOJENÉ ZAŘÍZENÍ a POČÍTAČ.

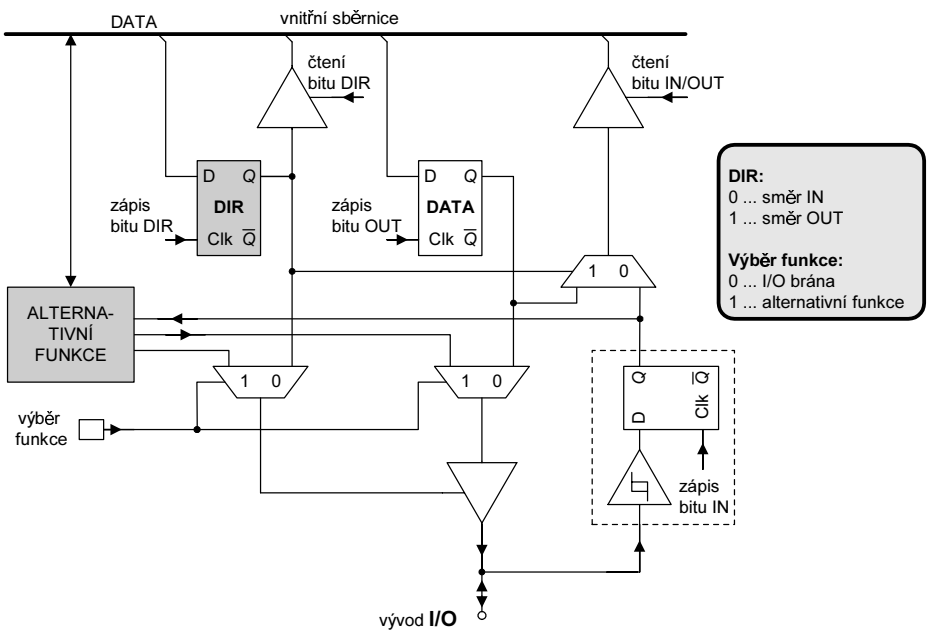
V praktických realizacích (např. rozhraní Centronics pro připojení tiskárny k počítači) jsou signály *ACK* a *RQ* z *obr. 9.3* mnohdy označeny jinými názvy a jsou případně negovány. Důvodem negací je zabezpečení připojeného zařízení proti nechtěnému spuštění při náhodném rozpojení konektoru spojovacího kabelu. Pokud byly obvody realizovány v technologii TTL, choval se rozpojený vstup jako by byl ve stavu 1 a řídicí signály pak musely být aktivní ve stavu 0. Tato konvence se dodržuje stále i při nyní převládající technologii CMOS.

Jednočipové mikropočítače nejsou běžně vybaveny doplňkovými obvody pro korespondenční provoz. Je-li to nutné, mohou se pro vytvoření signálů *RQ* a *ACK* využít vstupy a výstupy další standardní brány, ovládané programově.

U jednočipových mikropočítačů jsou brány zpravidla konstruovány jako dvousměrné (angl. bidirectional). Tím se zvyšuje univerzalita, ale roste složitost obvodů. Vedle výstupního datového registru patří k bráně ještě registr směru (DIR). Každý bit dat může být nezávisle na ostatních bitech určen jako vstupní nebo výstupní, v závislosti na příslušném bitu DIR – zpravidla stavem 0 je určen směr „vstup“ (IN), stavem 1 „výstup“ (OUT). Změna směru je možná i v běhu programu. Ke složitosti obvodů přispívá dále nutnost vícenásobného využití vývodů pouzdra, kdy kromě funkce brány mohou vývody sloužit ještě pro jiné alternativní funkce – mnohdy i dvě další. Přiřazení vývodu buď k bráně nebo k jiným obvodům je ovládáno některým řídicím registrem mikropočítače.

Registr směru je zásadně nastavován na směr IN při nulování počítače – tím se zabrání tomu, aby bit brány nebyl náhodně nastaven na směr OUT, ačkoliv byla předpokládána jeho funkce jako IN. Mohlo by tak dojít k poškození obvodů v bráně nebo ve zdroji vstupního signálu.

Detailnější pohled na obvody pro jeden bit dvousměrné brány ukazuje *obr. 9.4* Pro výstup slouží jeden klopný obvod datového registru (OUT), pro řízení směru jeden bit registru směru (DIR). Pro směrování jednotlivých signálů jsou v bráně přepínače (multiplexery). Nejprve předpokládejme, že bit „výběr funkce“ je ve stavu 0



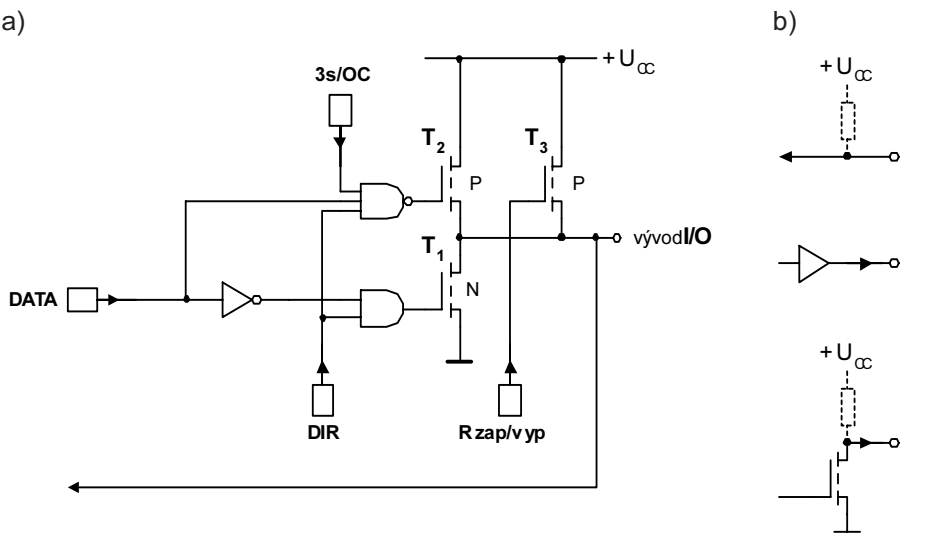
Obr. 9.4 Obvody pro jeden bit dvousměrné vstupní a výstupní brány

a vývodu I/O pouzdra je přiřazena funkce „brána“. Při DIR = 0 je výstupní třístavový obvod zablokován a je čten stav na vývodu I/O brány, při DIR = 1 je výstupní třístavový obvod odblokován a signál z datového registru se přivádí na vývod I/O. Je-li bit „výběr funkce“ je ve stavu 1, je ovládání směru i generace výstupního signálu plně pod kontrolou obvodů v bloku „ALTERNATIVNÍ FUNKCE“. Vstupní signál může být u dokonalejších architektur upraven (viz blok v čárkovaném rámečku). Obvod s hysterezí umožňuje zpracování i pomalu proměnných vstupních signálů. Následující klopný obvod vzorkuje vstupní signál s malým předstihem před vlastním čtením bitu IN/OUT, takže na datovou sběrnici se vždy dostává signál konstantní po dobu čtení. Toto uspořádání snižuje nároky na vlastnosti vstupního signálu, nezaručuje však vzájemnou synchronizaci se zdrojem signálu a počítačem. Nejedná se o doplňkové obvody pro korespondenční provoz.

Do registru směru lze zapisovat a lze jej pro kontrolu i číst. Do datového registru lze zapisovat a při směru OUT jej lze číst. To je důležité pro ovládání jednotlivých bitů výstupní brány. Procesor přečte obsah celého výstupního registru, s obsahem provede logickou operaci buď AND nebo OR s maskou, a výsledek vrátí do výstupního registru. Pokud se má např. bit D_4 výstupní brány nastavit do stavu 1, provede se operace OR s 00010000. Pokud se má bit D_4 vynulovat, provede se operace AND s 11101111. Celá operace je poněkud delší než prostý zápis, nevyžaduje však složité dekódovací obvody až na úroveň jednotlivých bitů.

Zapojení výstupního třístavového obvodu s možností volby několika režimů ukazuje *obr. 9.5a*.

Při DIR = 1 a 3s/OC = 1 se obvod chová jako dvoustavový, nenegující, s výkonovými tranzistory T_1 a T_2 (s malým odporem kanálu) na výstupu – tedy zdroj napětí U_L nebo U_H. Při DIR = 0 jsou oba tranzistory zavřeny a výstup je ve vyso-



Obr. 9.5 Výstupní obvod brány: a) schéma zapojení; b) náhradní schéma

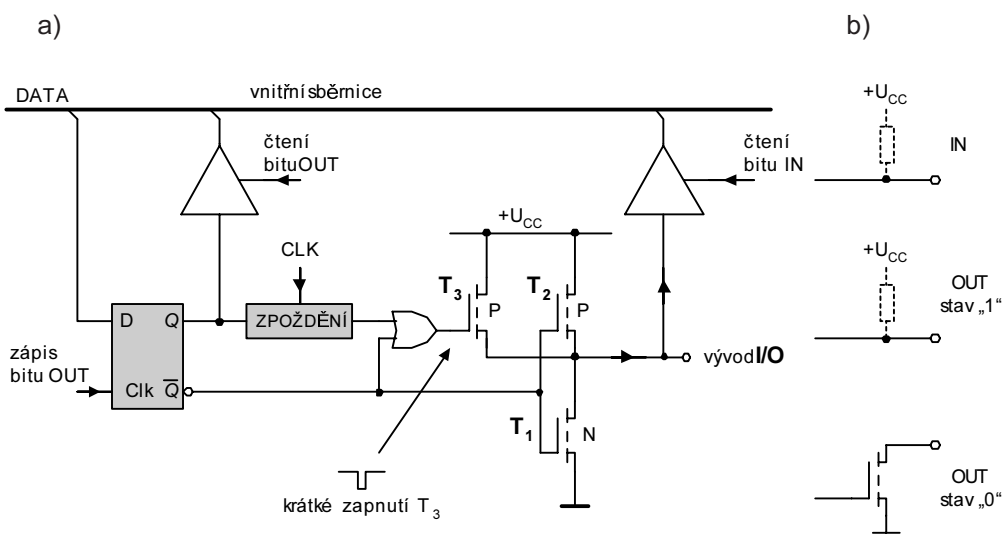
426410

koimpedančním stavu – vývod pouzdra I/O může být využit jako vstup. Při 3s/OC = 0 není T_2 nikdy vybuzen a výstup se tedy chová jako obvod s otevřeným kolektorem. Bit Rzap/vyp zapíná tranzistor T_3 s velkým odporem kanálu (desítky kΩ). Ten má funkci kolektorového rezistoru při konfiguraci výstupního obvodu jako otevřený kolektor. Při konfiguraci obvodů jako vstup může T_3 reprezentovat rezistor pro vnější obvody s otevřeným kolektorem. Náhradní schéma obvodů ukazuje *obr. 9.5b*. V některých případech existuje ještě další tranzistor mezi výstupem a zemí, ovládaný dalším řídicím bitem. Bity 3s/OC a Rzap/vyp existují jen ve vyspělých architekturách a jsou pak součástí některého z řídicích registrů periferního systému.

Celá brána, zahrnující obvody dle *obr. 9.4* a *obr. 9.5*, je značně složitým celkem. Někteří výrobci proto dali přednost jednoduchému zapojení částečně dvousměrné – „kvazi-dvousměrné“ (angl. quasi-bidirectional) – brány. Její zapojení je na *obr. 9.6a*.

Tranzistor T_1 v koncovém stupni má malý odpor kanálu a tím zajišťuje malý výstupní odpor ve stavu 0. Naopak T_2 má velký odpor kanálu (desítky kΩ) a tím je dán i výstupní odpor ve stavu 1. V tomto stavu nelze výstup brány zatěžovat – jako zátěž se předpokládají obvody CMOS nebo bipolární s malou spotřebou (LS, ALS). V důsledku velkého odporu T_2 je vlivem parazitních kapacit zpožďován přechod ze stavu 0 do 1. Pro urychlení přechodného děje je doplněn tranzistor T_3 s malým odporem kanálu (stovky Ω), zapínaný jen krátkodobě (na zlomky μs) při překlopení klopného obvodu v datovém registru brány ze stavu 0 do 1. Časovací obvod (blok ZPOŽDĚNÍ) je synchronního typu a využívá hodinové impulzy procesoru.

Brána nemá registr směru. Má-li mít vývod funkci vstupu, zapíše se předem do příslušného bitu datového registru hodnota 1. Tím je T_1 vypnut a T_2 se jeví

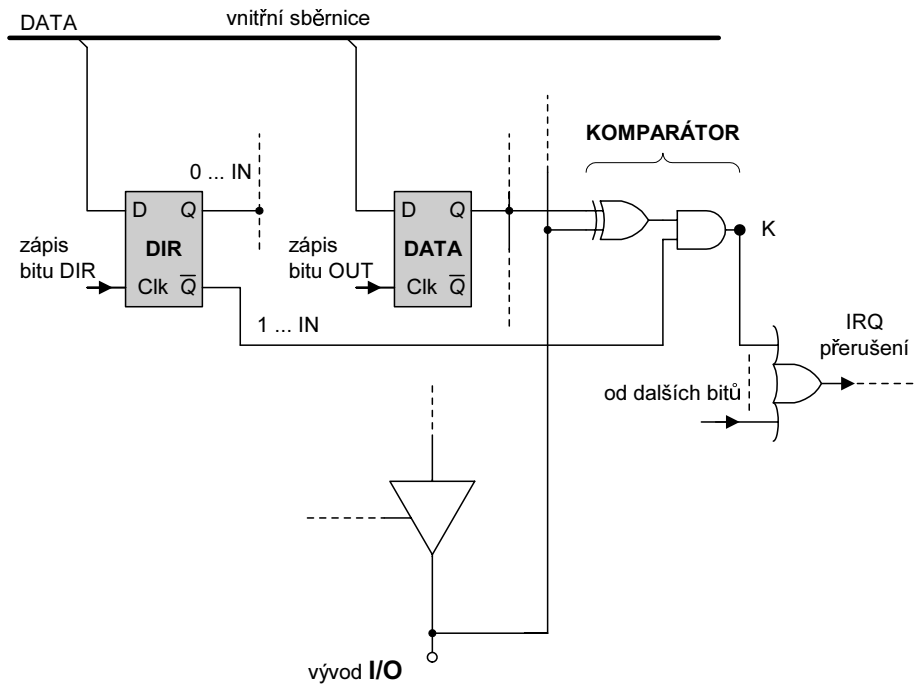


Obr. 9.6 Kvazi-dvousměrná brána: a) schéma zapojení; b) náhradní schéma

426410

jako rezistor o velké hodnotě mezi vývodem a napájecím napětím. Jeho přítomnost není pro převážnou většinu zdrojů vstupního signálu na závadu. *Obr. 9.6b* ukazuje náhradní zapojení ve všech stavech a režimech jednoho vývodu brány. Při nulování počítače jsou všechny bity datového registru nastaveny na stav 1, tedy směr IN. Výhodou kvazi-dvousměrné bránu je jednoduchost obvodů, nevýhodou jsou horší elektrické vlastnosti a náchylnost k rušení ve stavu 1 vlivem vysokého výstupního odporu.

Velmi užitečným, i když ne příliš častým doplňkem obvodů brány je detektor změn stavu na vstupech. Princip ukazuje *obr. 9.7*.



Obr. 9.7 Detektor změn vstupního signálu

Jedná se o jednobitový komparátor, vyhodnocující obvodem EX-OR neshodu mezi stavem na vývodu brány a na výstupu datového registru. V bodě *K* je stav 1 jen při neshodě a při směru brány IN. Signály *K* ze všech bitů brány jsou logicky sečteny a výsledný signál je zaveden do řadiče přerušení. Detekce změn stavu pak funguje tak, že nejprve program přečte stav na vývodech, nastavených na směr IN. Tentýž stav se pak zapíše do výstupního registru. Od tohoto okamžiku obvody reagují na jakoukoliv změnu na vstupech brány a při změně vyvolají přerušení.

Obvody, připojené na výstupy bran, mohou způsobit významný vzrůst výkonové ztráty celého integrovaného obvodu. To se sice obecně týká všech výstupů,

426410

paralelní brány však reprezentují zdaleka největší počet vývodů a jejich vliv je tak nejpodstatnější. Maximální výkonová ztráta celého obvodu vyplývá z povolené vnitřní teploty přechodů PN. K výpočtu slouží jednoduchý vztah

$$T_J = T_A + P_D \cdot \Theta_{JA}$$

kde T_J je vnitřní teplota přechodů, T_A je teplota okolí pouzdra, P_D je celková výkonová ztráta obvodu, Θ_{JA} je tepelný odpor mezi přechodem a povrchem pouzdra – jeho hodnotu vždy uvádí výrobce. Teplota okolí pouzdra se může vlivem nedostatečné ventilace v přístroji významně lišit od teploty v místnosti. Pro celkovou výkonovou ztrátu obvodu P_D platí vztah

$$P_D = P_{INT} + \Sigma P_{OUT}$$

kde P_{INT} je výkonová ztráta všech vnitřních obvodů (nezávislá na výstupních proudech bran) a ΣP_{OUT} značí dodatečnou výkonovou ztrátu, vzniklou výstupními proudy bran. Do sumy se započítají všechny vývody nastavené na směr OUT. Pokud je výstup ve stavu 0, je jeho příspěvek $U_{OL} \cdot I_{OL}$; je-li ve stavu 1, je jeho příspěvek $(U_{CC} - U_{OH}) \cdot I_{OH}$.

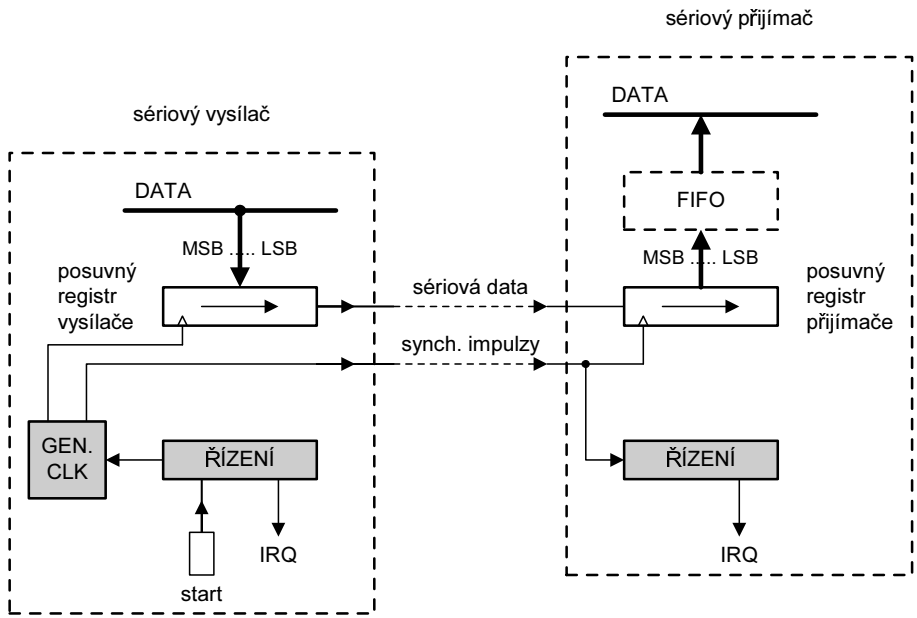
V dokumentaci výrobce jsou vždy uváděny maximální proudy I_{OL} a I_{OH}, při kterých jsou ještě zaručeny správné hodnoty výstupních napětí U_{OL} a U_{OH}. Dále je uváděn maximální proud, který ještě nezpůsobí trvalou poruchu obvodu. Tímto proudem smí být zatížen vždy jen jeden výstup. Správné hodnoty výstupních napětí přitom samozřejmě nejsou zaručeny.

9.3 SÉRIOVÉ VSTUPNÍ A VÝSTUPNÍ OBVODY

Sériové vstupní a výstupní obvody přenášejí data bit po bitu. Vyskytují se obě možnosti pořadí bitů – nejnižší bit napřed a nejvyšší bit napřed. Data jsou přenášena od sériového vysílače do sériového přijímače. Základní částí sériových obvodů je posuvný registr, do kterého se při příjmu s každým hodinovým impulzem vkládá jeden bit dat a současně se uložená data posunují o jednu pozici. Kromě posouvání obsahu umožňuje posuvný registr vysílače ještě paralelní zápis celého slova. Obdobně posuvný registr přijímače umožňuje i paralelní výstup celého slova. Jednoduché uspořádání pro sériový přenos ukazuje *obr. 9.8*.

Jádrem vysílače i přijímače je posuvný registr. Ve vysílači do něj lze paralelně (tj. všechny bity současně) zapsat celé slovo a následně je posunovat hodinovými impulzy. Z výstupu posuvného registru jsou data v sériovém kódu předávána přijímači. Pořadí bitů závisí na pozici nejvyššího bitu (MSB – Most Significant Bit) z datové sběrnice. Při MSB vlevo a nejnižším bitu (LSB – Least Significant Bit) vpravo, jak je vyznačeno na *obr. 9.8*, je nejprve vysílán LSB. Právě tak je možné i obrácené pořadí při opačné orientaci MSB a LSB. V přijímači data postupně zaplní posuvný registr a celé slovo je pak možné číst prostřednictvím datové sběr-

nice. Mezi posuvný registr a datovou sběrnici je zpravidla vložen alespoň jeden vyrovnávací datový registr, nebo jejich skupina organizovaná jako paměť fronty FIFO (angl. First In – First Out). Pokud se posuvný registr zaplnil a není včas programem přečten, může být jeho obsah přepsán dalšími sériovými daty a tudíž ztracen. Při přenosu znak za znakem bez časové prodlevy zbývá na přečtení posuvného registru málo času – jen zlomek taktu. Vyrovnávací registr, do kterého je obsah zaplněného posuvného registru vždy automaticky přepsán, může být čten po celou dobu během příjmu nového znaku, tedy podstatně déle. Vyrovnávací paměť FIFO, sestavená z několika desítek registrů, tuto toleranci ještě zvyšuje. Program pak může obsluhovat přijímač jen jednou za čas, kdy najednou přečte obsah celé vyrovnávací paměti. Někdy jsou vyrovnávací registry použity i u vysílačů. Zde sice nehrozí ztráta dat, ale při rychlém vysílači, kterému program nestačí dodávat data, dochází ke zbytečnému zpoždění komunikace. Vyrovnávací paměť je zaplněna celou delší zprávou jen jednou za čas.



Obr. 9.8 Obvody pro sériový přenos

9.3.1 Synchronizace přijímače s vysílačem

Při sériových přenosech hraje důležitou roli vzájemná synchronizace přijímače s vysílačem. Synchronizační impulzy dělí čas na jednotlivé takty. U sériových periferních obvodů mikropočítačů se přenáší jeden bit v jednom taktu. V telekomunikačních systémech se prostřednictvím vhodné modulace (vícestavové mo-

dulace) přenáší zpravidla více bitů v jednom taktu. Počet taktů za sekundu, tj. kmitočet synchronizačních impulzů, udává **modulační rychlost** v_m. Jednotkou je jeden Bd (Baud), což je jeden takt za sekundu. Odvozenou jednotkou je kBd. Modulační rychlosti jsou normalizovány. Prakticky nejnižší používaná je 1,2 kBd. Další jsou vždy dvojnásobky (výjimečně též poloviny), takže 2,4 kBd, 4,8 kBd, 9,6 kBd, 19,2 kBd, atd. Počet informačních bitů za sekundu udává **přenosová rychlost** v_p. Informační bity jsou jen ty bity, ve kterých se skutečně přenáší informace a nepočítají se k nim bity, které slouží k jiným účelům (např. k označení začátku a konce zprávy, apod.). U sériových periferních obvodů je tak běžně $v_p < v_m$, nerovnost pak $v_p > v_m$ platí u systémů s vícestavovou modulací. Je třeba poznamenat, že v anglické literatuře se tyto pojmy důsledně nerozlišují a používá se jednotka „bps“ (bit per second). Teprve za kontextu je zřejmé, o co se jedná.

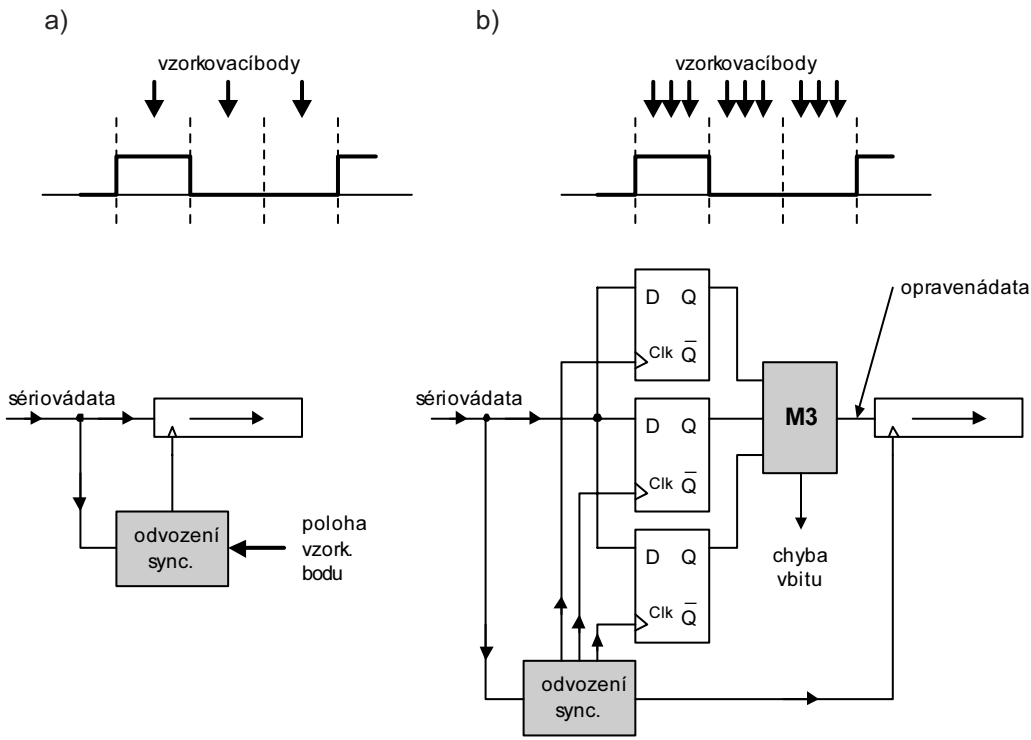
Na *obr. 9.8* je synchronizace zajištěna tak, že vedle sériových dat jsou přenášeny i synchronizačních impulzy, generované v obvodech vysílače. Tento způsob vede na jednoduché obvody přijímače, jsou ale zapotřebí dvě spojovací vedení, což zvyšuje cenu. Používá se proto jen pro sériové přenosy mezi obvody v malých vzdálenostech, nejčastěji na jedné desce plošného spoje.

Synchronizační impulzy lze též vytvořit až na místě přijímače. Obvody přijímače jsou pak složitější, odpadne však jedno spojovací vedení. Tento způsob je všeobecně používán pro přenosy na větší vzdálenosti. Přijímač si může vytvářet synchronizační impulzy dvěma způsoby: buď svým vlastním oscilátorem naladěným na stejný kmitočet jako má oscilátor vysílače, nebo odvozením synchronizačních impulzů od přenášených datových impulzů. První způsob je jednodušší, nezaručuje však dlouhodobou shodu kmitočtu oscilátoru vysílače a přijímače. Délka přenášených dat je proto podstatně omezena, v praxi nejčastěji na 7 až 8 bitů. Typicky se přenášejí jednotlivé slabiky (8 bitů), pro přenos znaků v nerozšířeném kódu ASCII stačí 7 bitů. Synchronizace se nazývá **znaková**. Druhý způsob odvozuje synchronizační impulzy od změn (hran) v datovém signálu. Přenášená data však mohou obsahovat úseky, v nichž nedochází ke změně – dlouhé sekvence samých nul nebo samých jedniček. Pak by scházela informace pro odvození synchronizačních impulzů. Tento problém se řeší buď vhodným kódováním dat ve vysílači tak, aby při každém taktu docházelo ke změně, nebo doplněním přijímače o obvody fázového závěsu. Fázový závěs se chová analogicky se setrvačníkem v mechanických soustavách a může tak tolerovat několik chybějících hran signálu. Výhodou odvození synchronizačních impulzů od datového signálu je možnost přenášení dat o principiálně neomezené délce. Synchronizace se nazývá **bitová**.

Běžné sériové periferní obvody jednočipových mikropočítačů data nijak nekódují a pracují se signálem v podobě NRZ (Non-Return to Zero) – tomu odpovídá i *obr. 9.8*. Synchronizační impulz na straně přijímače může v nejjednodušším případě data přímo zapisovat do posuvného registru. Vhodný okamžik zápisu – tzv. vzorkovací bod – je přibližně v polovině trvání bitu a je zřejmé, že synchronizační

impulzy vysílače a přijímače pak musí být vzájemně fázově posunuty. V jednoduchých periferních obvodech je tento posuv realizován v generátoru impulzů ve vysílači (jako na *obr. 9.8*). Ve složitějších obvodech, odvozujících synchronizaci od datových impulzů, jej lze nastavovat v přijímači. Je dokonce možné stanovit v jednom taktu postupně několik vzorkovacích bodů a vzorky porovnat. Neshoda signalizuje nespolehlivost přenosu (zpravidla vlivem rušení), může ale být opravitelná. Princip vzorkování s jedním a se třemi vzorkovacími body ukazuje *obr. 9.9*.

Při vzorkování se třemi vzorkovacími body jsou generovány v každém taktu postupně tři synchronizační impulzy, kterými jsou zapisovány vzorky do tří klopných obvodů. Za nimi následuje impulz pro zápis do posuvného registru. Majoritní obvod M3 má výstupní stav shodný se stavem na většině vstupů (realizuje „většinové hlasování“). Tak lze opravit chybu v jednom vzorkovacím bodě, ale dvě nebo tři chyby již opravit nelze. Pokud nejsou hodnoty všech vzorků shodné, je generován signál „chyba v bitu“ (angl. „bit error“). Tím je upozorněno na nespolehlivost přijatých dat.

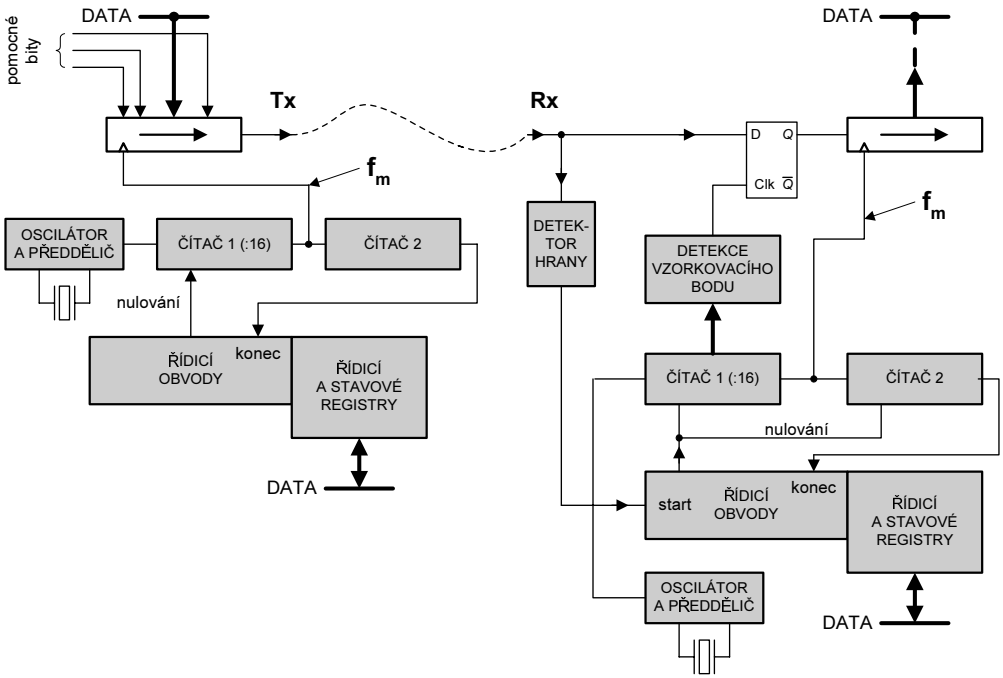


Obr. 9.9 Vzorkování vstupního signálu: a) jednoduché; b) vícenásobné

426410

9.3.2 Znaková synchronizace

Vytváření synchronizačních impulzů v přijímači je relativně jednoduché u znakové synchronizace. Vysílač i přijímač mají své oscilátory, pracující na (téměř) shodných kmitočtech. Oscilátory jsou zásadně řešeny jako krystalové s následujícím děličem kmitočtu pro dosažení požadované modulační rychlosti. Shodnost kmitočtů však nestačí – synchronizační impulzy vysílače a přijímače musí mít též správnou fázi. Protože krystalový rezonátor nelze přinutit ke změně fáze, musí být správná fáze nastavena v čítačovém děliči kmitočtu v přijímači – viz *obr. 9.10*.



Obr. 9.10 Znaková synchronizace ve vysílači a přijímači

V levé části *obr. 9.10* je vysílač (výstup *Tx*), v pravé je přijímač (vstup *Rx*). U obou shodně je modulační rychlost (kmitočet f_m) dána rezonančním kmitočtem krystalu, dělicím poměrem předděliče, a dělicím poměrem čítače 1. V klidovém stavu je čítač 1 zastaven (např. nulováním). Při zahájení vysílání je čítač ve vysílači spuštěn a data jsou bit za bitem vysouvána z posuvného registru. Čítač 2 odpočítává takty (periody f_m) a po jejich předvoleném počtu zastaví prostřednictvím řídicích obvodů další vysílání. V přijímači je čítač 1 spuštěn tehdy, když detektor hrany zaregistruje změnu stavu na vstupu přijímače, což nastane při vyslání prvého bitu. Tento bit – tzv. „start bit" – nepatří k datům a slouží výhradně k synchronizaci. Aby nedošlo vlivem případného rušení k mylnému zahájení příjmu, je start bit

vzorkován pro kontrolu ještě uprostřed taktu a nemá-li požadovanou hodnotu, vrací se řídicí obvody do počátečního stavu. Současným spuštěním obou čítačů ve vysílači i v přijímači je zaručena přibližně shodná fáze impulzů na výstupech čítače 1 u vysílače i přijímače. Přibližná proto, že do fáze oscilátoru nelze zasáhnout. Tím vzniká chyba fáze, která odpovídá maximálně jedné periodě na vstupu čítače 1 a je tedy tím menší, čím větší je dělicí poměr čítače 1. Ten bývá volitelný (např. 16 a 64). Obdobně jako u vysílače, i u přijímače je čítačem 2 odpočítán patřičný (předvolený) počet taktů a pak je příjem ukončen. Data jsou k dispozici v posuvném registru, případně ve vyrovnávacím registru či paměti FIFO (viz *obr. 9.8*). Ze stavu čítače 1 je přibližně v polovině periody f_m odvozen vzorkovací impulz a vzorek je dočasně uchován v klopném obvodu. Na konci periody je pak vsunut do posuvného registru. Je samozřejmě možné i vícenásobné vzorkování dle *obr. 9.9b*.

Po zahájení příjmu až do jeho ukončení již nedochází k žádné korekci synchronizace přijímače a případné tolerance v naladění obou oscilátorů se mohou časem projevit tak, že se vzorkovací bod posune z původního středu taktu až za jeho kraj. Tento způsob synchronizace je proto vhodný jen pro přenos malého počtu bitů, typicky kolem 10. Přípustný rozdíl kmitočtů oscilátorů pak činí 5 %, a s mírnou rezervou se vyžaduje max. 4 %. Tato velká tolerance je užitečná pro aplikace jednočipových mikropočítačů, jejichž částí jsou sériové periferní obvody. Jako oscilátor pro tyto periferie zde slouží základní oscilátor mikropočítače s vnitřními předděliči – ty jsou buď přímo součástí sériového přijímače a vysílače, nebo se využívá jedna ze sekcí čítačů/časovačů v režimu programovatelného generátoru impulzů. Může nastat situace, že při použitém krystalu základního oscilátoru nelze nalézt vhodný dělicí poměr pro přesné nastavení modulační rychlosti. Velká tolerance pak může problém vyřešit.

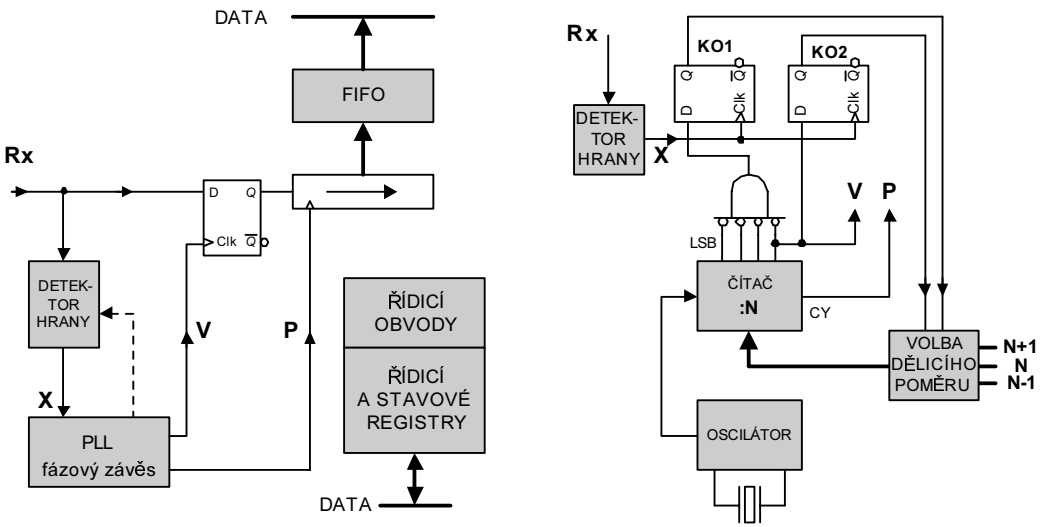
9.3.3 Bitová synchronizace

Synchronizační impulzy jsou vytvářeny v přijímači fázovým závěsem PLL (angl. Phase Locked Loop). Jednou z hlavních vlastností PLL, která je zde využívána, je schopnost dále generovat výstupní impulzy i při absenci vstupních impulzů. Absence vstupních impulzů nastává běžně při kódování NRZ a posloupnosti bitů o stejné hodnotě (samé 0 nebo samé 1). Princip bitové synchronizace ukazuje *obr. 9.11a*.

V ustáleném stavu je v bodě X kmitočet shodný s body V (vzorkovací impulz) a P (zápisový impulz do posuvného registru). Je samozřejmě možné i vícenásobné vzorkování dle *obr. 9.9*. Vstupní impulzy pro PLL jsou v detektoru hrany odvozeny od změn signálu *Rx*. Některé přijímače umožňují zvolit pro synchronizaci náběžné hrany, doběžné hrany, nebo i obě. Detektor může být po část taktu blokován (vyznačeno čárkovaně), což snižuje vliv případného rušení na synchronizaci.

Příklad fázového závěsu ve zjednodušeném zapojení ukazuje *obr. 9.11b*. Kmitočet oscilátoru je čítačem s proměnným dělicím poměrem vydělen tak, aby kmi-

točet vzorkovacích impulzů V na výstupu MSB čítače byl roven modulační rychlosti přenášených datových impulzů. Jeden takt je tak čítačem rozdělen na dílčí doby (segmenty), odpovídající jednotlivým stavům čítače. Vzorkovací bod je nastaven napevno na přechod ze stavu 0111 do 1000. K zápisu vzorku do posuvného registru dochází při přechodu z posledního stavu v cyklu čítače do 0000.



Obr. 9.11 Bitová synchronizace v přijímači: a) princip; b) detail fázového závěsu

Klopné obvody KO1 a KO2, ve spolupráci s obvodem volby dělicího poměru, řídí přeladování PLL. Pokud hrana Rx nastane při stavu čítače 0000, je PLL správně doladěn. Pokud se hrana vyskytne dříve, zřejmě se PLL opožďuje. Na nejvyšším bitu čítače je přitom jednička a ta se zapíše do KO2. Dělicí poměr se o jedničku zmenší, tím se zvýší výstupní kmitočet PLL. Pokud se naopak hrana vyskytne později, zřejmě se PLL předbíhá. Na MSB je nula a ta se zapíše do KO2. Dělicí poměr se o jedničku zvětší a tím se sníží výstupní kmitočet PLL. Při výskytu hrany ve stavu čítače 0000 je výstupu KO1 stav 1 a dělicí poměr čítače zůstává vždy nezměněn. Necitlivost tohoto typu PLL na polohu hrany, pokud se vyskytne kdykoliv během stavu čítače 0000, umožňuje pracovat s jistou tolerancí kmitočtu synchronizačních impulzů na straně vysílače a přijímače.

Pokud ve vstupním signálu nedojde ke změně, tj. chybí hrany, nemůže se změnit stav KO1 a KO2. Proto se nezmění ani dělicí poměr čítače a výstupní kmitočet PLL zůstává takový, jaký byl nastaven při poslední hraně. V důsledku neúplné shody mezi kmitočtem vysílače a PLL v přijímači tak může postupně narůstat fázová chyba, aniž by ji přijímač mohl korigovat. Výpadek hran je proto přípustný jen na omezenou dobu.

Tento problém se často řečí tzv. vsouváním bitů (angl. bit stuffing). Pokud se vyskytne posloupnost shodných bitů a délce větší než je povolená délka (např. 5 bitů), vysílač automaticky vkládá jeden bit opačné hodnoty. Přijímač sice vložený bit používá k synchronizaci, ale automaticky ho vyřazuje z posloupnosti datových bitů.

9.3.4 Ovládání sériového přijímače a vysílače

K řízení a informování o stavu sériového přijímače a vysílače slouží řídicí a stavové registry. I přes rozdíly v různých typech sériových přijímačů a vysílačů lze nalézt některé řídicí a stavové signály, které jsou všem společné.

Řídicí signály

U přijímače lze povolit či zakázat příjem. Vysílání je zpravidla povoleno trvale, zahajuje se automaticky při vložení dat do registru vysílače. Vysílač a přijímač mají vždy možnost vyvolat přerušení programu při dokončení vysílání či příjmu. Jednotlivá přerušení od vysílače i přijímače lze maskovat.

Stavové signály

U přijímače je stavovým bitem signalizováno ukončení příjmu dat, současně může být generována i žádost o přerušení. Dále jsou v několika stavových bitech signalizovány případné chyby v příjmu. U některých přijímačů mohou i tyto bity vyvolat přerušení. U vysílače je signalizováno ukončení vysílání dat, současně může být generována i žádost o přerušení.

Další řídicí a stavové bity již souvisejí se specifickými vlastnostmi jednotlivých vysílačů a přijímačů.

9.3.5 Spojení vysílačů a přijímačů

V nejjednodušším případě je spojen jeden vysílač s jedním přijímačem přímo, bez dalších obvodů. Takto lze spojit jen obvody v malé vzdálenosti, prakticky na jedné desce plošného spoje. Periferní obvody jsou vyráběny zásadně technologií CMOS a nejsou schopny dodávat dostatečné proudy do kapacitní zátěže, kterou by představovalo dlouhé vedení. Na větší vzdálenosti se proto vysílač i přijímač oddělují od vedení tzv. vazebními členy. Ty jsou schopny dodávat velké výstupní proudy a mohou pracovat s jinými napěťovými úrovněmi, než mají běžné obvody CMOS.

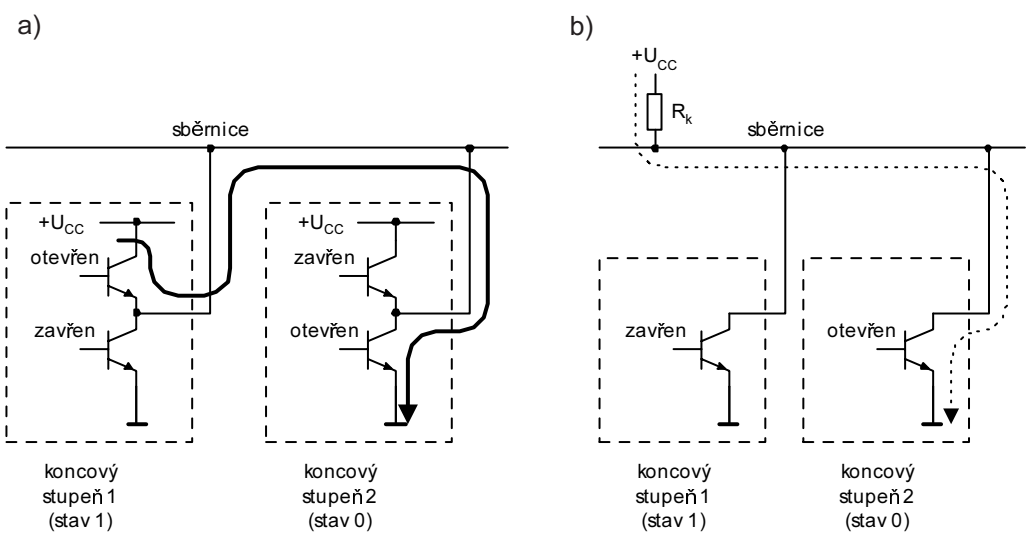
Síť vznikne spojením většího počtu uzlů, tvořených vysílači a přijímači. Může se jednat o seskupení přijímače s vysílačem (typicky mikrokontrolér), jen vysílač (např. inteligentní senzor, převodník A/Č doplněný patřičnými obvody, apod.), nebo jen přijímač (např. akční člen doplněný patřičnými obvody, apod.).

Velmi často se využívá síť s topologií sběrnice. Ta je konstrukčně jednoduchá a umožňuje snadné rozšiřování. Z hlediska elektrického obvodu se však jedná o spojení všech vysílačů a přijímačů do jednoho bodu. Problémem je paralelní

spojení vysílačů, neboť při současné aktivitě dvou nebo více vysílačů dojde ke kolizi. Její důsledky závisí na konstrukci výstupních obvodů vysílačů (jsou-li na sběrnici připojeny přímo), nebo jejich vazebních členů (jsou-li jimi od sběrnice odděleny). Výstupní obvody mohou být třístavové nebo s otevřenými kolektory – viz *obr. 9.12*.

Obr. 9.12a ukazuje spojení třístavových výstupních obvodů, přičemž jeden je ve stavu 0 a druhý ve stavu 1. Tlustou čárou je vyznačena smyčka vyrovnávacího proudu, který může ve velmi krátké době (milisekundy) způsobit přehřátí a zničení některých součástek ve smyčce. Pokud jsou výstupní obvody ve vysokoimpedančním stavu (vysílače neaktivní), jsou oba tranzistory zavřeny a zbytkové proudy jsou zcela zanedbatelné. Přístup všech vysílačů ke sběrnici musí být velmi spolehlivě řízen, aby nemohlo dojít ke kolizi.

Obr. 9.12b ukazuje spojení výstupních obvodů s otevřenými kolektory. Stav 0 je realizován otevřeným tranzistorem (zkratuje sběrnici na zem), stav 1 vypnutým tranzistorem, přičemž napětí U_H dodává kolektorový rezistor R_k. Jelikož stav 1 na sběrnici je zajišťován společným rezistorem R_k, je proud kterýmkoliv otevřeným tranzistorem maximálně roven U_{CC}/R_k. Konflikt tedy není destruktivní, i když samozřejmě naruší přenášená data. Přístup vysílačů ke sběrnici může být řízen tak, že tyto konflikty jsou dokonce využívány. Jestliže kterýkoliv z výstupních obvodů je ve stavu 0, bude na sběrnici 0 bez ohledu na ostatní obvody. Stav 0 je „dominantní“. Stav 1 však nastává jen tehdy, když žádný z výstupních obvodů není ve stavu 0. Stav 1 je „recesivní“.



Obr. 9.12 *Konflikt na sběrnici: a) při třístavových obvodech; b) při obvodech s otevřeným kolektorem*

V popisu činnosti uzlů se používá výraz „vstoupit“ či „připojit se“ na sběrnici, nebo naopak „odstoupit“ či „odpojit se“ od sběrnice. U třístavových výstupních obvodů vysílače (vazebních členů) se připojení či odpojení řídí výběrovým signálem $\overline{CS}$ (případně i jinak označeným). U výstupních obvodů s otevřenými kolektory stačí k odpojení vyvolat výstupní stav 1, žádný výběrový signál není zapotřebí. Uzel sítě, jehož vysílač má právo vstupovat na sběrnici a vysílat data, se nazývá „hlavní“ (angl. „master“) – bude označován symbolem **M**. Ten uzel, jehož vysílač smí vstoupit na sběrnici až na výzvu **M**, se nazývá „podřízený“ (angl. slave) – bude označován symbolem **S**. Je-li v síti jen jeden hlavní uzel (síť „monomaster“), je situace jednoduchá, neboť nemůže dojít ke konfliktům. **M** vstoupí na sběrnici, identifikuje **S**, se kterým chce komunikovat, a oznámí mu směr komunikace (**M** → **S** nebo **S** → **M**). Jedná-li se o směr **M** → **S,** následují ihned data pro identifikovaný **S**, který je přijme. Jedná-li se o směr **S** → **M,** odstoupí naopak **M** od sběrnice, na sběrnici se připojí identifikovaný **S** a vysílá data pro **M**, který data přijme. **S** pak ze sběrnice odstoupí. Identifikace uzlu sítě může spočívat ve výběrových signálech, generovaných **M** , a vedených po výběrových vodičích ke každému **S**. Tato metoda je evidentně vhodná jen pro nejkratší vzdálenosti, prakticky v rozsahu plošného spoje. Druhá metoda využívá adresu uzlu, vysílanou před daty. Každý uzel má jinou adresu a jen na ni reaguje – výjimkou může být adresa pro vysílání výzvy (angl. „broadcasting“), určené všem uzlům sítě.

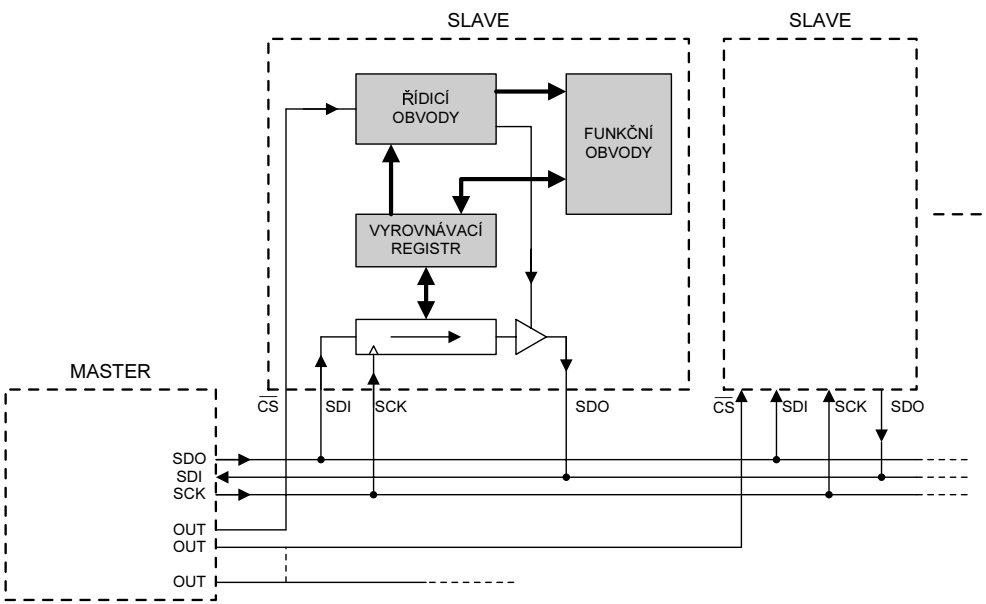
Existují též sítě typu „multimaster“, ve kterých je větší počet uzlů, schopných zahajovat vysílání (tedy typu **M**). V tomto případě by mohlo dojít ke konfliktům. Jsou-li výstupní obvody vysílače nebo vložených vazebních členů třístavové, musí být konflikt (v tomto případě destruktivní) vyloučen. Musí být proto velmi důkladně zajištěno, aby v síti v každém okamžiku existoval jen jeden aktivní uzel typu **M**. Je ovšem možné, aby uzel s momentální funkcí **M** se změnil na uzel s funkcí jen **S**, zatím co jiný uzel s momentální funkcí **S** se změní na uzel s funkcí **M**. Tato změna se nazývá „předání pověření“ (angl. token passing) a v sítích s topologií sběrnice je možná – síť je pak nazvána „token bus“. Tyto sítě mohou být po hardwarové stránce celkem jednoduché, řízení uzlů však vyžaduje rozsáhlý software.

Naopak při výstupních obvodech s otevřenými kolektory, kdy je konflikt nedestruktivní, může každý vysílač vstoupit na sběrnici kdykoliv. Může tím ovšem narušit již probíhající vysílání jiného uzlu. Přijímaná data jsou sice na straně přijímače kontrolována, jejich narušení bude přijímačem rozpoznáno a data budou odmítnuta, dojde však ke zdržení komunikace. Proto jsou využívány různé algoritmy arbitráže mezi vysílači. Velmi účinný je algoritmus CSMA/CD + AMP, využívaný v sítích CAN bus (viz další text). Zkratka pochází z anglického „Carrier Sense, Multiple Access/Collision Detection + Arbitration on Message Priority“ a dobře vystihuje použitý algoritmus. Všechny uzly trvale monitorují dění na sběrnici (zjišťují přítomnost datových impulzů). Uzel, který hodlá začít komunikaci, čeká se vstupem na sběrnici, pokud již komunikace probíhá. Pokud neprobíhá, uzel se připojí a vysílá, čímž vlastně blokuje sběrnici pro ostatní vysílače. Tento mechanizmus odvrací převážnou většinu konfliktů. Výjimečně však mohou na dosud volnou sběrnici vstoupit dva nebo i více vysílačů současně (přesně řečeno s ča-

sovým rozdílem menším než polovina taktu). Pak by byla narušena data vysílaná všemi vysílači. Pro tento případ existuje metoda arbitráže mezi vysílači. Spočívá v tom, že každým vysílačem je na začátku zprávy vysílán identifikátor, což je binární číslo, vysílané s MSB napřed. Každý uzel má svoji skupinu identifikátorů, rozdílných od identifikátorů ostatních uzlů. Každý uzel svým přijímačem průběžně kontroluje zprávu, kterou jeho vysílač vysílá, a v případě nesouhlasu vyvolá odstoupení vysílače od sběrnice. Jelikož výstupní obvody vysílače jsou s otevřeným kolektorem, kdy stav 0 je dominantní, nedojde k nesouhlasu u toho vysílače, který momentálně vysílal bit o hodnotě 0, a tento vysílač neodstoupí od sběrnice. Naopak dojde k nesouhlasu u všech vysílačů, které vysílaly bit o hodnotě 1, a tyto vysílače odstoupí od sběrnice. Vyšší prioritu zprávy zajistí identifikátor o nižší číselné hodnotě – stav 0 je totiž u něho vysílán dříve, než u identifikátorů s vyšší číselnou hodnotou.

9.3.6 Sériové vstupní a výstupní obvody SPI

Sériová periferie SPI (angl. Serial Peripheral Interface) je velmi jednoduchá a obsahuje ji převážná většina mikrokontrolérů. Využívá se pro spojení s blízkými obvody na jednom plošném spoji. Typicky se jedná o sériové paměti EEPROM, převodníky, různé senzory, hodiny reálného času, apod. Jednoduchost obvodů vyplývá z jednoduché synchronizace, kdy hodinové impulzy jsou generovány hlavním uzlem a jsou rozváděny ke všem podřízeným uzlům. Takové uspořádání ukazuje *obr. 9.13*.

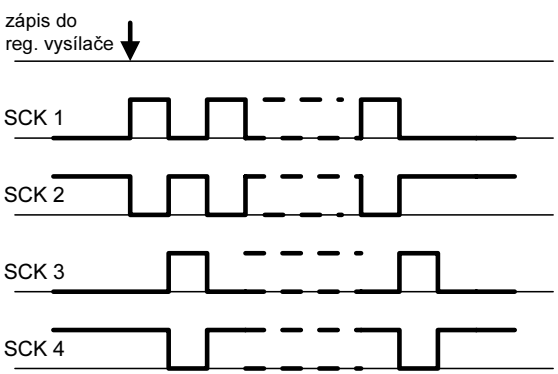


Obr. 9.13 Uspořádání obvodů SPI

Po třech společných vodičích se rozvádějí: synchronizační impulzy SCK, sériová výstupní data SDO, a sériová vstupní data SDI. Označení vývodů u **M** a **S** je vzájemně obrácené – co je SDI u **M,** je SDO u **S**, a co je SDO u **M**, je SDI u **S**. Zapojení vnitřních obvodů je shodné u všech uzlů, u **M** je navíc generátor synchronizačních impulzů. Před vlastním sériovým přenosem se nejprve vybere patřičný **S**. Výběr se děje výběrovými signály, generovanými **M**, a vedenými po výběrových vodičích ke každému **S**. Ke generaci výběrových signálů se využívají jednotlivé bity některé paralelní brány, programově ovládané. Tím odpadá nutnost přenášet v sériovém kódu adresu **S** a přenos se zkrátí. Okamžikem zápisu do registru vysílače začne přenos. Přenáší se osmice bitů s MSB napřed. Nejprve se od **M** přenáší příkaz pro vybraný **S**. Řídicí obvody **S** příkaz dekódují a pak provedou patřičnou akci s funkčními obvody (to je např. převodník, paměť, apod.). Jako druhé (případně i třetí) slovo se od **M** přenáší adresa pro práci s vnitřními funkčními obvody (např. pro paměť) – adresa může chybět, pokud funkční obvody žádnou nepotřebují (např. jednokanálový převodník). Naposled se přenášejí data ve směru, který vyplývá z předchozího příkazu. Jedná-li se o zápis, přenesou se data z **M** do **S**, jedná-li se o čtení, přenesou se data z **S** do **M** – i v tomto případě však SCK jsou generovány v **M**.

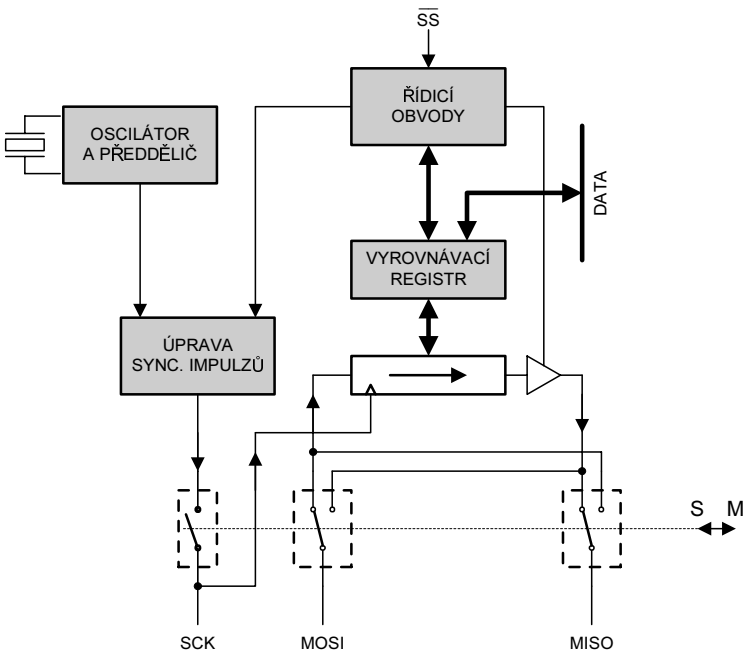
Obvody pro SPI jsou běžné a vyrábí je řada výrobců. Pro SPI není předepsán způsob řízení jednotlivých obvodů ani časování signálů – je nutné respektovat vlastnosti jednotlivých obvodů. Kmitočet synchronizačních kmitočtů se proto pohybuje od set kHz do několika MHz, rovněž průběh impulzů je různý. Detaily komunikace s jednotlivými vnějšími obvody jsou proto závislé na jejich skladbě. Na straně mastera (mikrokontroléru) to znamená programové nastavení kmitočtu i průběhu synchronizačních impulzů, obecně pro každý obvod jinak. Čtyři běžně používané varianty jejich průběhu jsou na *obr. 9.14*.

Ačkoliv periferie SPI nebyla původně určena pro sítě typu multimaster, moderní mikrokontroléry to v omezené míře umožňují. Pro změnu funkce z **M**



Obr. 9.14 Varianty synchronizačních impulzů SPI

na **S** a opačně je třeba obrátit smysl SDI a SDO, a vstup synchronizačních impulzů SCK změnit na výstup (a opačně). Aby vnější vodiče mohly zůstat připojeny na stále stejných vývodech pouzder, musí být jejich přepnutí realizováno na čipu. Vývody pak jsou jinak pojmenovány: vývod MISO značí Master In – Slave Out, vývod MOSI znamená Master Out – Slave In, SCK zůstává, a výběrový vstup se značí $\overline{SS}$ (Slave Select). Zapojení takovéhoto periferního obvodu ukazuje *obr. 9.15*. Přepínání signálů je samozřejmě realizováno elektronickými spínači. Změna funkce je možná programově jedním bitem řídicího registru periferie. V celé síti může v každém okamžiku být jen jeden uzel **M**, předávání funkce **M** v síti je plně pod kontrolou software.



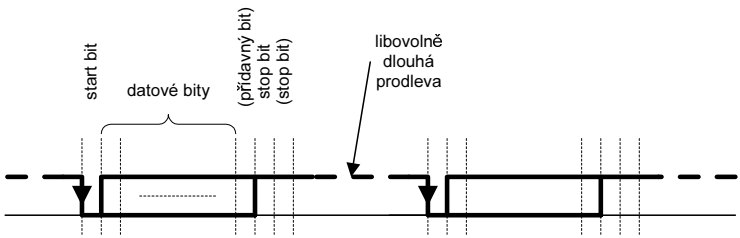
Obr. 9.15 Změna funkce M a S u SPI

9.3.7 Sériové vstupní a výstupní obvody UART

Sériová periferie UART (angl. Universal Asynchronous Receiver – Transmitter) se vyskytuje prakticky v každém počítači. Využívá asynchronní přenos jednotlivých znaků o délce 7 nebo 8 bitů. Synchronizace je znaková, jednotlivé znaky po sobě mohou následovat s libovolně dlouhou mezerou. Průběh signálu ukazuje *obr. 9.16*.

V klidovém stavu vydává vysílač stav 1, start bit má hodnotu 0. Za datovými bity následuje jeden či dva takty při stavu 1 – tzv. „stop bit" (bity). Stop bit umožňu-

je bezprostřední následování dalšího znaku, neboť je tak vždy zaručena změna z 1 do 0 na začátku start bitu. Počet stop bitů (1 nebo 2) je volitelný. Počet datových bitů (7 nebo 8) je rovněž volitelný. Vysílá se vždy LSB napřed. Volitelný je i „přídavný bit“. Ten může mít význam paritního bitu pro kontrolu, nebo může úplně chybět. Kontrola paritou je prováděna automaticky – ve vysílači je generátor parity, který zpracovává datové bity. Jeho výstup je vložen za poslední datový bit. V přijímači jsou obdobným generátorem parity zpracovány přijaté datové bity a výsledek je srovnán s přijatým paritním bitem. Rozdíl svědčí o chybě v příjmu. Kontrola však není spolehlivá, neboť parita nereaguje na sudý počet chyb. Rušení typu „série impulzů“ (angl. burst), které je velmi časté, může zasáhnout přenos celého znaku a pak pravděpodobnost sudého počtu chyb může být 50 %. Modulační rychlost je dána nastavením předděličů za základním oscilátorem a dělicím poměrem čítače (viz *obr. 9.10* a odstavec 9.3.2.).



Obr. 9.16 Průběh signálu z asynchronního vysílače

Asynchronní přijímače umožňují alespoň částečnou kontrolu přijímaných dat. Ve stavovém registru lze nalézt bity s významem:

- chyba bitu ... nesoulad vzorků při vícenásobném vzorkování,
- chyba parity ... přijatý paritní bit nesouhlasí s paritou přijatých dat,
- chyba rámce ... na místě stop bitu (bitů) je stav 0 místo 1,
- ztráta dat ... přijatý znak nebyl včas přečten (záleží na hloubce vyrovnávací paměti).

Periferie UART je většinou používána v rámci normy RS 232 C (Recommended Standard). Tato norma byla vyvinuta pro spojení prostřednictvím modemů a proto kromě sériových dat Tx a Rx je definována ještě řada dalších řídicích a stavových signálů pro modem. Kromě významů jednotlivých signálů jsou definovány i jejich elektrické parametry, konektory a jejich zapojení. Napěťové úrovně všech signálů jsou odlišné od úrovní logiky CMOS a tak jsou vazební členy nezbytné. Základní vlastnosti vazebních členů jsou v *tab. 9.1*.

Vazební členy signál negují (vyšší napětí odpovídá stavu 0). Zátěž, definovaná jako maximální kapacita, nestanovuje přípustnou délku kabelu – záleží na jeho vlastnostech. Prakticky lze bez problémů přenášet data s modulační rychlostí

19,2 kBd na mnoho desítek metrů. Záměrně malá strmost hran na výstupu potlačuje efekty z impedančně nepřizpůsobeného vedení – kabel tak není nutné zakončovat terminátory.

Tab. 9.1 Elektrické parametry vazebních členů RS 232

napěťové úrovně na výstupu	+5 V až +15 V pro stav 0, –5 V až –15 V pro stav 1
napěťové úrovně na vstupu	min. ±3 V
zátěž výstupu	max. 2500 pF
vstupní odpor	3 až 7 kΩ
strmost hran na výstupu	max. 30 V/μs

Vysoké napěťové úrovně na výstupu zvyšují odolnost proti rušení, vyžadují však další dvě napájecí napětí. Řada výrobců proto dodává členy, ve kterých jsou již integrovány napěťové měniče, takže postačuje napájení 5 V.

Vazební členy pro RS 232 jsou dvoustavové, takže je nelze od vedení odpojovat. Není proto možné spojovat paralelně několik vysílačů. Možnost spojení do sítě je tak podstatně omezena, přesto je alespoň částečně možná v konfiguraci „jeden vysílač – několik přijímačů“. Periferní obvody UART v některých mikrokontrolérech umožňují identifikovat (adresovat) přijímač. To se děje prostřednictvím přídavného (devátého) bitu, který má v tomto případě význam jiný než parita. V řídicím registru přijímače lze naprogramovat vyvolání přerušení při stavu 1 v devátém bitu, stav 0 ale přerušení nevyvolá. Přenos vypadá tak, že první slabika s významem adresy je vysílána s devátým bitem o hodnotě 1. Tím se vyvolá přerušení ve všech přijímačích. Programy v nich přečtou registry přijímačů a obsah porovnají se svými přidělenými adresami. Další data jsou již vysílána s nulou v devátém bitu, takže přijímače nevyvolávají přerušení. U toho přijímače, kde došlo k souhlasu adres, program periodicky testuje stavový registr a pokud je signalizováno naplnění vyrovnávacího datového registru, přečte jej a obsah zařadí do paměti. U ostatních přijímačů jsou přijímaná data ignorována.

Obvody UART nejsou určeny jen pro normu RS 232. S vazebními členy jiného typu, jako jsou třístavové členy podle RS 485, lze vytvářet plně funkční sítě typu monomaster nebo multimaster. Příkladem může být velmi rozšířená síť PROFIBUS.

9.3.8 Sériové vstupní a výstupní obvody CAN

Sériové periferní obvody CAN jsou podstatně složitější, než všechny výše uvedené. Mohou být začleněny do sběrnice, která je pro vysokou spolehlivost a rychlost přenosu prioritních zpráv velmi vhodná pro náročné úlohy v řízení. Sběrnice CAN bus (angl. Controller Area Network) byla původně vyvinuta pro řízení a sběr

dat ve vozidlech, postupně se ale rozšířila i do jiných oblastí automatizace. Stále větší procento nově konstruovaných mikrokontrolérů obsahuje vysílač a přijímač CAN jako standardní součást periferních obvodů. Kromě toho existují i samostatné řadiče CAN, které lze připojit na vnější sběrnice počítače.

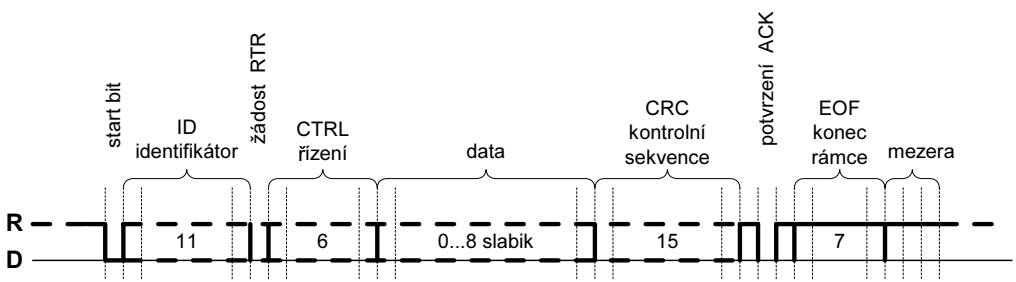
CAN používá arbitráž CSMA/CD + AMP, na začátku zprávy je vysílán identifikátor. Vazební členy mají otevřené kolektory. Existují dva typy – pro jednovodičovou a pro dvouvodičovou sběrnici. Prvý případ je vhodný pro krátké vzdálenosti a prostředí s malým rušením. Druhý případ je vhodný pro větší vzdálenosti a prostředí s velkým rušením. Generují se dva signály v protifázi a ve vstupních vazebních členech přijímačů jsou rozdílové zesilovače. Od vlastního přijímače a vysílače jsou vazební členy galvanicky odděleny optrony.

Přenášená data o volitelné délce až do 8 slabik jsou vložena v rámci. Rámec začíná start bitem SOF (angl. Start of Frame) obdobně jako u UART. Na rozdíl od něj se však používá bitová synchronizace a přijímač je vybaven fázovým závěsem, který se průběžně dolaďuje. Tolerance kmitočtů základních oscilátorů v jednotlivých uzlech sítě proto může být velmi mírná – uvádí se 5 %. Data nejsou kódována, výpadek synchronizačních hran je řešen vsouváním bitů (bit stuffing) tak, že v případě posloupnosti 5 shodných bitů je automaticky vkládán bit s opačnou hodnotou. Přijímač jej naopak vyřazuje. Modulační rychlost u CAN není normalizována, závisí na délce vedení přibližně podle vztahu

$$v_m < 4 \cdot 10^7 / l_v,$$

kde v_m je modulační rychlost v Bd a l_v je délka vedení v metrech. Maximální délka vedení závisí na vlastnostech vazebních členů a kabelu – 1 km však není problém. Modulační rychlosti pak vycházejí od 10 kBd do 1 MBd. Přijímač CAN umožňuje volbu polohy vzorkovacího bodu. Též rychlost přelaďování fázového závěsu lze volit. Jedná se o tzv. sychronizační skok SJW (angl. Synchronization Jump Width), o který se zkrátí či prodlouží takt při přelaďování. Průběh signálu v jednom rámci ukazuje *obr. 9.17*.

Uvnitř jednotlivých polí jsou vyznačeny jejich délky v počtech taktů. V diagramu nejsou stavy označeny jako 0 a 1, nýbrž jako R pro stav recesivní a D pro stav dominantní. Arbitráž mezi vysílači se děje v průběhu vysílání identifikátoru ID.



Obr. 9.17 Průběh signálu CAN

Význam identifikátoru a smysl recesivního a dominantního stavu byl vysvětlen v odstavci 9.3.5. Za ID je bit RTR (angl. Remote Transmission Request), který stavem D sděluje, že v rámci budou přenášena data. Stavem R žádá identifikovaný uzel o zaslání dat. Vysílání dat má prioritu. V poli řízení je informace o typu rámce a počtu slabik následujících dat. Na obrázku je znázorněn rámec CAN 2.0 A s 11bitovým polem ID. Existuje také rámec 2.0 B s 29bitovým ID. Následují data o počtu slabik, stanoveném v CTRL. Byl-li RTR = R, je délka dat nulová. V datovém poli je vysílána slabika z nejnižší adresy napřed, MSB napřed. Pole CRC obsahuje kontrolní sekvenci, pomocí které přijímače kontrolují všechny předcházející bity. Za CRC, mezi dvěma oddělovači ve stavu R, je bit ACK, ve kterém přijímače potvrzují stavem D bezchybný příjem. V tomto jediném taktu vysílač odstoupí od sběrnice a bit ACK vysílají přijímače. Není to potvrzení o tom, že vysílaná zpráva se dostala bezchybně na místo určení, nýbrž potvrzení o tom, že alespoň jeden přijímač zkontroloval zprávu a ta byla bezchybná. Je to tedy informace o bezchybném vyslání zprávy. Následuje označení konce rámce EOF a povinná mezera alespoň tří taktů – její horní mez není omezena. Vkládání bitů je aktivní jen během některých částí rámce – je to SOF, ID, CTRL, data, CRC. Netýká se EOF a mezery.

Spolehlivost přenosu dat je zabezpečena několika mechanismy. Během vysílání všechny přijímače (včetně přijímače sdruženého s právě aktivním vysílačem) průběžně kontrolují celý rámec a okamžitě reagují na chybu. Kterýkoliv uzel při zjištění chyby okamžitě vysílá chybový rámec (eng. error frame), který přeruší původní vysílání. Po zjištění chybového rámce nebo bitu ACK se stavem R vysílač opakuje vysílání původního rámce, dokud nebyl přenesen bez chyb. Kontroly během příjmu jsou následující:

- chyba v bitu ... nesoulad vzorků při vícenásobném vzorkování,
- chyba stuffingu ... špatně vkládané bity v polích SOF, ID, CTRL, DATA, CRC,
- chyba CRC ... kontrola pomocí zabezpečovací sekvence neúspěšná,
- chyba formátu ... v polích EOF, v mezeře a v oddělovačích kolem ACK musí být stav R,
- chybí potvrzení ... v ACK má být stav D.

V rozsáhlých systémech může být počet 11bitových identifikátorů nedostatečný. Proto byl zaveden alternativní rámec 2.0 B s 29 bity pole ID. Rámec 2.0 B se na začátku liší od rámce 2.0 A a přijímače, které mohou pracovat jen s rámcem 2.0 A, by hlásily chybu. Obvody modernější konstrukce umožňují proto práci s oběma rámci nebo alespoň plné ignorování rámce 2.0 B, takže není hlášena chyba.

V síti je velmi důležité, aby případná vadná stanice neohrozila funkčnost celé sítě neustálým vkládáním chybových rámců. V síti CAN je na to pamatováno. Každý přijímač má svůj čítač chyb, který je inkrementován při každé chybě v příjmu

a každý vysílač má svůj čítač, který je inkrementován při každé chybě ve vysílání. Pokud obsah jednoho z čítačů v uzlu přesáhne stanovenou mez, je pro daný uzel blokováno vysílání chybového rámce, není však omezeno vysílání datových rámců. Pokud se obsah čítačů ještě dále zvyšuje a dosáhne druhého limitu, je uzel odpojen a jeho opětné připojení je možné jen po novém resetu – existuje hardwarový i softwarový reset. Obsah čítačů se však může také snižovat, a to při úspěšném vysílání nebo příjmu. Snižování obsahu je pomalejší, než jeho zvyšování. Vychází se z filosofie, že ztráta „důvěryhodnosti" je vždy rychlejší, než její opětné získání. Uzel se tak může postupně propracovat do plně funkčního stavu. To je velmi důležité pro provoz s vysokými úrovněmi rušení, kdy chyby v příjmu či vysílání nejsou způsobeny selháními uzlu.

Celková spolehlivost hardware sítě CAN je extrémně vysoká. V literatuře se uvádí případ sítě s deseti uzly, s denním osmihodinovým provozem, s četností náhodných chyb v jednom rámci rovné 10^{-3}. Pravděpodobnost nezjištěné chyby je jedna za tisíc let. Není jistě nijak riskantní svěřit takovémuto systému řízení vozidla i s brzdovou soustavou, nebo třeba řízení letadla. Je ovšem nutné zabránit mechanickému poškození vodičů sběrnice. To je snadnější než zabránění mechanického poškození trubiček hydraulických soustav. Též je ale nutné zabránit kolapsům nadřazených programů. Chyby v software, kdy např. jeden uzel začne nevhodně vysílat zprávy s nejvyšší prioritou a potlačí tak všechny ostatní zprávy na sběrnici, jsou nejčastějším prohřeškem.

V každém přijímači existuje příjmový filtr (angl. acceptance filter), realizovaný komparátorem s maskováním. K němu patří dva registry – registr příjmového kódu ACR (angl. Acceptance Code Register) a registr příjmové masky AMR (angl. Acceptance Mask Register). Přijímač sice průběžně přijímá a kontroluje všechny zprávy, ale do svého vyrovnávacího registru (nebo rozsáhlejší vyrovnávací paměti) vloží jen ty zprávy, jejichž identifikátory vyhovují nastavení filtru. Vložení zprávy je procesoru signalizováno stavovým bitem, případně též přerušením. Jednotlivé bity identifikátoru musí být shodné s bity ACR. Ignorovány jsou bity, maskované registrem AMR. Lze tak naprogramovat celou podmnožinu identifikátorů, přípustných pro daný přijímač. U některých přijímačů existuje několik příjmových filtrů a vyrovnávací paměť dvoubránová. Jednotlivé zprávy tak přijímač automaticky třídí podle ID a zařazuje do patřičných schránek.

Přidělení identifikátorů jednotlivým zprávám je klíčové pro spolehlivou činnost sítě. Tuto službu a další služby, nutné pro distribuované řízení v reálném čase, zajišťuje tzv. aplikační vrstva. Jedná se o vrstvu 7 podle modelu ISI-OSI, zatím co vlastní přijímač a vysílač CAN realizuje první a druhou vrstvu (fyzická a linková). Existuje několik normalizovaných komunikačních protokolů, zajišťujících všechny potřebné služby (např. protokol CAL, CANopen, aj.).

9.3.9 Sériové vstupní a výstupní obvody IIC

Periferní obvody IIC nebo I^2C (angl. Inter-Integrated Circuit Bus) jsou používány jen výjimečně, převážně v obvodech televizních přijímačů. Nejsou tak samozřejmou součástí mikrokontrolérů, jako je SPI nebo UART, a neumožňují vytváření sítí, tak jako je umožňuje CAN. Jsou zde proto uvedeny jen stručně, pro úplnost.

Sběrnice IIC je částečně podobná CAN. Není to náhoda, IIC předcházela CAN. V CAN pak byly některé principy IIC významně zdokonaleny, ovšem na úkor podstatně vyšší složitosti obvodů. IIC používá dva vodiče – datový (SDA) a synchronizační (SCL). Po obou je možný dvousměrný přenos. Jednotlivé obvody na sběrnici jsou vybírány adresou, která je přenášena jako první část zprávy. Jednotlivé uzly mají na výstupu otevřené kolektory jak pro data, tak pro synchronizační impulzy, zvláštní vazební členy se nepoužívají. Celá sběrnice je určena pro propojení obvodů na malé vzdálenosti, prakticky na jedné desce plošného spoje. Přístup na sběrnici přes otevřené kolektory umožňuje jednoduchou arbitráž bez destruktivních konfliktů. Použitý způsob arbitráže umožňuje provoz sítě typu „multimaster“, používá se však zřídka. Běžné uspořádání je mikrokontrolér jako **M** a skupina obvodů typu **S**. Arbitráž mezi uzly **M**, které současně vstoupily na sběrnici, se děje během vysílání adresy obdobně jako u CAN během vysílání ID. Adresa s nižší číselnou hodnotou tedy má vyšší prioritu. Existuje formát se 7bitovou adresou a formát s 10bitovou adresou.

Přenášená zpráva je členěna na 8bitové slabiky, jejichž počet je libovolný. Přenáší se napřed MSB. Začátek rámce je signalizován značkou „start“. V prvé slabice po ní je vysílána adresa (7 bitů), v posledním bitu slabiky je informace o směru dalšího přenosu (**M** bude přijímat data nebo je bude vysílat). Na konci datového pole je značka „stop“, kterou je ukončen rámec. Obě zmíněné značky využívají takové kombinace stavů na vodičích SDA a SCL, které se jinak nevyskytují. Za každou slabikou je vsunut bit potvrzení příjmu. V něm přijímač (stavem 0) potvrzuje, že přijal slabiku jemu adresovanou. Jiná kontrola bezchybnosti přenášených dat není hardwarově implementována. Je záležitostí programů, jak a zda vůbec budou data zabezpečena.

Co se týče kmitočtu synchronizačních impulzů, existují tři verze IIC: standardní (100 kHz), rychlá (400 kHz) a vysoce rychlá (do 3,4 MHz). IIC má navíc zajímavou vlastnost, jak pomalý přijímač může pozastavit příliš rychlé hodinové impulzy. Vyplývá to z připojení na sběrnici přes otevřené kolektory. Přijímač může udržet na vodiči SCL stav 0, i když vysílač by již sám o sobě generoval stav 1. Jakmile vysílač tuto situaci zjistí, pozastaví generaci dalších impulzů až do okamžiku, kdy přijímač uvolnil zkrat na SCL.

Sběrnice IIC, i přes některé shodné rysy s CAN, není k CAN alternativou. Je ale alternativou k SPI. Řada výrobců dodává obvody se stejnou funkcí, ale ve verzi buď pro SPI, nebo pro IIC.

426410

9.4 ANALOGOVÉ VSTUPNÍ A VÝSTUPNÍ OBVODY

Analogové obvody počítače mohou mít podobu speciální jednotky (desky), nebo mohou být integrovány na jednom čipu s ostatními obvody. Prvý případ je typický pro personální počítače a zvláště pro počítače řídicí, druhý případ je typický pro mikrokontroléry. Samostatná jednotka umožňuje využití optimální sestavy analogových obvodů, kterých je v současné době na trhu velký výběr. Je tak možné dosáhnout podle potřeby velké přesnosti, velké rychlosti, nebo speciálních funkcí. Naopak integrace analogových obvodů v mikrokontroléru musí sledovat hlavně kritérium velké univerzality.

9.4.1 Analogové vstupní obvody

Základní tvar analogových vstupních obvodů ukazuje *obr. 9.18*. Vstupní kanály jsou voleny analogovým multiplexerem, sestaveným z přesných elektronických spínačů CMOS (na obrázku jsou pro názornost značeny jako spínače mechanické). Řídicí obvody sepnou vždy jen jeden spínač podle naprogramovaného čísla kanálu. Následuje vzorkovač, který v nejjednodušším případě sestává ze spínače a paměťového kondenzátoru. Řídicí obvody krátkodobě sepnou spínač, pak ho vypnou a na kondenzátoru zůstává konstantní napětí pro analogově-číslicový převodník (A/Č). Řídicí obvody převodník spustí a testují jeho stavový signál, kterým je oznamován konec převodu. Dodávají převodníku též potřebné synchronizační impulzy. Výsledky převodu jsou uloženy ve výstupních registrech, odkud je lze programem číst. Činnost řídicích obvodů je definována obsahem řídicích registrů, stav obvodů lze zjistit čtením stavových registrů.

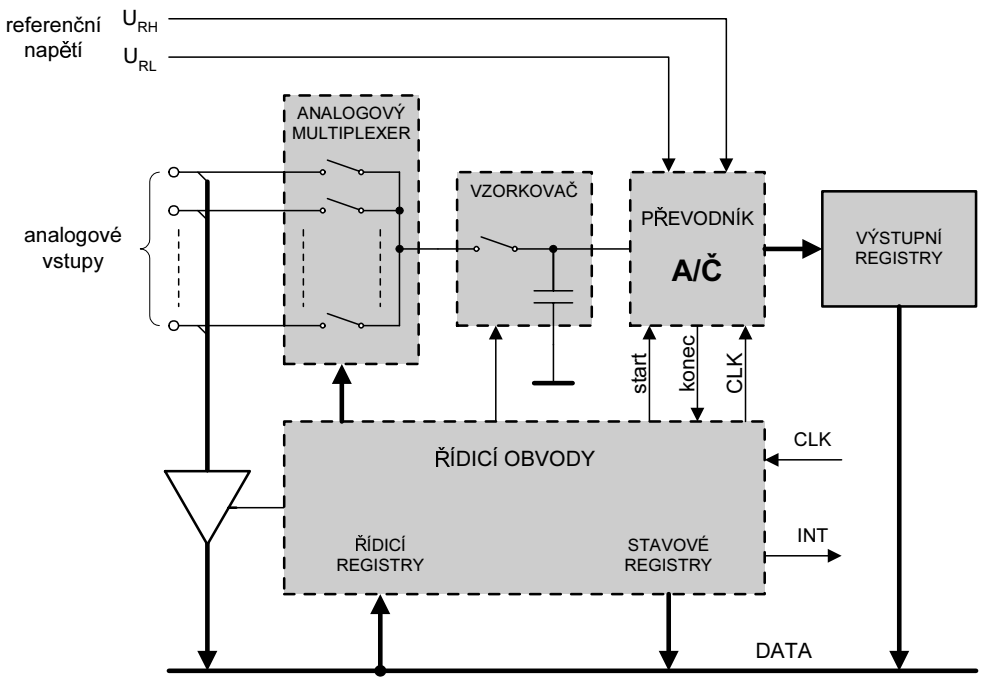
Obvody na *obr. 9.18* představují minimální sestavu. V samostatných jednotkách jsou podstatně rozšířeny a zdokonaleny:

- Vzorkovač je vybaven operačními zesilovači, takže proud pro nabíjení paměťového kondenzátoru není hrazen ze zdroje vstupního napětí a vstupní proud převodníku není hrazen z paměťového kondenzátoru.
- Mezi multiplexer a vzorkovač je vložen zesilovač s programovatelným zesílením. To umožňuje pracovat s kanály o velmi malém vstupním napětí řádu mV vedle kanálů s napětím jednotek V, aniž by utrpěla přesnost převodu A/Č. Zesilovač bývá řešen jako rozdílový a jak jeho invertující, tak i neinvertující vstup má svůj multiplexer. Vstupní kanály jsou tedy řešeny jako rozdílové. To je užitečné pro potlačení rušení, které může pronikat do vstupních přívodů a může znehodnotit celý systém.
- Převodníky mohou být různých typů podle aplikace celého systému. Pro nejrychlejší převod při digitalizaci video signálu se používají paralelní převodníky („flash converter“, 8 až 12 bitů s dobou převodu kolem 10 až 15 ns, pro nejvyšší přesnost převodníky sigma-delta (až 24 bitů s dobou převodu

až stovek ms), pro univerzální použití převodníky aproximační (kompromis rychlosti a přesnosti, 8 až 16 bitů s dobou převodu několika set ns na bit).

- Je umožněna individuální úprava signálů ještě před vstupy multiplexeru. Signály o příliš vysoké amplitudě lze rezistorovým děličem snížit a omezit, malé signály lze zesílit, střídavé signály usměrnit, do vstupů lze zařadit filtry pro omezení rušení a antialiasing, atd.
- Výstupní data převodníku jsou ukládána do vyrovnávací paměti, řešené buď jako FIFO, nebo jako paměť dvojbránová. Analogová jednotka tak může pracovat značně nezávisle na procesoru, který si jen dle potřeby vybírá data z vyrovnávací paměti. Řídicí obvody umožňují autonomní činnost analogové jednotky.

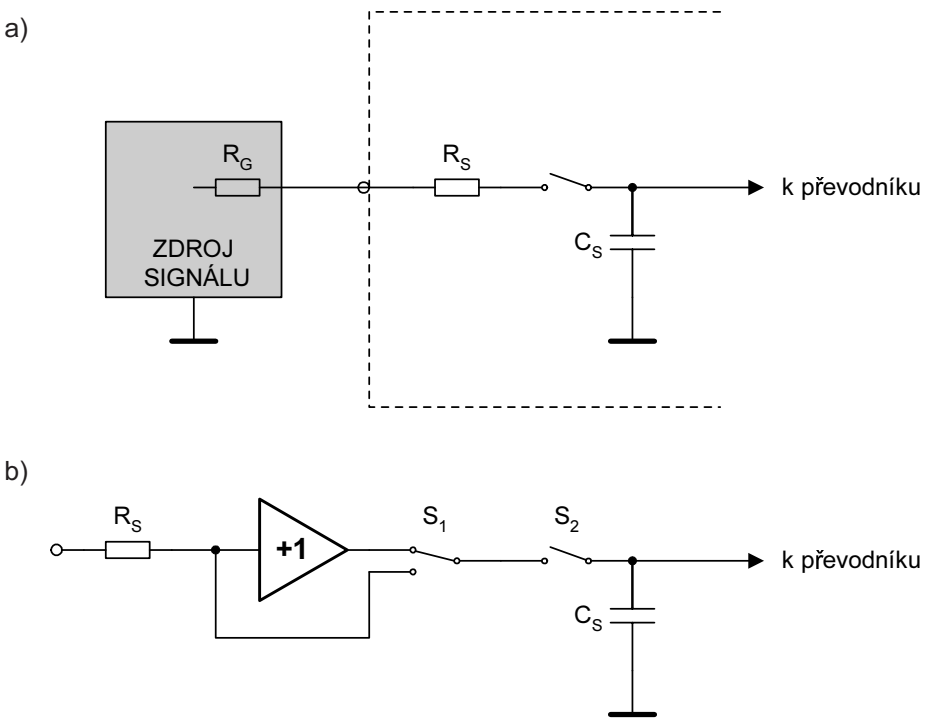
Analogové obvody se šestnácti- a vícebitovými převodníky představují velmi náročný systém. Pro ilustraci – při rozsahu vstupních napětí 0 až +5 V a 16bitovém převodníku odpovídá nejnižšímu bitu napěťový přírůstek pouhých 76 μV. Nemá-li být přesnost převodníku degradována, musí být sumární hodnota rušení, offsetů a driftů použitých operačních zesilovačů a úbytků na vodičích ještě menší. To vyžaduje dokonalé stínění, perfektní plošné spoje a velmi kvalitní součástky. Tomu samozřejmě odpovídá i cena. Návrh přesných analogových obvodů však spadá mimo rozsah této publikace.



Obr. 9.18 Analogové vstupní obvody

426410

U jednočipových řešení je technologicky náročné vyrobit na jednom čipu jednak rychlé číslicové obvody, jednak přesné analogové obvody (včetně operačních zesilovačů). Obvodové řešení u mikrokontrolérů má proto své zvláštnosti, které je třeba při návrhu systému respektovat. Vzhledem k omezeným možnostem realizace přesných analogových obvodů je u jednoduchých konstrukcí vzorkovač zapojen jen jako spínač s paměťovým kondenzátorem. Situaci ukazuje *obr. 9.19a*. Zdroj signálu má vnitřní odpor R_G, rezistor R_S reprezentuje odpor sepnutého spínače multiplexeru ve vybraném kanálu a další odpory. Při sepnutí spínače se kondenzátor C_S nabíjí exponenciálně a jeho nabití vyžaduje čas – chyba jednoho LSB vzhledem k ustálené hodnotě je u 10bitového převodu dosažena až po 7 časových konstantách $(R_G + R_S) \cdot C_S$, u osmibitového převodu postačí 5 časových konstant. Doba nabíjení je u některých mikrokontrolérů programovatelná, u jiných ale ne. Ze známé doby nabíjení je nutné spočítat maximální přípustný odpor zdroje signálu – jeho větší hodnota zvyšuje chybu převodu. Realistické hodnoty jsou řádu několika kΩ. Velikost paměťového kondenzátoru jsou desítky pF, doba sepnutí spínače je kolem 1 μs. Špička vstupního proudu je omezena v konstrukci vnitřních obvodů podle *obr. 9.19b*, kde v prvé fázi je paměťový kondenzátor nabíjen přes jednoduchý zesilovač, ve druhé fázi pak rovnou z multiplexeru. Většinu náboje kondenzátoru tak dodá zesilovač, ale konečná hodnota napětí není závislá na vlastnostech zesilovače (offset, drifty).



Obr. 9.19 Vzorkovač: a) jednoduché zapojení; b) zapojení s omezením proudové špičky

426410

Na *obr. 9.18* je vyznačeno i propojení vstupů pro analogové signály s vnitřní datovou sběrnicí. Je tak možné využít vývody pouzdra alternativně pro analogové nebo pro číslicové signály (vždy jen vstup). Volba se provádí jedním z bitů řídicího registru, kterým se při práci s analogovými signály blokují číslicové vstupní obvody (tj. odpojí se od napájení). Blokování je důležité pro omezení příkonu – analogové napětí se totiž může dlouhodobě pohybovat právě kolem rozhodovací úrovně číslicových obvodů, které v tomto režimu mají vysokou spotřebu napájecího proudu. Naopak při využití vývodů pouzdra pro číslicové signály lze pro úsporu příkonu vypnout analogové obvody tím, že jedním bitem řídicího registru se blokují jejich hodinové impulzy.

Převodníky, používané v jednočipových verzích, jsou zásadně aproximační – buď se sítí s přepínanými rezistory (R-2R), nebo se sítí s přepínanými váhovými kondenzátory na principu redistribuce náboje. Převod je spuštěn řídicím signálem „start“, konec převodu je signalizován stavovým signálem „konec“ (skutečné názvy se budou lišit), činnost je synchronizována hodinovými impulzy. Ty jsou odvozeny od hodinových impulzů procesoru přes předdělič a lze je blokovat. Na *obr. 9.18* jsou naznačeny ještě vstupy dvou referenčních napětí U_{RL} a U_{RH}. Napětí U_{RL} definuje vstupní napětí, které bude převedeno na číslo 00...0. Rozdíl obou referenčních napětí určuje pracovní rozsah převodníku. Velmi často se U_{RL} spojuje s analogovou zemí. Platí:

$$\Delta U_{IN} = (U_{RH} - U_{RL})/2^n,$$

kde ΔU_{IN} je přírůstek vstupního napětí odpovídající jednomu LSB a n je počet bitů převodníku. Rozdíl referenčních napětí je vhodné nastavit pokud možno vysoký, aby ΔU_{IN} bylo co nejvyšší vhledem k vnějšímu i vnitřnímu rušení. Horní mezí pro referenční napětí je velikost napájecích napětí analogových obvodů. U vyspělejších konstrukcí jsou napájecí napětí vnitřních číslicových obvodů a napájecí napětí vnitřních analogových obvodů zavedena na oddělené vývody pouzdra. To je v souladu se zásadami konstrukce smíšeného (tj. analogového a číslicového) systému na jedné desce. Je třeba i důsledně oddělovat součástky analogových obvodů od součástek číslicových obvodů a spojovat je s odpovídajícím typem napájecích napětí. Napájecí napětí citlivých analogových obvodů je filtrováno.

Mikrokontroléry mají nejčastěji 8 analogových vstupů a převodníky 8bitové nebo 10bitové, výjimečně 12bitové. U 10 nebo 12 bitů není zaplněno celé slovo (16 bitů) a proto lze řídicí obvody nastavit na zarovnání výsledku zprava (angl. right justified) nebo zleva (angl. left justified). Též bývá možné volit buď hrubý a rychlý 8bitový převod, nebo převod pomalejší, ale s plným počtem bitů. Typický analogový vstupní systém může pracovat v několika volitelných režimech:

- **Převod jednoho vstupního kanálu** (angl. single channel conversion). Kanál zvoleného čísla je převeden a výsledek uložen do výstupního registru. U jednoduchých mikrokontrolérů existuje jen tento režim.

- **Opakovaný převod jednoho vstupního kanálu** (angl. single channel continuous conversion). Kanál zvoleného čísla je automaticky opakovaně převáděn a skupina výstupních registrů je postupně zaplněna.
- **Postupné převádění skupiny vstupních kanálů** (angl. auto scan). Postupně je převáděn jeden kanál za druhým a výsledky jsou postupně ukládány do výstupních registrů. Po převedení předvoleného počtu kanálů činnost končí.
- **Opakované převádění skupiny vstupních kanálů** (angl. auto scan continuous). Postupně je převáděn jeden kanál za druhým a výsledky jsou postupně ukládány do výstupních registrů. Po převedení předvoleného počtu kanálů se celý postup automaticky opakuje.

Začátek činnosti je vyvolán zápisem do řídicího registru na pozici bitu „start". Opakované převody se zastaví vynulováním tohoto bitu, při převodech jednorázových (jednoho kanálu nebo jejich skupiny) je tento bit vynulován automaticky. U některých systémů je převod spuštěn samotným zápisem čísla kanálu. Po spuštění již řídicí obvody generují potřebné signály – nejprve přepnou multiplexer, pak zapamatují vzorek signálu a následně spustí vlastní převodník. Po dokončení převodu zapíší jeho výstupní data do patřičného výstupního registru. Při skupinových převodech využívají k adresaci registrů svůj vnitřní čítač.

Ukončení převodu dle zvoleného režimu je signalizováno jednak ve stavovém registru, jednak vyvoláním přerušení. To samozřejmě lze maskovat. Výstupní registry je nutno přečíst dříve, než je spuštěn nový převod. To není problémem u jednorázových převodů, neboť jak čtení, tak i nové spuštění je programově řízeno. U automaticky opakovaných převodů však k tomu dojít může. Ve stavovém registru je proto bit, signalizující ztrátu dat (angl. overrun error).

Režim „převod jednoho vstupního kanálu" je univerzální a vyhovuje plně programovému řízení. Při potřebě opakovaných převodů však vede na časové ztráty. Opakované převody jednoho kanálu se využívají např. pro omezení šumu zprůměrováním. Převod skupiny kanálů s minimálním vzájemným časovým odstupem je zapotřebí např. v úlohách počítačového řízení pohonů.

9.4.2 Analogové výstupní obvody

Problematika analogových výstupních obvodů je daleko jednodušší. Číslicově-analogový převodník (Č/A) je v tomto případě založen zpravidla na rezistorové síti R-2R s přesnými napěťovými spínači. Tento převodník lze připojit rovnou na výstupní paralelní bránu bez potřeby jakéhokoliv dalšího ovládání. Každý výstupní kanál však vyžaduje jeden převodník. Z tohoto hlediska je vstupní systém mnohem úspornější, neboť jeden převodník postačí pro větší počet kanálů, přičemž multiplexer nepředstavuje významnou komplikaci.

U jednočipových řešení jsou výstupní analogové obvody poměrně vzácné a jsou-li vůbec realizovány, pak ve skromném rozsahu (1–2 kanály). Často se nahrazují

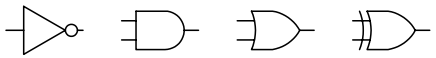
výstupem PWM s následujícím filtrem typu dolnofrekvenční propust. Malá potřeba analogových výstupů vyplývá z typické aplikace mikrokontrolérů: z řízené soustavy jsou získány signály (analogové i číslicové), ty jsou číslicově zpracovány, a na základě výsledků zpracování jsou ovládány akční členy, zpravidla dvoustavové. Analogové obvody v samostatných jednotkách však mohou obsahovat i výstupní část s velmi rozmanitou funkcí.

426410

SEZNAM GRAFICKÝCH SYMBOLŮ, POUŽÍVANÝCH V OBRÁZCÍCH

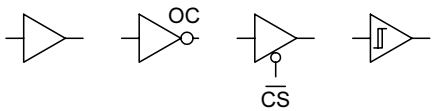
Základní logické členy:

Zleva člen realizující negaci, logický součin (AND), logický součet (OR), exklusivní součet (XOR).



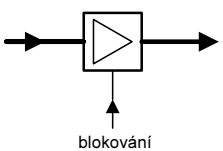
Členy se zvláštními vlastnostmi:

Zleva nenegující člen (oddělovač, sledovač signálu – mění jen elektrické parametry signálu, zvláště výstupní proud či napětí), člen s otevřeným kolektorem, třístavový člen (blokovací vstup zde značen jako $\overline{CS}$), člen s hysterezní převodní charakteristikou.



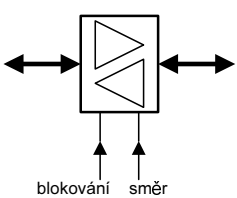
Skupina jednosměrných třístavových členů (budičů sběrnice):

Slouží k připojení zdroje dat na sběrnici. Všechny budiče skupiny jsou ovládány jedním společným signálem blokování (uvedení do vysokoimpedančního stavu).



Skupina dvousměrných třístavových členů (budičů sběrnice):

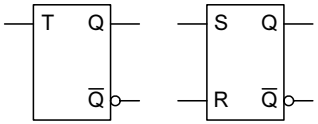
Slouží k dvousměrnému spojení zdroje či příjemce dat se sběrnicí. Signálem směr se vyvolává vysokoimpedanční stav na jedné nebo na druhé straně, signálem blokování se vyvolá na obou stranách současně.



426410

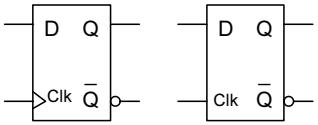
Asynchronní klopné obvody:

Vlevo klopný obvod typu T, vpravo RS.



Synchronní klopný obvod (zde typu D):

Vlevo obvod řízený náběžnou hranou, vpravo obvod řízený hladinou (aktivní ve stavu 1).



Řídicí bit:

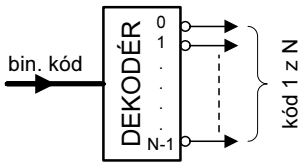
Jednobitový signál, generovaný klopným obvodem. Může být samostatný nebo součástí skupiny klopných obvodů (řídicího registru).

Typ reakce na vstupní signál:

Zleva – na náběžnou hranu, na doběžnou hranu, na obě hrany, na hladinu (po celou dobu trvání stavu 1).

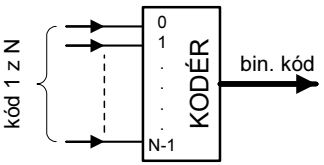
Dekodér:

Převádí binární kód na kód 1 z N. Hodnota binárního čísla odpovídá poloze výstupu, na kterém je stav 0 (u nenegovaných výstupů stav 1).



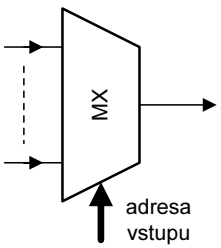
Kodér:

Opak funkce dekodéru. Jen na jednom vstupu smí být stav 1. Pokud by mohl být stav 1 na více než jednom vstupu, jednalo by se o prioritní kodér.



Číslicový multiplexer:

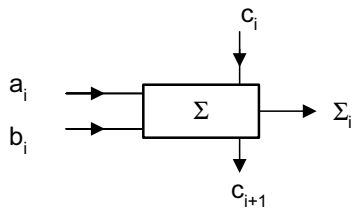
Adresou (v binárním kódu) je vybrán jeden vstup a převeden na výstup.



426410

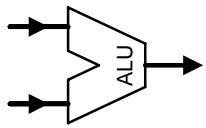
Jednobitová sčítačka:

Na výstupu Σ_i je aritmetický součet bitů a_i a b_i, a přenosu z nižšího řádu c_{i-1}. Na výstupu c_{i+1} je přenos do vyššího řádu.



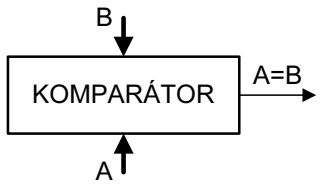
Aritmeticko-logická jednotka:

Vstupem jsou dvě čísla o stejném počtu bitů, výstup je zpravidla uložen do registru A (akumulátor, střadač). Provádí aritmetické operace, logické operace, posuny a rotace.



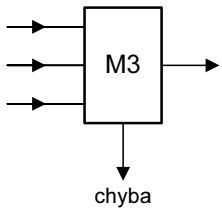
Číslicový komparátor:

Porovnává dvě čísla o stejném počtu bitů. Na výstupu je stav 1 při shodě obou čísel.



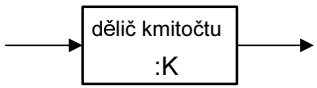
Majoritní obvod:

Výstup má takový stav, jaký má většina vstupů. Těch musí být lichý počet, takže existuje obvod M3, M5, atd. Na druhém výstupu může být signalizována chyba, když všechny vstupy nejsou ve stejném stavu.



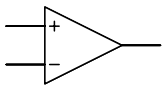
Dělič kmitočtu:

Realizován binárním čítačem, dělicí poměr je nejčastěji dán jako mocnina dvou.



Zesilovač (operační zesilovač):

Vstup označený „+“ je neinvertující, vstup označený „–“ je invertující.

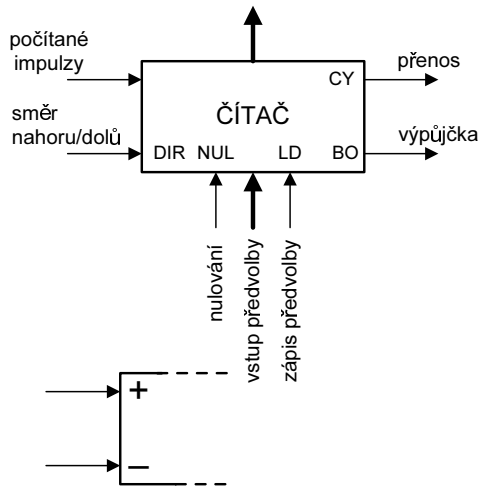


426410

Čítač:

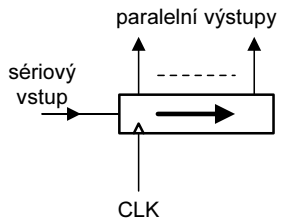
Směr počítání je řízen vstupem DIR, nebo existují dva vstupy impulzů (viz dolní obrázek) – pro počítání nahoru slouží vstup „+“, pro počítání dolů vstup „–“. Vstupem NUL se čítač nuluje, vstupem LD se zapíše číslo, přiváděné na vstupy předvolby. Na výstupu přenosu CY je impulz při přechodu čítače z nejvyššího čísla do nuly, na výstupu výpůjčky (záporného přenosu) BO je impulz při přechodu čítače z nuly do nejvyššího čísla.

Některé z uvedených vstupů a výstupů mohou případ od případu chybět.



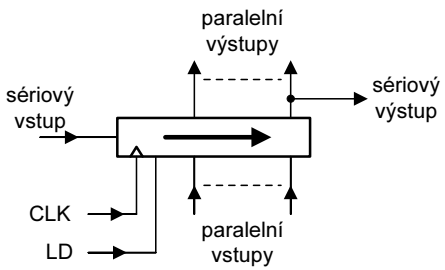
Posuvný registr:

Obsah je s každým hodinovým impulzem posouván doprava a zleva je doplňován ze sériového vstupu. Po zaplnění je k dispozici na paralelních výstupech. Typické použití je při převodu sériového kódu na paralelní kód.



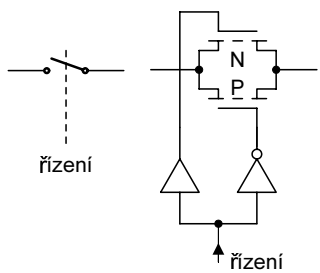
Posuvný registr s možností nastavení počátečního stavu:

Kromě naplnění bit za bitem ze sériového vstupu umožňuje i naplnění paralelní. Stav na paralelních vstupech se zápisovým impulzem LD přesune do registru. Pak může být impulzy CLK posunován doprava.



Přesný spínač analogového signálu:

Vlevo symbol spínače, vpravo elektronický spínač v technologii CMOS.



426410

REJSTŘÍK

Ř

S

T

U

V

Z

426410

LITERATURA

[1] Brandejs M.: Mikroprocesory INTEL 8086-80486. GRADA Praha, 1991. ISBN 80-85424-27-4.

[2] Šnorek M.: Standardní rozhraní PC. GRADA Praha, 1992. ISBN 80-85424-80-0.

[3] Vrátil Z.: Architektura PC na bázi Pentia. GETHON, 1994.

[4] Pechal S.: Monolitické mikropočítače. BEN – technická literatura Praha, 1995.

[5] Vlach J., Vlachová V.: Počítačová rozhraní – přenos dat a řídicí systémy. BEN – technická literatura Praha, 1995.

[6] Kainka B.: Využití rozhraní PC. Měření, řízení a regulace standardních portů PC. HEL Ostrava, 1997. ISBN 80-902059-3-3.

[7] Cady F.: Microcontrollers and microcomputers. Principles of software and hardware engineering. Oxford University Press, 1997. ISBN 0-19-511008-0.

[8] Skalický P.: Procesory řady 8051. BEN – technická literatura Praha, 1998. ISBN 80-86056-39-2.

[9] Hrbáček J.: Komunikace mikrokontroléru s okolím. BEN – technická literatura Praha, 1999. ISBN 80-86056-42-2.

[10] Ganssle J.: The art of designing embedded systems. Newnes, 1999. ISBN 0-7506-9869-1.

[11] Ličev L., Morkes D.: Procesory architektura, funkce, použití. Kompletní průvodce procesory a souvisejícími hardwarovými komponenty. Computer Press Brno, 1999. ISBN 80-7226-172-X.

[12] Morton T.: Embedded microcontrollers. Prentice Hall, 2001. ISBN 0-13-9077577-1.

[13] Ball S.: Analog interfacing to embedded microprocessors. Real world design. Newnes, 2002. ISBN 0-7506-7339-7.

[14] Firemní literatura Atmel

[15] Firemní literatura Hitachi

[16] Firemní literatura Intel

[17] Firemní literatura Microchip

[18] Firemní literatura Mitsubishi

[19] Firemní literatura Motorola

[20] Firemní literatura Siemens

[21] Firemní literatura Texas Instruments

[22] Firemní literatura Zilog.

[23] Minihofer O., Kratochvílová J.: Anglicko-český slovník výpočetní techniky. SNTI Praha, 1987.

[24] Vitovský A.: Anglicko-český výkladový slovník softwaru. AV SOFTWARE Praha, 1992. ISBN 80-901428-0-X.

[25] Poláček D.: Technické kreslení podle mezinárodních norem, 3. díl. MONTANEX Ostrava, 1995. ISBN 80-85780-28-3.

Knihy nakladatelství BEN – technická literatura:

EDICE PC & elektronika

Udělejte si z PC – 1. díl

– generátor, čítač, převodník, programátor…
Měření, řízení a regulace pomocí
sériového portu PC a sběrnice I²C

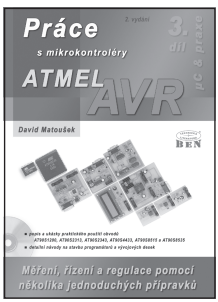
Autor Ing. David Matoušek, 176 stran B5 + CD ROM,
obj. číslo 121069, MC 249 Kč.

Udělejte si z PC – 2. díl

Měření, řízení a regulace pomocí portů PC
prostřednictvím několika jednoduchých přípravků
Komunikace PC s aplikacemi mikrokontrolérů řady AT89C2051,
stavba jednoduchého programátoru mikrokontroléru AT89C2051

Autor Ing. David Matoušek, 180 stran B5 + CD ROM,
obj. číslo 121114, MC 299 Kč.

EDICE µC & praxe



Měření, řízení a regulace pomocí několika jednoduchých přípravků

Práce s mikrokontroléry Atmel AT89C2051, 1. díl

Autor Ing. David Matoušek, 248 stran B5 + CD ROM,
obj. číslo 121093, MC 349 Kč.

Práce s mikrokontroléry Atmel AT89S8252, 2. díl

Autor Ing. David Matoušek, 304 stran B5 + CD ROM,
obj. číslo 121112, MC 399 Kč.

Práce s mikrokontroléry Atmel AVR, 3. díl (2. vydání)

Autor Ing. David Matoušek, 376 stran B5 + CD ROM,
obj. číslo 121130, MC 499 Kč.

DOPORUČUJEME

- **Mikrokontroléry Atmel AVR – vývojové prostředí**
- **Mikrokontroléry Atmel AVR – popis procesoru a instrukční soubor**
- **Mikrokontroléry Atmel AVR – programování v jazyce C**
- **Mikrokontroléry Atmel AVR – programování v jazyce Pascal**
- **Mikrokontroléry Atmel AVR – programování v jazyce Bascom**
- **Začínáme s mikrokontroléry Motorola HC08 Nitron**

- **Měření, řízení a regulace prostřednictvím PC**
- **Měření, řízení a regulace s Delphi**
- **Udělejte si z PC v DELPHI 1. díl**

Pár slov o nakladatelství

*Nakladatelství **BEN – technická literatura** se věnuje vydávání převážně počítačové a elektrotechnické literatury. Nakladatelství je součástí stejnojmenné firmy, která se zabývá prodejem a distribucí veškeré technické a počítačové literatury, jež v poslední době vyšla. Dále pak prodejem a distribucí zejména českých titulů na CD ROM a DVD. Přehledy dostupné literatury – ediční plány, vydávané několikrát ročně, obdržíte na našich prodejnách, na vyžádání je zasíláme poštou.*
Celková nabídka je soustředěna do několika specializovaných prodejen po celé České republice. Jejich adresy najdete na předcházející straně.

Adresa této knihy na Internetu:
http://shop.ben.cz/121158

Jiří Pinker
MIKROPROCESORY A MIKROPOČÍTAČE

Vydalo nakladatelství BEN – technická literatura, Praha 2008
1. dotisk 1. vydání

Vedoucí nakladatelství Libor Kubica
Vedoucí redakce a DTP Hana Züglerová
Odpovědná a technická redaktorka Iveta Kubicová

Sazba	Iveta Kubicová
Obálka	layout & foto Libor Kubica
Rozsah	160 stran
Tisk	Marten s.r.o.

objednací číslo	121158
EAN	9788073001100
ISBN	978-80-7300-110-0 (tištěná kniha)
	978-80-7300-353-1 (elektronická kniha v PDF)

426410