Obsah

Kapitola 1

Úvod

Elektronický procesor označovaný v prostředí výpočetní techniky zkratkou CPU (Central Processing Unit) je hlavní výkonná součástka počítačového systému. Jedná se o číslicový obvod, který svým vnitřním uspořádáním vykonává základní aritmetické a logické operace a operace přesunu dat. Na základě předem definované posloupnosti těchto operací (instrukcí) je procesor schopen vykonávat libovolný algoritmus. Procesor je univerzální programovatelný stroj schopný vykonávat libovolný algoritmus. Mikroprocesor je speciální případ procesoru, který je uzavřen do jednoho integrovaného obvodu.

Procesor lze posuzovat z několika hledisek:

- Taktovací frekvence procesoru
- Šířka vnitřních sběrnic

Taktovací frekvence procesoru udává kolik operací je procesor schopen udělat za jednu vteřinu. U daného procesoru je pak ještě důležité kolik instrukčních cyklů je potřeba k vykonání jedné instrukce. To se různí u procesorů s instrukčním souborem RISC a CISC. Slovo je u procesorů nejmenší počet bitů, se kterým procesor pracuje když zpracovává data. Délka slova je ovlivněna několika parametry počítače, například šířkou vnitřních datových sběrnic, velikostí vnitřních registrů, ... Pojem šířka vnitřních sběrnic se používá k popisu tří parametrů procesoru:

- Šířka vnitřních registrů - s jakou nejvyšší hodnotou je procesor schopen pracovat v jedné instrukci.
- Šířka vstupních a výstupních sběrnic pro data - jakou největší hodnotu lze po sběrnici přenést v jenom kroku.
- Šířka adresové sběrnice - jakou maximální paměť lze adresovat.

Výsledný výkon procesoru ale nelze posuzovat jen za pomoci taktovací frekvence nebo šírky vnitřních sběrnic. Procesory mají obecně různou vnitřní architekturu. Například jeden instrukční cyklus může zabrat dva taktovacími cykly, nebo při jednom instrukčním cyklu lze přečíst více instrukcí současně... Pro účely porovnávání výkonu mikroprocesorů se

využívají jednotky **MIPS** - Millions of Instructions Peer Secound, které specifikují kolik miliónů instrukcí je daný procesor schopen vykonat za sekundu. Například u procesoru s 2 taktovacími cykly na jeden instrukční cyklus a taktovací frekvencí 100MHz disponuje daný procesor výkonem $\frac{100}{2} = 50MIPS$. Procesor který má 3 taktovací cykly na jeden instrukční cyklus a taktovací frekvencí 150MHz je výkon $\frac{150}{3} = 50MIPS$ je tedy stejně výkoný jako první zmiňovaný procesor.

1.1 Rozdělení číslicových obvodů

Číslicové obody lze podle konstrukce rozdělit do několika skupin:

- Kombinační obvod
- Automat
- Mikrokontroler
- Mikroprocesor

1.1.1 Kombinační obvody

Kombinační obvody jsou základní číslicové obvody, které se skládají z jednotlivých logických hradel. Kombinační logické obvody tvoří základ pro všechny ostatní logické obvody. Kombinační logické obvody jsou specifiké vztahem mezi vstupem a výstupem - výstup logického kombinačního obvodu je přesně dán jeho vstupem.

1.1.2 Automaty

Logický automat vzniká přidáním paměťové části ke kombinačnímu obvodu. Díky tomu jeho výstup není přímo závislí na jeho aktuálním vstupu, ale také na hodnotách předchozích vstupů. Paměťová část tvoří vnitřní stavy obvodu, které jsou definovány předchozími vstupními a výstupními hodnotamy a tím ovlivňují výstup následující.

1.1.3 Mikrokontroler

Mikrokontroler je složitý programovatelný číslicový obvod (procesor), který v sobě obsahuje vše nezbytné pro uložení programu a jeho vykonávání - paměti, CPU, registry, ... Předpona *mikro* vyjadřuje, že celý obvod se nachází na jednom integrovaném čipu. Jedná se o označení z dob kdy procesor byl rozdělen do několika samostatných integrovaných obvodů. Mikrokontrolery se využívají především k řízení periferií, strojů, sběr dat, komunikační uzly, ... Mikrokontrolery pro svou činnost využí-

vají malé hardwarové požadavky (paměť, synchronizační takt, menší šířka sběrnic, ...).

1.1.4 Mikroprocesor

Mikroprocesor je podobný mikrokontroleru. Jeho zaměření je ale jiné. Mikroprocesor je stejně jako mikrokontroler složitý číslicový obvod (procesor), který je ulože do jednoho pouzdra integrovaného obvodu. Stanardně však neobsahuje RAM a ROM paměť, a některé další periferie. Ty jsou přidány jako externí periferie, díky čemu je možné nastavit parametry procesoru na míru dané aplikaci. Mikroprocesory jsou využívány především pro výpočty - operační systém, matematické programy, grafické programy, ... To vyžaduje v porovnání s mikrokontrolery potřebu větší paměti RAM, vyšší synchronizační takt, větší šířku sběrnic, ...

Kapitola 2

Koncepce počítačů

Počítač je zařízení, které slouží ke zpracování informací, k jejich ukládání a třídění. Pro tyto informace se používá označení **data**. Data jsou posloupností čísel, symbolů, ... Zpracováním dat vznikají data nová, uspořádaná do nových posloupností (řídící signály, textové dokumenty, ...). Ke zpracování těchto dat slouží obvody, které vykonávají aritmetické a logické operace, řídící funkce, přesuny dat, ... Posloupnosti dílčích operací, které vykonávají nějakou předem definovanou činnost se říká **algoritmus**. Záznam, který říká jaká operace se má kdy vykonat se nazývá **program**, který v sobě uchovává daný algoritmus.

Všechny architektury počítačů jsou založeny na jedné důležité myšlence - procesor zpracovává nějaká data podle předem připraveného programu, který je sám reprezentován daty. Díky tomu lze daný program různě modifikovat a měnit - počítač je programovatelný a nemá jednu předem danou funkci (základní myšlenka počítačů). Z toho vyplývá, že uvnitř počítače je potřeba paměť pro uložení dat a systém komunikace s pamětí (čtení a zápis).

Koncepce nebo také uspořádání počítače určuje jakým způsobem vzájemně spolupracují jednotlivé části ze kterých se daný počítač skládá. Z historického hlediska jsou nejvýznamnější **von Neumanova koncepce** a **Harwardská koncepce**. Rozdělení počítačů podle koncepce architektury je již v současnosti méně významné kritérium, protože současné počítače využívají prvky z obou koncepcí. Současné počítače tedy využívají hybridní koncepci.

2.1 Von Neumanova koncepce počítače

Von Neumanova koncepce počítače jako první přináší důležitou myšlenku - počítač zpracovává nějaká data pomocí programu, který je také reprezentován daty. To má veliký dopad na vývoj počítačových systémů, protože díky tomu je možné počítačový program upravovat, modifikovat, rozšiřovat. Díky této myšlence je počítač **programovatelný** a nemá přesně danou funkci (jednoúčelový), stává se flexibilním, protože pouhou změnou programu lze změnit funkci daného počítače.

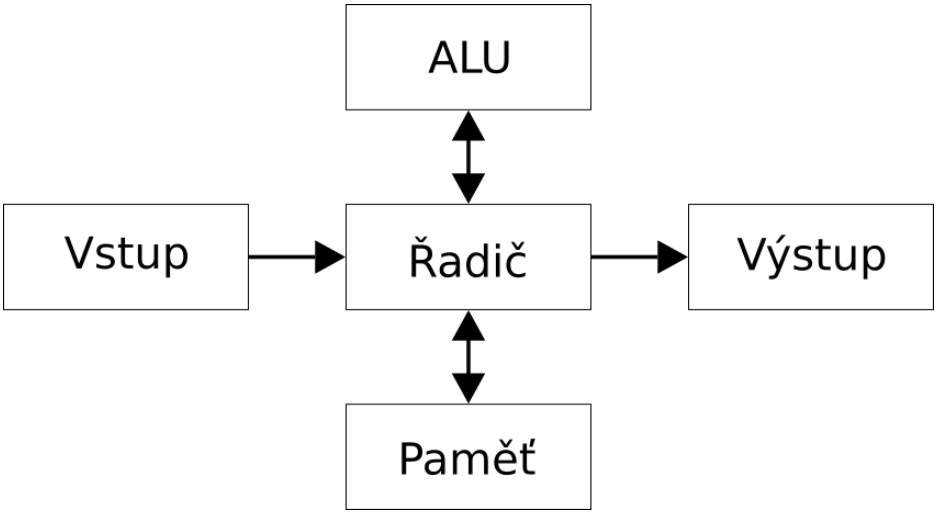
Von Neumannova koncepce je označení pro jednoduché schéma programovatelného počítače, které používá jednu sběrnici, na kterou jsou

připojeny všechny aktivní prvky (procesor, paměť, vstupy a výstupy). Jedná se tedy o architekturu se společnou pamětí pro data i program, což znamená, že zpracování programu je sekvenční (nelze v jednu chvíli číst i zapisovat z pamětí). Sdílená paměť pro program a pro data umožňuje v jednu chvíli pouze číst program a nebo číst/zapisovat data, ale nikoli obojí současně.

Celá von Neumanova koncepce se skládá z pěti částí:

- **Operační paměť** ve které jsou uchována data i program
- **Programový řadič**, který řídí a synchronizuje komunikaci jednotlivých částí počítače
- **Aritmeticko-logická jednotka** vykonává aritmetické a logické výpočty s daty
- **Vstupy** (vstupní periferie), které předávají počítači vstupní data, které následně zpracovává pomocí daného programu
- **Výstupy** (výstupní periferie), které slouží k předávání výsledků, zpracovaných dat uživateli počítače

Procesor počítače se skládá z řídící a výkonné (aritmetickologické) jednotky. Řídící jednotka zpracovává jednotlivé instrukce uložené v paměti, přičemž jejich vlastní provádění nad daty má na starosti aritmetickologická jednotka. Vstup a výstup dat zajišťují vstupní a výstupní jednotky.



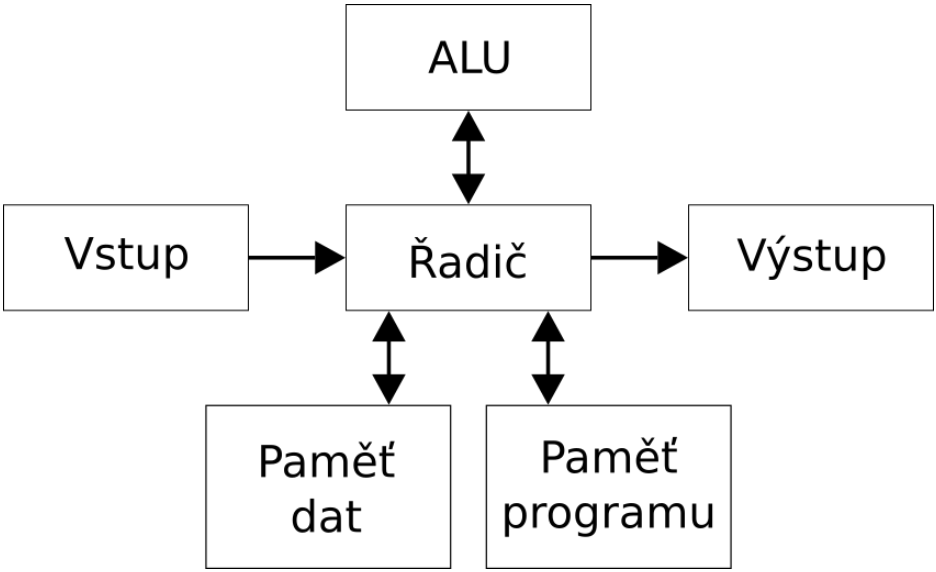
V případě von Neumanovi koncepce počítače je pomocí vstupního zařízení do operační paměti přes aritmetickologickou jednotku nahrán program, který bude provádět výpočty. Obdobným způsobem jsou do stejné paměti umístěna i data, se kterými program pracuje. Jednotlivé kroky programu provádí aritmetickologická jednotka. Ta je v průběhu výpočtu spolu s ostatními moduly řízena řadičem a jednotlivé dílčí výpočty jsou ukládány do operační paměti. Po skončení programu je výsledek poslán přes aritmetickologickou jednotku na výstupní zařízení.

Rychlost zpracování instrukcí dnešními procesory je výrazně vyšší než rychlost komunikace s pamětí. Komunikace s pamětí se tak stává nejslabším článkem řetězu ve von Neumannově architektuře. Tuto nevýhodu částečně řeší tzv. paměťové cache, což jsou rychlé mezipaměti, do kterých se potřebná data a instrukce z pomalejší hlavní paměti načítají dříve, než jsou při zpracování potřeba.

2.2 Harwardská koncepce počítače

Harvardská architektura počítače fyzicky odděluje paměť programu a dat a jejich spojovací obvody. To umožňuje paralelní přístup k oběma pamětem (lze v jednu chvíli číst instrukce programu z programové paměti a číst/zapisovat do datové paměti), což zvyšuje rychlost vykonávání programu.

U harvardské architektury není potřeba mít paměť stejných parametrů a vlastností pro data a pro program. Paměti můžou být naprosto odlišné, mohou mít různou délku slova, časování, technologii a způsob adresování a umožňuje v podstatě zdvojnásobení velikosti paměti oproti von Neumanově architektuře při stejně veliké adresové sběrnice.



Dost často se ale vyskytují hybridní koncepce, které v sobě kombinují prvky z von Neumanovy a Harwardské koncepce. Například koncepce společné paměti pro data i instrukce programu, která obsahuje paměťové rozhraní pro čtení dat a pro čtení instrukcí. Instrukce a data jsou sice uloženy ve stejné paměti, ale je možné k nim přistupovat současně.

Kapitola 3

Součásti mikroprocesoru

Pro lepší popis funkce a konstrukce procesoru je jeho konstrukce rozdělena na několik základních částí:

- Řadič
- Aritmetickologická jednotka
- Soubor registrů
- Sběrnice
- Periferní obvody
- Paměti a adresové prostory

Reálné procesory obsahují ještě další části, které umožňují zrychlit vykonávání programů, ale jedná se o dodatečné části, které nejsou potřeba k vykonávání základní funkce.

3.1 Řadič

Hlavním úkolem řadiče procesoru je prostřednictvím čítače instrukcí vyzvednout instrukci z hlavní paměti a uložit do dekodéru instrukce. Tam se dekóduje operační znak instrukce a na jeho základě se vygenerují řídící signály.

Dalším úkolem řadiče je zpracování adresní části instrukce. Adresní část instrukce obsahuje údaje, které určují operandy, s nimiž se mají vykonat operace. Zjišťuje se také jestli se instrukce skládá z jednoho, dvou, nebo tří slov.

Řadič může být řešen dvěma způsoby:

- Řadič s pevnou logikou
- Mikroprogramový řadič

Řadič s pevnou logikou je realizován číslicovým obvodem, který přijímá pouze pevně daný set číslicových kombinací, které reprezentují jednotlivé instrukce, operace procesoru. Výhodou řadiče s pevnou logikou je vysoká rychlost zpracování instrukcí, daných programem v paměti. Nevýhodou pak je omezený instrukční soubor, který je omezený pouze na základní operace.

Mikroprogramový řadič vykonává instrukce jako posloupnosti jednodušších mikroinstrukcí. To znamená, že pro každou instrukci tvoří taková posloupnost mikroprogram. Mikroprogramy všech instrukcí jsou uloženy v paměti mikroprogramů, do které mikroprocesor vstupuje při dekódování a provádění jednotlivých instrukcí. Jestliže je tedy každá instrukce reprezentována vlastní binární číselnou hodnotou, je možné do daného řadiče naprogramovat libovolný instrukční soubor. Nevýhodou je ale nižší rychlost vykonávání instrukcí daného programu.

Řadič procesoru většinou obsahuje speciální registry pro uložení operačního kódu instrukce - **instrukční registr**. Na základě obsahu instrukčního registru je řízen další průběh provádění instrukce. Má-li instrukce adresovou část, je tato část uložena do **pomocného adresového registru**, který bude během následujícího provádění instrukce dodávat adresu pro paměť dat. Má-li instrukce datovou část, je tato část uložena do **pomocného datového registru**, do kterého jsou uloženy konstanty z paměti programu, které jsou dostupné během provádění instrukce. Tyto pomocné registry jsou zpravidla programátorovi nepřístupné a neviditelné. Data jsou do nich načítány a čteny po vnitřních datových sběrnicích.

Operační kód instrukce je zpracováván uvnitř řadiče instrukčním dekodérem, který na základě obsahu instrukčního registru generuje signály, které řídí jednotlivé části procesoru.

Řadič inkrementuje hodnotu programového čítače, tak aby vždy ukazoval na následující instrukci, která se má vykonat. V tomto případě je potřeba také počítat s délkou instrukce. Tedy pokud má instrukce délku 4 bajty, je potřeba inkrementovat hodnotu programového čítače o hodnotu 4, jinak by programový čítač ukazoval na adresu operandů instrukce. Výjimku tvoří instrukce skoku, která způsobí skok v programu, tedy úpravu hodnoty v registru programového čítače.

3.2 Aritmetickologická jednotka

Aritmetickologická jednotka, označována zkratkou ALU (arithmetic logic unit) je hlavní výkonná část procesoru, ve které se vykonávají aritmetické a logické operace. Je ovládána řídícími signály z řadiče, generované na základě operačního kódu instrukce, které určují druh prováděné operace. U jednoduchých typů mikroprocesorů ALU dokáže většinou jen sčítat, odečítat, provádět přesuny dat a některé logické operace jako AND, OR, rotace vpravo a vlevo, popřípadě některé další. Ostatní operace jsou řešeny softwarově kombinací základních operací. U výkonných

mikroprocesorů ALU kromě základních operací dokáže také násobit a dělit v délce slova odpovídající šířce vnitřních datových sběrnic mikroprocesoru nebo ve dvojnásobné přesnosti.

Aby ALU mohla vykonávat požadované operace je nutné dopravit na její vstupy operandy a někam uložit výsledek operace. Do aritmetických obvodů vstupují dva operandy A a B. Příprava na provádění výpočtů v ALU probíhá ve dvou taktech:

- V prvním taktu jsou nahrány operandy do pomocných registrů.
- V druhém taktu dojde k samotnému výpočtu a uložení výsledku do cílového registru.

Protože je u některých instrukcí zapotřebí nějaký druh aritmetické operace nad operandy, například úprava cílové adresy, je možné k těmto operacím buď využít obvody aritmetickologické jednotky, což výrazně zpomaluje provádění programu (ALU nemůže vykonávat více operací najednou, protože má pouze jeden vstup a výstup, ale zároveň toto řešení výrazně zjednodušuje danou architekturu procesoru) a nebo je možné doplnit vnitřní obvody procesoru o dodatečné jednoúčelové aritmetické obvody, které vykonávání programu výrazně zrychlí (například sčítačka/odčítačka u programového čítače, pro účely skoků v programu).

Výsledek aritmetickologické jednotky bývá doplněn redundantními informacemi, které informují o dodatečných vlastnostech operace. Těmto informacím se říká příznaky a nacházejí se v příznakovém registru. Příznaky dokumentují správnost výsledku a používají se jako podmínky pro větvení programu. Základní příznaky jsou:

- Přetečení výsledku aritmetické operace (výsledek je větší než je možné uložit do cílového registru) - OV (owerflow)
- Nulový výsledek (výsledkem aritmetické operace je nula) - Z (zero)
- Přenos do vyššího řádu - C (carry)
- Znaménko výsledku - S (sign)
- Parita operace - P (parity)

3.3 Soubor registrů

Registr je malá paměťová buňka, která má obvykle velikost v násobcích čísla dvě, která je přímo přístupná procesoru. Obsah registru může mikroprocesor načíst rychleji než data uložená jinde. Velikost vnitř-

ních registrů určuje kolik informací je procesor schopen zpracovat během jednoho instrukčního cyklu a jakým způsobem jsou data v procesoru přesouvána. Procesor při vykonávání aritmetických a logických operací využívá registry pro rychle přístupné úložiště pro mezivýsledky, čímž urychluje svou činnost. Některé procesory mají větší vnitřní registry než šířku vnější datové sběrnice. Takový procesor musí naplňovat registry ve více cyklech a teprve poté data zpracuje. Ale existují i procesory, které mají datovou sběrnici větší než vnitřní registry. V takovém případě je datová sběrnice rozdělena do několika větví čímž procesor může zpracovat více instrukcí během jednoho instrukčního cyklu → superskalární architektura. Procesor často obsahuje několik typů registrů, které mohou být klasifikovány podle jejich obsahu nebo instrukcí sloužící pro práci s nimi. V procesoru se nacházejí registry, které mají své speciální využití. Dohromady se těmto registrům řídká **soubor registrů procesoru**

Registry jsou dnes obvykle implementovány jako soubor registrů v procesoru. Procesor často obsahuje několik typů registrů, které mohou být klasifikovány podle jejich obsahu nebo instrukcí sloužící pro práci s nimi:

- Uživatelsky přístupné registry. Nejčastější rozdělení je na **datové** a **adresové** registry.
- Datové registry jsou využívány pro uložení mezivýsledků aritmetických operací a uložení jiných číselných hodnot. Adresové registry v sobě uchovávají adresy do paměti, které využívají instrukce pro nepřímé adresování.
- Jiné registry

Registry procesoru lze podle jejich funkce rozdělit do několika kategorii:

- Registry pro všeobecné použití (pracovní registry)
- Ukazatelé na zásobník
- Adresové (linkové) registry
- Konfigurační registry
- Příznakový registr
- Programový čítač
- Registry pro speciální použití

3.3.1 Registry pro všeobecné použití

Registry pro všeobecné použití nebo také pracovní registry jsou registry, které se používají pro ukládání mezivýpočtů ALU, adres do pomněti a jiným účelům. Jedná se o nejvíce používané registry. Obvykle se jich v procesoru nachází několik, protože je třeba v jednu chvíli uchovávat více než jedu běhovou hodnotu.

3.3.2 Ukazatel na zásobník

Zásobník je speciální datová struktura, v pojetí hardwaru je to speciální paměť, jejíž přístup je typu LIFO (Last In Frist Out) tedy poslední dovnitř, první ven. Pro manipulaci s uloženými daty slouží tzv. **ukazatel na zásobník**. Zásobník se používá především k uložení stavu pracovních registrů (kontext procesoru) při vstupu do podprogramu, přepínání mezi softwarovými vlákny nebo celými procesy. Zásobník může být realizován speciální paměti v procesoru a nebo jako vymezená část v hlavní operační paměti (paměti dat).

Ukazatel na zásobník je speciální registr procesoru, který v sobě uchovává paměťovou adresu ukazující na naposledy vložená data. Pro každá data je v zásobníku rezervováno stejně velké místo, bez ohledu na velikost těchto dat. Pro ukládání a čtení dat slouží speciální instrukce PUSH a POP. Tyto instrukce mají za úkol vložit data na následující pozici v zásobníku a zároveň upravit adresu v ukazateli na zásobník - přičíst/odečíst k aktuální hodnotě v registru hodnotu x, která definuje počet čtených, nebo ukládaných dat. To lze zajistit také kombinací instrukcí pro uložení/načtení dat a inkrementací/dekrementací hodnoty v ukazateli na zásobník, ale instrukce PUSH a POP jsou atomické, což znamená, že se vykonají celé a nebo vůbec. Nehrozí tedy přerušení procesu ukládání dat do zásobníku.

Adresa aktuálně uložená v registru ukazatele na zásobník se nazývá **strop zásobníku** a adresa na které se nachází data, která byla do zásobníku uložena jako první se nazývá **dnem zásobníku**.

Při startu procesoru se vždy registr SP nastavuje na určitou hodnotu, která definuje začátek zásobníku. Tuto hodnotu lze přednastavit v inicializační sekvenci bootloaderu, která také nahrává programové vybavení procesoru do hlavní operační paměti. Tato počáteční adresa (hodnota) musí být vhodně zvolená, aby nedošlo ke kolizi s jinou částí paměti. **Hloubka zásobníku** definuje počet vložených dat (počet vnoření) do zásobníku.

3.3.3 Adresové registry

Adresový registr má několik využití. Typicky se používá k uložení

nějaké paměťové adresy, která je použita v určité instrukci procesoru. Přitom se instrukce na tuto adresu odkazuje pouze jménem adresového registru. To je ve vyšších programovacích jazycích využíváno ukazateli na adresu.

Adresové registry jsou také využívány pro uchovávání návratové adresy podprogramu. Před vstupem do podprogramu je do adresového registru vložena aktuální hodnota programového čítače, ke které se přičte jednička, jinak by čítač opět ukazoval na instrukci volání podprogramu. Následně se uloží stavy vybraných registrů procesoru do zásobníku a při návratu z podprogramu se opět obnoví a do programového čítače se uloží hodnota z adresového registru. Program pak pokračuje následující instrukcí za voláním podprogramu.

3.3.4 Programový čítač

3.3.5 Konfigurační registry

3.3.6 Příznakový registr

3.3.7 Registry pro speciální použití

3.3.8 Endianita

Endianita vyjadřuje způsob uložení čísel v operační paměti počítače, který definuje, v jakém pořadí se uloží jednotlivé bajty číselného datového typu. Jde tedy o to, v jakém pořadí jsou v operační paměti uloženy jednotlivé řády čísel, které zabírají více než jeden bajt.

3.4 Sběrnice

Sběrnici tvoříé skupina signálových vodičů sloužících k posílání dat mezi dvěma zařízeními, popřípadě obvody, které lze podle druhu přenášených dat rozdělit na několik druhů:

- **Datová sběrnice** - pro přenos dat
- **Adresový sběrnice** - pro přenos adres k adresovému dekodéru
- **Řídící sběrnice** - pro přenos řídících signálů, které mají vyvolat nějakou odezvu, nebo akci určitého zařízení či obvodu.

Jednotlivé sběrnice mohou tvořit stromovou strukturu, kdy jedna sběrnice může vést k více zařízením a nebo na jiné sběrnice. V takovém

případě musí platit, že na pomalejší sběrnice jsou napojeny sběrnice rychlejší aby nedocházelo ke zpomalení přenosu.

Na sběrnici mohou být připojeny periferie, které mohou pracovat v jednom ze třech režimů:

- vysílač - periferie, která může data pouze vysílat
- přijímač - periferie, která může data pouze přijímat
- přijímač a vysílač - periferie, která data může přijímat i vysílat

V případě, že je sběrnice sdílena mezi více periferiemi může po nich v jednu chvíli vysílat pouze jedno zařízení. Periferie, která data po sběrnici vysílá je **řídící periefie** a periferi, která data na sběrnici přijímá je **podřízená periferie**.

3.4.1 Adresová sběrnice

Vždy je ale nutný mechanismus adresování, který určí cíl přenášených dat. K určení cílového zařízení slouží **adresový dekodér**. Jedná se o číslicový obvod který na základě dané adresy přivede na n-tý (kód 1 z n) výstupní vodič logickou hodnotu jedna. To aktivuje danou periferii, která přijme a zpracuje data z data sběrnice. Pokud je cílovou periferií samotný integrovaný obvod, bude se zřejmě jednat o jeho vstup CS.

Rozlišuje se **úplné adresové dekódování** a **částečné adresové dekódování**. V případě částečného dekódování jsou na adresový dekoder přivedeny pouze nejvyšších vodiče (bity) adresové sběrnice. Výstupní vodiče adresového dekodéru tvoří jednu z řídících sběrnic. Zbylé nižší bity paměťové adresy jsou využity pro adresování v rámci dané periferie (například paměť, registr, ...). To umožňuje jednoduší zacházení se složitějšími periferiemi, ale zároveň se jedná o způsob, který je neefektivní k využití paměťového rozsahu daného procesoru (u některých periferií nemusí být využity všechny adresy).

U úplného adresového dekódování jsou na vstup adresového dekodéru přivedeny všechny vodiče adresové sběrnice. Díky tomu jsou využity všechny paměťové adresy z daného rozsahu. V případě, že je nutné na dané periferii vybírat mezi více částmi, je na danou periferii vyvedeny vyvedeno více vstupů CS na které jsou navedeny výstupy z adresového dekoderu.

Adresová sběrnice je vždy jednosměrná, není tedy nutné řešit režimy čtení a zápisu.

3.4.2 Datová sběrnice

Datové sběrnice lze rozdělit na **jednosměrné** a **obousměrné**. V případě jednosměrné sběrnice mohou data proudit pouze jedním směrem. To znamená, že z dané periferie lze pouze číst.

V případě, že je nutné aby data proudila oběma směry jsou vyvedeny dva nezávislé vodiče, které paralelně realizují čtení a zápsi. Takové řešení je jednoduché (není třeba řešit režimy čtení a zápisu) a umožňuje rychlejší přenos dat. Nevyhodou je nutnost vedení jednoho vodiče navíc. V případě jednosměrné datové sběrnice je výhodou nižší hardwarové nároky na signálové vedení, ale složitější řešení obousměrné komunikace. Obousměrná komunikace více periferií po jednom vodiči vyžaduje komunikační protokol, který zamezuje kolizím při komunikaci. Aby se

Čtení a zápis jsou řízeny a přesně časovány pomocí řídících signálů. Vstupy periferií pro řízení čtení nebo zápisu se většinou označují **RD** pro čtení a **WR** pro zápis. Posloupnost činností (zápis adresy na adresovou sběrnici, dekódování adresy, ...), během kterých dojde právě jednou ke čtení nebo zápisu prostřednictvím sběrnic, se nazývá **sběrnicový cyklus**. 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 ... Dále je možné rozlišovat umístění sběrnic. **Vnitřní sběrnice** jsou umístěni uvnitř počítače (v procesoru, základní desce, ...) a slouží ke komunikaci mezi jednotlivými částmi (periferiemi) počítače. **Vnější sběrnice** jsou umístěny vně počítače a slouží ke komunikaci s externími zařízeními nebo s jinými počítači.

3.5 Periferní obvody

Periferní obvody tvoří velice rozmanitou skupinu obvodů. Příkladem periferních obvodů jsou vstupní a výstupní zařízení, čítače, časovače, ... Velmi často je několik periferních obvodů sdruženo do jednotek. Například několik vstupně výstupních bran pinů.

Ze strany datové sběrnice s nimi komunikuje procesor, z druhé strany pak okolí počítače. Jelikož v jednotce bývá sdruženo několik periferních obvodů, jsou data ukládána do několika datových registrů (pro každý obvod jeden). Velmi často je nutné činnost těchto obvodů nějak řídit. K tomu slouží skupina **řídicích registrů**. Dále je třeba zjišťovat stav těchto obvodů. K tomu slouží skupina **stavových registrů**. Procesor může prostřednictvím datové sběrnice zapisovat do řídicích registrů a číst stavové registry. Jednotka může vyžadovat obsluhu též prostřednictvím přerušení programu. K tomu slouží jeden nebo více **signálů INT** (in-

terupt). Podnětem k vyvolání přerušení je dokončení požadované operace nebo případné hlášení chyby. Do jednotky je vedeno několik (nejnižších) adresových bitů pro rozlišení mezi různými datovými, řídicími a stavovými registry. Z tohoto důvodu musí uvnitř jednotky existovat ještě **lokální adresový dekodér**, který vybírá mezi jednotlivými registry. Signál 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.

3.6 Paměti a adresové prostory

Kapitola 4

Instrukční soubor procesoru

ISA - Instrukction Set Architecture je sada základních operací, které je procesor schopen vykonávat na hardawarové úrovni. Úkolem každého procesoru je vykonávat instrukce. Instrukce je nejmenší jednotkou ze které lze sestavit nějaký program - algoritmu vykonávaný procesorem. **Instrukční soubor** definuje základní skupinu operací, které je schopen procesor vykonat na hardwarové úrovni. Tyto operace jsou realizovány pomocí speciálních číslicových obvodů. V procesorech se obecně vyskytuje několik druhů instrukcí. Ve všech případech je ale hlavní částí instrukce **operační kód**, který definuje typ (provádění) dané operace.

Některé instrukce obsahují pouze operační kód. V takovém případě jsou prováděny pouze vnitřní operace v procesoru, které nepotřebují žádné dodatečné informace. V dalším případě může instrukce obsahovat dodatečné informace. Takovými informacemi může být adresa operandu v paměti dat (načti hodnotu z paměti do registru) a přímá data, konstanty, které jsou součástí programu (nacházejí se v paměti programu, načti danou hodnotu do registru). Instrukce mohou být i složitější a mohou obsahovat i různý počet argumentů. To znamená, že instrukce mají obecně různou délku.

Každá operace se skládá z tzv. mikroinstrukcí. To jsou jednotlivé mezi operace, které dohromady vykonávají určitou operaci v procesoru. Příkladem je načtení operandů, provedení aritmetického výpočtu a uložení výsledku do cílového registru. Na principu práce s mikroistrukcemi je založena architektura CISC. Operační znak určuje nejen druh operace, ale může obsahovat i další informace, například délku instrukce nebo adresy operandů.

Souhrn všech činností, potřebných pro vykonání jedné instrukce se nazývá **instrukční cyklus**. Instrukční cyklus lze dále rozdělit na jednotlivé mezikroky, které tvoří danou instrukci, které se nazývají **strojové cykly**. V každém strojovém cyklu se právě jedna mikroinstrukce (zápis, čtení, přesun, ...) Ale vždy první strojový cyklus představuje čtení operačního kódu instrukce z paměti programu. Následující strojové cykly jsou určeny druhem vykonávané instrukce. Na každý strojový cyklus připadá jeden takt vnitřního synchronizačního (taktovacího, hodinového) obvodu (oscilátor).

Vykonávání instrukce se skládá ze dvou částí - čtení instrukce a provedení instrukce. Při čtení instrukce se vykonávají čtecí cykly z paměti programu tak, aby se přečetly všechny části instrukce. Počet čtecích cyklů odpovídá délce instrukce (jeden čtecí cyklus může načíst pouze jednu část instrukce). Adresa instrukce je adresa na které se nachází operační kód instrukce. Při provádění instrukce se vykonává vnitřní logika procesoru včetně získávání potřebných operandů instrukce a uložení výsledků (součet, součin, přesuny, ...).

4.1 Typy instrukcí a jejich vykonávání

4.2 Architektura RISC a CISC

4.3 Adresace paměti

Adresa slouží k určení zdroje, nebo cíle dat. Existuje několik způsobů adresace:

• **Bezprostřední data** jsou obsažena již v instrukci, která v tomto případě obsahuje datovou část. Žádná adresa zde není potřeba.
• **Přímá adresa** je obsažena v instrukci podobně jako v případě bezprostředních dat. Hodnota v tomto případě ale nehraje roli konstanty, ale adresy. Instrukce v tomto případě obsahuje adresovou část.
• **Nepřímá adresa** neobsahuje samotnou adresu, ale pouze místo (ukazatel na data) na kterém se tato adresa nachází. Tímto místem může být registr, nebo oblast v paměti. Během provádění instrukce je tento zdroj přečten a tak je teprve získána cílová adresa. V případě, že je nepřímá adresa obsažena v některém registru procesoru, je její čtení a dekódování rychlé. Nevýhodou je ale malý počet míst kam je možné adresu uložit. Naopak v případě, že je nepřímá adresa uložena v paměti, je počet možností kam adresu uložit výrazně vyšší. Nevýhodu je ale jeden čtecí cyklus navíc, protože adresu paměti, kde je uložena nepřímá adresa je nutné nejprve přivést na adresový dekodér a teprv pak přečíst hodnotu nepřímé adresy.
• **Relativní adresa** obsahuje adresovou část, takzvaný **offset**, jejíž obsah se přičte k obsahu čítače instrukcí. To se využívá nikoli ke čtení, nebo zápisu dat, ale k získání adresy instrukce, která se má vykonat v následujícím instrukčním cyklu. To se využívá například při skoku v programu, programové smyčky, ... Offset je vždy chápán jako číslo se znaménkem a proto může být skok v programu jak o určitý počet instrukcí kupředu

tak o určitý počet instrukcí zpět. Velikost offsetu bývá většinou kratší než přímá adresa, takže se jedná o paměťově úspornou adresaci, ale vzdálenost skoku, kterou lze skokem překlenou je tak omezena.
• **Indexová adresa** používá indexový registr v řadiči procesoru, k jehož hodnotě je přičtena hodnota operandu instrukce a tento součet je teprve použit jako adresa pro data (přímá indexová adresace) nebo jako zdroj adresy (nepřímá indexová adresace). Jedná se o podobný způsob jako relativní adresace, ale s tím rozdílem, že indexová adresa je používána pro výpočet adresy pro data. Indexová adresa je výhodná pro práci s datovou strukturou pole.

Některé programové algoritmy vyžadují procházet postupně určitý úsek paměti, například průchod polem. To vyžaduje postupné zvyšování, nebo snižování paměťové adresy. K tomu v procesorech slouží **autoinkrementace** nebo **autodekrementace** paměťové adresy. To je vnitřně prováděno instrukcí inkrementace, popřípadě dekrementace některého z adresových registrů. Počet inkrementačních cyklů je dán nějakou podmínkou v programu. Hodnota uložená v adresovém registru následně slouží jako přímá adresa do paměti.

Kapitola 5

Architektury procesoru

Architektura procesoru založená na práci s jednotlivými operandy instrukcí může být řešena různými způsoby, které mají v různých situacích své výhody i nevyhody. V praxi nebývá procesor, který využívá čistě jednu architektonický model, ale je sožen z několika architektur, které můžou být využívány v daných situacích, pro které jsou optimální.

5.1 Architektura load store

Architektura **load store** je architektura procesoru, která umožňuje přístup do paměti pomocí instrukcí pro načítání (load) a ukládání (store). Pokud jsou tyto hodnoty požadovány jako parametry nějaké instrukce je třeba je nejprve překopírovat do nějakého z pracovních registrů procesoru (argumentem instrukce **nemůže** být adresa do paměti). Výsledek dané instrukce je opět uložen do některého z pracovních registrů odkud je možné ho pomocí specializované instrukce překopírovat do paměti.

5.1.1 Register base architektura

5.1.2 Akumulator base architecture

5.2 Architektura register memory

5.2.1 Stack-base architektura

Stack-base architecture využívá k adresování operandů instrukce zásobník. Jedná se o datovou strukturu LIFO (Last In Firest Out). Díky tomu, že zásobník je v procesoru přístupný prostřednictvím jednoho speciální registru - **Stack pointer**, instrukce nevyžaduje pro vykonávání operací nad operandy žádné paměťové adresy - vše se odehrává nad registry uvnitř ALU. Průběh vykonávání instrukce tedy může vypadat nějak takto:

```
POP x
POP y
ADD x,y, result
PUSH result
```

Kapitola 6

Čítače a časovač

Čítače a časovače jsou obvody, které slouží k přesnému časování, nebo čítání vnějších událostí nebo synchronizaci programu s okolím. Díky tomu lze pracovat s procesorem v reálném čase. Čítače a časovače lze nahradit softwarovými konstrukcemi, ale samostatné hardwarové periferie zvyšují přesnost operace, nezatěžují procesor a tím zvyšují jeho výkon v dané aplikaci. Čítač a časovač bývá označován zkratkou **TC** (Timer and Counter). TC jsou většinou tvořeny 8-bitovými nebo 16-bitovými čítacími registry. Reálné procesory jsou vybaveny většinou více než jedním TC obvodem.

Čítač a časovač tvoří jeden obvod, který plní danou funkci podle jeho nastavení v konfiguračním registru. **Čítač** spravuje nějaké paměťové místo, v hardwarové realizaci většinou číslicový čitač, k jehož uložené hodnotě je možné přičítat nebo odečítat jedničku (inkrementovat/dekrementovat) na základě zjištěné náběžné nebo sestupné hraně nějakého sledovaného signálu. Sledovaným signálem může být například výstupní signál nějakého čidla. **Časovač** stejně jako čítač umožňuje přičítat nebo odečítat od uložené hodnoty jedničku, ale v jeho případě je sledovaný signál vnitřní hodinový signál procesoru s předem známým průběhem a konstatní frekvencí. Pomocí časovače lze měřit čas nezávisle na běhu programu (časovač pracuje paralelně se zbytkem procesoru). Díky tomu lze vytvořit časová zpoždění a prácovat v reálném čase. Důležité je, že daný obvod může v jednu chvíli pracovat buď jako čítač a nebo jako časovač, ale ne obojí současně.

Rozdíl mezi čítačem a časovačem je ve zdroji budícího signálu. Zdroj hodinového signálu může být vnitřní (hodinový signál procesoru), nebo vnější napojený na speciální vstupní pin.

Hodnotu z čítacího registru lze číst nebo ji měnit podle potřeb běžícího programu. **Rozlišení TC** je dáno počtem bitů n čítacího registru. Jeho vnitřní hodnota se pak může pohybovat v intervalu $0, (2^n - 1)$. V případě, že dojde k přetečení čítané hodnoty, dojde k vyvolání přerušení. Toho lze využít v případě časování na pozadí procesoru, nastavením požadované počátení hodnoty čítače. V případě potřeby cyklyckého časování je nutné po přetečení čítače jej opět nastavit (v obsluze přerušení) na jeho počáteční hodnotu jinak dojde k inkrementaci/dekrementaci od jeho maximální/minimální hodnoty.

6.1 Režim čítání impulsů

Čítač není možné číst kdykoli. V případě, že by se v době čtení hodnota čítače právě inkrementovala/dekrementovala, mohlo by dojít k přečtení špatné hodnoty. K řešení tohoto problému slouží několik řešení. Prvním řešením je synchronizace čítacích impulsů čítače a čtecích cyklů vnitřní sběrnice. Další možností je

V případě že je čítač více bytový a není možné jej přečít v jedné instrukci může nastat problém kdy dojde přeteční nižšího bytu čítače do vyššího bytu čítače a výsledek čtení bude chybná hodnota (hodnota čítače je 0x00FF, v následujícím cyklu dojde k přetečení na hodnotu 0x0100, v případě přečtení spodního bytu je přečtena hodnota 0xFF a při následujícím čtení bude přetena hodnota 0x01, výsledek čtení tedy bude 0x1FF). Tento problém je řešen registrovým bufferem, který v sobě uchovává hodnotu vrchního bytu. V případě přečtení spodního bytu čítače je současně do tohoto bufferu načtena hodnota horního bytu čítače. Hodnota uložená v bufferu se nemění současně s čítačem a je možné jej bezezměny přečíst v dalším cyklu.

6.2 Režim časování

TC je ve výchozím nastavení časován pomocí vnitřního časovacího signálu, který řídí také ostatní části procesoru. Na TC může být také přiveden externí časovací signál. V případě, že je na vstup TC přiveden vnitřní časovací signál, jedná se vždy o relativně vysokou frekvenci, která způsobý přetečení čítacího registru ve velmi krátkém časovém okamžiku. Pro snížení frekvence řídícího signálu je využita předdělička, která sníží frekvenci v určitém poměru. Předdělička je v podstatě další čítací registr, který je modulován určitou hodnotou. To znamená, že je možné nastavit jeho maximální hodnotu. V případě jeho přetečení vyšle signál na vstup čítacího registru TC. Díky tomu dojde ke snížení výsledné časovací frekvence.

Zrnitost (granularita, anglicky granularity) časovače vyjadřuje časový interval potřebný pro jednu inkrementaci hodnoty čítacího registru TC. V případě 8-bitového registru, který je časován frekvencí 1MHz je doba potřebná pro jeho přetečení rovna $\frac{255}{1000000s} = 255\mu s$, to znamená, že granularita tohoto časovače je rovna $\frac{255\mu s}{2^8-1} = 1\mu s$. Pomocí předděličky lze snížit frekvenci například 1024-krát. Granularita časovače pak bude rovna $\frac{1000000}{1024} \doteq 976\mu s \doteq 1ms$.

6.3 Parametry čítače a časovače

V případě jednodušších TC je možné detekovat buď pouze náběžnou hranu a nebo sestupnou hranu. V případě komplexnějších TC je ale možné mezi těmito režimi přepínat. To samé platí o směru čítání. Jednodušší čítače umožňují čítat pouze jední přesně definovaným směrem, ale komplexnější TC umožňují čítat oběma směry (inkrementovat i dekrementovat). Tento nedostatek u jednodušších TC bývá řešen více obovdy TC, kde jeden umožňuje čítat nahoru a druhý směrem dolu.

Dalším parametrem je velikost daného číte, která definují jakou maximální hodnotu může daný čítač uložit.

Pro snížení celkové spotřeby procesoru je možné každý jednotlivý TC obvod vypnout.

6.4 Nastavení čítače a časovače

6.5 PWM

Kapitola 7

Watchdog timer

Watchdog timer (nebo jen **watchdog**), zkráceně **WDT** (hlídací pes) je periferie mikroprocesoru, která umožňuje resetovat celý mikroprocesor v případě, že dojde k vynulování časovače. Watchdog slouží jako hlídací obvod, který umožňuje detekovat jakékoli anomálie, které nastanou při běhu programu. Bez Watchdogu by bylo nutné zaseknutý procesor ručně resetovat. Watchdog na tyto anomálie zareaguje resetováním procesoru, nebo vyvoláním přerušení a nebo obojí současně. Watchdog je spuštěn buď bezprostředně po resetu procesoru a nebo někdy později z vykonávaného programu. Jedná se o samostatnou periferii, a proto není závislá na činnosti zbytku procesoru. Pokud dojde k uváznutí procesoru, watchdog pracuje bezezměny dál. Timeout watchdogu musí být zvolen větší než nejdelší předpokládaný čas na vykonání hlídané rutiny.

Watchdog timer svou činností upravuje příznakové bity v příznakových registrech různých periferií procesoru. Především se jedná o bity WDT iterupt flag, WDT reset. Ty můžou být využívány jako příznaky těchto událostí v dalším vykonávání programu. Typyckým příkladem je zálohování běhových dat v obsluze přerušení do nevolatilní paměti. Na začátku programu se následně zkontrolují dané příznaky a běhová data se obnoví. V případě modu přerušení dojde po prvním vyvolání přerušení k automatickému zakázání přerušení v případě následujících timeoutů watchdogu (je nutné jej v obsluze přerušení opět povolit).

V případě, že watchdog nebude v programu využíván je vhodné, aby nedošlo v průběhu vykonávání kódu k neočekávanému resetu způsobeného obvodem watchdog, na začátku programu (sekce setup) nastavit watchdot timer do neaktivního stavu. Počáteční kód tak vždy odpojí watchdog a nehrozí tak neočekávané chování programu.

7.1 Popis watchdog timeru

Základem watchdog timeru je n-bitový čítač, napojený na hodinový signál s frekvencí y xHz, který odčítá v pravidelných časových intervalech s uloženou počáteční hodnotu dokud nedosáhne nuly. V případě, že není čítač resetován dojde po uplynutí časové doby k přetečení čítače. Doba ve které je nutné čítač resetovat je dána vztahem:

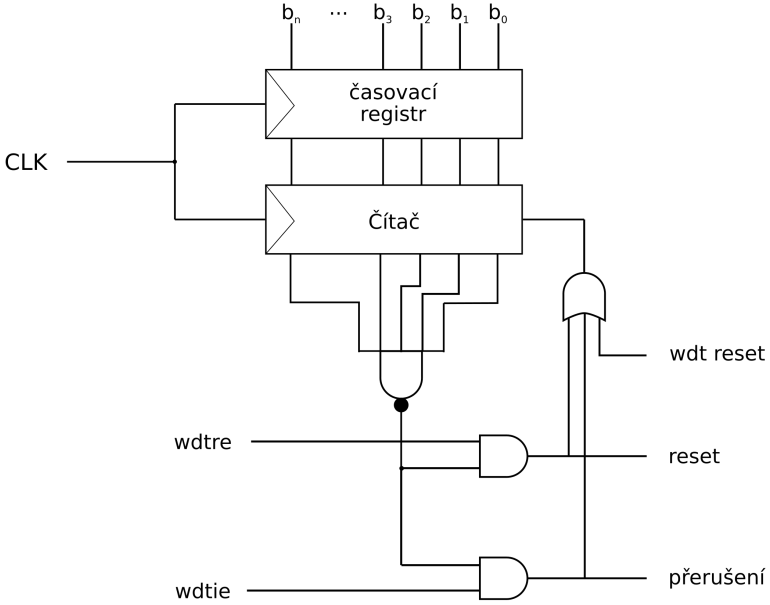
$$t = \frac{2^{n-1} + 2^{n-2} + ... + 2^0}{y \cdot 10^x Hz}$$

Výsledný čas je v sekundách. Po přetečení čítače dojde k resetu procesoru nebo vyvolání přerušení nebo obojí současně.

Pro resetování watchdog timeru slouží buď nějaký servisní signál (bit), který je uložen v řídící registru (nastavení servisního bitu na danou hodnotu resetuje watchdog a zároveň invertuje hodnotu servisního bitu), nebo speciální instrukce, která resetuje watchdog.

Watchdog timer je speciálním případem časovače, který je schopen vyvolat hardwarový reset procesoru.

Watchdog timer je možné realizovat více způsoby Jedním ze způsobů je čítač/časovač na který jsou přivedeny vstupy **časového registru**, který v sobě drží hodnotu času, ve kterém může program watchdog resetovat. Při resetování watchdogu nebo celého procesoru (popřípadě při vyvolání přerušení) se z časového registru opět nahraje hodnota do čítače, který začne odčítat hodnoty. Při nahrání hodnot do čítače se automaticky deaktivují výstupy *reset* a *přerušení*. V případě manuálního vyvolání resetu watchdogu se zároveň po resetu automaticky deaktivuje vstup *wdt reset*.



V případě vynulování čítače dojde k aktivování watchdogu a v závislosti na aktivovaném vstupu dojde buď k vyvolání přerušení, nebo k resetu procesoru, nebo k obojímu současně, přičemž nejprve dojde k přerušení a následně k resetu procesoru. Přerušení před resetem procesoru slouží k ošetření stavu procesoru před resetem (uložení hodnot registrů, nastavení režimů, ...).

Jinou možností je realizovat watchdog timer jako čítač, s fixní délkou do který se nastavý vždy plná hodnota (všechny bity má nastaveny na hodnotu log 1). Následně je před hodinový signál přivedena signálová předdělička, která umožní zpomalit jeho frekvenci a nastavit tak čas odpočtu. Výhodou tohoto řešení je jednodušší reset watchdogu (přivedením signálu na vstup reset čítače, popřípadě přetečením čítače), ale nevýhodou je omezená možňost časování - čas odpočtu je možné nastavit pouze v poměru násobku mocniny dvou a frekvence časovacího signálu. Místo hodnoty času se následně nastavují hodnoty na předděličce, která upraví frekvenci vstupního řídícího signálu. Výsledný čas pro odpočet je dán vztahem:

$$t = \frac{2^n}{y \cdot 10^x Hz}$$

Na watchdog timer je přiveden buď hodinový signál procesoru a nebo disponuje vlastním oscilačním obvode, který generuje vždy stejnou frekvenci bez ohledu na vnitřní časovací frekvenci procesoru.

kde n je počet bitů čítače a y je frekvence řídího signálu

7.2 Využití watchdog timeru

Využití watchdogu je založeno na jednoduché myšlence. Do programu jsou vloženy příkazy resetující čítač watchdogu (uvedou ho do původního stavu) - program tak pravidelně hlásí svůj stav. V případě, že program pracuje správně nemůže dojít k vypršení časového limitu, ale v případě, že program, nebo některá z periferií procesoru nepracují správně, nedojde k včasnému resetování čítače a watchdog vyvolá reset procesor (přerušení). Typickým případem použití je zacyklení programu, kdy reset programu je jediný způsob jak uvolnit zaseknutý program.

Další možné využití watchdogu je jako **wakeup timer**, který umožňuje přejít z režimu spánku procesoru do aktivního režimu. V případě, že daný procesor disponuje obvody pro režim spánku, je možné mikroprocesoru sníži spotřebu, čehož je využíváno především u přenosných zařízení, která jsou napájeny bateriemi.

7.3 Nastavení watchdog timeru

Při nastavení watchdogu je nutné nejprve vypnout přerušení programu, aby nedošlo k přerušení před dokončením nastavení. Dále je vhodné resetovat čítač watchdogu aby nedošlo k time outu v průběhu nastavování. Watchdog disponuje určitými vrstvamy ochrany, aby nedošlo k nechtěným změnám při vykonávání programu. Typycký je uzamikací bit wdtce(Watchdog Timer Change Enable) řídícího registru, který musí být nastaven na určitou logickou úroveň, aby byly všechny ostatní bity odemčeny pro upravování. Tento bit se typicky po několika instrukčních cyklech opět překlopí a uzamkne řídící registr.

Je důležité před prvním použitím watchdogu v programu po nahrání časového intervalu do *časového registru* celý watchdog resetovat nastavením vstupu *wdt reset* nebo speciální instrukcí procesoru, aby došlo k nastavení času do čítače a následně teprve aktivovat celý obvod nastavením alespoň jednoho ze vstupů *wdtie* (watchdog timer interupt enable) nebo *wdtre* (watchdog timer reset enable).

Stav watchdogu je řízen pomocí registru **wdtcsr** (watchdog timer control and status register).

wdtre	*wdtie*	*reset*	*přerušení*
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

Hodnota časového registru se po resetu procesoru nebo watchdogu nijak nenuluje (bylo by nutné ji po resetu opět nastavovat) a proto se vždy nachází v předchozím stavu. Po spuštění procesoru se tedy nachází v nulovém stavu (všechny bity časového registru jsou v hodnotě log nula).

V případě registru *wdtcsr* se po resetu procesoru a watchdogu nuluje pouze bit *wdt reset*, který způsobuje nahrání hodnoty z časového registru do čítače watchdogu.

Kapitola 8

Obvod RESET

Obvody reset slouží k resetování stavu procesoru - nulování procesoru. Reset procesoru zahrnuje nulování pracovních registrů, nastavení konfiguračnícha řídících registrů do výchozí hodnoty, ... Jedná se o proces, který nastaví procesor do počátečního stavu. Obvykle se nastaví programový čítač na první adresu v paměti programu. Na první adrese v programové paměti nachází instrukce skoku (tabulka vektorů přerušení), která má za úkol zavést obsluhu přerušení resetu. Obsluha přerušení resetem má za úkol nastavit procesor do výchozího stavu.

Vsupy do obvodu RESET jsou nejčastěji tři:

- watchdog timer - vynulování watchdogu
- on power reset - reset po připojení napájení
- RESET MCU - nastavením vstupu RESET MCU na dobu delší než 50 ms
- Brown-out reset - při poklesu vstupního napětí

Za určitých okolností je potřeba rozlišovat k jakému typu resetu došlo. K tomu slouží registr **MCUSR** (MCU Status Registr). Pro každý typ resetu je definován jeden status bit, který je nastaven v případě, že nastane daný reset. Zároveň jsou všechny ostatní bity definující stav resetu nastaveny na logickou nulu.

8.1 Power-on reset

Power-on reset je speciální obvod, který zajišťuje reset procesoru po připojení napání. Důležitou součátí tohoto obvodu je časovač, který zajistí zpoždění spuštění programu dokud se vstupní napětí neustálí na požadované hodnotě.

8.2 RESET MCU

Vnější reset je nejčastěji generován logickou nulovou přivedenou na vstupu RESET daného procesoru. Reset je generován i v případě, že vnitřní hodinový sygnál není v činnosti. Po opětovném nastavení log 1 na vstup RESET je vnitřní reset generován ještě po dobu $t_{resetout}$. Díky vstupu RESET MCU lze při nějaké chybě ručně daný procesor resetovat.

8.3 Brown-out reset

Procesor vybavený detekcí napětí umožňuje sledovat hladinu vstupního napětí, které při poklesu pod prahovou úroveň způsobí aktivaci resetu. Detektor napětí umožňuje definovat určitou histerezi, díky které se zabraní aktivaci resetu při mírném zakolísání vstupního napětí:

$$U_{+p} = U_p + \frac{U_{HYST}}{2}, U_{-p} = U_p - \frac{U_{HYST}}{2}$$

Ke spuštění a obnovení normální činnosti procesoru dojde po ustálení napětí nad prahovou hodnotou - sestupná hrana resetu.

Kapitola 9

Přerušení

Přerušení (interrupt) je způsob asynchdoního obsluhu událostí, jehož principem je přerušení aktuálně vykonávané operace přepnout na předem definovanou obsluhu přerušení. Po vykonání obsluhy přerušení proceesor pokračuje v předchozí činnosti.

9.1 Obsluha přerušení

Obsluha přerušení je speciální podprogram, který je vyvolán při přerušení procesoru. Přijde-li do procesoru signalizace přerušení, je v případě, že obsluha přerušení je povolena, nejprve dokončena právě rozpracovaná strojová instrukce. Pak je na zásobník uložena adresa následující strojové instrukce, která by měla být zpracována, kdyby k přerušení nedošlo. Pak je podle tabulky přerušení vyvolána obsluha přerušení, která obslouží událost, kterou přerušení vyvolalo. Obsluha přerušení je zodpovědná za to, aby na jeho konci byl uveden stav procesoru do stavu jako na jejím začátku, aby výpočet přerušené úlohy nebyl ovlivněn, což se z důvodu vyšší rychlosti obvykle dělá softwarově (některé procesory umožňují uložit svůj stav pomocí speciální strojové instrukce). Na konci obsluhy přerušení je umístěna instrukce návratu (RET, někdy speciální IRET), která vyzvedne ze zásobníku návratovou adresu a tak způsobí, že z této adresy bude vyzvednuta následující strojová instrukce. Přerušená úloha tak až na zpoždění nepozná, že proběhla obsluha přerušení.

Všechna přerušení je třeba aktivovat nastavením řídícího bitu v řídícím registru dané periferie a bitu **I** ve stavovém registru SREG. V případě, že řadič přerušení nepodporuje vnoření přerušení je v případě aktivování přerušení, automaticky vynulován bit **I**, aby v průběhu vykonávání obsluhy přerušení nemohlo být vyvoláno jiné přerušení. Bit **I** je možné v obsluze přerušení opět nastavit a tak v rámci dané obsluhy přerušení ostatní přerušení povolit. Při návratu z obsluhy přerušení opět bit **I** automaticky nastaven, čímž jsou opět povolena globální přerušení.

V případě, že řadič přerušení podporuje vnoření přerušení může nastat při vykonávání obsluhy přerušení vyvolání dalšího přerušení.

9.2 Typy přerušení

Existuje několik druhů přerušení:

- Vnější přerušení
- Vnitřní přerušení
- Softwarové přerušení

9.2.1 Vnější přerušení

Vnější přerušení (též hardwarové přerušení) je označováno podle toho, že přichází ze vstupně-výstupních zařízení (tj. z pohledu procesoru přicházejí z vnějšku). Vstupně-výstupní zařízení tak má možnost si asynchronně vyžádat pozornost procesoru a zajistit tak svoji obsluhu ve chvíli, kdy to právě potřebuje bez ohledu na právě aktuálně procesorem zpracovávanou úlohu.

Vnější přerušení jsou do procesoru doručována prostřednictvím řadiče přerušení, což je specializovaný obvod, který umožňuje stanovit prioritu jednotlivým přerušením, rozdělovat je mezi různé procesory a další související akce.

9.2.2 Vnitřní přerušení

Vnitřní přerušení vyvolává sám procesor, který tak signalizuje problémy při zpracování strojových instrukcí a umožňuje nadřazenému systému (operačnímu systému) na tyto události nejvhodnějším způsobem zareagovat. Jedná se například o pokus dělení nulou, porušení ochrany paměti, nepřítomnost matematického koprocesoru, výpadek stránky a podobně.

9.2.3 Softwarové přerušení

Softwarové přerušení je speciální strojová instrukce procesoru, která umožňuje záměrně vyvolat přerušení aktuálně prováděného programu. Tento typ přerušení je na rozdíl od druhých dvou typů synchronní, je tedy vyvoláno zcela záměrně umístěním příslušné strojové instrukce přímo do prováděného programu. Jedná se o podobný způsob, jako vyvolání klasickému podprogramu (podprogramem je zde ISR uvnitř operačního systému), avšak procesor se může zachovat jinak. Instrukce softwarového přerušení se proto využívá pro vyvolání služeb operačního systému z běžícího procesu (systémové volání). Uživatelská úloha tak sice nemůže skočit do prostoru jádra operačního systému, ale může k tomu využít softwarové přerušení (kterých je omezené množství a vstupní body lze snadno kontrolovat). Při využití privilegovaného režimu může softwarové přerušení aktivovat privilegovaný stav operačního systému nebo celého procesoru.

9.3 Interupt ReQuest

Interupt ReQuest - IRQ označuje signál, kterým daná periferie (vnitřní vnější) požádá procesor o přerušení programu. Daná periferie vyšle IRQ řadiči přerušení v procesoru a požádá ho vyvolání přerušení. Řadič přerušení předá informaci o čekajícím přerušení procesoru. V případě, že daný procesor má povolené globální přerušení a zároveň daná periferie má také povolené přerušení programu, procesor dokončí následující instrukci a vyvolá obsluhu přerušení.

Přerušení je identifikované jedinečným indexem, které označuje adresu v tabulce přerušení. Dané přerušení je v zápisech (technických, programových) označováno zkratkou IRQ a indexem daného přerušení - IRQ8.

9.4 Tabulka přerušení

Tabulka přerušení nebo také **vektorová tabulka** je datová struktura, která uchovává seznam všech přerušení a adres korespondujících oblužných rutin přerušení. Umíst

Vektor přerušení je adresa paměti kde se nachází obsluhy daného přerušení nebo index pole tabulky vektorů přerušení, která tyto adresy obsahuje. Vektor přerušení je přirovnáván ke geometrickému vektoru, který je definován svou velikostí a směrem. Vektor přerušení je stejně jako geometrický vektor definován svou velikostí vyjadřující vzdálenost adresy obsluhy přerušení od umístění vektoru přerušení v paměti a směr, který je závislí na umístění tabulky vektorů přerušení v paměti. Umístění tabulky vektorů v paměti vyjadřuje směr umístění obsluhy přerušení v paměti programu.

Umístění vektoru přerušení v tabulce vektoru v paměti programu je možné získat vynásobením indexu daného přerušení a počtem bytů které tvoří paměťová adresa (šířka adresové sběrnice v bytech):

$$addr = \frac{index}{bus\ width}$$

Vektorová tabulka bývá nejčastěji umístěna na začátku a nebo na konci paměti programu. Platí že čím nižší index přerušení tím je jeho priorita vyšší. Nejvyšší prioritu má tedy reset přerušení, které je na indexu 0.

9.5 Průběh přerušení

Průběh přerušení závisí na jeho typu.

9.6 Řadič přerušení

Řadič přerušení je obvod, který umožňuje vyřizovat přicházející přerušení podle jejich nastavené priority.

9.7 Výjimky

Výjimky (Exception) procesoru je speciálním typem přerušení, kterým procesor reaguje na vzniklé chyby za běhu programu. Po vyvolání výjimky je zavolána obsluha výjimky, která je ekvivalentem obsluhy přerušení. Typickým příkladem výjimky je dělení nulou, přetečení, přístup na neexistující paměťovou adresu, ...Výjimky tak slouží k obnovení normálního stavu procesoru, aby nebyl narušen běh uživatelského programu.

Kapitola 10
Vstupní a výstupní obvody

10.1 Jednotka správy paměti

Kapitola 11
Hodiny reálného času

Kapitola 12
Programovací obvody

12.1 Paralelní programování

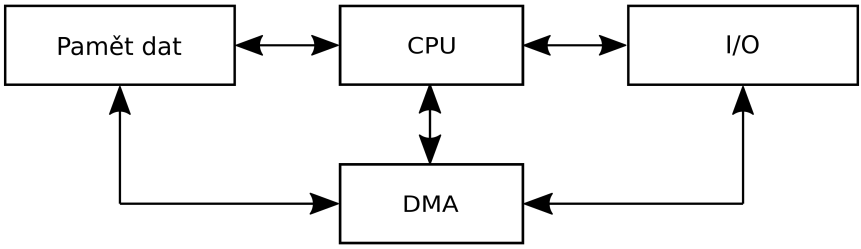
12.2 Sériové programování

Kapitola 13

Obvody zrychlující činnost procesoru

13.1 DMA

DMA - direct memory access je speciální obvod zabudovaný buď přímo v procesoru a nebo realizovaný jako externí integoravý obvod, který umožňuje urychlit přenos dat mezi vnitřní datovou pamětí procesoru a periferiemi. Díky DMA je zároveň zvýšen výkon procesoru, který se nemusí zabývat přesouváním většího objemu dat a může dál vykonávat svůj program.



Procesor je tak omezen pouze obsazením systémové sběrnice při potřebě čtení nebo zápisu dat z datové paměti. To je řešeno samostatnými sběrnicemi mezi pamětí a obvodem DMA. ú

13.1.1 Popis činnosti DMA

DMA je pomocí několika instrukcí naprogramován tak, aby překopíroval popřípadě vymazal ze zdroje data z určité počáteční adresy až do určitého offsetu na jinou adresu v paměti. Přenos může probíhat v režimu paměť-paměť, nebo paměť-výstup a nebo vstup-paměť. Pokud jsou I/O registry mapovány do paměťového rozsahu adres je nezáleží na režimu přenosu - jednotný přístup. V případě, že I/O registry nejsou přístupné pomocí paměťových adres, je nutné vést dodatečný výběrový signál coby adresu zdroje popřípadě cíle, který aktivuje čtení popřípadě zápis do daného registru.

Do DMA je zadána počáteční adresa v paměti dat a offset kam až

se kopírovaná data nacházejí. Následně se zadá počáteční cílová adresa kam se mají data v paměti nakopírovat. Činnost DMA lze dále ovlivnit pomocí stavového a řídícího registru, který umožňuje definovat zda je offset relevantní pro zdrojovou adresu (paměť-výstup) nebo pro cílovou adresu (vstup-paměť) a nebo pro oba (paměť-paměť).

Poté co je DMA procesorem naprogramán se jeho činnost skládá z několika kroků:

- Výpočet a nastavení zdrojové adresy
- Přečtení dat a uložení do vnitřního pracovního registru
- Výpočet a nastavení cílové adresy
- Nakopírování dat z vnitřního pracovního registru do cílové adresy

13.1.2 Popis DMA

DMA je samostatný komplexní obvod, který disponuje vlastní komunikační sběrnicí, pracovními a řídícími registry, a adresovým dekoderem. Jedná se o programovatelný obvod, který není zaměřený na matematické a logické výpočety jako samostatný procesor, ale je zaměřený na přesuny dat.