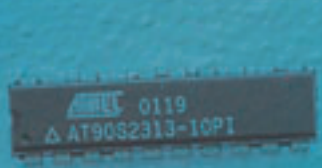
Mikrokontroléry

ATMEL AVR

Vladimír Váňa

popis procesoru a instrukční soubor



TECHNICKÁ LITERATURA
BEN

Mikrokontroléry ATMEL AVR se v poslední době staly spolu s mikrokontroléry PIC firmy MICROCHIP a jednočipovými mikropočítači s jádrem x51 nejrozšířenějšími „jednočipáky“ v řadě amatérských i profesionálních konstrukcí u nás. Z této trojice nejpoužívanějších řad jsou nejmladší a tak dosud neměl český uživatel, programátor či konstruktér možnost seznámit se (s výjimkou informativních časopiseckých článků) podrobně s jejich architekturou a kompletním popisem instrukčního souboru v češtině.

Proto si publikace klade za cíl seznámit českého čtenáře především s architekturou AVR a se souborem instrukcí. V části popisující architekturu seznamuje s jádrem AVR MCU, s organizací paměťového systému a adresovacími módy, s I/O prostorem, popisem periferií integrovaných v AVR MCU jako jsou porty, čítače/časovače, SPI, UART, A/D a popisem jejich registrů, což jsou údaje potřebné při programování i pro návrh praktických zapojení. Tato publikace má tedy spíše poskytovat teoretický základ, potřebný při čtení dalších publikací, týkajících se již praktických konstrukcí či programování.

Protože je v současné době i nedostatek publikací o AVR zaměřených spíše prakticky, jsou v této úvodní publikaci o AVR zařazeny i ukázky praktických konstrukcí – ISP programátoru s AT90S1200 a jednoduchého vývojového kitu s AT90S8515, včetně obrazce jednostranných plošných spojů a dále i stručnou informaci o potřebném vývojovém software, včetně odkazů na Internetu, kde lze tento software zdarma získat.

Vladimír Váňa

Mikrokontroléry ATMEL AVR – popis procesoru a instrukční soubor

Vladimír Váňa: Mikrokontroléry ATMEL AVR – popis procesoru a instrukční soubor
BEN – technická literatura, Praha 2003
1. vydání

ISBN 978-80-7300-083-0 (tištěná kniha)
ISBN 978-80-7300-380-7 (elektronická kniha v PDF)

426410

OBSAH

SEZNAM INSTRUKCÍ

(řazen dle očíslovaných stran v knize)

SLOVO AUTORA

V sedmdesátých letech minulého století, kdy spatřil světlo světa první mikroprocesor a kdy technologie výroby složitých integrovaných obvodů byla v plenkách, byli návrháři touto technologií omezeni a také jim scházely zkušenosti z použití těchto nových prvků. Spolu se zlepšující se technologií a zkušenostmi z nasazení prvních mikroprocesorů se návrhářům dařilo navrhovat stále dokonalejší obvody.

Koncem sedmdesátých let se pak vývoj těchto obvodů rozdělil do dvou směrů.

První směr vedl k stále se zvyšujícímu výkonu mikroprocesorů, větších kapacit i rychlostí pamětí, lepších obvodů I/O apod. Objevovaly se 16, 32 a 64bitové mikroprocesory s architekturou RISC i CISC. Příkladem může být intelovská řada používaná v osobních počítačích PC či pracovních stanicích počínajíc typem 8086, přes obvody 80286, 80386 a 80486 po procesory Pentium, …, Pentium 4. Současně postupoval i vývoj software využívaný těmito mikroprocesory a to jak operační systémy (DOS, UNIX, LINUX, Windows apod.), tak i nástroje pro tvorbu programů počínajíc assemblerem (v těchto počítačích dnes již téměř nepoužívaným), přes vyšší programovací jazyky jako Pascal či C umožňující psát strukturované programy, C++, Java, C# atd. pro objektově orientované programování, rovněž se zlepšovalo prostředí těchto vývojářských nástrojů, což vedlo k nástupu nástrojů RAD (Visual Basic, Delphi, C++ Builder, JBulder atd.). Objevily se i nástroje CASE. Velké množství zpracovávaných dat vedlo ke vzniku SQL a jeho používání v databázích jako např. MS SQL server či Oracle atd., nástup sítí pak k rozvoji distribuovaných aplikací, koncepci klient/server a později i vícevrstvové architektuře. Dalším fenoménem byl rozvoj Internetu a intranetů, koncepce .NET či .COM, e-bussiness atd.

Druhý směr vedl k integraci celého počítače na jeden čip. Na tomto čipu je integrován jak vlastní mikroprocesor, tak paměti i obvody I/O. Většinou je přitom oddělena paměť programu od paměti pro data (Harwardská architektura), na rozdíl od společné paměti pro program i dat u počítačů prvního směru (von Neumanovská architektura). Tyto počítače, umístěné na jednom čipu, je někdy možné rozšířit např. o další vnější paměť atd. Je to umožněno tím, že jejich vnitřní sběrnice je vyvedena na vývody integrovaného obvodu. Těmto prvkům pak obvykle říkáme *jednočipové mikropočítače.* V řadě případů však vystačíme s jedním obvodem, není potřeba rozšiřovat jeho paměť atd. Vlastnosti tohoto počítače k vytvoření jednoduché aplikace bohatě stačí, přitom je potřeba, aby cena tohoto obvodu byla co nejmenší. Toho lze docílit mj. i tím, že se co nejvíce sníží počet vývodů obvodu a není vyvedena sběrnice. Potom obyčejně mluvíme o *jednočipových řadičích (či mikrokontrolérech MCU)*. Hranice mezi jednočipovými řadiči a jednočipovými mikropočítači, jak jsme si ji definovali, není ostrá a jednotlivé produkty ji překrývají. Jednočipové mikropočítače, schopné vytvářet vnější sběrnici, se mohou často omezit na vnitřní paměť programu i dat a mohou tak být použity (při vyšší ceně prvku i spoje) ve funkci jednočipového

řadiče. Někteří výrobci dokonce dodávají levné varianty jednočipových mikropočítačů v pouzdrech s malým počtem vývodů, u kterých je vnější rozšíření vyloučeno. Jako příklad si můžeme uvést jednočipový „řadič" Philips 83C752 (87C572) v pouzdře DIL28 (PLCC28), který je modifikací jednočipového mikropočítače Philips 83C552. Na druhou stranu, řady jednočipových řadičů jsou rozšiřovány o prvky dovolující vytvořit vnější sběrnici. Takové modifikace podstatně zjednodušují vývoj programového vybavení, které se jinak musí opírat pouze o programové simulátory a opakované programování vnitřní paměti EPROM či EEPROM, případně o drahé emulační čipy. Jako příklad rozšíření řady jednočipových řadičů směrem k jednočipovým mikropočítačům si můžeme uvést prvek Microchip 16C71, který rozšiřuje řadu řadičů Microchip 16C5x. Připojení vnější paměti umožňuje i řada AVR mikrokontrolérů ATMEL, jimiž se budeme dále zabývat v této knize.

Doménou jednočipových řadičů jsou hromadně vyráběné řídicí obvody pro domácí spotřebiče, domácí audio-vizuální techniku, zabezpečovací zařízení, telefonní přístroje. Typickými aplikacemi je i obsluha vstupních zařízení počítačů (klávesnice, myši), modemy, řízení zobrazovacích panelů v automobilech, zpracování signálů v inteligentních senzorech a jednoúčelové řízení motorů pro průmyslovou automatizaci. Vzhledem k nízké ceně jsou i vhodnou alternativou k logickým obvodům tam, kde není vyžadována vysoká rychlost (vhodnou alternativou k logickým obvodům v případech, kdy požadujeme vysokou rychlost a současně i možnost jejich naprogramování/přeprogramování jsou např. obvody FPGA; ostatně firma ATMEL má v rodině AVR, o které pojednává tato kniha, i mikroprocesor s programovatelným polem FPGA. Tato kombinace, umožňuje jednak uživateli naprosto volnou definici periferií dle potřeby konstruktéra, jednak redukuje množství různých variant, které musí výrobce vyrábět, aby uspokojil různorodé požadavky zákazníků). Zatímco ještě před pár lety byly jednočipové řadiče typicky 4bitové a jejich aplikační použití se omezovalo na logické řízení, dnes se setkáváme převážně s mikrokontroléry 8bitovými. Pro náročnější aplikace (řízení automobilových motorů, ABS systémy) lze počítat s příchodem jednočipových řadičů 16bitových.

Běžné jednočipové mikropočítače jsou 8bitové, jejich výkon je však nedostatečný pro řadu zajímavých aplikací. Typickým příkladem je řízení vstřikování paliva a řízení zapalování spalovacích motorů nebo ochrana brzdového systému proti zablokování (systémy ABS). V těchto aplikacích je nepostačující i schopnost časovačů měřit časové parametry vstupních signálů a generovat výstupní signály s požadovanou přesností. Požadavky, kladené na tyto aplikace vedly k rozvoji 16bitových jednočipových mikropočítačů, jejichž typickými představiteli jsou mikropočítače řady Intel 8096/80C196.

Šestnáctibitové jednočipové mikropočítače pokrývají současné potřeby řízení v reálném čase. Oblast aplikací, na které již nestačí jednak výpočetní kapacitou, jednak rozsahem použitelné paměti, je zpracování zvukového a obrazového signálu (případně signálů ultrazvukových a rentgenových snímačů ve zdravotnictví či průmyslu, radarových systémů v dopravě a vojenství). Řada těchto aplikací je doménou signálových procesorů vybavenou výkonnou aritmetickou jednotkou. Jiné vyžadují

426410

provést množství operací nad rozsáhlými daty a to je typická doména 32bitových mikroprocesorů a mikropočítačů. Omezujícím faktorem při výběru současných 32bitových mikroprocesorů pro vestavěné (Embedded) aplikace je nutnost vytvoření velmi rychlé a dosti komplikované sběrnice a paměťového subsystému. K tomu, abychom mohli mluvit o skutečných jednočipových mikropočítačích také chybí efektivní obvody rozhraní, které jsme mohli poznat u 8bitových a 16bitových mikropočítačů (čítače a časovače, moderní převodníky A/D).

Pokud jde o 8bitové jednočipové mikropočítače a mikrokontroléry patří u nás mezi nejpoužívanější dvě řady. Jednou z nich je mikropočítač 8051, který pochází z roku 1980 a je vývojově relativně starým. U návrhářů však dosáhl takové obliby, že i v současné době se řada výrobců, včetně firmy Atmel, orientuje na výrobu procesorů s jádrem 8051, které je rozšířené o další periferie. Je to procesor s architekturou jádra CISC.

Druhou řadou jsou mikrokontroléry PIC firmy Microchip s architekturou jádra mající silné prvky architektury RISC. Jsou vhodné hlavně pro jednoduché aplikace. Kromě nízké ceny jsou výhodné i tím, že firma Microchip k nim poskytuje zdarma vývojový software a to jak assembler, tak i jazyk C. Kromě řady x51 a PIC se u nás občas ještě používá např. Zilog Z8 či Motorola 68HC11, popř. další. O jejich hw i vývoji jejich sw vyšlo v nakladatelství BEN – technická literatura dosti publikací, v časopisech jako Amatérské rádio, Praktická elektronika či Rádio plus KTE byla publikována řada konstrukcí zejména s procesory PIC či x51. V poslední době se i u nás v maloobchodním prodeji a v konstrukcích objevují MCU AVR firmy ATMEL. Protože zatím v češtině nejsou AVR příliš popsány, napsal jsem tento text pro potřebu výuky předmětu „Elektronické počítače“ na střední průmyslové škole elektrotechnické s touto specializací. MCU AVR firmy Atmel jsou totiž mj. i dalším, moderním prvkem pro vytváření tzv. embedded aplikací. Je předpovídán velký rozvoj těchto aplikací, zejména v souvislosti s mobilními komunikacemi, propojení s Internetem (jistě jste četli např. úvahy o tom, jak kdejaká lednička bude mít vlastní IP adresu, bude automaticky doobjednávat svůj obsah atd., či o tom, že prvním takto realizovaným zařízením je nějaký topinkovač). Svou příležitost zde nalézají i velké firmy, tvořící původně jen software pro počítače PC a větší. Je to mj. i Microsoft se svými eMbedded Visual Tools či embedded doplňkem k VisualStudio.NET, nebo SUN se svou Javou J2ME.

Mikroprocesory rodiny AVR

Když firma Atmel viděla, jaký úspěch na trhu zaznamenal Intel se svojí rodinou 80C5x, uvedla kolem roku 1993 svoji inovaci tohoto oblíbeného mikroprocesoru. Tou inovací bylo použití paměti flash jako programové paměti, namísto do té doby používané paměti EPROM. Byl to velmi dobrý tah, neboť do té doby, aby bylo možné do mikroprocesoru nahrát (přeprogramovat) nové programové vybavení, musel být zapouzdřen do velmi drahého keramického pouzdra s okénkem. Po odladění

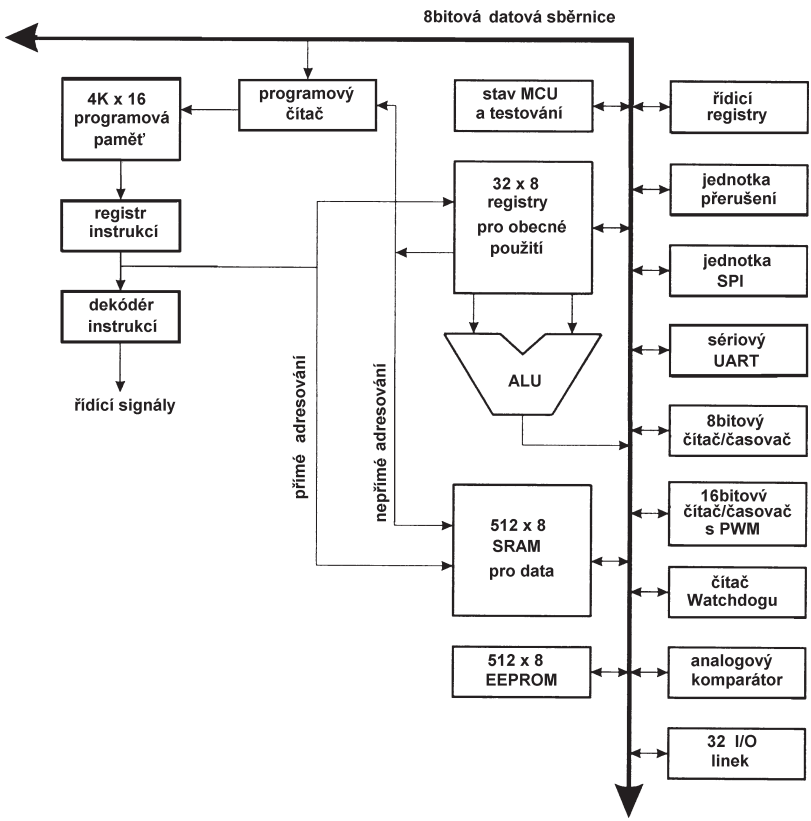
programového vybavení bylo pak pro větší série možné použít ten samý procesor v laciném plastovém pouzdru, který však neměl možnost přeprogramování, neboť pouzdro nemohlo mít okénko nutné pro vymazání paměti EPROM pomocí UV záření. Díky velmi dobře zvládnuté technologii flash slavila firma Atmel velké úspěchy s tímto mikroprocesorem. To vedlo firmu k uvedení vlastních odvozených typů. Na jedné straně některé aplikace nepotřebují tolik vstupů/výstupů jako má mikroprocesor v pouzdře DIL40, takže se objevil např. v pouzdře DIL20 typ AT89C2051, na druhé straně nárůst složitosti aplikací spolu s častějším používáním vyšších programovacích jazyků donutily výrobce implementovat do mikroprocesoru stále větší paměť programu a v menší míře i větší program dat. Složitější aplikace taktéž vyžadovaly nové a nové periferie. Proto v této řadě mikroprocesorů s jádrem 8051 najdeme obvody s integrovaným obvodem Watchdog, analogovým komparátorem, pamětí EEPROM pro konfigurační či kalibrační data, pamětí programu až 32 kB, rozhraní SPI, dvojitým data pointerem. I když tyto inovace spolu se zlepšující se technologií přinesly zvýšení hodinového kmitočtu procesoru až na 33 MHz a tím zvýšení výpočetního výkonu až na 2,75 Mips, přece jen potřebám některých aplikací přestaly tyto procesory stačit. Proto se v norském vývojovém centru Nordic VLSI v Trondheimu začátkem devadesátých let minulého století skupina návrhářů hw spolu s programátory rozhodla navrhnout novou strukturu mikrokontroléru tak, aby struktura tohoto mikrokontroléru vyhovovala překladačům vyšších programovacích jazyků, zejména široce používaného jazyka C. Výsledkem práce této skupiny bylo optimalizované jádro nové řady mikroprocesorů s Harwardskou architekturou nesoucí hlavní charakteristiky mikroprocesorů s redukovanou instrukční sadou (RISC – Reduced Instruction Set Controlers). Před několika lety koupila firma Atmel od norských vývojářů tuto koncepci, na které je založena rodina AVR, kterou také uvedla v druhé polovině devadesátých let na trh.

Přes zdánlivou podobnost s jádrem mikroprocesoru 8051 najdeme zde podstatné odchylky. Především je to šíře instrukčního slova zvětšená na 16 bitů. Zvětšení šířky instrukčního slova na jednu stranu zvětšilo požadavky na velikost paměti, na druhou stranu však umožnilo zrychlit načtení mnoha instrukcí, neboť kromě několika výjimek vystačí instrukce s jedním slovem, tj. mikroprocesor je dokáže načíst během jednoho hodinového cyklu. Druhým velkým rozdílem je propojení ALU s polem 32 pracovních registrů. To umožnilo snížit počet hodinových taktů potřebných na provedení téměř všech instrukcí na pouhé dva takty, načtení + dekódování a vykonání. To, že instrukce vystačí s jedním slovem, umožnilo implementaci jednoduchého překrývání instrukcí (pipelining). Počet hodinových taktů potřebných na vykonání instrukce typu registr-registr se tímto snížil na pouhý 1 hodinový takt, tj. na 1 MHz hodinového kmitočtu připadá výkon 1 Mips. Srovnáme-li toto se základním instrukčním cyklem řady 8051, vidíme, že jádro nové řady mikroprocesorů AVR dokáže poskytnout až 12× větší výpočetní výkon při shodném hodinovém kmitočtu.

1

ARCHITEKTURA MIKROKONTROLÉRŮ AVR

Architekturu typického mikrokontroléru AVR ukazuje *obr. 1.1* (konkrétně jde o typ AT90S8515).



Obr. 1.1 *Příklad architektury MCU AVR (AT90S8515)*

426410

Jádro řady AVR se podobá jádru většiny RISC-procesorů, které jsou dostupné na trhu. Jádro AVR se skládá ze 32 stejných 8bitových registrů, které mohou obsahovat jak data, tak adresy. K přístupu k těmto univerzálním registrům stačí jeden hodinový cyklus. Vzhledem k propojení těchto registrů s ALU (Arithmetic Logic Unit) to znamená, že během jednoho hodinového cyklu se provede jedna ALU operace. Přitom vstupem jsou dva operandy uložené v souboru registrů, výstup operace je pak uložen nazpět do registru. Posledních 6 registrů můžeme ve dvojici použít jako ukazatele adresy pro nepřímé adresování paměti dat. Tyto registry označované písmeny X, Y a Z dovolují libovolné ukládací operace (load/store). Programátor má například na výběr, zda se ukazatel adresy bude po zpracování určité instrukce inkrementovat nebo se před zpracováním této instrukce bude dekrementovat. Užitečné je pro adresování využít možnosti 6bitového posunu v ukazateli adresy v dvojitých registrech Y a Z. ALU umožňuje aritmetické a logické operace mezi registry, nebo mezi registrem a konstantou. Také umožňuje operace s jediným registrem. K registrovým operacím se mohou přidat i operace aplikující obvyklé paměťové adresní módy na soubor registrů. Je to umožněno tím, že soubor registrů zabírá dolních 32 adres datového prostoru ($00 až $1F), což dovoluje přístup k registrům jako by to byly běžné paměťové buňky.

Paměťový prostor dále obsahuje 64 adres I/O registrů sloužící k periferním funkcím jako jsou řídicí registry, čítače/časovače, A/D převodníky a další I/O funkce. Rovněž k této části adresového prostoru je možný přístup, tentokrát v rozsahu adres $20 až $5F.

Mikrokontroléry AVR využívají koncepci Harwardské architektury – oddělené paměti pro program a pro data. Jak již bylo zmíněno, program umístěný v programové paměti je prováděn s jednoduchým překrýváním instrukcí (pipeline). Zatímco jedna instrukce je prováděna, druhá je přesouvána z programové paměti. Programová paměť je „In-System Programmable Flash memory“. Znamená to, že kromě klasického, paralelního naprogramování této paměti, je možné i sériové naprogramování přímo v systému. Při paralelním programování, které se používá u většiny jednočipových mikropočítačů či mikrokontrolérů se využívá toho, že příslušný obvod je navržen tak, aby po připojení programovacího napětí na určitý vývod obvodu se provedlo přepnutí vývodů, které jsou v normálním režimu vývody I/O portů, tak, že nyní jsou tyto vývody obvodu připojeny k adresovým a datovým vývodům vnitřní programové paměti, takže lze do této paměti paralelně zaznamenat data, což je vlastně příslušný program. Po naplnění této paměti snížení napětí na vývodu umožňujícím přepínaní do/z programovacího módu a uvedení procesoru do počátečního stavu (resetování) pak mikropočítač či mikrokontrolér pracuje podle právě naprogramovaného programu. Je zřejmé, že při tomto způsobu programování je třeba, aby při programování byl k I/O vývodům připojen programátor a naopak aby byly odpojeny od jakýchkoli jiných obvodů, např. periferií. Proto je při každém programování potřeba obvod vyjmout z objímky, popř. vyletovat z plošného spoje a vložit do programátoru. Tato nevýhoda odpadá při sériovém programování, kdy mikrokontrolér zůstává v aplikaci a pomocí několika signálů (u většiny AVR MCU jsou to signály MOSI, MISO, SCK a RESET) připojených k programátoru se dá jednoduše naprogramovat.

426410

Při provádění relativních skoků či instrukcí volání je přímo přístupný adresový prostor. Většina AVR instrukcí má formát jednoho 16bitového slova. Každá adresa programové paměti obsahuje 16 nebo 32bitovou instrukci.

Při provádění obsluhy přerušení a volání podprogramu se návratová adresa programového čítače (PC tj. Program counter) ukládá do zásobníku. Zásobník je umístěn v datové paměti SRAM a tudíž je omezen jenom velikostí paměti SRAM a jejím volným místem. Všechny uživatelské programy musí inicializovat SP v inicializační (reset) části programu, před prováděním podprogramů nebo obsluhy přerušení. Šestnáctibitový ukazatel zásobníku je přístupný pro čtení i zápis v I/O prostoru.

Architektura AVR má pět adresovacích módů pro paměť dat:

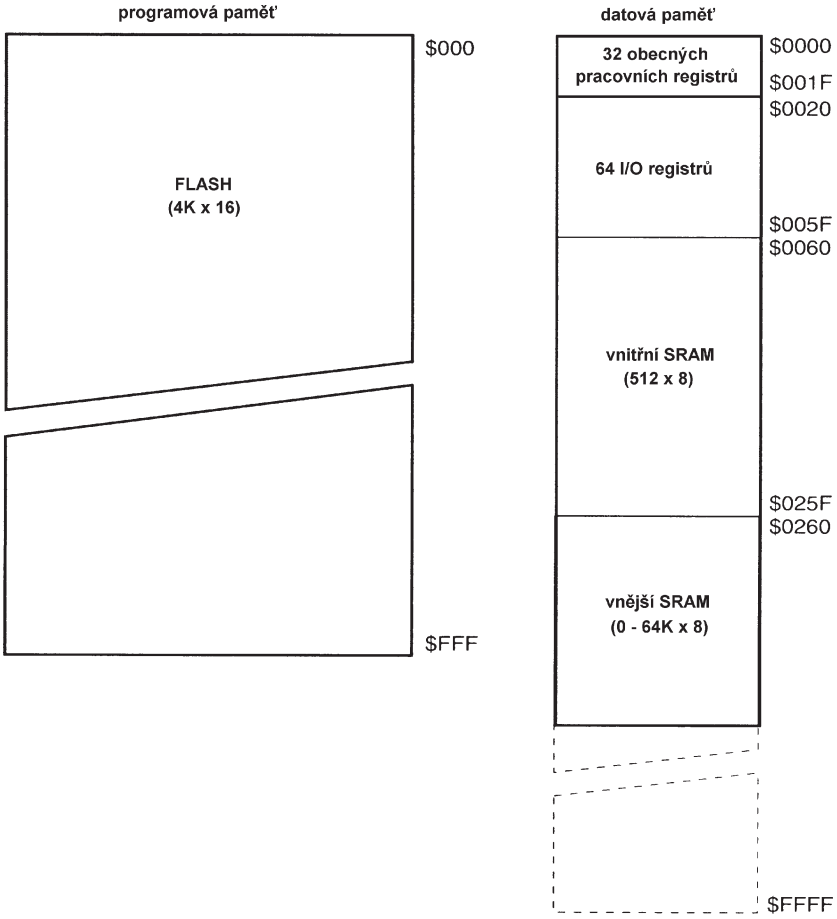
- přímé adresování,
- nepřímé adresování s posunutím (6bitový posun),
- nepřímé adresování,
- nepřímé adresování s dekrementací ukazatele adresy před zpracováním instrukce,
- nepřímé adresování s inkrementací ukazatele instrukce po zpracování instrukce.

Systém přerušení má vlastní řídicí registry umístěné v I/O prostoru a navíc bit ve stavovém registru pro zákaz/povolení všech přerušení. Všechna různá přerušení mají oddělený vektor přerušení v tabulce vektorů přerušení umístěné na začátku programové paměti. Priorita těchto přerušení je dána umístěním jejich vektorů v tabulce přerušení. Čím nižší má vektor přerušení adresu, tím větší má prioritu.

Příklad paměťového prostoru (AT90S8515) ukazuje *obr. 1.2.*

Na tomto obrázku vidíme, že prvních 32 adres v datové paměti patří souboru registrů. Třebaže tyto registry nejsou implementovány jako součást SRAM, umožňuje tato organizace paměťového prostoru značnou flexibilitu v přístupu k registrům. Adresy těchto registrů ukazuje *obr. 1.3*.

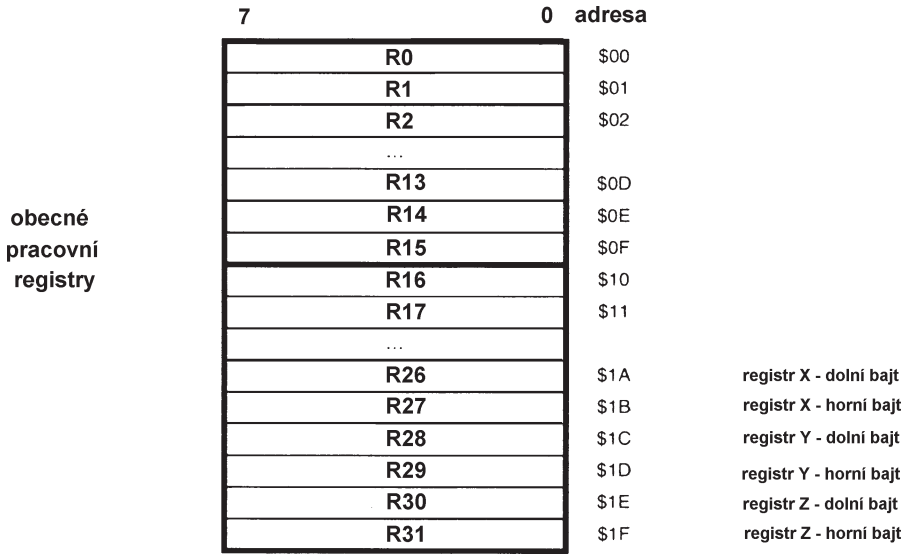
Při pohledu na *obr. 1.2* si můžeme všimnout možnosti připojení externí paměti dat. Týká se to typů počínajíc AT90S8515. Jde totiž o to, že i když má mikroprocesor celkem 256/512 byte vnitřní datové paměti, pro některé aplikace je to málo. Na tyto případy je tento MCU též připraven, neboť umožňuje připojení externí paměti dat až do velikosti 64 kB. Rozhraní pro připojení externí paměti je obdobné standardu známého u řady 89C5x. Rozhraní pro externí paměť dat používá 16bitovou adresovou sběrnici, jejích spodních 8 bitů je multiplexováno a daty (8 bitů). Aby činnost mikroprocesoru byla zpomalována co nejméně, trvá přístup do externí paměti pouze tři takty systémového kmitočtu. To klade vyšší požadavky jednak na rychlost paměti, jednak na rychlost externího registru spodních 8 bitů adresy, který musí být schopen zachytit adresu za pouhých 22 ns (při řídicím kmitočtu 8 MHz). Mikroprocesor má možnost přidání jednoho taktu systémového kmitočtu, tj. přístup k paměti pak trvá celkem čtyři takty. Sníží se tím nároky na přístupové doby paměti, ale ne samotného registru adres. Též se tímto řešením sníží výkon mikroprocesoru, protože instrukce



Obr. 1.2 *Příklad datového prostoru MCU AVR (AT90S8515)*

pracující s externí pamětí budou pomalejší (instrukce LD, ST, LDS, STS, PUSH a POP). Tyto instrukce v případě přístupu k externí paměti budou o jeden takt pomalejší, popř. o dva takty pokud přístup do paměti je s jedním cyklem navíc. Taktéž instrukce volání podprogramu budou pomalejší a to o dva, případně čtyři takty, neboť je nutno uschovat 2 byte adresy do registru PC.

Pokud jde o paměť programu typu flash, je organizována jako N × 16, kde N je závislé na typu mikrokontroléru. Např. pro AT90S8515 je N = 4K, takže tento typ vystačí s programovým čítačem PC 12 bitů širokým, kdežto třeba pro typ ATmega128 je PC 16bitový. Pro programovou paměť flash zaručuje firma Atmel alespoň 1000 programovacích cyklů. Již jsme se zmínili o programování této paměti paralel-



Obr. 1.3 *Obecné pracovní registry MCU AVR*

ně v přípravku nebo sériově jako ISP. Novinkou je i možnost programování této paměti přímo mikrokontrolérem, jehož je součástí. Tuto možnost však využívají až MCU řady ATmega, využívající rezidentní Boot Leader Program.

426410

2

AVR – ŘADY MIKROKONTROLÉRŮ

Atmel vyrábí mikrokontroléry AVR ve třech řadách:

Základní

Obsahuje typy AT90S1200, AT90S2313, AT90S2323, AT90S2343, AT90S4433, AT90S4434, AT90S8515, AT90S8534 a AT90S8535. Tyto typy (s výjimkou AT90S1200) mají 118 instrukcí a jsou u nás dostupné v maloobchodní siti. Typ AT90S1200 má 89 instrukcí a oproti dalším typům nemá paměť SRAM. Proto se k napsání sw pro tento typ nedají použít některé překladače vyšších jazyků. Na druhé straně výhodou tohoto typu je jeho nízká cena.

ATtiny

Obsahuje typy ATtiny11, ATtiny12, ATtiny15, ATtiny28 a ATtiny26. S výjimkou posledního typu ATtiny26, který má 118 instrukcí, mají ostatní 90 instrukcí. Rovněž typy této rodiny jsou u nás v maloobchodě dostupné. Jsou to nejlacinější typy mikrokontrolérů AVR.

ATmega

Obsahuje jednak starší řadu ATmega103, ATmega161, ATmega163 a ATmega323 a novější řadu ATmega8, ATmega16, ATmega64 a ATmega128. Tyto mikrokontroléry mají (s výjimkou ATmega103) 130 instrukcí. Oproti základní řadě obsahuje navíc instrukce pro násobení. Novější řada je navíc vybavena rozhraním JTAG pro ladění sw přímo v aplikaci. Většina novinek v oblasti AVR se týká právě řady ATmega a to jak nové typy mikrokontrolérů, tak novinky v oblasti vývojových prostředků.

Při získávání prvních praktických zkušeností s ATMEL AVR MCU pravděpodobně použijeme to, co je v současné době k dispozici v maloobchodních prodejnách, tedy MCU základní řady. Jejich stručný popis je obsahem *tab. 2.1*.

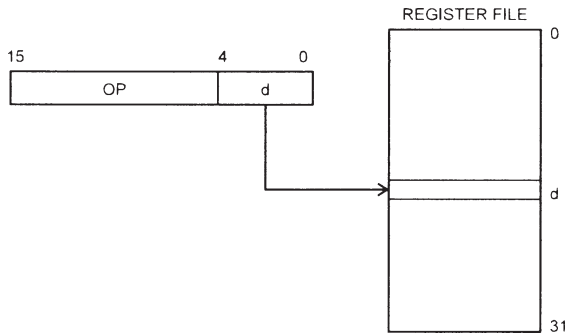
Tab. 2.1 *Vlastnosti MCU základní řady*

	1200	**2313**	**2343**	**4433**	**8515**	**8535**
Max. kmitočet CPU v [MHz]	4 popř. 12	4 popř. 10	1; 4; 10	4 popř. 8	4 popř. 8	4 popř. 8
Počet instrukcí	89	118	118	118	118	118
Počet obecných registrů	32	32	32	32	32	32
FLASH	1kB	2kB	2kB	4kB	8kB	8kB
SRAM	-	128B	128B	128B	512B	512B
EEPROM	64B	128B	128B	256B	512B	512B
Čítač/časovač0	8 bit	8 bit	8 bit	8 bit	8 bit	8 bit
Čítač/čacovač1	-	16 bit	-	16 bit	16 bit	16 bit
Čítač/časovač2	-	-	-	-	-	8 bit
Watchdog	ano	ano	ano	ano	ano	ano
A/D převodník	-	-	-	10 bit	-	10 bit
Analog. komp.	ano	ano	ano	ano	ano	ano
UART	-	ano	ano	ano	ano	ano
SPI sér. interface	ano	ano	ano	ano	ano	ano
PORT A	-	-	-	-	ano	ano
PORT B	ano	ano	ano	ano	ano	ano
PORT C	-	-	-	ano	ano	ano
PORT D	ano	ano	-	ano	ano	ano

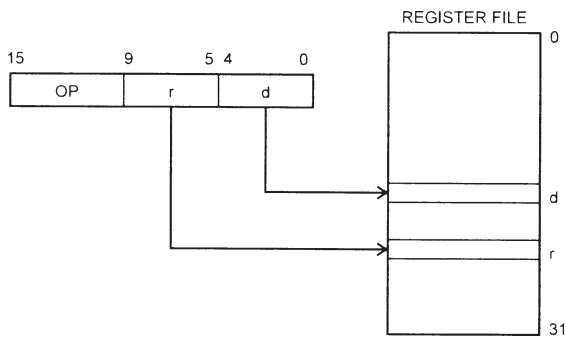
3

ADRESOVACÍ MÓDY

Touto kapitolou se vracíme k popisu architektury AVR MCU a ukážeme si adresovací módy pro přístup k programové paměti (flash) a k datové paměti (SRAM, soubor registrů a I/O paměť). Na obrázcích popisující tyto módy je OP zkratka pro část instrukčního slova obsahující operační kód instrukce.

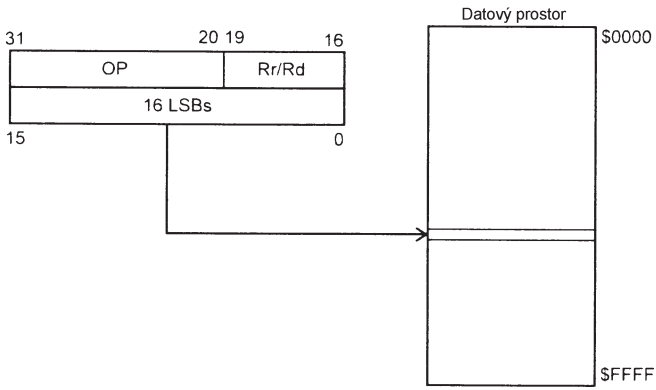


Obr. 3.1 *Přímé adresování jednoho registru*



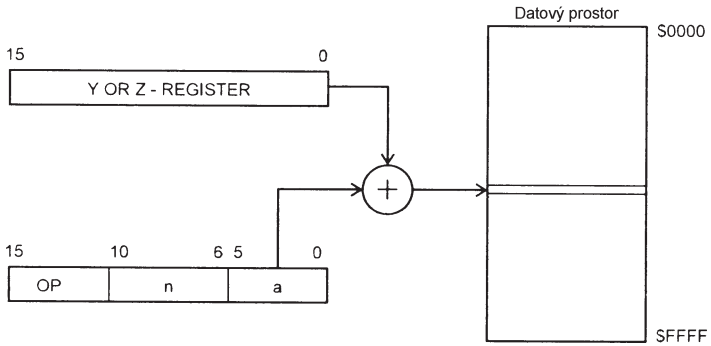
Obr. 3.2 *Přímé adresování, dva registry*

Operandy jsou obsaženy v registru r (Rr) a d (Rd). Výsledek je umístěn do cílového registru d (Rd).



Obr. 3.3 *Přímé adresování dat*

Šestnáctibitová adresa dat je obsažena v dolních 16 bitech (LSB) dvouslovové instrukce. Rd/Rr je cílový nebo zdrojový registr.



Obr. 3.4 *Nepřímé adresování dat s posunem*

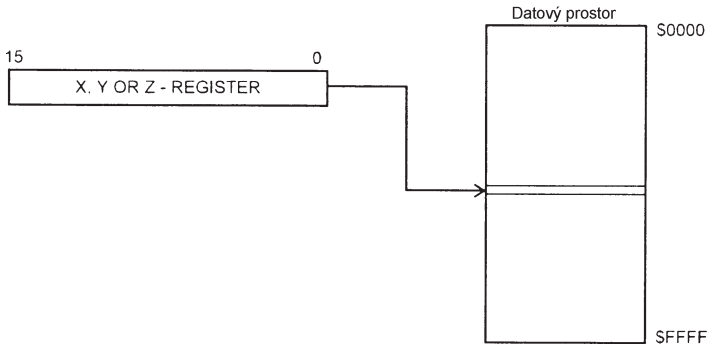
Adresa operandu se získá přičtením obsahu registru Y nebo Z k adrese umístěné v 6 bitech instrukčního slova (*obr. 3.4*).

Adresa operandu je obsahem registru X, Y nebo Z (*obr. 3.5*).

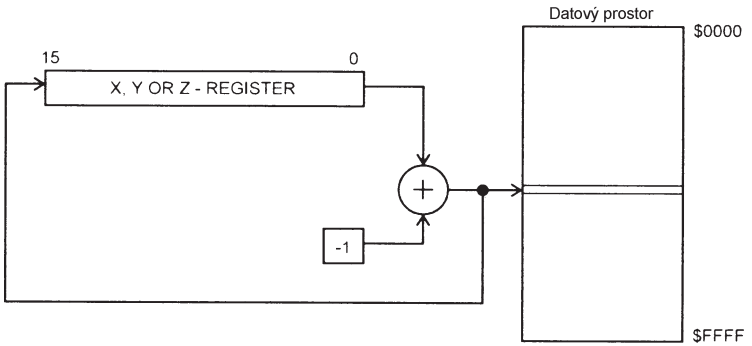
Registr X, Y či Z je dekrementován před provedením operace. Adresa operandu je dekrementovaný obsah registru X, Y nebo Z (*obr. 3.6*).

Registr X, Y nebo Z je inkrementován po provedení operace. Adresa operandu je obsah registru X, Y nebo Z před inkrementací (*obr. 3.7*).

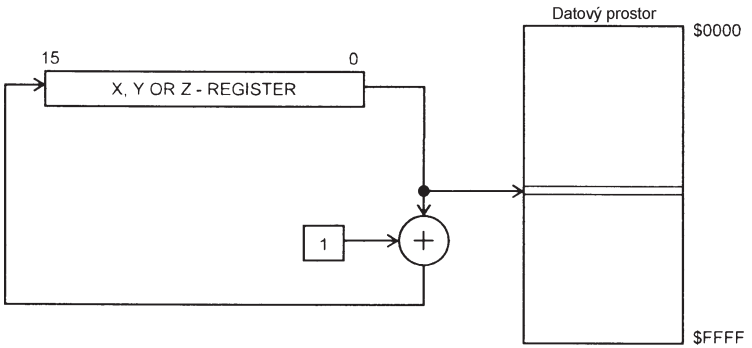
Adresa konstanty je určena obsahem registru Z. Patnáct horních bitů (MSB) vybírá adresu slova v rozsahu 0 až 4K, dolní bit (LSB) vybírá dolní byte (když LSB = 0) nebo horní byte (LSB-1) *obr. 3.8*.



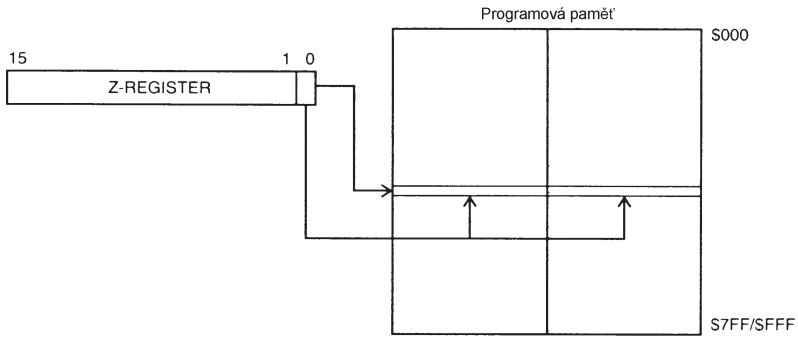
Obr. 3.5 *Nepřímé adresování dat*



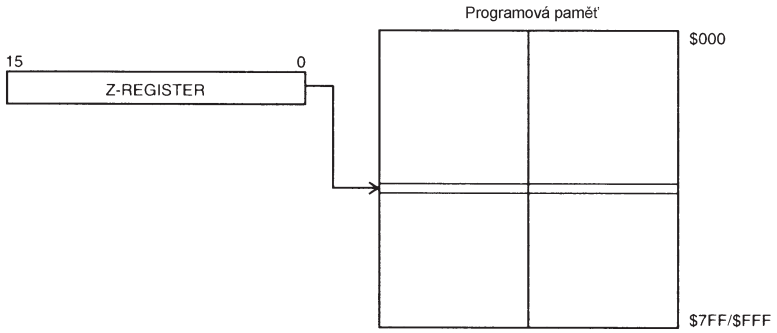
Obr. 3.6 *Nepřímé adresování dat s pre-dekrementací*



Obr. 3.7 *Nepřímé adresování dat s post inkrementací*

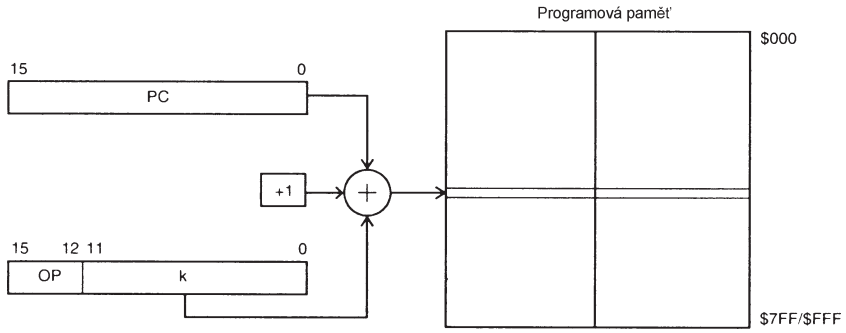


Obr. 3.8 *Adresování konstanty užitím instrukce LPM*



Obr. 3.9 *Nepřímé adresování programové paměti, instrukce IJMP a ICALL*

Provádění programu pokračuje na adrese obsažené v registru Z, tj. programový čítač PC je naplněn obsahem registru Z.



Obr. 3.10 *Relativní adresování v programu, instrukce RJMP a RCALL*

Provádění programu pokračuje na adresa PC+k+1. Relativní adresa *k* je v rozsahu –2048 až 2047.

4

DALŠÍ PRVKY ARCHITEKTURY AVR

Kromě paměťového systému a ALU jsou součástí MCU i další obvody, viz *obr. 1.1*. Nyní si je alespoň stručně popíšeme. Jde především o obvody Watchdog, časovače, analogový komparátor, A/D převodník, UART a A/D převodník. Rovněž si ukážeme, jak vypadá přerušovací systém a obvod resetu. Poté, co se seznámíme se základními částmi architektury MCU AVR si ukážeme několik konkrétních typů ze základní řady.

Hlídací obvod Watchdog

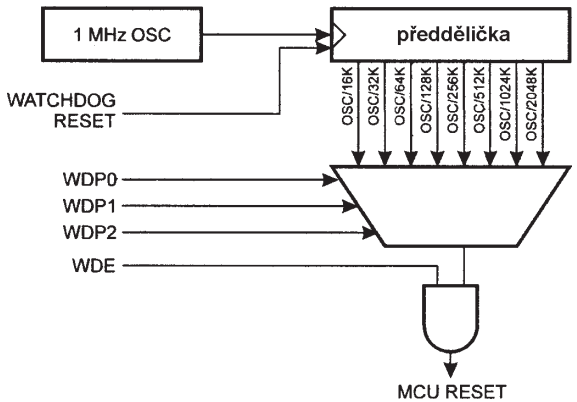
Termínem hlídací (nebo dohlížecí) obvody obvykle označujeme obvody, které dohlížejí na správný běh programu. Hlídací obvod *Watchdog* je časovač, který je odstartován buď bezprostředně po *resetu* mikropočítače nebo někdy později z programu. Jeho časová konstanta může být pevně svázána s hodinami mikropočítače, nebo může být nějakým způsobem nastavena (např. programem). Po vypršení časového limitu časovač vyvolá automatický reset, pokud ho ovšem dříve nějakým podnětem (signálem, hodnotou zapsanou do řídicího registru, speciální instrukcí) nevynulujeme, tj. nevrátíme do počátečního stavu.

Využití hlídacího obvodu Watchdog je založeno na jednoduché myšlence – do programu jsme schopni vložit příkazy nulující časovač hlídacího obvodu Watchdog tak, že při jeho správné funkci nedojde k vypršení časového limitu. Porucha, způsobená např. náhodnou změnou dat v paměti, nebo chyba v programu, například neošetřená určitá kombinace vstupních dat, často vyvolá změnu chování programu. V řídicích systémech lze sice některé chyby tolerovat, když nevedou k problémům s řízením systému, jiné chyby však tolerovat nelze. Je to např. zacyklení programu, chybné zamaskování přerušovacího signálu nebo překonfigurování periferních obvodů. A právě v těchto případech je obvod Watchdog užitečný. Program v nekonečném cyklu či program, který není pravidelně aktivován vnějšími podněty nevynuluje do vypršení časového limitu časovač obvodu Watchdog, a ten program z nežádoucího stavu vyvede resetem. Je to vlastně technická implementace toho,

426410

co např. provádíme u počítače PC, který se v důsledku náhodné chyby technických prostředků, operačního systému či aplikačního programu „zakousne“.

Blokové schéma obvodu Watchdog použitého v MCU AVR ukazuje *obr. 4.1*.



Obr. 4.1 *Blokové schéma hlídacího obvodu Watchdog*

Jedná se o spojení RC oscilátoru s kmitočtem cca 1 MHz a děličky s nastavitelným dělicím poměrem. Díky nastavitelné děličce si můžeme vybrat pro danou aplikaci nejvhodnější dobu cyklu obvodu Watchdog (výběr pomocí WDP0, WDP1 a WDP2). Aby nebyl obvod Watchdog závislý na externím oscilátoru, který díky nějaké chybě může přestat kmitat a tím pozastavit celou funkci hlídacího obvodu, je jako zdroj využit interní oscilátor o kmitočtu 1 MHz. I když není tento RC oscilátor příliš stabilní, neboť je zejména citlivý na změnu napájecího napětí, má konstruktér možnost využít tento oscilátor též jako zdroj systémového taktu pro celý MCU. Programově lze tento obvod ovládat prostřednictvím obsahu řídicího registru WDTCR (Watchdog Timer Control Register).

Watchdog Timer Control Register - WDTCR
řídicí registr hlídacího obvodu

bit	7	6	5	4	3	2	1	0	
$21 ($41)	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
čtení/zápis	R	R	R	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 4.2 *Řídicí registr hlídacího obvodu*

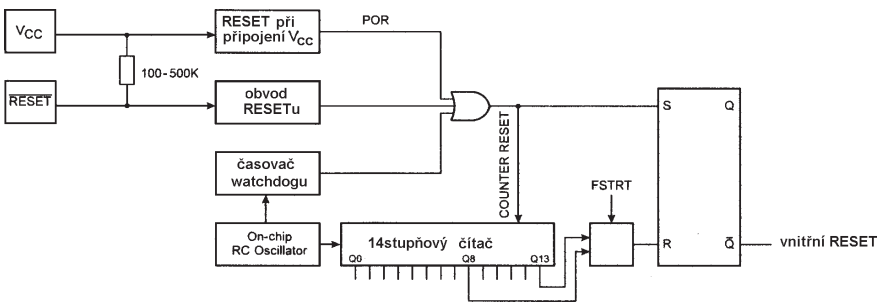
Bity 5, 6 a 7 nejsou použity. Bit 4 je označen **WDTOE** (Watchdog Turn-off Enable) a musí být nastaven na nulu, když má být bit WDE vynulován, v opačném případě nebude povolena činnost obvodu Watchdog. Nastavený Bit 4 vynuluje hardware po čtyřech hodinových cyklech. Bit 3 je označen **WDE** (Watchdog Enable), a je-li nastaven na jedničku, povoluje činnost obvodu Watchdog, při nule ji zakazuje. Bity 0, 1 a 2, označené WDP2, WDP1 a WDP0 slouží k nastavení dělicího poměru děličky a tím nastavení doby cyklu (Time-out) obvodu Watchdog. Vliv jejich nastavení na dobu cyklu Watchdogu ukazuje *tab. 4.1*.

Tab. 4.1 *Vliv bitů WDPO, WDP0, WDP1 a WDP2 na dobu cyklu*

WDP2	WDP1	WDP0	Počet cyklů oscilátoru WDT	Typický Time-out při Vcc = 3 V	Typický Time-out při Vcc = 5 V
0	0	0	16K cyklů	47 ms	15 ms
0	0	1	32K cyklů	94 ms	30 ms
0	1	0	64K cyklů	0,19 s	60 ms
0	1	1	128K cyklů	0,38 s	0,12 s
1	0	0	256K cyklů	0,75 s	0,24 s
1	0	1	512K cyklů	1,5 s	0,48 s
1	1	0	1024K cyklů	3 s	0,97 s
1	1	1	2048K cyklů	6 s	1,9 s

Výše popsaný obvod Watchdog patří do obvodu Reset, jehož celkové blokové schéma můžeme vidět na *obr. 4.3*.

Obvod RESETu



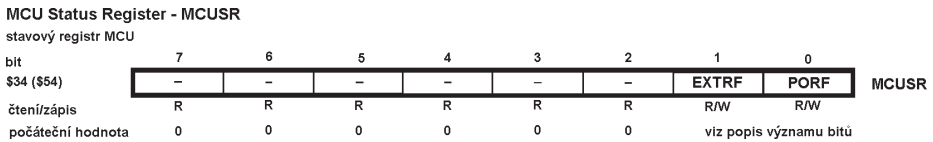
Obr. 4.3 *Schéma obvodu RESETu*

Z tohoto obrázku je vidět, že tento obvod má tři zdroje:

- již zmíněný Watchdog,
- automatický reset po připojení k napájení POR (Power-on Reset),
- externí reset přivedením logické nuly na vývod /RESET MCU na dobu delší než 50 ns.

MCU stavový registr – MCUSR

MCU stavový registr poskytuje informaci o tom, co bylo zdrojem RESETu.



Obr. 4.4 *Stavový registr MCU*

Bit 1 – EXTRF: External Reset Flag

Po automatickém RESETu po připojení k napájení není hodnota tohoto bitu definována. Je nastavena na jedničku externím resetem. Watchdog reset nechává tento bit nezměněn.

Bit 0 – PORF: Power-on Reset Flag

Tento bit je nastaven na jedničku jedině automatickým resetem po připojení k napájení. Watchdog nechává tento bit nezměněn.

Čítače/časovače

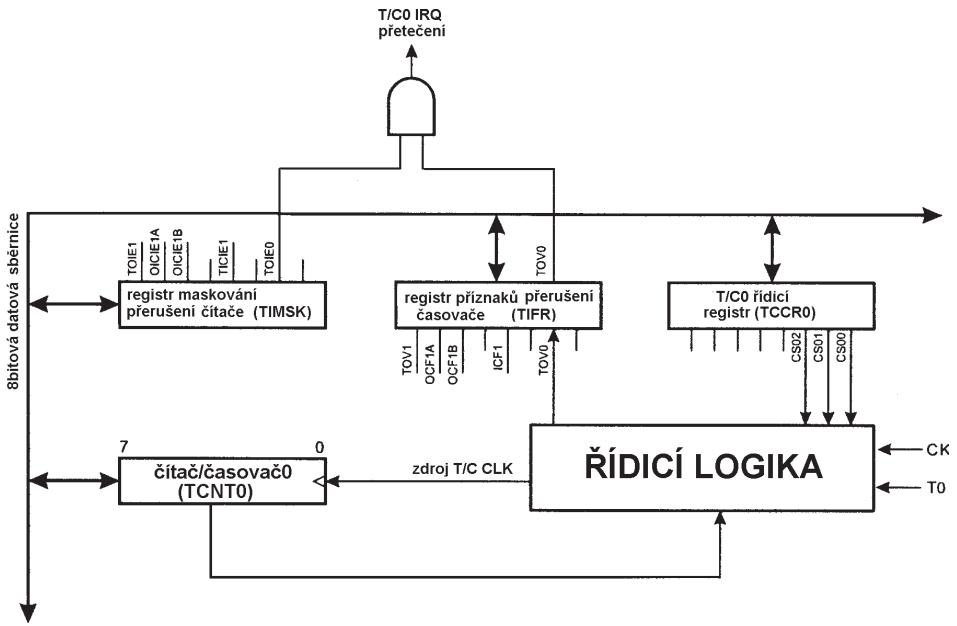
Důležitou funkcí jednočipových mikropočítačů a mikrokontrolérů je čítání vnějších událostí, časových intervalů mezi nimi a/nebo jejich kmitočet. Vnější události jsou na vstupech systému běžně prezentovány úrovněmi nebo změnami logických signálů. Od jednočipového mikropočítače či řadiče obvykle vyžadujeme, aby byl schopen v určitých časových okamžicích generovat řídicí signály pro vnější prostředí jako např. různé spínače. Někdy je i nutná vazba na reálný, astronomický, čas (např. pomocí vazby na DCF77).

Čítání vnějších událostí lze u pomalejších dějů zajistit programem, a to buď průběžným sledováním hodnoty binárního signálu na příslušném vstupu, nebo lépe reakcí na přerušení vyvolané změnou hodnoty tohoto signálu. Přímé měření nebo generování časových intervalů programem, při kterém vycházíme ze známého kmitočtu hodin procesoru a počtu hodinových cyklů, které vyžaduje provedení každé instrukce, je možné jen ve velmi omezené míře. V praxi se setkáme s takovým řešením např. když potřebujeme, s nepříliš velkou přesností, časově omezit čekání v čekací smyčce, kdy potřebujeme definovat časový limit – *time-out.* Ve většině aplikací je měření a generování delších časových intervalů programem nepřijatelné a proto jsou do struktury jednočipových mikropočítačů a řadičů zařazeny podpůrné obvody schopné čítat vnější události a měřit a generovat časové intervaly nezávisle na procesoru.

Proto i AVR MCU mají v sobě zabudované 8bitové a 16bitové čítače/časovače. Vybavení časovači není u všech typů MCU AVR stejné. I v nejjednodušším případě mají alespoň jeden 8bitový čítač označovaný *čítač/časovač 0* (Timer/Counter0) – u typů ATtiny11, ATtiny12, ATtiny28, AT90S1200, AT90S2323 a AT90S2343,

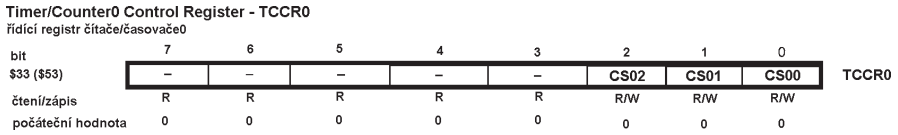
Typ ATtiny15 má dva 8bitové čítače/časovače. Většina zbývajících AVR MCU má jeden 8bitový *čítač/časovač 0* a jeden *čítač/časovač 1* – typy ATtiny26, AT90S2313, AT90S4433, AT90S8515, AT90S8534, popř. navíc ještě další 8bitový *čítač/časovač 2* – u typů AT90S8535, ATmega 16, ATmega103, ATmega161, ATmega163 a ATmega323. Typy ATmega64 a ATmega128 kromě toho ještě další, *čítač/časovač 3*, tedy celkem čtyři čítače/časovače.

Nyní si jednotlivé čítače/časovače u MCU AVR popíšeme. *Obr. 4.5* ukazuje *čítač/časovač 0*.



Obr. 4.5 *Čítač/časovač0*

Jelikož 8bitový čítač/časovač nedovoluje velkou flexibilitu v možnosti volby časových intervalů (v režimu čítání hodinového kmitočtu procesoru, signál CK), je vlastnímu čítači/časovači předřazen předdělič s možností výběru dělicího poměru 1 až 1024 v krocích 1, 8, 64, 256 a 1024. *Čítač/časovač 0* může rovněž čítat vnější impulzy přivedené na vstup T0 (společný s PB0). Je-li čítán vnější signál, musí každý jeho stav trvat nejméně jednu periodu hodinového cyklu oscilátoru a obsah čítače je zvětšen s každou jeho vzestupnou hranou. Řídicím registrem I/O pro tento čítač je TCCR0 (Timer/Counter0 Control Register). Příznak přetečení je umístěn v registru TIFR (Timer/Counter Interrupt Flag Register), povolení/zakázání přerušení z *čítače/časovače 0* je součástí registru TIMSK (Timer/Counter Interrupt Mask Register). Dále si popíšeme význam jednotlivých bitů řídicího registru TCCR0.



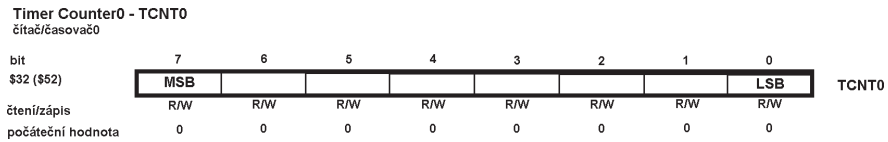
Obr. 4.6 *Řídicí registr čítače/časovače 0*

Bity 3 až 7 jsou nastaveny na 0, bity 0, 1 a 2 určují zdroj signálu či dělicí poměr předděličky a jsou popsány *tab. 4.2*.

Tab. 4.2 *Zdroj signálu či dělicí poměr předděličky*

CS02	CS01	CS00	Význam
0	0	0	Zastavení čítání, *časovač/čítač 0* je zastaven
0	0	1	CK
0	1	0	CK/8
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	Externí vstup T0, sestupná hrana
1	1	1	Externí vstup T0, vzestupná hrana

Vlastní obsah čítače je přístupný jako registr **Timer Counter 0 – TCNT0**:



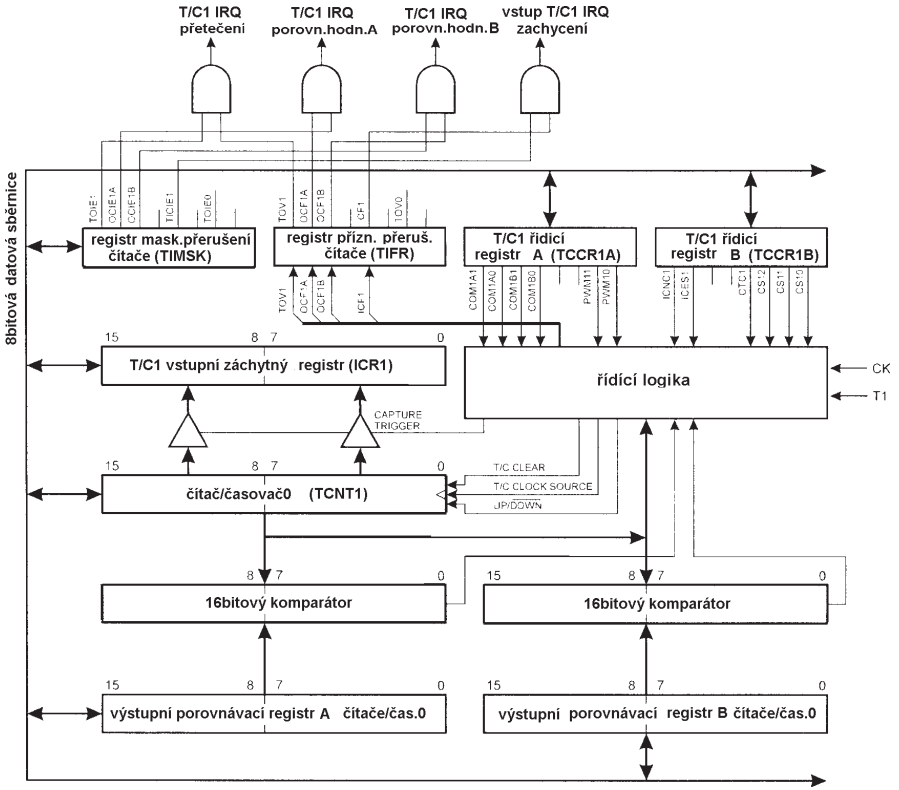
Obr. 4.7 *Registr Timer Counter 0*

Čítač/časovač je realizován jako čítač čítající vpřed s možností čtení i zápisu do TCNT0. Po zápisu do TCNT0, lze pokračovat čítáním v dalším hodinovém cyklu následujícím po operaci zápisu.

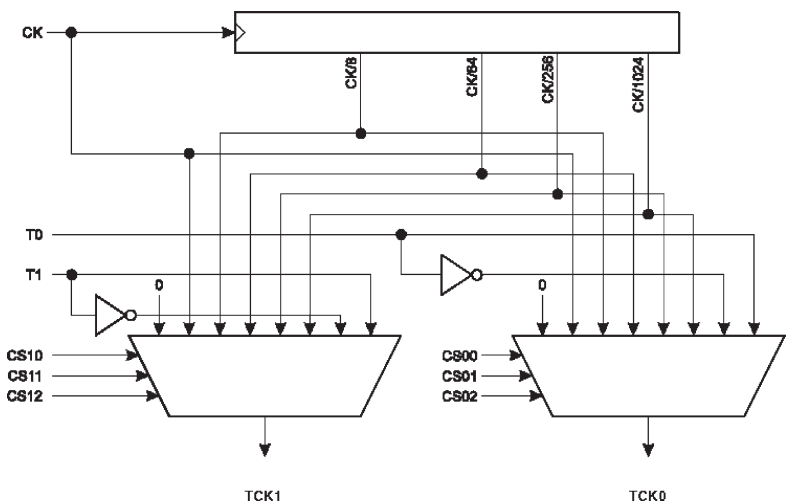
Dále si popíšeme *čítač/časovač 1* (který je součástí jen některých typů – viz výše). Tento čítač je multifunkční a je ho možno využít jednak jako klasický čítač/časovač, jednak jako 16bitovou jednotku compare/capture. Poslední funkcí, kterou je tato jednotka schopna plnit, je generování signálu pomocí pulzně-šířkové modulace PWM. Blokové schéma *čítače/časovače 1* ukazuje *obr. 4.8*.

Tento čítač/časovač není pouhým rozšířením předchozího, 8bitového čítače na 16bitový čítač. Mj. obsahuje navíc i **záchytný registr** a dva **komparační registry.** Stručně si popíšeme jejich funkci.

Pokud potřebujeme určit čas, ve kterém došlo k určité vnější události a máme přitom k dispozici univerzální čítač/časovač (pozn. tj. čítač/časovač se strukturou jako má *čítač/časovač 0*), který nám vytváří časovou základnu, postupujeme obvykle tak, že v obslužné rutině aktivované vnější událostí přečteme okamžitou hodnotu



Obr. 4.8 *Čítač/časovač1*



Obr. 4.9 *Předdělička čítače/časovače*

čítače/časovače a vypočteme příslušný časový údaj. Doba, která uplyne mezi vnější událostí a čtením časovače/čítače je chybou metody. Chybu lze kompenzovat pouze částečně, protože k žádosti o přerušení může dojít v době, kdy je obsluhováno jiné přerušení (s vyšší prioritou), nebo kdy je přerušovací systém zamaskován. Tento problém je odstraněn tím, že volně běžící časovač, jehož přetečení je indikováno procesoru přetečením, je doplněn o *záchytný registr* (Capture Register). Signál informující o vnější události vygeneruje žádost o přerušení a současně přepíše hodnotu čítače do záchytného registru. Aktivovaná přerušovací rutina čte časový údaj ze záchytného registru. Zpoždění způsobené vlastním kódem rutiny a skutečností, že rutina nemohla být spuštěna okamžitě, je tak eliminována. Záchytný registr dovolí přesně změřit délku časového intervalu, metoda toleruje i značně zpožděný start obslužné rutiny a navíc zjednodušuje programování. **Zajímavou funkcí, kterou může uživatel použít v situaci, kdy vstupní signál je zrušen, je filtrování vstupního signálu tak, že za správný stav se považuje situace, kdy všechny čtyři vzorky stavu na vstupním pinu ICP mají shodnou hodnotu**. Rychlost vzorkování stavu je ekvivalentní hodinovému kmitočtu mikroprocesoru.

Podobného zpřesnění a zjednodušení lze dosáhnout i při generování časových intervalů (jako výstupních signálů) použitím *komparačních (srovnávacích) registrů (Compare Register).* Obsah komparačního registru je komparátorem srovnáván s okamžitou hodnotou volně běžícího čítače. Při shodě je jednak realizována přeprogramovaná změna výstupního signálu (přechod do nuly, přechod do jedničky, změna hodnoty), jednak je vygenerován požadavek na přerušení. Aktivovaná přerušovací rutina připraví další hodnotu v komparačním registru a (případně) přeprogramuje další změnu výstupního signálu. Komparační registr a jeho podpůrné obvody dovoluje eliminovat zpoždění způsobené přerušovací rutinou při generování signálů. Samozřejmostí je také eliminace vlivu prodlevy mezi vypršením jednoho časového intervalu a startem dalšího, přesně lze realizovat i značně složité periodické signály.

Zatím jsme si popsali tři funkce *čítače/časovače 1*:

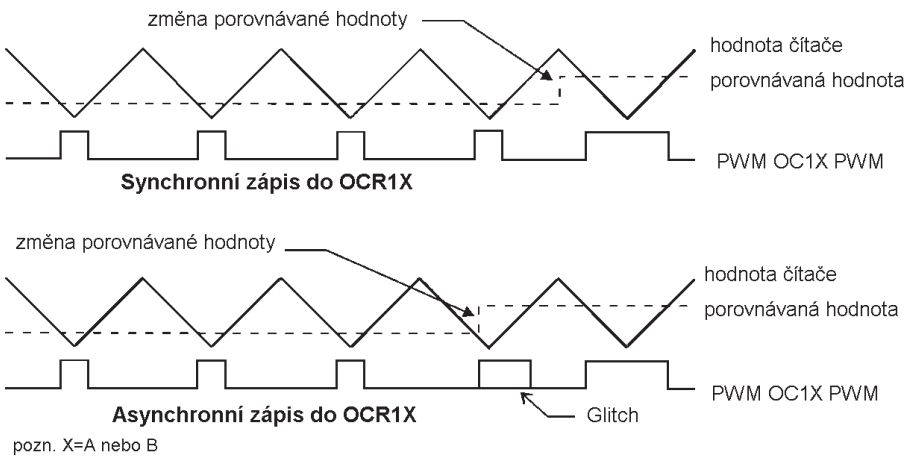
- Režim čítače/časovače ve kterém je jeho chování totožné s čítačem/časovačem 0 s tím rozdílem, že *čítač/časovač 1* je 16bitový. V případě, že je čítán externí signál, musí být doba mezi dvěma změnami tohoto signálu minimálně rovna nebo být větší jedné periodě hodinového signálu. Jinými slovy to znamená, že maximální měřený kmitočet externího signálu může být maximálně roven 1/2 kmitočtu CPU a to v případě střídy 1:1.
- Druhou funkcí, kterou může zmiňovaný blok vykonávat, je porovnávání obsahu čítače s obsahem registru *Output Compare Registr* a generování přerušení v momentu dosažení shody.
- Třetí funkcí, kterou můžeme využít, je zachycení stavu čítače v momentu, kdy se změní stav vstupního pinu ICP do aktivní úrovně.

Čtvrtou funkcí, kterou může tento blok plnit, je generování signálu PWM. V tomto případě *čítač/časovač 1* pracuje jako vzestupný/sestupný čítač, kdy nejdříve čítá vzestupně, po dosažení hodnoty TOP (viz dále) pak čítá sestupně. Hodnoty TOP určují jednak rozlišení PWM, jednak kmitočet opakování. Vztah mezi těmito veličinami ukazuje *tab. 4.3*.

Tab. 4.3 *Závislost PWM rozlišení a kmitočtu na TOP*

PWM rozlišení	Hodnota TOP časovače	Kmitočet
8 bit	0x00FF	$f_{TC1}/510$
9 bit	0x01FF	$f_{TC1}/1022$
10 bit	0x03FF	$f_{TC1}/2046$

Signál PWM generovaný tímto blokem je vyveden na vývod OC1A nebo OC1B v závislosti na nastavení obsahu registru TCCR1A (**Timer/Counter1 Control Register A**). Například při nastavení 10bitového PWM rozlišení získáme na výstupech OC1A nebo OC1B šířkově modulovaný signál o kmitočtu $f_{PWM} = f_{TC1}/2046$, kde f_{TC1} je kmitočet taktovacího signálu *čítače/časovače 1.* Generování signálu PWM probíhá tak, že nejprve čítač čítá vzestupně, v momentu dosažení shody s hodnotou v komparačním registru je výstup nastaven do požadovaného stavu. Po dosažení maximální hodnoty čítače (hodnota TOP časovače v tabulce) počne tento čítač čítat sestupně. V momentu dosažení shody s hodnotou v komparačním registru je výstup nastaven na původní úroveň. Nesynchronní zápis nové hodnoty do registru OCR1X (X = A nebo B) může způsobit vznik nežádoucího výstupního pulzu o těžko definované šířce, tzv. glitch. Proto návrháři vybavili mikroprocesor, pro tuto situaci, záchytným registrem. Mikroprocesor si pak novou hodnotu do registru OCR1X zapíše sám ve správný okamžik a uživatel není nucen tuto situaci vůbec řešit, viz příklad na *obr. 4.10*.



Obr. 4.10 *Generování signálů PWM*

K tomu, abychom mohli *čítač/časovač 1* používat v našich programech, potřebujeme ještě znát význam bitů v jednotlivých registrech. Proto si nyní provedeme jejich podrobnější popis.

Čítač/časovač 1 může čítat buď přímo hodinové impulzy mikrokontroléru CK (computer clock), nebo hodinové impulzy vydělené v předděličce, popř. impulzy přivedené z externího zdroje na pin MCU. Rovněž lze i zastavit čítání, což se určuje obsahem **řídicích registrů** *čítače/časovače 1* **TCCR1A** a **TCCR1B**. Stavové příznaky pro přetečení, docílení komparační úrovně či naplnění záchytného registru jsou obsaženy v **Timer/Counter Interrupt Flag Register (TIFR)**. Řídicí signály obsahují **Timer/Counter1 Control Registers (TCCR1A** a **TCCR1B)**. Povolení/zakázání přerušení pro *čítač/časovač 1* je provedeno v **Timer/Counter Interrupt Mask Register (TIMSK)**.

Timer/Counter1 Control Register A - TCCR1A
řídicí registr A čítače/časovače1

bit	7	6	5	4	3	2	1	0	
$2F ($4F)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	PWM11	PWM10	TCCR1A
čtení/zápis	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 4.11 *Řídicí registr A Čítače/časovače 0*

Bity 7, 6 – COM1A1, COM1A0: Compare Output Mode1A, bity 1 a 0

Řídicí bity COM1A1 a COM1A0 určují činnost výstupních pinů po dosažení shody při komparaci v *čítači/časovači 1*. Činnost výstupních pinů ovlivňuje pin OC1A (output CompareA). Jelikož je to alternativní funkce I/O portu, musí být řídicí bit směru nastaven na 1 (1 nastavuje odpovídající bit I/O portu jako výstup, 0 jako vstup).

Bity 5, 4 – COM1B1, COM1B0: Compare Output Mode1B, bity 1 a 0

Řídicí bity COM1B1 a COM1B0 určují činnost výstupních pinů po dosažení shody při komparaci v *čítači/časovači 1*. Činnost výstupních pinů ovlivňuje pin OC1B (output CompareB).

Řídicí konfigurace pro bity 4, 5, 6, a 7 je uvedena v *tab. 4.4*, přitom X = A nebo B.

Tab. 4.4 *Nastavení činnosti výstupních pinů po dosažení shody*

COM1X1	COM1X0	Popis
0	0	*Čítač/časovač 1* je odpojen od výstupu OC1X
0	1	Překlopí výstup OC1X
1	0	Vynuluje výstup OC1X
1	1	Nastaví výstup OC1X na jedničku

Bity 1, 0 – PWM11, PWM10: nastavení modulátoru PWM

Tyto bity řídí činnost PWM *čítače/časovače 1*, což je popsáno v *tab. 4.5*.

Tab. 4.5 *Nastavení modulátoru PWM*

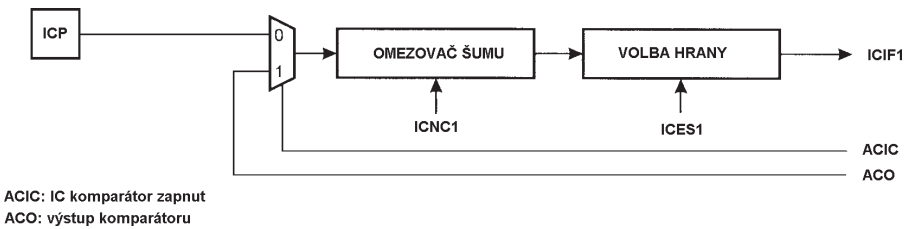
PWM11	PWM10	Popis
0	0	Režim PWM čítače/časovače1 není povolen
0	1	Režim 8bitový PWM
1	0	Režim 9bitový PWM
1	1	Režim 10bitový PWM

Timer/Counter1 Control Register B - TCCR1B
řídící registr B čítače/časovače1

bit	7	6	5	4	3	2	1	0	
$2E ($4E)	ICNC1	ICES1	–	–	CTC1	CS12	CS11	CS10	TCCR1B
čtení/zápis	R/W	R/W	R	R	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 4.12 *Řídicí registr B Čítače/časovače 1*

Obsah bitu 6 a 7 souvisí s možností již zmíněné filtrace vstupního signálu přivedeného na vstup ICP (Input Capture Pin). Tento pin je připojen k logice zobrazené na *obr. 4.13*



Obr. 4.13 *Logika filtrace šumu ze vstupního signálu*

Bit 7 – ICNC1: Input Capture 1 Noise Canceler – ovládací vstup omezovače šumu

Když ICNC1 je nastaven na nulu, je vypnuto omezování šumu. Vstup je vzorkován při první vzestupné/sestupné hraně na ICP podle toho, zda ICES1 = 0 (pak se vzorkuje se sestupnou hranou) nebo ICES1 = 0 (se vzestupnou hranou). Je-li ICNC1 = 1, provádí se filtrování tím, že za nezašuměný signál je považován takový, jehož následující čtyři vzorky jsou stejné.

Bit 6 – ICES1: Input Capture1 Edge Select

Volba vzorkování se vzestupnou (ICES1 = 0), či sestupnou hranou (ICES1 = 1)

Bit 3 / CTC1: Clar Timer/Counter1 on Compare Match

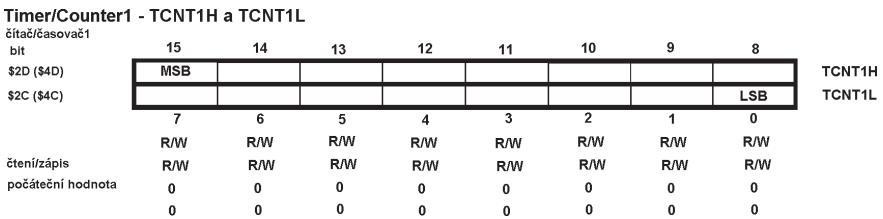
Je-li CTC1 = 1, je *čítač/časovač 1* vynulován na $0000 při hodinovém cyklu následujícím po dosažení shody v komparátoru. Když je CTC1 = 0, tak *čítač/časovač1* pokračuje v čítání a dosažení shody v komparátoru na něj nemá vliv.

Bity 2, 1, 0 – CS12, CS11 a CS10, Clock Select1

Tyto bity určují zdroj signálu či dělicí poměr předděličky a jsou popsány v *tab. 4.6*.

Tab. 4.6 *Určení zdroje signálu či dělicího poměru předděličky*

CS02	CS01	CS00	Význam
0	0	0	Zastavení čítání, *časovač/čítač 1* je zastaven
0	0	1	CK
0	1	0	CK/8
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	Externí vstup T1, sestupná hrana
1	1	1	Externí vstup T1, vzestupná hrana



Obr. 4.14 *Registr TCNT1*

Tento registr obsahuje přednastavenou hodnotu 16bitového *čítače/časovače 1*. Protože při zápisu či čtení 16bitového registru by mohlo dojít k chybě, způsobené tím, že např. při přečtení nižšího byte čítače, který je v činnosti, není zaručeno, že nedojde ke změně vyššího bytu dříve, než dojde k jeho přečtení. Jinými slovy hodnota vyššího byte by již nepatřila k nižšímu byte.

Na toto návrháři pamatovali a vybavili vyšší byte 16bitového registru pomocným TEMP 8bitovým registrem. Pokud budeme při čtení postupovat tak, že přečteme nejdříve nižší byte, dojde při čtení nižšího byte *čítače/časovače 1* ve stejný okamžik k zachycení okamžitého stavu vyššího byte *čítače/časovače 1* do záchytného TEMP registru, takže následné čtení vyššího byte přečte jeho hodnotu ne z registru *čítače/časovače 1*, ale z jeho záchytného registru. Postup při zápisu 16bitové hodnoty je přesně opačný. Nejdříve je nutné zapsat hodnotu vyššího byte, přičemž tato hodnota se nezapíše přímo do příslušného registru, ale do záchytného (TEMP) registru. Při následném zápisu hodnoty do nižšího byte 16bitového registru dojde ve shodný okamžik k zapsání zachycené hodnoty ze záchytného registru do vyššího byte registru.

Výstupní komparátorové registry jsou 16bitové registry pro zápis i čtení. Obsahují data, která jsou nepřetržitě porovnávána s obsahem *čítače/časovače 1*. Jaká je

Timer/Counter1 Output Compare Register - OCR1AH a OCR1AL

porovnávací výstupní registr čítače/časovače1

bit	15	14	13	12	11	10	9	8	
$2B ($4B)	MSB								OCR1AH
$2A ($4A)								LSB	OCR1AL
	7	6	5	4	3	2	1	0	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Timer/Counter1 Output Compare Register - OCR1BH a OCR1BL

porovnávací výstupní registr čítače/časovače1

bit	15	14	13	12	11	10	9	8	
$29 ($49)	MSB								OCR1BH
$28 ($28)								LSB	OCR1BL
	7	6	5	4	3	2	1	0	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Obr. 4.15 *Porovnávací výstupní registr Čítače/časovače 1*

reakce na dosažení shody porovnávaných dat v komparátoru je určeno v řídicím a stavovém registru *čítače/časovače 1*. Reakce na dosažení shody porovnávaných dat však nastává pouze dojde-li ke shodě tím, že čítač načítá svůj obsah na hodnotu OCR. Nastavení stejných hodnot TCTN1 a OCR1A nebo OCR1B softwarově však negeneruje shodu porovnávaných dat.

Dosažení shody porovnávaných dat nastaví příznak přerušení při CPU hodinovém cyklu následujícím po dosažení shody. Protože výstupní komparátorové registry OCR1A a OCR1B jsou 16bitové, je opět užit pomocný TEMP registr, stejně jako v případě registru TCNT1. Rovněž postup při zápisu či čtení tohoto registru je stejný, jak již bylo popsáno. TEMP registr je používán i při přístupu k registru ICR1.

Timer/Counter1 Input Capture Register - ICR1H a ICR1L

vstupní záchytný registr čítače/časovače1

bit	15	14	13	12	11	10	9	8	
$25 ($45) popř. $27 ($47)	MSB								ICR1H
$24 ($44) popř. $26 ($46)								LSB	ICR1L
	7	6	5	4	3	2	1	0	
	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

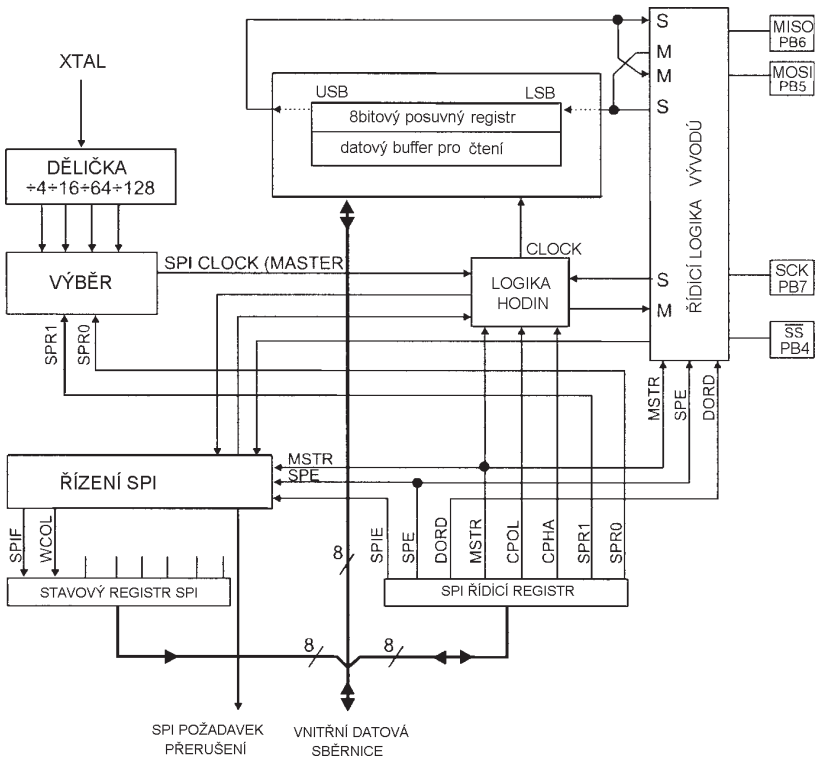
Obr. 4.16 *Vstupní záchytný registr Čítače/časovače 1*

Vstupní záchytný registr je registr určený pouze ke čtení (read only).

Když je detekována aktivní nebo sestupná hrana (v závislosti na nastavení ICES1) vstupního signálu z ICP, je okamžitá hodnota *čítače/časovače 1* přenesena do Input Capture Registru ICR1. Současně je nastaven příznak ICF1 na 1. Protože registr ICR1 je 16bitový a pro přístup k tomuto registru je použit dočasný registr TEMP, **je nutné** použít postup pro čtení/zápis 16bitového registru tak, jak byl dříve popsán.

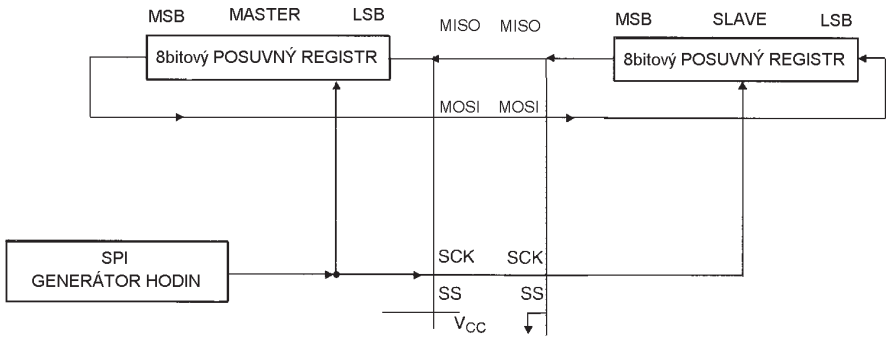
Serial Periphal Interface SPI – synchronní sériový port

Jelikož je škoda mít v mikroprocesoru téměř celý synchronní port (kanál) jen pro účely programování vnitřních pamětí, vybavili návrháři tento mikroprocesor tak, aby synchronní kanál používaný při programování vnitřních pamětí (ISP), měl všechny potřebné části potřebné i pro komunikaci při běhu programu (toto tvrzení platí pro většinu typů AVR MCU, výjimkou je např. ATmega103, u kterého se k sériovému programování používají signály SCK, RxD a TxD). Rozhraní SPI umožňuje nastavit typ zařízení, tj. zda je typu Master nebo Slave, lze nastavit, který bit, MSB či USB bude vysílán jako první, pomocí vlastního generátoru přenosových rychlostí lze nastavit čtyři přenosové rychlosti, aniž by uživatel přišel o jediný čítač/časovač. Rozhraní je též schopné detekovat kolizi na sběrnici a v případě, že je v módu Slave, dokáže „vzbudit" celý mikroprocesor z IDLE módu. Celkové blokové schéma synchronního sériového kanálu je na *obr. 4.17*.



Obr. 4.17 *Blokové schéma synchronního sériového kanálu*

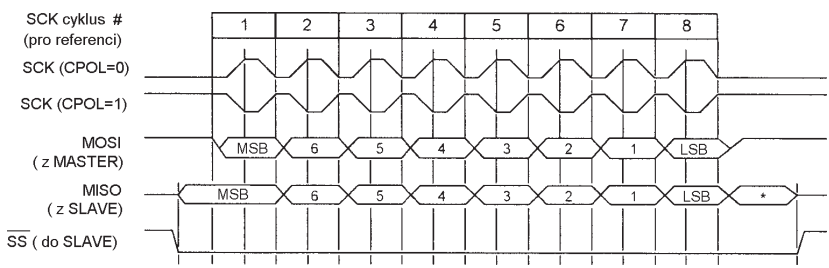
Spojení mezi CPU, které je Master a CPU, které je Slave ukazuje *obr. 4.18*



Obr. 4.18 *SPI spojení MASTER a SLAVE*

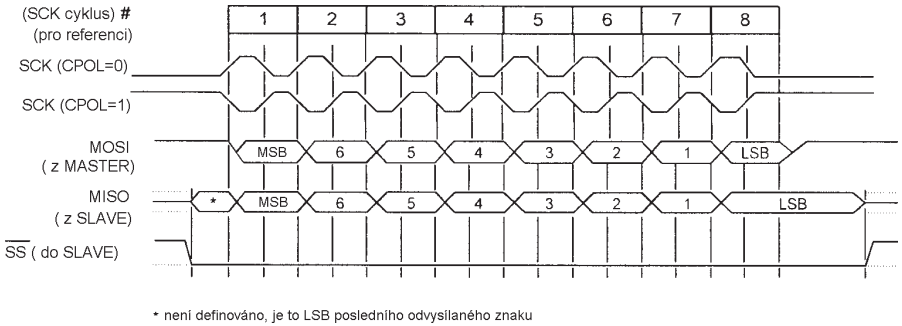
Pin PB7 (SCK) je v režimu Master zdrojem hodinových pulzů, v režimu Slave je naopak vstupem, tyto impulzy přijímajícím. Zápisem do SPI datového registru v CPU Master se začnou generovat hodinové impulzy a zapsaná data se posouvají z posuvného registru přes výstupní pin PB7 (MOSI – Master Out Slave In) a vstupují do vstupního pinu PB5 CPU Slave. Po přesunutí jednoho byte se SPI hodinový generátor zastaví a nastaví se příznak **konec přenosu** SPIF. Je-li v registru SPCR nastaven bit SPIE (povolení SPI přerušení), je po ukončení zmíněného přenosu jednoho byte vyslán požadavek na přerušení. Vstup Slave Select (PB4) slouží k výběru příslušného SPI Slave a to nastavením úrovně na tomto vstupu na nulu. Na *obr. 4.18* je také vidět, že posouvají-li se data z CPU Master do CPU Slave, posouvají se současně i data z CPU Slave do CPU Master, tudíž se během jednoho posouvacího cyklu data vzájemně přehodí.

Celkem jsou možné čtyři módy přenosu dat dané čtyři kombinacemi fáze a polarity SCK vzhledem k sériovým datům. Tyto módy zobrazuje *obr. 4.19* a *obr. 4.20*.



* není definováno, je to MSB právě přijatého znaku

Obr. 4.19 *SPI formát přenosu pro CPHA = 0 a DORD = 0*



Obr. 4.20 *SPI formát přenosu pro CPHA = 1 a DORD = 0*

SPI Control Register - SPCR
řídící registr SPI

bit	7	6	5	4	3	2	1	0	
$0D ($2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 4.21 *Řídicí registr SPI*

Bit 7 – SPIE: SPI Interrupt Enable

Je-li tento bit nastaven na 1 a je-li současně povoleno globální přerušení (tj. povolení všech přerušení), je povoleno SPI přerušení.

Bit 6 – SPE: SPI Enable

Je-li tento bit nastaven na 1, je SPI povolen. Tento bit musí být nastaven, má-li se provádět libovolná SPI operace.

Bit 5 – DORD: Data Order – pořadí dat

Je-li DORD nastaven na 1, přenáší se nejdříve LSB datového slova, je-li DORD vynulován na 0, přenáší se nejprve MSB.

Bit 4 – MSTR: Master/Slave Select – výběr funkce jako master nebo slave

Tento bit vybírá SPI master mód v případě nastavení na jedničku. Je-li vynulován, vybírá Slave SPI mód. Pokud je $\overline{SS}$ nakonfigurován jako vstup a je na něm nastavena nula, poté co MSTR je nastaven, MSTR bude vynulován a SPIF v SPRS bude nastaven na jedničku.

Bit 3 – CPOL – Clock Polarity – polarita hodinových impulzů

Je-li tento bit nastaven na jedničku, SCK je v klidovém stavu na úrovni jedna, je-li CPOL vynulován, je SCK v klidovém stavu na nule.

Bit 2 – CPHA – Clock Phase – fáze hodin

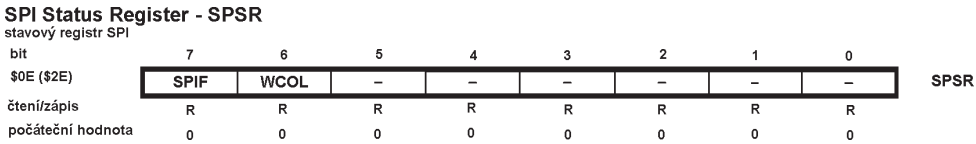
Význam je zřejmý z *obr. 4.19* a *obr. 4.20*

Bit 1 – SPR1, SPR0: SPI Clock Rate Select 1 a 0

Tyto dva bity řídí kmitočet SCK u zařízení nakonfigurovaném jako master. Nemá vliv při nakonfigurování jako slave. Vztah mezi kmitočtem CSK a kmitočtem oscilátoru f_{CL} ukazuje *tab. 4.7.*

Tab. 4.7 *Nastavení kmitočtu SCK*

SPR1	SPR0	Kmitočet SCK
0	0	$f_{CL}/4$
0	1	$f_{CL}/16$
1	0	$f_{CL}/64$
1	1	$f_{CL}/128$



Obr. 4.22 ***Stavový registr SPI***

Bit 7 – SPIF: SPI Interrupt Flag

Po ukončení sériového přenosu je bit SPIF nastaven na jedničku a je generováno přerušení, když SPIE v SPCR je nastaven na jedničku a současně je povoleno globální přerušení (povolení všech přerušení). Je-li $\overline{SS}$ vstupem a je na něm úroveň nula a SPI je v master módu, nastaví také příznak SPIF. Alternativně, SPIF je hardwarově vynulován při provádění odpovídajícího vektoru přerušení. SPIF je vynulován prvním čtením SPI stavového registru, v případě že WCOL je nastaven na jedničku a pak proveden přístup do SPI Datového registru.

Bit 6 – WCOL: Write Collision Flag

Bit WCOL je nastaven, když během přenosu dat je naplněn SPI datový registr (SPDR). Bit WCOL a bit SPIF jsou vynulovány při prvním čtení SPI datového registru při nastaveném WCOL na jedničku a pak při přístupu k SPI datovému registru.

Bit 5 až 0 – nevyužito, rezerva

SPI Data Register - SPDR
datový registr SPI

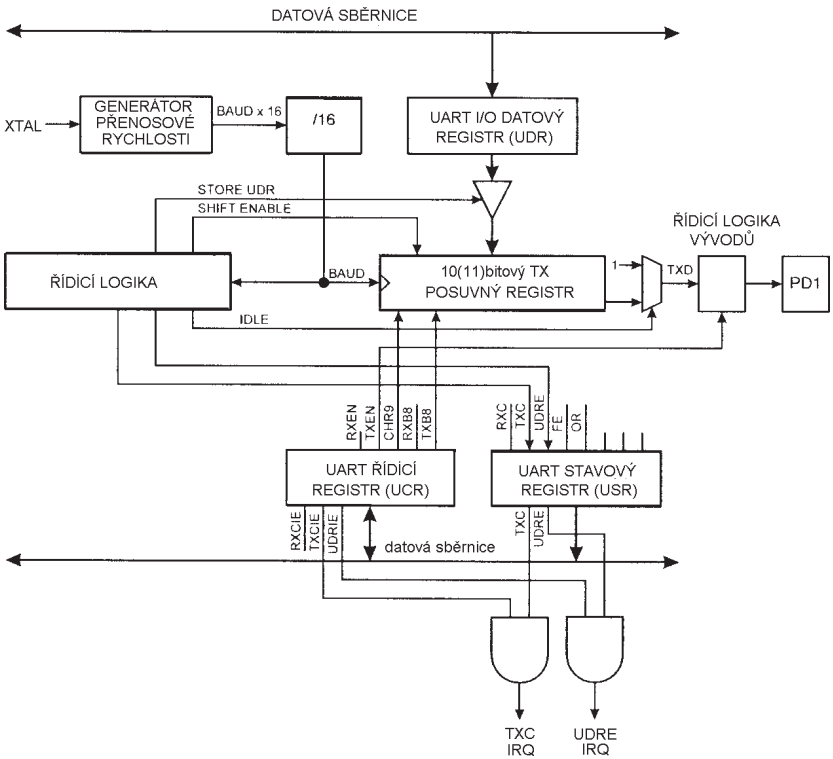
bit	7	6	5	4	3	2	1	0	
$0F ($2F)	MSB							LSB	SPDR
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	x	x	x	x	x	x	x	x	nedefinovaná

Obr. 4.23 *Datový registr SPI*

SPI datový registr je registr pro čtení i zápis používaný k přenosu dat mezi souborem registrů a SPI posuvným registrem. Zápisem do registru se zahajuje datový přenos. Čtení registru způsobí, že je čten přijímací posuvný registr.

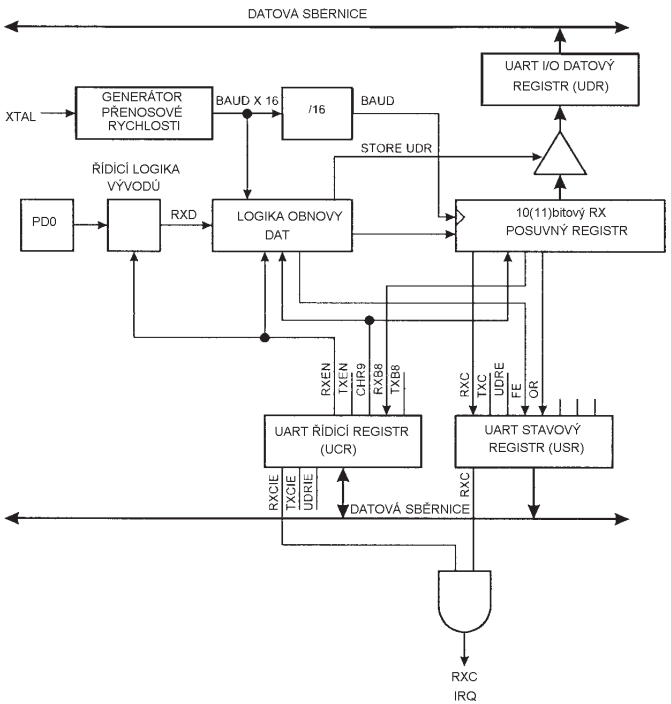
UART

Univerzální sériový přijímač/vysílač, asynchronní kanál, je běžnou částí AVR MCU. Je to plně duplexní sériový kanál, umožňující komunikaci ve standardním 8 a 9bitovém asynchronním režimu. Obsahuje i detekci falešného start bitu, detekci chybného znaku a přetečení datového registru, filtraci šumu. Aby byla obsluha



Obr. 4.24 *Vysílací část*

sériového portu z hlediska programátora snadnější, disponuje sériový kanál celkem třemi samostatnými vektory přerušení: vysílání dokončeno (transmitter register empty), vysílací vyrovnávací registr prázdný (data register empty) a příjem kompletní (receive complete). Jelikož se sériová komunikace používá v mnoha případech pro multiprocesorovou komunikaci, nebo komunikaci mezi více zařízeními, která komunikují po jedné sběrnici (např. RS-485), podporuje sériový kanál 9bitovou komunikaci, která umožňuje snadno oddělit data od povelů. Aby uživatel nepřišel o jeden čítač/časovač při použití sériového kanálu, vybavili návrháři sériový kanál vlastním generátorem přenosových rychlostí. Díky tomu, že generátor využívá přímo kmitočet hlavního oscilátoru, lze využít vysokých přenosových rychlostí i při relativně nízkém systémovém taktu. Celková bloková schémata obou částí, přijímací i vysílací, jsou na *obr. 4.24* a *obr. 4.25*.



Obr. 4.25 *Přijímačová část*

Řídicí registry UARTu

UART I/O Data Register - UDR
I/O UART datový registr

bit	7	6	5	4	3	2	1	0	
$0C ($2C)	MSB							LSB	UDR
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 4.26 *I/O UART datový registr*

UDR registr jsou ve skutečnosti fyzicky oddělené registry sdílející tytéž I/O adresy. Pří zápise do UDR registru se zapisuje do UART Data Transmit registru, při čtení z UDR se čtou data z UART **receive data** registru.

UART Status Register - USR
stavový registr UARTu

bit	7	6	5	4	3	2	1	0	
$0B ($2B)	RXC	TXC	UDRE	FE	OR	–	–	–	USR
čtení/zápis	R	R/W	R	R	R	R	R	R	
počáteční hodnota	0	0	1	0	0	0	0	0	

Obr. 4.27 *Stavový registr UARTu*

USR registr slouží pouze ke čtení, poskytuje informace o stavu UARTu.

Bit 7 – RXC: UART Receive Complete

Tento bit je nastaven na jedničku, pokud je přijatý znak přesunut z přijímacího posuvného registru do UDR. Bit je nastaven bez ohledu a případnou detekci chyb přenosu rámce. Je-li RXCIE bit v UCT nastaven, bude provedeno UART přerušení **příjem úplný**, pokud je RXC nastaven na jedničku. RXC je vynulován čtením UDR. Proběhne-li přerušení vyvolané příjmem dat, tak obslužná rutina musí číst UDR a vymazat RXC, jinak nastane nové přerušení před ukončením rutiny.

Bit 6 – TXC: UART Transmit Complete

Tento bit je nastaven na jedničku, pokud celý znak včetně stop bitu byl z vysílacího posuvného registru posunut ven a žádná data nejsou zapsána do UCR. Tento příznak je velmi důležitý v interface pro poloviční duplex, kdy vysílací aplikace musí spustit přijímací režim a musí uvolnit komunikační kanál bezprostředně po ukončení vysílání.

Bit 5 – UDRE: UART Data Register Empty

Tento bit je nastaven na jedničku, v případě že znak zapsaný do UDR je přenesen do **Transmit Shift** registru. Nastavení tohoto bitu indikuje, že vysílač je připraven přijmout nový znak k vysílání. Když UDRIE bit v UCR je nastaven, UART přerušení **přenos úplný** bude prováděno pokud je UDRE nastaven. UDRE je vynulován při zápisu UDR. Používá-li se datový přenos řízený přerušením, UART Data Register Empty Interrupt rutina musí zapsat UDR, aby vynuloval UDRE, jinak by nastalo nové přerušení jakmile skončí obslužná routina.

UDRE je nastaven na jedničku během resetu, aby indikoval, že vysílač je připraven.

Bit 4 – FE: Framing Error

Tento bit je nastaven, pokud je detekován Framing Error, tj. když stop bit přicházejícího znaku je nula. FE bit je vynulován, je-li stop bit přijímaných dat jednička.

Bit 3 OR: Overrun

Tento bit je nastaven v případě, že znak, který je již přítomný v UDR registru, není přečten před tím, než je další znak posunut do přijímacího posuvného registru. Bit OR je překlopen, což znamená, že bude nastaven dokud platná data v UDRE jsou čtena.

Bit 2 až 0 rezervované

Jsou nevyužity, při čtení se přečte nula.

UART Control Register - UCR
řídící registr UARTu

bit	7	6	5	4	3	2	1	0	
$0A ($2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UCR
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R	W	
počáteční hodnota	0	0	0	0	0	0	1	0	

Obr. 4.28 *Řídicí registr UARTu*

Bit 7 – RXCIE: RX Complete Interrupt Enable

Pokud je tento bit nastaven na jedničku, nastavení bitu RXC v UCR vyvolá Receive Complete Interrupt rutinu, je-li ovšem povoleno globální přerušení (= povolení všech přerušení).

Bit 6 – TXCIE: TX Complete Interrupt Enable

Je-li tento bit nastaven, nastavení bitu TXC v UCR vyvolá Transmit Complete Interrupt rutinu v případě, že je povoleno globální přerušení (= povolení všech přerušení).

Bit 5 – UDRIE: UART Data Register Empty Interrupt Enable

Pokud je tento bit nastaven, nastavení bitu UDRE v USR vyvolá UART Data Register Empty rutinu, je-li ovšem povoleno globální přerušení (= povolení všech přerušení).

Bit 4 – RXEN : Reciever Enable

Tento bit, je-li nastaven, povoluje, zapíná, přijímač UARTu. Je-li přijímač vypnut, stavové příznaky TXC, OR a FE se nemohou nastavit. Jsou-li tyto příznaky nastaveny, vypnutí RXEN nenastane.

Bit 3 – TXEN: Transmitter Enable

Je-li tento bit nastaven, zapíná vysílač UARTu. Pokud vypíná vysílač během doby, kdy vysílač vysílá znak, vysílač není vypnut před tím, než jsou znaky v posuvném registru a libovolný znak v UDR kompletně odvysílány.

Bit 2 – CHR9: I-bit Characters

Je-li tento bit nastaven, vysílané i přijímané znaky jsou dlouhé 9 bitů, plus start a stop pulzy. Devátý bit je čten a zapisován užitím bitu RXB8 resp. TXB8 v UCR. Devátý bit může být také užit jako další stop bit nebo bit parity.

Bit 1 – RXB8: Receive Data Bit 8

Pokud je CHR9 nastaven, RXB8 je devátý datový bit přijímaného znaku.

Bit 0 – TXB8: Transit Data Bit 8

Je-li CHR9 nastaven, TXB8 je devátý datový bit v znaku, který se má vysílat.

Generátor (telegrafní) rychlosti (BAUD generator)

Generátor přenosové rychlosti je dělička kmitočtu, která generuje přenosové rychlosti podle následujícího vzorce

$$BAUD = \frac{F_{CK}}{16\,(UBRR + 1)}$$

kde BAUD je přenosová rychlost, F_{CK} je kmitočet krystalu, UBRR je obsah UART Baud Rate registru v rozmezí 0 až 255

Tab. 4.8 ukazuje pro nejčastěji používané kmitočty krystalů a často používané přenosové rychlosti hodnoty UBRR, které poskytuje tyto přenosové rychlosti s chybou max. 2 %

Tab. 4.8 *Hodnoty UBRR*

Baud Rate	1 MHz		%Error	1.8432 MHz		%Error	2 MHz		%Error	2.4576 MHz		%Error
2400	UBRR=	**25**	**0.2**	UBRR=	**47**	**0.0**	UBRR=	**51**	**0.2**	UBRR=	**63**	**0.0**
4800	UBRR=	**12**	**0.2**	UBRR=	**23**	**0.0**	UBRR=	**25**	**0.2**	UBRR=	**31**	**0.0**
9600	UBRR=	6	7.5	UBRR=	**11**	**0.0**	UBRR=	**12**	**0.2**	UBRR=	**15**	**0.0**
14400	UBRR=	3	7.8	UBRR=	**7**	**0.0**	UBRR=	8	3.7	UBRR=	10	3.1
19200	UBRR=	2	7.8	UBRR=	**5**	**0.0**	UBRR=	6	7.5	UBRR=	**7**	**0.0**
28800	UBRR=	1	7.8	UBRR=	**3**	**0.0**	UBRR=	3	7.8	UBRR=	4	6.3
38400	UBRR=	1	22.9	UBRR=	**2**	**0.0**	UBRR=	2	7.8	UBRR=	**3**	**0.0**
57600	UBRR=	0	7.8	UBRR=	**1**	**0.0**	UBRR=	1	7.8	UBRR=	2	12.5
76800	UBRR=	0	22.9	UBRR=	1	33.3	UBRR=	1	22.9	UBRR=	**1**	**0.0**
115200	UBRR=	0	84.3	UBRR=	**0**	**0.0**	UBRR=	0	7.8	UBRR=	0	25.0

Baud Rate	3.2768 MHz		%Error	3.6864 MHz		%Error	4 MHz		%Error	4.608 MHz		%Error
2400	UBRR=	**84**	**0.4**	UBRR=	**95**	**0.0**	UBRR=	**103**	**0.2**	UBRR=	**119**	**0.0**
4800	UBRR=	**42**	**0.8**	UBRR=	**47**	**0.0**	UBRR=	**51**	**0.2**	UBRR=	**59**	**0.0**
9600	UBRR=	**20**	**1.6**	UBRR=	**23**	**0.0**	UBRR=	**25**	**0.2**	UBRR=	**29**	**0.0**
14400	UBRR=	**13**	**1.6**	UBRR=	**15**	**0.0**	UBRR=	16	2.1	UBRR=	**19**	**0.0**
19200	UBRR=	10	3.1	UBRR=	**11**	**0.0**	UBRR=	**12**	**0.2**	UBRR=	**14**	**0.0**
28800	UBRR=	**6**	**1.6**	UBRR=	**7**	**0.0**	UBRR=	8	3.7	UBRR=	**9**	**0.0**
38400	UBRR=	4	6.3	UBRR=	**5**	**0.0**	UBRR=	6	7.5	UBRR=	7	6.7
57600	UBRR=	3	12.5	UBRR=	**3**	**0.0**	UBRR=	3	7.8	UBRR=	**4**	**0.0**
76800	UBRR=	2	12.5	UBRR=	**2**	**0.0**	UBRR=	2	7.8	UBRR=	3	6.7
115200	UBRR=	1	12.5	UBRR=	**1**	**0.0**	UBRR=	1	7.8	UBRR=	2	20.0

***Tab. 4.8** (pokračování) Hodnoty UBRR*

Baud Rate	7.3728 MHz		%Error	8 MHz		%Error	9.216 MHz		%Error	11.059 MHz		%Error
2400	UBRR=	191	0.0	UBRR=	207	0.2	UBRR=	239	0.0	UBRR=	287	-
4800	UBRR=	95	0.0	UBRR=	103	0.2	UBRR=	119	0.0	UBRR=	143	0.0
9600	UBRR=	47	0.0	UBRR=	51	0.2	UBRR=	59	0.0	UBRR=	71	0.0
14400	UBRR=	31	0.0	UBRR=	34	0.8	UBRR=	39	0.0	UBRR=	47	0.0
19200	UBRR=	23	0.0	UBRR=	25	0.2	UBRR=	29	0.0	UBRR=	35	0.0
28800	UBRR=	15	0.0	UBRR=	16	2.1	UBRR=	19	0.0	UBRR=	23	0.0
38400	UBRR=	11	0.0	UBRR=	12	0.2	UBRR=	14	0.0	UBRR=	17	0.0
57600	UBRR=	7	0.0	UBRR=	8	3.7	UBRR=	9	0.0	UBRR=	11	0.0
76800	UBRR=	5	0.0	UBRR=	6	7.5	UBRR=	7	6.7	UBRR=	8	0.0
115200	UBRR=	3	0.0	UBRR=	3	7.8	UBRR=	4	0.0	UBRR=	5	0.0

UART BAUD Rate Register - UBRR
UART registr přenosové (telegrafní) rychlosti

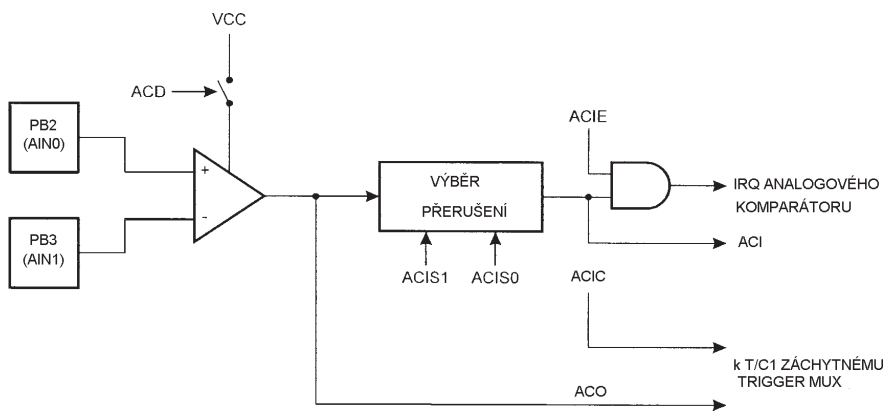
bit	7	6	5	4	3	2	1	0	
$09 ($29)	MSB							LSB	UBRR
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

***Obr. 4.29** Registr UART přenosové rychlosti*

UBRR registr je 8bitový registr pro zápis i čtení, který určuje přenosovou rychlost podle rovnice uvedené výše.

Analogový komparátor

Jelikož mnoho signálů, které mikroprocesory zpracovávají, jsou analogové, musí se nejdříve převést do digitální formy. Tomuto účelu slouží A/D převodníky. Pro mnoho aplikací je však přítomnost drahého A/D převodníku z hlediska ceny vlastního mikroprocesoru nevhodná. Pro řadu aplikací postačují méně přesné metody převodu, např. pomocí analogového komparátoru. Blokové schéma analogového komparátoru je na *obr. 4.30*



***Obr. 4.30** Blokové schéma analogového komparátoru*

Analog Comparator Control and Status Register - ACSR
řídící a stavový registr analogového komparátoru

bit	7	6	5	4	3	2	1	0	
$08 ($28)	ACD	–	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
čtení/zápis	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	N/A	0	0	0	0	0	

Obr. 4.31 *Řídicí a stavový registr analogového komparátoru*

Bit 7 – ACD: Analog Comparator Disable

Nastavení tohoto bitu na jedničku způsobí odpojení napájení analogového komparátoru. Tento bit může být kdykoliv nastaven, aby vypnul analogový komparátor. Umožňuje to snižovat spotřebu. Provádí-li se změna bitu ACD, musí být zakázáno přerušení analogového komparátoru vynulováním bitu ACIE v ACSR, jinak by došlo k přerušení.

Bit 6 – rezerva

Bit 5 – ACO: Analog Comparator Interrupt Flag

ACO je přímo spojeno s výstupem komparátoru.

Bit 4 – ACI: Analog Comparator Interrupt Flag

Tento bit je nastaven na jedničku v případě, že výstup komparátoru způsobí spuštění přerušení definované pomocí ACI1 a ACI0. Obslužná rutina **Analog Comparator** přerušení je provedena, pokud bit ACIE je nastaven na jedničku a bit I v SREG je nastaven na jedničku. ACI je vynulován hardwarem, provede-li se odpovídající vektor přerušení. Alternativně, ACI je vynulován zapsáním jedničky do příznaku. Ještě si všimněme, že pokud další bit v tomto registru je modifikován užitím instrukce SBI nebo CBI, bude ACI vynulován v případě, dojde-li k tomu již před operací.

Bit 3 – ACIE – Analog Comparator Interrupt Enable

Je-li tento bit nastaven na jedničku a bit I je ve stavovém registru nastaven na jedničku, přerušení analogového komparátoru je aktivováno. Je-li vynulován, přerušení je zakázáno.

Bit 2 – ACIC: Analog Comparator Input Capture Enable

Je-li tento bit nastaven na jedničku, bit zapíná Input Capture funkci *čítače/časovače 1* být spouštěn analogovým komparátorem. Výstup komparátoru je v tomto případě přímo spojen s Input Capture logikou.

Bit 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select

Tyto bity určují, které přerušení vyvolá analogový komparátor, viz *tab. 4.9.*

Tab. 4.9 *Nastavení módu přerušení*

ACIS1	ACIS0	Mód přerušení
0	0	Přerušení při překlápění výstupu
0	1	rezervováno
1	0	Přerušení při sestupné hraně výstupu
1	1	Přerušení při vzestupné hraně výstupu

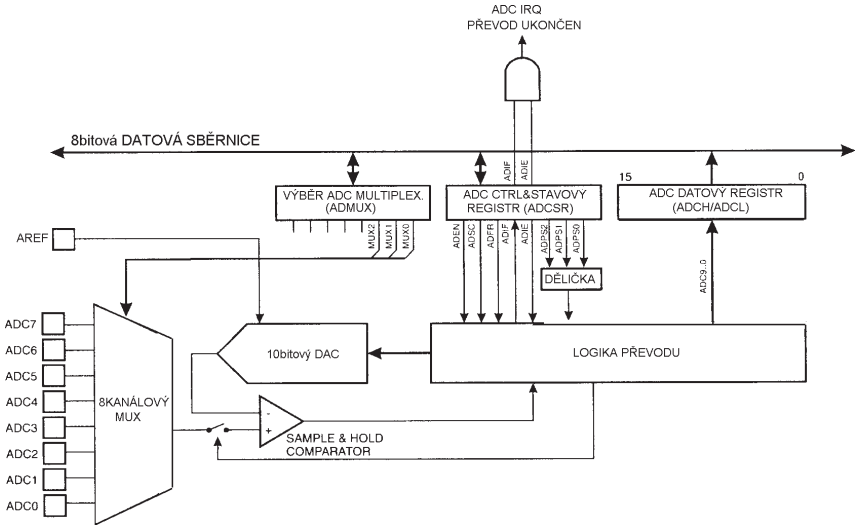
A/D převodník

A/D převodník je periferie, kterou obsahují některé typy AVR MCU. Např. mikrokontrolér AT90S8535 je vybavený vstupním analogovým multiplexerem na který je připojen A/D převodník pracující na principu postupné aproximace s rozlišením až 10 bitů. Výrobce zaručuje ±0,5 LSB maximální nelinearitu a 2 LSB absolutní přesnost. Převodník je schopen poskytnout maximálně 15 ksps při zachování maximální přesnosti. Protože převodník potřebuje ke své činnosti zdroj řídicího kmitočtu, disponuje převodník vlastní předděličkou, která je schopna poskytnout požadovaný kmitočet vydělením kmitočtu systémového. Pro dosažení maximální přesnosti se musí řídicí kmitočet pro A/D převodník pohybovat v rozmezí 50 kHz až 200 kHz. V rozmezí těchto kmitočtů se doba převodu pohybuje v rozmezí 260 µs až 65 µs. Pokud nepotřebujeme maximální přesnost, lze zvýšit řídicí kmitočet až na 2 MHz. Na druhé straně, pokud potřebujeme dosáhnout maximální přesnosti, doporučuje výrobce několik opatření počínaje dobrou filtrací napájecího napětí pro analogovou část MCU přes vhodný návrh desky plošných spojů s vhodným rozmístěním součástek, až po možnost uvést MCU do úsporného režimu, kdy zůstane v činnosti pouze část periferií. Uvedením MCU do úsporného režimu se výrazně omezí vnitřní rušení signály z digitální části MCU. Blokové schéma A/D převodníku ukazuje *obr. 4.32* a *obr. 4.33* ukazuje předděličku.

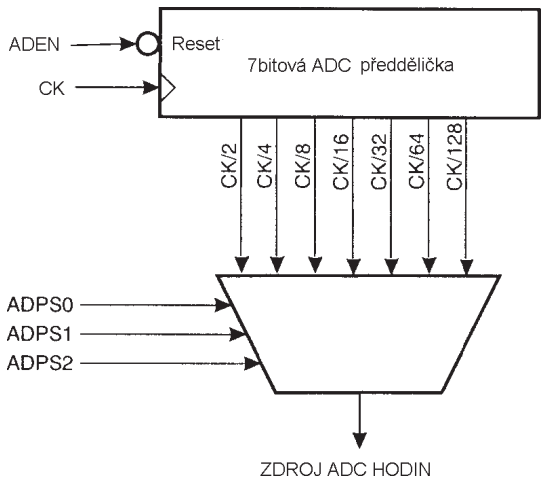
A/D převodník (ADC) pracuje ve dvou módech – s jedním převodem a volně běžící. V módu s jedním převodem je každý převod inicializován uživatelem. Ve volně běžícím módu je A/D převodník pravidelně vzorkován a obnovuje data v ADC datovém registru. Přepínání mezi těmito dvěmi módy provádí bit ADFR v ADCSR registru.

ADC je zapnut, povolen, ADC Enable bitem, ADEN v ADCSR. Výběr vstupního kanálu nebude funkční, dokud nebude ADEN nastaven. ADC nemá žádnou spotřebu, když je ADEN vynulován, proto se doporučuje vypnout ADC před spuštěním sleep módu se sníženou spotřebou.

ADC poskytuje 10bitový výsledek, který je uložen v ADC datovém registru, ADCH a ADCL. Při čtení dat se musí nejdříve přečíst ADCL, pak ADCH, aby se zajistilo to, že údaj, který se takto přečte, patří k témuž převodu. Při čtení ADCL je blokován přístup ADC k datovému registru. To znamená, že když ADCL byl přečten a převod dokončen předtím, než ADCH je přečten, žádný registr není přepsán a výsledek



Obr. 4.32 *Blokové schéma A/D převodníku*



Obr. 4.33 *Předdělička pro ADC*

převodu je ztracen. V případě přečtení ADCH, je přístup ADC k ADCH a ADCL registrům znovu povolen.

ADC má vlastní přerušení, které může být spuštěno, když je dokončen převod. Když přístup ADC k datovým registrům je zakázán mezi čtením ADCH a ADCL, bude spuštěno přerušení dokonce když výsledek je ztracen.

Omezovač šumu v ADC

ADC obsahuje možnost omezit šum při převodu během Idle módu způsobený indukcí z MCU jádra. K tomu, aby se dal omezit šum, musí být splněno:

ADC je povolen a není ve stavu **busy converting**.

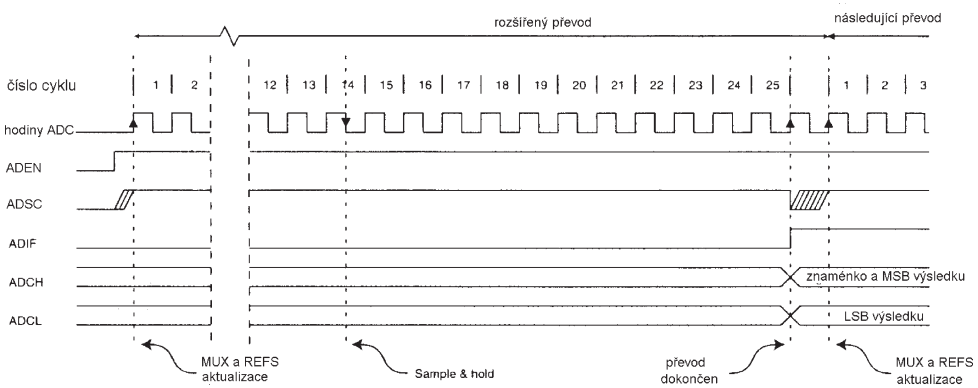
ADEN = 1

ADSC = 0

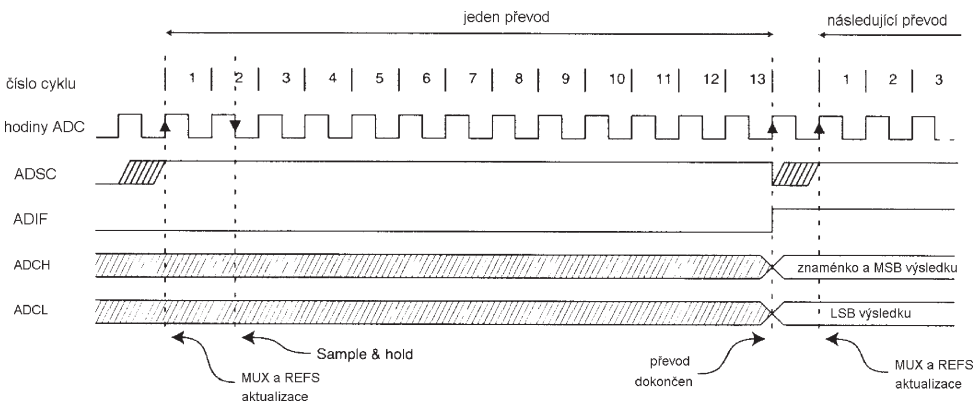
ADFR = 0

ADIE = 1

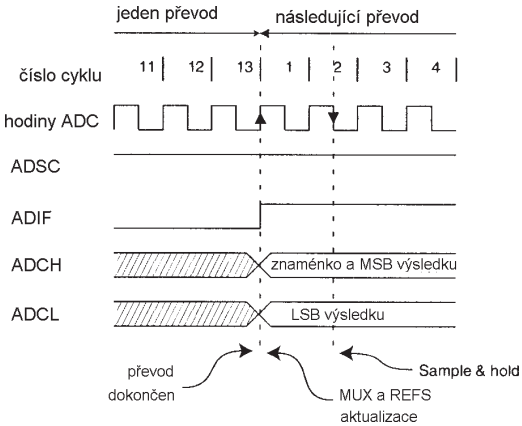
Nastaví se Idle mód, ADC spustí převod jakmile byl CPU zastaven.



Obr. 4.34 *ADC časový diagram, rozšířený převod (mód s jedním převodem)*



Obr. 4.35 *ADC časový diagram, jediný převod*



Obr. 4.36 *Časový diagram, volně běžící ADC*

Když nenastane další přerušení před dokončením ADC převodu, tak ADC přerušení ukončí sleep mód MCU a provede ADC Conversion Complete obsluhu přerušení.

ADC Multiplexer Select Register - ADMUX

registr výběru ADC multiplexeru

bit	7	6	5	4	3	2	1	0	
$07 ($27)	–	–	–	–	–	MUX2	MUX1	MUX0	ADMUX
čtení/zápis	R	R	R	R	R	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 4.37 *Registr výběru ADC multiplexu*

Bit 7 až 3 rezerva

Bit 2 až 0 MUX2 až MUX0: Analog Channel Select Bits 2-0

Hodnoty těchto tří bitů vybírají, který analogový vstup ADC7 až 0 je spojen s ADC. *Tab. 4.10* toto podrobně popisuje. Změní-li se tyto bity během převodu, změna se neprojeví do té doby, dokud není konverze dokončena (ADIF v ADCSR je nastaven).

Po skončení ADC převodu je výsledek umístěn v těchto dvou registrech. Při čtení ADCL se obsah datového registru nezmění, dokud je ADCH čten. Tudíž je třeba číst nejdříve ADCL a potom ADCH.

ADC 9 až 0 výsledek převodu

Tyto bity představují výsledek převodu. $000 představuje analogovou nulu, $3FF zvolené referenční napětí zmenšené o hodnotu odpovídající jednomu bitu (LSB).

Tab. 4.10 *Řízení připojení vstupů k ADC*

MUX2,0	Připojený vstup
000	ADC0
001	ADC1
010	ADC2
011	ADC3
100	ADC4
101	ADC5
110	ADC6
111	ADC7

ADC Control and Status Register - ADCSR
řídící a stavový registr ADC

bit	7	6	5	4	3	2	1	0	
$06 ($26)	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSR
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 4.38 *Řídicí a stavový registr ADC*

5

VNITŘNÍ PAMĚŤ EEPROM

Mnoho programů potřebuje uchovat některá data tak, aby zůstala zachována i při vypnutí přístroje, a aby bylo možné je kdykoliv přepsat. Tomuto požadavku vyhovují paměti EEPROM. Konstruktér má sice možnost přidat vnější paměť EEPROM, ale obsadí tím některé vstupní/výstupní piny MCU a jelikož jsou tyto paměti sériové, jsou pomalé a vyžadují velkou programovou podporu. Tyto problémy je možné vyřešit tím, že je potřebná paměť umístěna uvnitř vlastního MCU. Veškerá komunikace s vnitřní pamětí EEPROM se děje, stejně jako u ostatních periferií, které jsou mapované do oblasti I/O portů, tj. pomocí instrukcí IN a OUT. K přístupu k EEPROM je potřeba datový, adresový a řídicí registr, jako příklad uvedeme 512 bytovou EEPROM MCU AT90S8535:

EEPROM Address Register - EEARH a EEARL

adresový registr EEPROM

bit	15	14	13	12	11	10	9	8	
$1F ($3F)	–	–	–	–	–	–	–	EEAR9	EEARH
$1E ($3E)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
čtení/zápis	R	R	R	R	R	R	R	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	X	
	X	X	X	X	X	X	X	X	

Obr. 5.1 *Adresový registr EEPROM*

obsah EEPROM adresového registru (EEARH a EEARL) určuje adresu paměti EEPROM v adresovém prostoru EEPROM (např. u paměti 512 byte je to rozsah 0 až 511).

EEPROM Data Register - EEDR

EEPROM datový registr

bit	7	6	5	4	3	2	1	0	
$1D ($3D)	MSB							LSB	EEDR
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 5.2 *Datový registr EEPROM*

Bity 7 až 0, EEDR7 až 0, EEPROM Data

Při zápisu do EEPROM se do EEDR registru ukládají data, která se mají zapsat do EEPROM, při čtení z EEPROM obsahuje tento registr přečtená data z EEPROM.

EEPROM Control Register - EECR
řídící registr EEPROM

bit	7	6	5	4	3	2	1	0	
$1C ($3C)	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
čtení/zápis	R	R	R	R	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 5.3 *Řídicí registr EEPROM*

Bit 7 až 4 – rezerva

Bit 3 EERIE: EEPROM Ready Interrupt Enable

Pokud bity I jsou v SREG a EERIE nastaveny na jedničku, je EEPROM Ready Interrupt povolen. Je-li vynulován, přerušení je zakázáno.

Bit 2 – EEMWE: EEPROM Master Write Enable

Při nastavení bitu EEMWE na jedničku je možné zapisovat do paměti při EEWE nastaveném na jedničku. Je-li EEMWE nulový, nemá nastavení EEWE na jedničku žádný efekt. Po nastavení EEMWE na jedničku programem, je tento bit hardwarově vynulován po čtyřech hodinových cyklech.

Bit 1 – EEWE: EEPROM Write Enable

EEWE signál je strobovacím signálem pro zápis do EEPROM. Při zápisu do EEPROM se postupuje podle následující procedury:

- čeká se, než bude EEWE nulové,
- zapíše se nová EEPROM adresa do EEARL a EEARH,
- zapíší se nová data do EEDR,
- zapíše se jednička do bitu EEMWE v EECR (aby to bylo možné, musí být v témže cyklu zapsána nula do bitu EEWE),
- během čtyř hodinových cyklů po nastavení EEMWE, zapsat logickou jedničku do EEWE.

6

I/O PORTY

MCU řady AT90 v základní řadě mohou být vybaveny až čtyřmi 8bitovými obousměrnými branami PORT A, PORT B, PORT C a PORT D. U řady ATmega mohou být ještě PORT E a PORT F. Brány jsou mapovány každá na tři adresy – vstupní vývody, výstupní registr a směrový registr. Jelikož základní funkce těchto tří registrů je stejná u všech portů, bran, popíšeme si jen registry pro PORT A.

Port A Inputs Pins adresa (PINA) není registr, tato adresa umožňuje přístup

Port A Data Register - PORTA
datový registr PORTu A

bit	7	6	5	4	3	2	1	0	
$1B ($3B)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Port A Data Direction Register - DDRA
registr směru přenosu dat PORTu A

bit	7	6	5	4	3	2	1	0	
$1A ($3A)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Port A Input Pins Address - PINA
adresa vstupních pinů PORTu A

bit	7	6	5	4	3	2	1	0	
$19 ($39)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
čtení/zápis	R	R	R	R	R	R	R	R	
počáteční hodnota	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Obr. 6.1 *Registry PORTu A*

k fyzické hodnotě vlastního portu, na adrese vstupních/výstupních vývodů jsou přístupné skutečné úrovně na vývodech integrovaného obvodu, na adresu výstupního registru lze zapisovat výstupní data. Data zapsaná do směrového registru definují funkci příslušného vývodu. Vliv dat zapsaných do registru dat a směrového registru na chování vývodu shrnuje *tab. 6.1*.

Všechny bity v režimu výstupu mohou být v log. 0 zatíženy proudem až 20 mA.

Tab. 6.1 *Stav vývodu*

DDAn	PORTAn	I/O	Stav vývodu
0	0	Vstup	Velká impedance (otevřený kolektor v log.1)
0	1	Vstup	Připojen zatěžovací odpor, vstup může být zdrojem proudu
1	0	Výstup	Výstup v log.0, otevřený kolektor
1	1	výstup	Výstup v log.1, otevřený kolektor

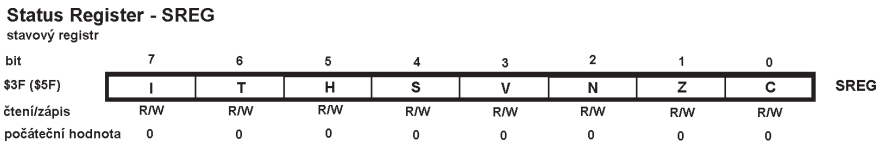
Přístup na I/O porty je základní dovednost nezbytně nutná pro jakoukoliv činnost MCU. AVR umožňují přístup bitový i bytový. Pro bitový přístup k portu se využívají dva registry. Pro čtení jednotlivých bitů se používají např. instrukce SBIC a SBIS (např. SBIS PINA, 4), pro zápis např. instrukce SBI a CBI (např. SBI PORTA, 0). Pro bytový přístup používáme instrukce IN a OUT (např. IN DATA, PINA).

Jednotlivé bity bran mohou mít přiřazeny alternativní funkce nastavením konfiguračních bitů v registrech příslušných periferií, popř. v kombinaci s nastavením bitů ve směrovém registru brány. Alternativní funkce jsou přiřazeny podle typu pouzdra a MCU, nastavení nutné pro výkon těchto funkcí je však u všech typů shodné.

Kromě vstupně/výstupních bran, portů, jsou součásti I/O prostoru i stavový registr, či registry související s prací vestavěných periferií. Proto si teď stručně popíšeme funkci zbývajících registrů I/O prostoru:

Stavový registr – SREG

AVR stavový registr zaujímá v I/O prostoru pozici $3F ($5F)



Obr. 6.2 *Stavový registr*

Bit 7 – Global Interrupt Enable – povolení všech přerušení

Tento bit musí být nastaven na jedničku, aby povoloval všechna přerušení. Povolení jednotlivých přerušení se provádí v příslušných oddělených řídicích registrech. Je-li bit I vynulován, jsou zakázána všechna přerušení, nezáleží přitom na případných povolení přerušení v individuálních řídicích registrech. Bit I je nulován hardwarově poté, co dojde k přerušení. Nastavován je instrukcí RETI.

Bit 6 – T: Bit Copy Storage, příznak Transfer Bit

Instrukce provádějící kopírování bitu BLD (BitLoaD) a BST (Bit Store) používají bit T jako zdrojový či cílový bit.

Bit 5 – H: Half-carry Flag

Přenos mezi třetím a čtvrtým bitem (využíván BCD aritmetikou).

Bit 4/S: Sign Bit, S = N ⊕ V

Tento bit je výlučným součtem (exclusive or) mezi příznakem záporného výsledku N a příznakem přetečení V, to znamená, že určuje znaménko výsledku.

Bit 3 – V: Two's Complement Overflow Flag

Příznak přetečení dvojkového doplňku.

Bit 2 – N: Negative Flag

Příznak záporného výsledku aritmetických a logických operací.

Bit 1 – Z: Zero Flag

Příznak Z indikuje nulový výsledek aritmetických nebo logických operací.

Bit 0 – C: Carry Flag

Příznak C indikuje přenos při aritmetických a logických operacích

Ukazatel na zásobník SP (Stack Pointer)

Ukazatel zásobníku je implementován jako dva 8bitové registry v I/O prostoru. Počet použitých bitů závisí na typu MCU. Uvedeme si SP pro AT90S8535

Stack Pointer SP
ukazatel na zásobník

bit	15	14	13	12	11	10	9	8	
$3E ($5E)	–	–	–	–	–	–	SP9	SP8	SPH
$3D ($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
	R	R	R	R	R	R	R	R	
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Obr. 6.3 *Ukazatel na zásobník*

Ukazatel SP ukazuje na zásobník umístěný v SRAM. Prostor zásobníku v SRAM musí být definován programem před jeho používáním voláním podprogramů či obsluhy přerušení. Musí být nastaven nad $60 (důvod je zřejmý z *obr. 1.2* – nižší adresy nepatří SRAM).

426410

General Interrupt Mask Register - GIMSK

registr maskování všech přerušení

bit	7	6	5	4	3	2	1	0	
$3B ($5B)	INT1	INT0	–	–	–	–	–	–	GIMSK
čtení/zápis	R/W	R/W	R	R	R	R	R	R	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 6.4 *Registr maskování všech přerušení*

Bit 7 – INT1: External Interrupt Request 1 Enable

Je-li bit INT1 nastaven na jedničku a současně je nastaven bit I ve stavovém registru SREG, je povoleno vnější přerušení 1. Bity 0 a 1 **Interrupt Sense Control1** (ISC11 a ISC10) v MCU obecném řídicím registru (MCUCR) definují, zda externí přerušení je aktivováno vzestupnou nebo sestupnou hranou signálu na pinu INT1 a nebo úrovní signálu. Odpovídající přerušení je obslouženo z programové paměti na adrese $002.

Bit 6 – INT0: External Interrupt Request 0Enable

Je-li bit INT0 nastaven na jedničku a současně je nastaven bit I ve stavovém registru SREG, je povoleno vnější přerušení 0. Bity 0 a 1 Interrupt Sense Control1 (ISC01 a ISC00) v MCU obecném řídicím registru (MCUCR) definují, zda externí přerušení je aktivováno vzestupnou nebo sestupnou hranou signálu na pinu INT0 a nebo úrovní signálu. Odpovídající přerušení je obslouženo z programové paměti na adrese $001.

General Interrupt Flag Register - GIFR

registr příznaků všech přerušení

bit	7	6	5	4	3	2	1	0	
$3A ($5A)	INTF1	INTF0	–	–	–	–	–	–	GIFR
čtení/zápis	R/W	R/W	R	R	R	R	R	R	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 6.5 *Registr příznaků všech přerušení*

Bit 7 – INTF1: External Interrupt Flag 1

Je příznak vnějšího přerušení. Pokud hrana nebo změna úrovně na pinu INT1 spustí požadavek na přerušení, INTF1 se nastaví na jedničku. Tento příznak je vynulován pokud pin je nakonfigurován pro přerušení nízkou úrovní, protože stav přerušení nízkou úrovní je určen čtením registru PIN.

Pokud je bit I v SREG nastaven, stejně jako bit INT1 v GIMSK, MCU provede skok na adresu přerušení $002. Tento příznak je shozen po provedení obslužné rutiny přerušení.

Bit 6 – INTF0 : External Interrupt Flag 0

Příznak vnějšího přerušení. V případě, že hrana nebo změna úrovně na pinu INT0 spustí požadavek na přerušení, INTF0 se nastaví na jedničku. Tento příznak je vynulován, pokud pin je nakonfigurován pro přerušení nízkou úrovní, jelikož stav přerušení nízkou úrovní je určen čtením registru PIN.

Je-li bit I v SREG nastaven, stejně jako bit INT0 v GIMSK, MCU provede skok na adresu přerušení $001. Tento příznak je shozen po provedení obslužné rutiny přerušení.

Timer/Counter Interrupt Mask Register - TIMSK
registr maskování přerušení čítače/časovače

bit	7	6	5	4	3	2	1	0	
$39 ($59)	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 6.6 *Registr maskování přerušení čítače/časovače*

Bit 7 – OCIE2:
Timer/Counter2 Output Compare Match Interrupt Enable

Je-li bit OCIE2 nastaven na jedničku a současně je nastaven na jedničku i bit I ve stavovém registru SREG, je povoleno přerušení Timer/Counter2 Output Compare Match. Odpovídající přerušení (na vektoru $003) nastane, pokud dojde k rovnosti obsahu *čítače/časovače 2* a obsahu porovnávaného registru (tj. když je nastaven bit OCF2 v TIFR).

Bit 6 – TOIE2: Timer/Counter2 Overflow Interrupt Enable

V případě, že je bit TOIE2 nastaven na jedničku a současně je nastaven na jedničku i bit I ve stavovém registru SREG, je povoleno přerušení Timer/Counter2 Overflow (přetečení *čítače/časovače 2*). Odpovídající přerušení (na vektoru $004) je provedeno, pokud dojde k přetečení v *čítači/časovači 2* (tj. když je nastaven bit TOV2 v TIFR).

Bit 5 – TICIE1: Timer/Counter1 Input Capture Interrupt Enable

Přerušení záchytného vstupu *čítače/časovače 1*. Je-li bit TICIE1 nastaven a současně je nastaven na jedničku i bit I ve stavovém registru SREG, je povoleno přerušení Timer/counter1 Input Capture Enble. Odpovídající přerušení (na vektoru $005) je provedeno, pokud dojde k události zachycení na pinu ICP (tj. když je bit ICF1 nastaven v TIFR).

Bit 4 – OCE1A:
Timer/counter1 Output CompareA Match Interrupt Enable

Je-li bit OCIE1A nastaven a současně je nastaven na jedničku i bit I ve stavovém registru SREG, je povoleno přerušení Timer/Counter1 CompareA Match. Odpovídající přerušení (na vektoru $006) je provedeno, nastane-li shoda čítače s příslušným registrem (tj. když je nastaven bit OCF1A v TIFR)

bit 3 – OCE1B:
Timer/counter1 Output CompareB Match Interrupt Enable

Je-li bit OCIE1B nastaven a současně je nastaven na jedničku i bit I ve stavovém registru SREG, je povoleno přerušení Timer/Counter1 CompareA Match. Odpovídající přerušení (na vektoru $007) je provedeno, nastane-li shoda čítače s příslušným registrem (tj. když je nastaven bit OCF1B v TIFR)

Bit 2 – TOIE1: Timer/counter1 Overflow Interrupt Enable

Je-li bit TOIE1 nastaven a současně je nastaven na jedničku i bit I ve stavovém registru SREG, je povoleno přerušení Timer/Counter1 Overflow. Odpovídající přerušení (na vektoru $008) je provedeno, nastane-li přetečení *čítače/časovače 1* (tj. když je nastaven bit TOIE1 v TIFR).

Bit 0 – TOIE0: Timer/counter0 Overflow Interrupt Enable

Je-li bit TOIE0 nastaven a současně je nastaven na jedničku i bit I v stavovém registru SREG, je povoleno přerušení Timer/Counter0 Overflow. Odpovídající přerušení (na vektoru $009) je provedeno, nastane-li přetečení *čítače/časovače 0* (tj. když je nastaven bit TOIE0 v TIFR).

Timer/Counter Interrupt Flag Register - TIFR
registr příznaků přerušení čítače/časovače

bit	7	6	5	4	3	2	1	0	
$38 ($58)	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	TIFR
čtení/zápis	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
počáteční hodnota	0	0	0	0	0	0	0	0	

Obr. 6.7 *Registr příznaků přerušení čítače/časovače*

Bit 7 – OCF2: Output Compare Flag 2

Bit OCF2 je nastaven, nastane-li shoda *čítače/časovače 2* s daty v OCR2 (Output Compare Register 2). OCF2 je vynulován hardwarově po provedení odpovídajícího přerušovacího vektoru. OCF2 je vymazán zapsáním logické jedničky do příznaku. Jsou-li nastaveny bity I v SREG, OCIE2 v TIMSK a OCF2 v TIFR, je Timer/Counter2 Compare Match Interrupt přerušení provedeno.

Bit 6 – TOV2: Timer/Counter2 Overflow Flag

Bit TOV2 je nastaven na jedničku, dojde-li k přetečení v *čítači/časovači 2*. TOV2 je vynulován hardwarově po provedení odpovídajícího vektoru přerušení. TOV2 je vymazán zapsáním logické 1 do příznaku. Jsou-li nastaveny bity I v SREG, TOIE2 v TIMSK a TOV2 v TIFR, je provedena obsluha přerušení přetečení *čítače/časovače 2*. Při práci v PWM módu je tento bit nastaven, když čítač/časovač začíná na $00.

Bit 5 – ICF1: Input Capture Flag 1

Bit ICF1 je nastaven na jedničku do příznaku události zachycení vstupu, indikující, že hodnota čítače/časovače 1 byla přenesena do ICR1. ICF1 je vynulován hardwarově po provedení odpovídající obsluhy přerušení. OCF1 je vynulován zapsáním logické 1 do příznaku. Jsou-li nastaveny na jedničku bity I v SREG, TICIE1 v TIMSK a ICF1 v TIFR, je provedena obsluha přerušení Timer/Counter1 Capture.

Bit 4 – OCF1A: Output Compare Flag 1A

Bit OCF1A je nastaven, nastane-li shoda *čítače/časovače 2* s daty v OCR1A (Output Compare Register 1A). OCF1A je vynulován hardwarově po provedení odpovídajícího přerušovacího vektoru. OCF1A je vymazán zapsáním logické jedničky do příznaku. Jsou-li nastaveny bity I v SREG, OCIE1A v TIMSK a OCF1A v TIFR, je Timer/Counter1 CompareA Match Interrupt přerušení provedeno.

Bit 3 – TOV1: Timer/Counter1 Overflow Flag

Bit TOV1 je nastaven na jedničku, dojde-li k přetečení v *čítači/časovači 1*. TOV1 je vynulován hardwarově po provedení odpovídajícího vektoru přerušení. TOV1 je vymazán zapsáním logické 1 do příznaku. Jsou-li nastaveny bity I v SREG, TOIE1 v TIMSK a TOV1 v TIFR, je provedena obsluha přerušení přetečení *čítače/časovače 1.* Při práci v PWM módu je tento bit nastaven, když čítač/časovač začíná na $0000.

Bit 0 – TOV0: Timer/Counter0 Overflow Flag

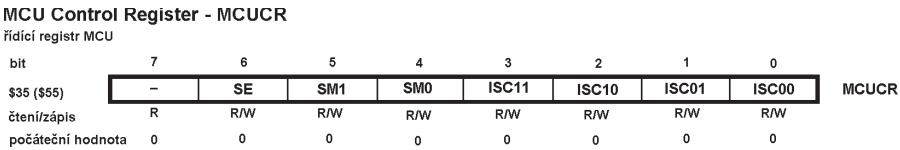
Bit TOV0 je nastaven na jedničku, dojde-li k přetečení v *čítači/časovači 1*. TOV0 je vynulován hardwarově po provedení odpovídajícího vektoru přerušení. TOV0 je vymazán zapsánín logické 1 do příznaku. Jsou-li nastaveny bity I v SREG, TOIE0 v TIMSK a TOV0 v TIFR, je provedena obsluha přerušení přetečení *čítače/časovače 0*.

Úsporný režim (sleep mode)

Ke zmenšení spotřeby má MCU možnost pracovat v jednom ze tří úsporných režimů, nastavitelných pomocí bitů SM1/SM0 řídicím registru MCUCR.

MCU Control Register – MCUCR

Tento registr obsahuje řídicí bity pro obecné MCU funkce.



Obr. 6.8 *Řídicí registr MCU*

Bit 7 – SE: Sleep Enable

Bit SE musí být nastaven na jedničku, aby bylo možné převést MCU do úsporného režimu po provedení SLEEP instrukce.

Bity 5, 4 – SM1/SM0: Slep Mode Select Bits 1 a 0

Tyto bity vybírají mezi třemi úspornými režimy (*tab. 6.2*):

Tab. 6.2 ***Úsporné režimy MCU***

SM1	SM0	Slep Mode
0	0	Idle
0	1	Rezerva (nepoužito)
1	0	Power-down
1	1	Power save

Bity 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 bits 1 and 0

Externí přerušení 1 je aktivováno externím pinem INT1, když příznak I v SREG a odpovídající maska přerušení v GIMSK je nastavena. Význam těchto dvou bitů je popsán v *tab. 6.3*.

Tab. 6.3 *Význam bitů ISC11 a ISC10*

ISC11	ISC10	Popis
0	0	Nízká úroveň na INT1 vyvolává požadavek na přerušení
0	1	Rezervováno
1	0	Sestupná hrana na INT1 vyvolává požadavek na přerušení
1	1	Vzestupná hrana na INT1 vyvolává požadavek na přerušení

Bity 1, 0 – ISC01, ISC00: Interrupt Sense Control 0 bits 1 and 0

Externí přerušení 0 je aktivováno externím pinem INT0, pokud příznak I v SREG a odpovídající maska přerušení v GIMSK je nastavena. Význam těchto dvou bitů je popsán v *tab. 6.4*.

Tab. 6.4 *Význam bitů ISC01 a ISC00*

ISC01	ISC00	Popis
0	0	Nízká úroveň na INT0 vyvolává požadavek na přerušení
0	1	Rezervováno
1	0	Sestupná hrana na INT0 vyvolává požadavek na přerušení
1	1	Vzestupná hrana na INT0 vyvolává požadavek na přerušení

Zatím jsme popisovali význam bitů registrů, které se zobrazují do I/O prostoru. To vše si nyní shrneme do *tab. 6.5* (pro typ 8535):

Tab. 6.5 *I/O prostor MCU AT90S8535*

I/O adresa (SRAM adresa)	Jméno	Funkce
$3F($5F)	SREG	Stavový registr
$3E($5E)	SPH	Ukazatel zásobníku – vyšší bity
$3D($5D)	SPL	Ukazatel zásobníku – nižší bity
$3B($5B)	GIMSK	Mask. registr vnějších přerušení
$3A($5A)	GIFR	Registr příznaků vnějších přerušení
$39($59)	TIMSK	Mask. registr přerušení čítačů/časovačů
$38($58)	TIFR	Registr příznaků přerušení čítačů/časovačů
$35($35)	MCUCR	MCU řídicí registr
$34($54)	MCUSR	MCU stavový registr
$33($53)	TCCR0	Řídicí registr *čítače/časovače 0*
$32($52)	TCNT0	*Čítač/časovač 0* (8 bitů)
$2F($4F)	TCCR1A	Řídicí registr *čítače/časovače 1A*
$2E($4E)	TCCR1B	Řídicí registr *čítače/časovače 1B*
$2D($4D)	TCNT1H	*Čítač/časovač 1* – vyšší bity
$2C($4C)	TCNT1L	*Čítač/časovač 1* – nižší bity
$2B($4B)	OCR1AH	Porovnávací registr *čítače/časovače 1A* – vyšší bity
$2A($4A)	OCR1AL	Porovnávací registr *čítače/časovače 1A* – nižší bity
$29($49)	OCR1BH	Porovnávací registr *čítače/časovače 1B* – vyšší bity
$28($48)	OCR1BL	Porovnávací registr *čítače/časovače 1B* – nižší bity
$27($47)	ICR1H	Záchytný registr *čítače/časovače 1* – vyšší bity
$26($46)	ICR1L	Záchytný registr *čítače/časovače 1* – nižší bity
$25($45)	TCCR2	Řídicí registr *čítače/časovače 2*
$24($44)	TCNT2	*Čítač/časovač 2* (8 bitů)
$23($43)	OCR2	Porovnávací registr *čítače/časovače 2*
$22($42)	ASSR	Stavový registr asynchronního módu
$21($41)	WDTCR	Řídicí registr časovače Watchdog

***Tab. 6.5** (pokračování) I/O prostor MCU AT90S8535*

$1F($3F)	EEARH	Adresový registr EEPROM – vyšší bity
$1E($3E)	EEARL	Adresový registr EEPROM – nižší bity
$1D($3D)	EEDR	Datový registr EEPROM
$1C($3C)	EECR	Řídicí registr EEPROM
$1B($3B)	PORTA	Datový registr, port A
$1A($3a)	DDRA	Směrový registr, port A
$19($39)	PINA	Vývody portu A
$18($38)	PORTB	Datový registr, port B
$17($37)	DDRB	Směrový registr, port B
$16($36)	PINB	Vývody portu B
$15($35)	PORTC	Datový registr, port C
$14($34)	DDRC	Směrový registr, port C
$13($33)	PINC	Vývody portu C
$12($32)	PORTD	Datový registr, port D
$11($31)	DDRD	Směrový registr, port D
$10($30)	PIND	Vývody portu D
$0F($2F)	SPDR	Datový I/O registr SPI
$0E($2E)	SPSR	Stavový registr SPI
$0D($2D)	SPCR	Řídicí registr SPI
$0C($2C)	UDR	I/O datový registr UART
$0B($2B)	USR	Stavový registr UART
$0A($2A)	UCR	Řídicí registr UART
$09($29)	UBRR	Registr rychlostí UART
$08($28)	ACSR	Řídicí a stavový registr analogového komparátoru
$07($27)	ADMUX	ADC registr výběru multiplexeru
$06($26)	ADCSR	ADC řídicí a stavový registr
$05($25)	ADCH	ADC datový registr – vyšší bity
$04($24)	ADCL	ADC datový registr – nižší bity

Tab. 6.5 tedy popisuje I/O prostor nejlépe vybaveného AVR MCU základní řady – AT90S8535. Pro typy mající méně periferií je odpovídající tabulka příslušně zredukována, obsahuje jen řádky popisující periferie zabudované v příslušném typu MCU. Ve většině případů jsou adresy v I/O prostoru nezávislé na typu MCU. Např. datový registru PORTu B je vždy na adrese $18, obdobně pro PORT A, PORT C, PORT D (jsou-li ovšem součástí příslušného typu MCU). Nezávislost adresy I/O registrů na typu MCU není bohužel stoprocentní. Našel jsem čtyři výjimky (ICR1H, ICR1L, UCSRA a UCSRB) porušující nezávislost adresy I/O registru na typu. Tato skutečnost je jen vadou na kráse elegantního návrhu řady AT90, které si v praxi vůbec nevšimneme. Téměř vždy při tvorbě programu pro MCU AVR bude zdrojový kód obsahovat hlavičkový soubor pro příslušný typ MCU obsahující definice konstant jako PORTA, DDRB. Pro programování v assembleru poskytuje tyto hlavičkové soubory (jako např. 8515def.inc) přímo výrobce AVR firma ATMEL. Jména konstant přitom jsou zvolena stejně jako jména odpovídajících registrů v dokumentaci k MCU ATMEL. Pro úplnost si ještě uvedeme tabulku adres I/O portů pro již zmiňovanou šestici AVR MCU základní řady (*tab. 6.6)*.

426410

Tab. 6.6 *I/O adresy AVR MCU základní řady*

I/O adresa (SRAM adresa)	1200	2313	2343	4433	8515	8535
$3F($5F)	SREG	SREG	SREG	SREG	SREG	SREG
$3E($5E)	-	-	-	-	SPH	SPH
$3D($5D)	-	SPL	SPL	SPL	SPL	SPL
$3B($5B)	GIMSK	GIMSK	GIMSK	GIMSK	GIMSK	GIMSK
$3A($5A)	-	GIFR	GIFR	GIFR	GIFR	GIFR
$39($59)	TIMSK	TIMSK	TIMSK	-	TIMSK	TIMSK
$38($58)	TIFR	TIFR	TIFR	-	TIFR	TIFR
$35($35)	MCUCR	MCUCR	MCUCR	MCUCR	MCUCR	MCUCR
$34($54)	-	-	MCUSR	MCUSR	-	MCUSR
$33($53)	TCCR0	TCCR0	TCCR0	TCCR0	TCCR0	TCCR0
$32($52)	TCNT0	TCNT0	TCNT0	TCNT0	TCNT0	TCNT0
$2F($4F)	-	TCCR1A	-	TCCR1A	TCCR1A	TCCR1A
$2E($4E)	-	TCCR1B	-	TCCR1B	TCCR1B	TCCR1B
$2D($4D)	-	TCNT1H	-	TCNT1H	TCNT1H	TCNT1H
$2C($4C)	-	TCNT1L	-	TCNT1L	TCNT1L	TCNT1L
$2B($4B)	-	OCR1AH	-	OCR1AH	OCR1AH	OCR1AH
$2A($4A)	-	OCR1AL	-	OCR1AL	OCR1AL	OCR1AL
$29($49)	-	-	-	-	OCR1BH	OCR1BH
$28($48)	-	-	-	-	OCR1BL	OCR1BL
$27($47)	-	-	-	ICR1H	-	ICR1H
$26($46)	-	-	-	ICR1L	-	ICR1L
$25($45)	-	ICR1H	-	-	ICR1H	TCCR2
$24($44)	-	ICR1L	-	-	ICR1L	TCNT2
$23($43)	-	-	-	-	-	OCR2
$22($42)	-	-	-	-	-	ASSR
$21($41)	WDTCR	WDTCR	WDTCR	WDTCR	WDTCR	WDTCR
$1F($3F)	-	-	-	-	EEARH	EEARH
$1E($3E)	EEARL	EEARL	EEARL	EEARL	EEARL	EEARL
$1D($3D)	EEDR	EEDR	EEDR	EEDR	EEDR	EEDR
$1C($3C)	EECR	EECR	EECR	EECR	EECR	EECR
$1B($3B)	-	-	-	-	PORTA	PORTA
$1A($3a)	-	-	-	-	DDRA	DDRA
$19($39)	-	-	-	-	PINA	PINA
$18($38)	PORTB	PORTB	PORTB	PORTB	PORTB	PORTB
$17($37)	DDRB	DDRB	DDRB	DDRB	DDRB	DDRB
$16($36)	PINB	PINB	PINB	PINB	PINB	PINB
$15($35)	-	-	-	PORTC	PORTC	PORTC
$14($34)	-	-	-	DDRC	DDRC	DDRC
$13($33)	-	-	-	PINC	PINC	PINC
$12($32)	PORTD	PORTD	-	PORTD	PORTD	PORTD
$11($31)	DDRD	DDRD	-	DDRD	DDRD	DDRD
$10($30)	PIND	PIND	-	PIND	PIND	PIND
$0F($2F)	-	-	-	SPDR	SPDR	SPDR
$0E($2E)	-	-	-	SPSR	SPSR	SPSR
$0D($2D)	-	-	-	SPCR	SPCR	SPCR
$0C($2C)	-	-	-	UDR	UDR	UDR
$0B($2B)	-	USR	-	UCSRA	USR	USR

***Tab. 6.6** (pokračování) I/O adresy AVR MCU základní řady*

$0A($2A)	-	UCR	-	UCSRBk	UCR	UCR
$09($29)	-	UBRR	-	UBRR	UBRR	UBRR
$08($28)	ACSR	ACSR	-	ACSR	ACSR	ACSR
$07($27)	-	-	-	ADMUX	-	ADMUX
$06($26)	-	-	-	ADCSR	-	ADCSR
$05($25)	-	-	-	ADCH	-	ADCH
$04($24)	-	-	-	ADCL	-	ADCL
$03($23)	-	-	-	UBRRHI	-	-

7

PŘERUŠOVACÍ SYSTÉM

Řadiče AT90S mohou mít v základní řadě až 16 zdrojů přerušení (v řadě ATmega až 34). Adresy jejich vektorů přerušení jsou uspořádány za sebou od počátku paměťového systému (na nejnižší adrese je RESET systému) a jejich pořadí odpovídá i jejich priorita, s níž jsou vykonávána. Přehled přerušení a jejich vektorů ve v *tab. 7.1* (příklad pro typ AT90S8535 ze základní řady):

Tab. 7.1 *Vektory přerušení MCU AT90S8535*

Vektor č.	Adresa programu	Zdroj	Popis
1	$000	RESET	Inicializace systému
2	$001	INT0	Vnější přerušení 0
3	$002	INT1	Vnější přerušení 1
4	$003	TIMER2 COMP	Shoda čítače 2 s OCR2
5	$004	TIMER2 OVF	Přetečení *čítače/časovače 2*
6	$005	TIMER1 CAPT	Vnější událost-zachycení obsahu čítače 1
7	$006	TIMER1 COMPA	Shoda čítače 1 s OCR1A
8	$007	TIMER1 COMPB	Shoda čítače 1 s OCR1B
9	$008	TIMER1 OVF	Přetečení *čítače/časovače 1*
10	$009	TIMER0 OVF	Přetečení *čítače/časovače 0*
11	$00A	SPI, STC	Sériový přenos dokončen
12	$00B	UART, RX	UART, příjem dokončen
13	$00C	UART, UDRE	UART, prázdný registr dat
14	$00D	UART, TX	UART, vysílání dokončeno
15	$00E	ADC	ADC převod dokončen
16	$00F	EE_RDY	EEPROM připravena
17	$010	ANA_COMP	Analogový komparátor

Každé přerušení je povoleno, je-li součin bitu I ve stavovém registru SREG a bitu povolení přerušení v řídicím registru příslušné periferie, která má být jeho zdrojem, roven log.1.

Pro nezávislé přerušovací vstupy INT0 a INR1 jsou povolovací bity INT0 a INT1 umístěny v registru GIMSK. Příslušný vývod musí být ovšem nastaven jako vstupní (vynulováním svého bitu ve směrovém registru brány, ve které se nachází). Druh události na vstupech, při které dojde k přerušení je kódován bity ICS00, ICS01, popř. ICS10, ICS11 v registru MCUSR. Bity umožňující přerušení od čítačů/časovačů se nacházejí v registru TIMSK. Bit TOIE1 povoluje přerušení při přetečení čítače 1. Současně se nastaví bit TOV1 v registru návěští TIFR.

426410

Bity OCE1A a OCE1B povolují přerušení při rovnosti obsahu čítače 1 s porovnávanými registry. Přitom jsou nahozeny i bity OCF1A, popř. OCF1B v TIFR. Bit TICIE1 umožňuje přerušení při události spouštějící zachycení obsahu čítače 1. Zároveň je nastaveno návěští ICF1 v TIFR.

Bity TOIE0, OCIE0, TOV0 a OCF0 jsou obdobou bitů TOIE1, OCIE1,TOV1 a OCF1 pro čítač 0. Analogicky pro čítač 2, je-li součástí MCU.

Každé z návěští je po spuštění obslužného podprogramu přerušení nulováno hardwarově.

Odezva na přerušení trvá tři takty oscilátoru, návrat z obslužné rutiny je stejně dlouhý. Do zásobníku je ukládána pouze jeho návratová adresa, vše ostatní je nutno v případě potřeby obsloužit softwarově. Řadiče MCU mají zásobník adresovaný osmi a více bity v registrech ukazatele (v závislosti na typu MCU, jeho SRAM; např. typ AT90S2313 má zásobník adresovaný 8 bity, typ AT90S8515 16 bity) a typ AT90S1200 má pouze hardwarově obsluhovaný zásobník pro tři návratové adresy.

Kapitolku o přerušení zakončíme jednou velmi důležitou poznámkou:

V tabulce vektorů přerušení pro typ AT908535 (*tab. 7.2)* si povšimneme toho, že adresy vektorů přerušení následují za sebou. V případě, kdy použijeme jiný typ AVR MCU, který oproti typu 8535 např. bude mít méně periferií, budou adresy vektorů přerušení opět tvořit souvislou řadu, takže **stejný vektor přerušení u různých typů AVR často mívá jinou adresu**. Pro dostupné typy AVR základní řady si této skutečnosti můžeme všimnout v *tab. 7.2* adres vektorů přerušení.

Tab. 7.2 *Vektory přerušení MCU AVR základní řady*

	1200	**2313**	**2343**	**4433**	**8515**	**8535**
RESET0	$000	$000	$000	$000	$000	$000
INT0	$001	$001	$001	$001	$001	$001
INT1	-	$002	-	$002	$002	$002
TIMER2COMP	-	-	-	-	-	$003
TIMER2OVF	-	-	-	-	-	$004
TIMER1CAPT	-	$003	-	$003	$003	$005
TIMER1COMPA	-	-	-	-	$004	$006
TIMER1COMPB	-	-	-	-	$005	$007
TIMER1COMP	-	$004	-	$004	-	-
TIMER1OVF	-	$005	-	$005	$006	$008
TIMER0OVF	$002	$006	$002	$006	$007	$009
SPI,STC	-	-	-	$007	$008	$00A
UART RX	-	$007	-	$008	$009	$00B
UART UDRE	-	$008	-	$009	$00A	$00C
UART TX	-	$009	-	$00A	$00B	$00D
ADC	-	-	-	$00B	-	$00E
EE_RDY	-	-	-	$00C	-	$00F
ANA_COMP	$003	$00A	-	$00D	$00C	$010

Instrukční soubor MCU AVR

Soubor instrukcí RISCového AVR mikroprocesoru obsahuje, v závislosti na typu MCU, 89 až 130 instrukcí. Jejich seznamy, podrobný popis jejich činnosti je uveden v příloze.

8

PŘÍKLADY KONKRÉTNÍCH AVR MCU

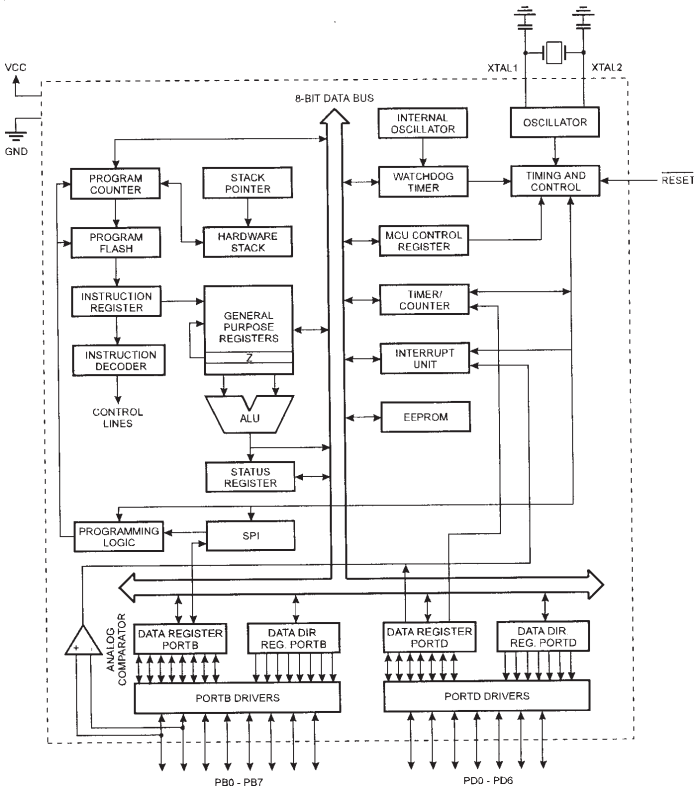
Již jsme provedli výčet všech AVR MCU z rodiny AT90 v současné době vyráběných v základní řadě, v řadě ATtiny a v řadě ATmega. Jednotlivé typy se liší svým provedením, kapacitou pamětí, vybavením periferií. Je zbytečné podrobně si popisovat všechny typy, když některé jsou téměř stejné a liší se jen tím, že jeden typ má třeba 4K flash programové paměti a druhý 8K paměti Flash. Proto se uvedeme jen čtyři typické typy ze základní řady, dostupné v maloobchodní síti (v červnu 2002 v Praze). V praktické části této publikace bude uveden ISP programátor. Jednodeskové počítače s těmito MCU budou uvedeny v dalších publikacích nakladatelství BEN – technická literatura.

AT90S1200

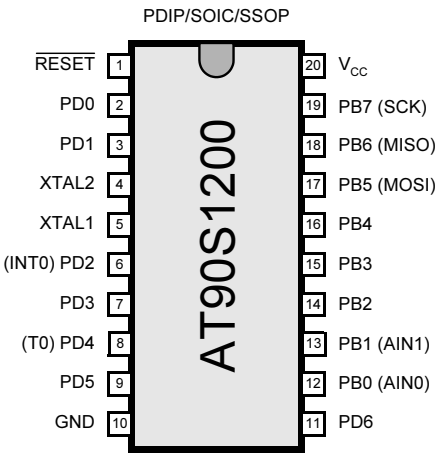
Je to nejmenší zástupce základní řady. Má velmi malou paměť složenou jen z 32 pracovních registrů. Paměť SRAM, běžná u dalších typů, mu zcela chybí. Má 1 Kbyte flash paměť pro program a 64bitovou paměť EEPROM. To, že jde opravdu co do schopností o nejmenšího zástupce, můžeme usoudit i z toho, že byl implementován pouze tříúrovňový hardwarový zásobník návratových adres SP, a též podle toho, že byl implementován pouze jeden registr (registr Z) pro nepřímé adresování a podle toho, že má pouze 89 instrukcí. Pro mnoho jednoduchých aplikací je požadavkem co nejmenší počet součástí. Protože je tento MCU směřován právě do této oblasti, najdeme na čipu MCU implementován obvod interního oscilátoru s kmitočtem 1 MHz, se kterým můžeme v mnoha aplikacích vystačit. Při požití vnějšího krystalu je kmitočet 0 až 4 MHz u typu AT90S1200-4, popř. 0 až 12 MHz u AT90S1200-12.

MCU AT90S1200 je vyráběn v 20pinovém pouzdru PDIP/SOIC/SSOP, *obr. 8.2.*

I tento nejjednodušší typ AVR MCU má možnost sériového programování programové paměti Flash i paměti EEPROM přes rozhraní SPI, které vystačí s třemi piny portu B (PB5, PB6 a PB7).

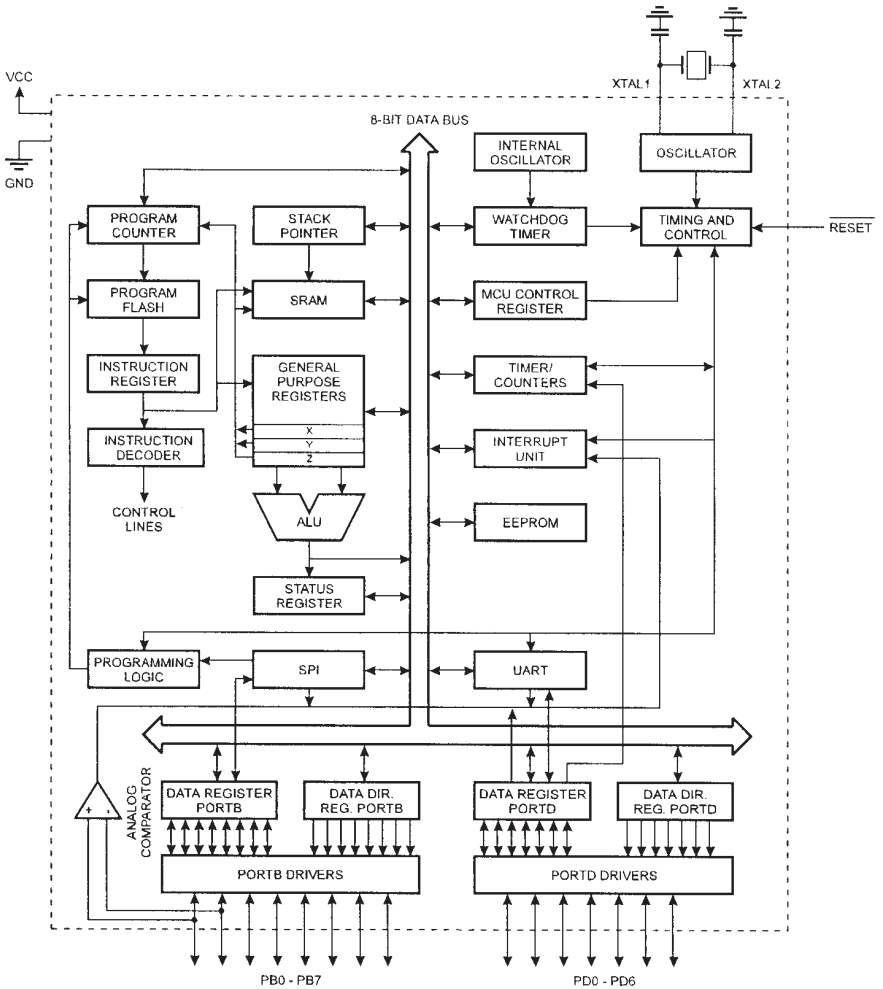


Obr. 8.1 *Blokové zapojení MCU AT90S1200*



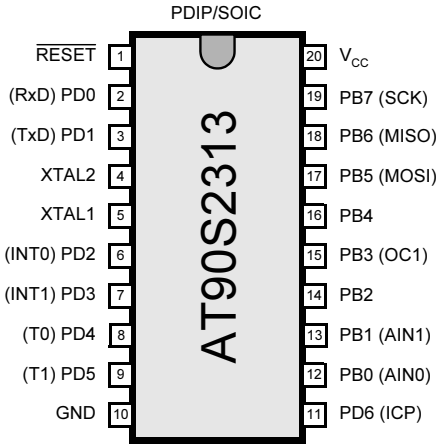
Obr. 8.2 *Zapojení vývodů AT90S1200*

Tento MCU má 2 Kbytů programové paměti flash, 128 byte SRAM a 128 byte EEPROM. Tento typ má již tři registry pro nepřímé adresování (X, Y a Z). Každý z těchto registrů je tvořen dvojicí 8bitových registrů. S ohledem na velikost paměti tohoto typu MCU by se zdálo, že 2bytové registry jsou zde zbytečné. Důvodem jejich použití je programová kompatibilita s většími MCU této rodiny. Stejně jako tyto větší MCU, má i AT90S2313 soubor 118 instrukcí. Vnější krystal může být v rozmezí 0 až 4 MHz u typu AT90S2313-4 a 0 až 10 MHz u typu AT90S2313-10.



Obr. 8.3 *Blokové schéma MCU AT90S2313*

Rovněž tento typ má 20pinové pouzdro, *obr. 8.4*



Obr. 8.4 *Zapojení vývodů AT90S2313*

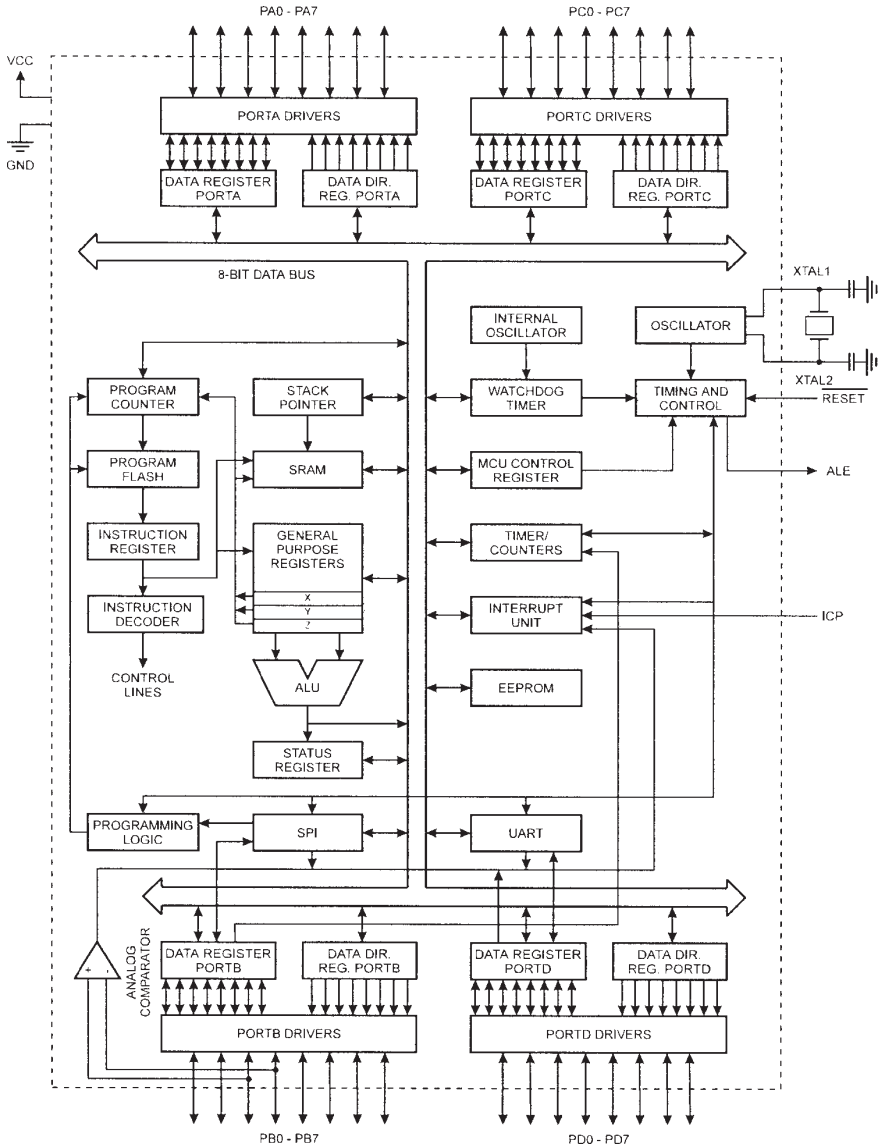
Zbývající dva typy, se kterými se seznámíme, mají již 40 PDIP, popř. 44 pin PLCC nebo TQFR.

AT90S8515

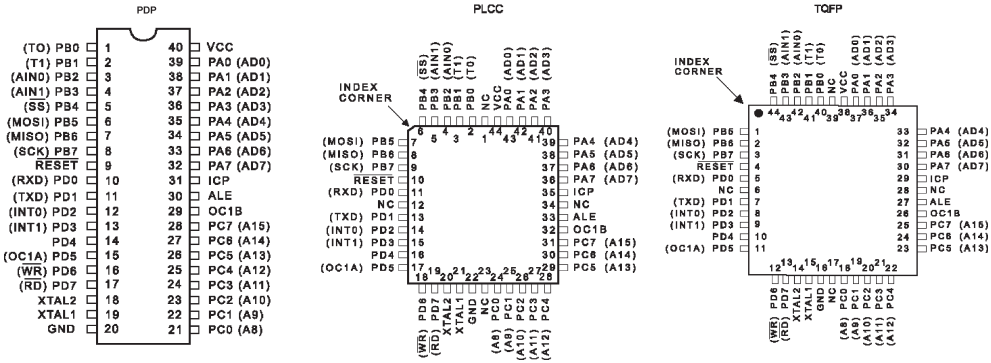
Tento MCU má 8 Kbytů programové paměti flash, 512 byte SRAM a 512 byte EEPROM. Dále obsahuje jeden 8bitový čítač/časovač a jeden 16bitový čítač/časovač. Dále obsahuje jeden analogový komparátor, UART, Watchdog, kmitočet krystalu 0 až 4 MHz u typu AT90S8515-4, a 8 MHz u typu AT90S8515-8. Blokové schéma ukazuje *obr. 8.5* a zapojení pouzdra *obr. 8.6.*

RISCový obvod AT90S8515 je vývodově kompatibilní s CISCovým obvodem AT89S8252 s jádrem x51 a to včetně signálů MOSI, MISO a SCK pro rozhraní SPI umožňující sériové programování paměti flash pro program, kterou ATMEL používá i v CISCovém AT89S8252. Jediné, čím se (ovšem kromě jádra) při pohledu z vnějšku liší je úroveň RESET signálu. Praktický důsledek je možnost navrhnout pro oba typy, RISCový i CISCový, identický plošný spoj. Oba typy lze pak programovat stejným programátorem ISP a i pro typ AT89S8252 s jádrem x51 používat při programování ATMEL sw **AVR studio**, který je však navržen především pro práci s RISCovými AVR MCU.

S tímto MCU je vývodově kompatibilní i další MCU z řady AT90 a to AT90S8535, který oproti typu AT90S8515 obsahuje navíc i A/D převodník a ještě jeden čítač/časovač. Na druhé straně nemá možnost připojení externích pamětí (nemá vyvedené sběrnice).



Obr. 8.5 *Blokové schéma AT90S8515*



Obr. 8.6 *Zapojení vývodů AT90S8515*

AT90S8535

Tento MCU obsahuje 8 Kbyte paměť flash pro program, 512 byte SRAM i 512 byte EEPROM. Dále obsahuje UART, dva 8bitové čítače/časovače, jeden 16bitový čítač/časovač, a hlavně 8kanálový, 10bitový A/D převodník. Blokové schéma tohoto MCU ukazuje *obr. 8.7* a zapojení pouzdra *obr. 8.8.*

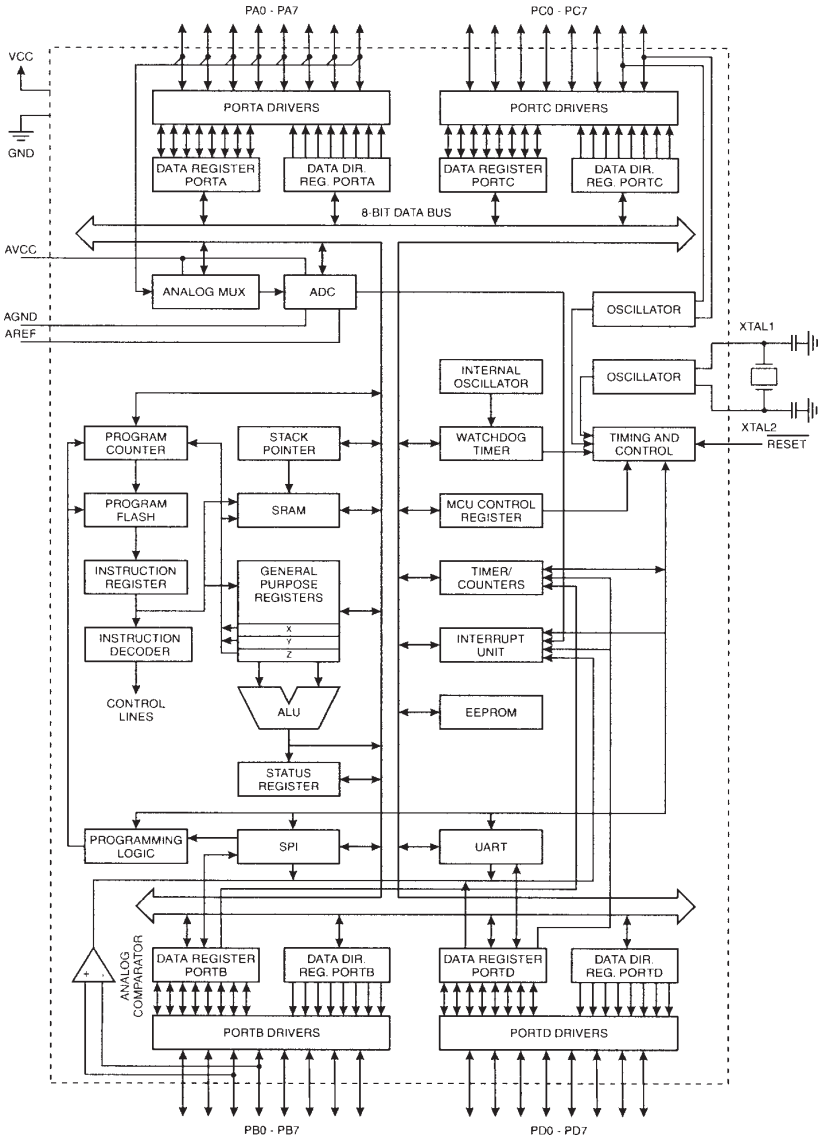
Programování pamětí na čipu MCU

Čipy lze programovat dvojím způsobem. Paralelní programování je shodné např. s programováním čipů řady x51 se všemi nevýhodami s tím souvisejícími. Řada AT90 a kromě toho i dva čipy z rodiny x51 AT89S8252 a AT89S53 lze programovat sériově, přímo v zapojení, ve kterém se pak bude provozovat aplikace. Vžila se pro toto zkratka ISP – In Systém Programming. Paměť programu i dat je přístupná po bytech od adresy 0000 až po svoji jmenovitou velikost (1, 2, 4 a 8 Kbyte paměť programu a 128, 256 či 512 byte paměť dat). K vlastnímu programování slouží pouhé tři signály Serial ClocK (SCK), Master In-Slave Out (MISO) a Master Out – Slave In (MOSI). Kmitočet signálu přiváděného na vývod SCK smí být nejvýše 1/40 kmitočtu hodin procesoru. Ten musí být nejméně 1 MHz a nejvýše 10 či 20 MHz v závislosti na typu MCU.

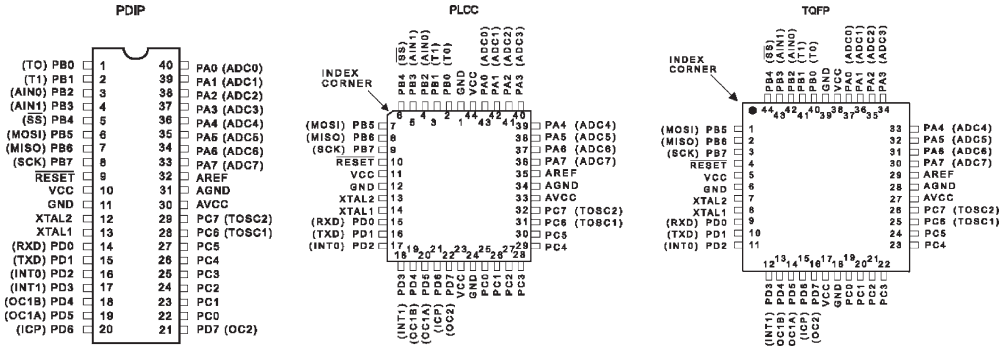
Programovací sekvence je pak následující:

- Připojit napájecí napětí a uvést RESET do log.1 a vyčkat nejméně 10 ms.
- Na vývod MOSI vyslat instrukci povolující sériový zápis do paměti.
- Vysláním příslušně formované instrukce zápisu naprogramovat 1 byte.

426410



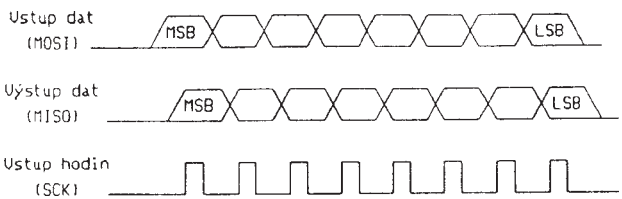
Obr. 8.7 *Blokové schéma AT90S8535*



Obr. 8.8 *Zapojení vývodů AT90S8535*

- Zápis lze verifikovat zasláním instrukce pro čtení, která na vývodu MISO vrátí obsah žádané adresy.
- Po poklesu RESET do log.0 se MCU vrátí k běžné činnosti.

Sériové programování lze znemožnit naprogramováním pojistky v paralelním módu. Časové průběhy signálu při programování jsou na *obr. 8.9.*



Obr. 8.9 *Časové průběhy při programování ISP*

Soubor instrukcí pro sériové programování je uveden v *tab. 8.1*.

Tab. 8.1 *Instrukce pro ISP*

Instrukce	Byte 1	Byte 2	Byte 3	Byte 4	Operace
Programování povoleno	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Povoluje sériové programování po /RESET = 0
Vymazání obvodu	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Vymazání celé Flash i EEPROM i lock bitů
Čtení programové Flash paměti	0010 H000	0000 aaaa	bbbb bbbb	oooo oooo	Čte data o z adresy a:b paměti programu
Zápis do Flash paměti	0110 H000	0000 aaaa	bbbb bbbb	iiii iiii	Zapisuje data o z adresy a:b paměti programu

***Tab. 8.1** (pokračování)*

Čtení EEPROM	1010 0000	0000 0000	bbbb bbbb	oooo oooo	Čte data o z adresy a:b paměti dat
Zápis do EEPROM	1110 0000	0000 0000	bbbb bbbb	i i i i i i i i	Zapisuje data o z adresy a:b paměti dat
Uzamčení paměti	1010 1100	111x x*21*x	xxxx xxxx	xxxx xxxx	Zapsání zamykacích bitů, nastavení *1,2* na ‚0' (lock bity)
Čtení kódu zařízení	0011 0000	xxxx xxxx	0000 00bb	oooo oooo	Čtení kódu zařízení

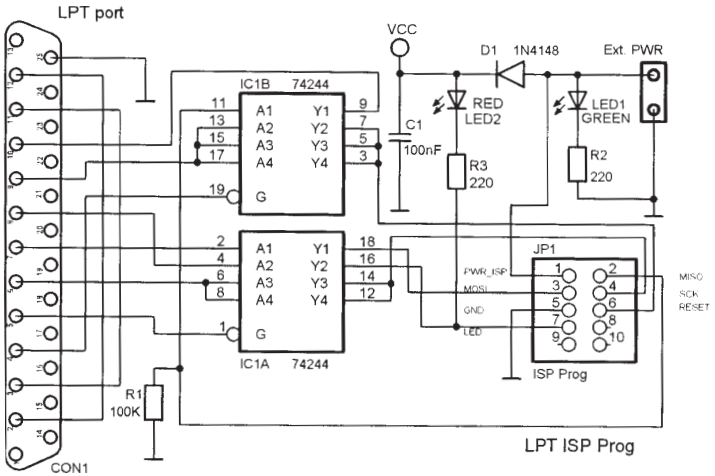
Poznámka:
a = vyšší bity adresy,
b = nižší bity adresy,
H = 0 – dolní byte, 1 – horní byte,
o = výstupní data,
i = vstupní data,
x = úroveň není určena.

Příklad probíhající komunikace při sériovém programování, význam kódu zařízení a další podrobnosti uveřejnil ATMEL v aplikační poznámce AVR910, kterou s dalšími aplikačními poznámkami, podrobným popisem všech MCU vyráběných firmou ATMEL, software atd. se dá stáhnout z firemní stránky http://www.atmel.com nebo z firemního CD. To, že ATMEL uveřejnil veškeré informace o ISP řady AT90 umožnilo vytvořit řadě konstruktérů, profesionálních či amatérských, velký počet různých programátorů AVR a napsat příslušný obslužný software. Tyto konstrukce jsou většinou volně přístupné na Internetu.

Většinou jde spíše než o programátor o jednoduché interface, umožňující programování pomocí sériového či paralelního portu PC. V řadě případů jde o pouhé připojení k LPT portu přes několik odporů, či k COM portu rovněž přes několik odporů s připojenými zenerovými diodami na napětí cca 5 V. V lepším případě jde o interface s obvodem 7414 či 74244. Pro správnou funkci ISP je však potřeba zaručit poměrně přísné podmínky týkající se strmosti hran signálu, impedance vedení atd. Proto se může stát, že některé zapojení může dobře pracovat na jednom PC a na druhém nikoli, či být závislé na konstrukčním provedení, velikosti zatěžovacích či zdvihacích (pull-up) odporů atd. Propojení se pak např. doporučuje kroucenou dvoulinkou apod. Řešení těchto problémů se věnují i různé diskuzní skupiny na Internetu. Přesto je použití takovýchto programátorů velice rozšířené. Jako velice dobrý obslužný program lze uvést Pony Prog pracující jak pod Windows, tak existuje i verze pro Linux.

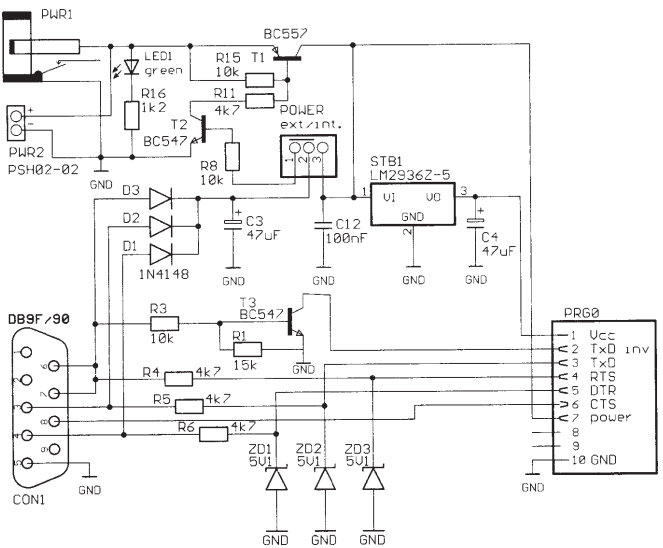
Pro úplnost si ještě uvedeme zapojení programátorů – interface používané pro spolupráci s programem Pony Prog. Na *obr. 8.10* je zapojení programátoru AVR MCU, který připojujeme k paralelnímu LPT portu. Zapojení tohoto programátoru je odvozeno od programátoru firmy KANDA Systems, který se používá pro programování AVR MCU pracujícího na start kitu STK200 této firmy. Schéma STK200 najdete

na CD příloze ke knize AVR Assembler, stejně jako aplikační listy firmy ATMEL s programy pro STK200. Lze tedy programátor z *obr. 8.10* používat nejen ve spojení s Pony Prog, ale i se sw podporujícím programátor STK200. Jsou to např. nižší verze AVR studia, BasCom či CodeVisionAVR překladač C.



Obr. 8.10 *Programátor AVR MCU kompatibilní s STK200 programátorem*

Obr. 8.11 ukazuje schéma programátoru, který připojujeme k sériovému COM portu a který je podporován Pony Prog.

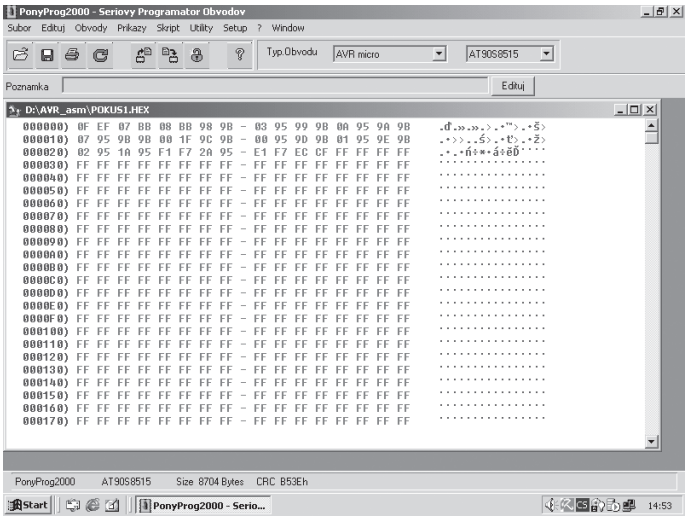


Obr. 8.11 *Programátor AVR ovládaný přes sériový port PC*

426410

Zapojení obou programátorů (*obr. 8.10* a *obr. 8.11*) jsou převzata z Internetu, kde najdete i několik verzí obrazců plošných spojů. Najdete je např. na domovské stránce Pony Prog či na českém hw serveru http://www.hw.cz.

Zmíněný Pony Prog ukazuje *obr. 8.12.*



Obr. 8.12 *Pony Prog*

Bližší informace o tomto sw, včetně možnosti stažení najdete na http://www.LancOS.com

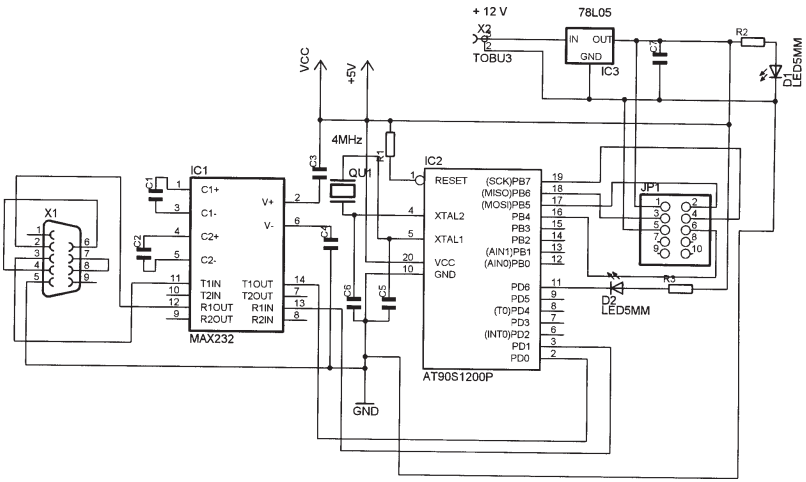
AVR studio

Pokud budeme používat pro programování AVR MCU programátory a software, o němž jsme se zmínili v předchozí kapitole, budeme muset nejdříve pomocí jiného software, assembleru, jazyka C, Basicu atd. napsat zdrojový kód aplikace. Poté spustíme příslušný překladač, který vygeneruje binární či hex soubor, který potom použijeme jako vstupní pro obslužný program nějakého programátoru, jako je např. zmíněný Pony Prog. Jiný program použijeme při odlaďování a zase jiný při simulaci či emulaci AVR MCU. Používáme přitom řadu různých programů, s rozdílnými prostředími. Je to obdoba tvorby software pro „velké“ počítače včetně PC v počátcích jejich existence. Požadavek na zvýšení efektivity práce programátorů těchto počítačů vedl k zlepšování programátorských nástrojů, mj. je dnes při psaní programů pro „velké“ počítače běžné používat **integrované programátorské prostředí**.

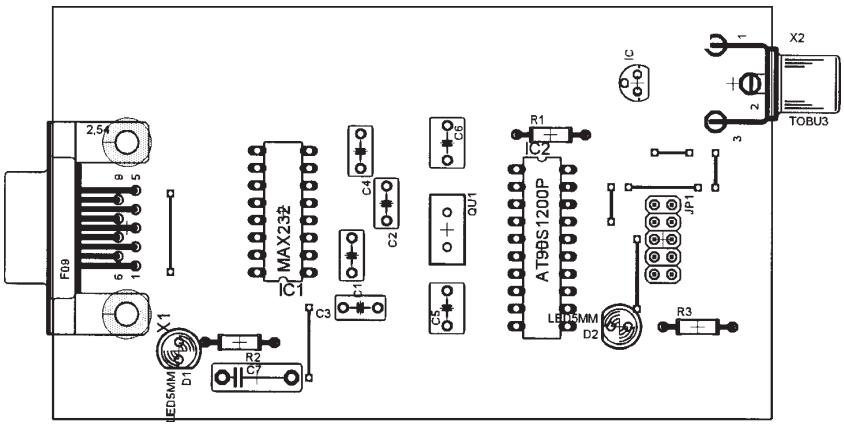
Takovéto prostředí existuje i pro programování MCU AVR a poskytuje ho zdarma přímo výrobce mikrokontrolérů AVR firma ATMEL. Je to **AVR studio**, které je pro profesionální práci naprosto ideální a proto ho budeme dále používat. K tomu, abychom mohli používat AVR studio však potřebujeme i programátor, který tento sw

podporuje. AVR studio podporuje zejména Start kity firmy Atmel STK100 až STK500. Kromě toho podporuje i tzv. **AVR ISP programátor AVR prog**, jehož základem je AT90S1200. Zapojení publikoval ATMEL ve svých aplikačních listech AVR910. Rovněž publikoval zdrojový kód pro AT90S1200 v programátoru. Kromě toho, že tento programátor je podporován AVR studiem, má ještě jednu velkou výhodu – nemá nedostatky některých jednoduchých programátorů způsobené tvarem a zpožděním signálů. Proto jsem navrhl programátor, který vychází ze zapojení firmy ATMEL. Zapojení tohoto programátoru ukazuje *obr. 8.13*, rozložení součástí *obr. 8.14* a obrazec spojů na *obr. 8.15*.

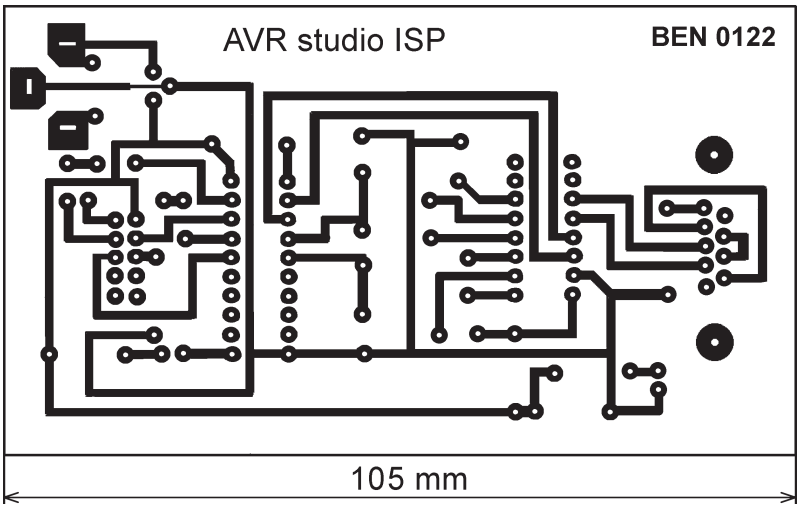
Spoj byl navržen jako jednostranný s několika drátovými propojkami. Zdrojový kód k tomuto programátoru publikoval ATMEL jako AVR910-2_0.asm, již přeložený



Obr. 8.13 *Zapojení AVR Prog*



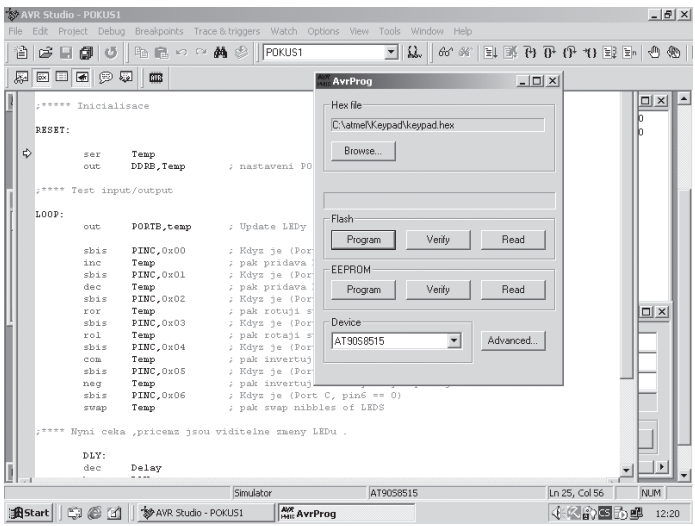
Obr. 8.14 *Rozložení součástí AVR Prog*



Obr. 8.15 *Schéma plošných spojů AVR Prog, skutečná velikost (105 mm × 60 mm) – BEN 0122*

kód jako AVR910-2_0.hex. Bohužel neumožňuje programování rozšířeného typu AT90S8535, proto je upravený kód s poznámkami AVR 910 umožňující naprogramovat typ 8535 uveden na CD příloze ke knize AVR Assembler nebo je možno jej zdarma stáhnout ze stránek nakladatelství BEN – technická literatura z přímé adresy knihy na Internetu (uvedeno v tiráži knihy).

Na *obr. 8.16* vidíme již zmiňované AVR studio.



Obr. 8.16 *AVR studio*

Popis práce s AVR studiem, bude součástí učebnice programování MCU řady AVR. Samozřejmostí je i dosti podrobný popis – help, který je součástí vlastního AVR studia. O používání jazyka C v AVR studiu pak informuje aplikační poznámka ATMELu AVR030: Getting Started with C for AVR.

426410

9

POZNÁMKA NA KONEC

Mikrokontroléry ATMEL AVR jsou poměrně nové mikroelektronické prvky, které se v poslední době slušně zabydlují v různých konstrukcích profesionálních i amatérských konstruktérů. Některým z nich však může vadit nedostatek literatury o ATMEL AVR v češtině. Proto tato knížka obsahuje převážně informace, jejichž zdrojem je firemní dokumentace firmy ATMEL. Tato dokumentace je však dosti rozsáhlá a její pouhý překlad by vydal na několik knih. Proto při snaze zhustit alespoň základní informace do jedné, české publikace může něco scházet. Vynechal jsem např. podrobnosti o řadě ATmega. Snažil jsem se ale uvést podstatné informace o základní řadě. Protože četba firemní dokumentace, katalogových listů, je mnohdy stejně záživná, jako četba telefonního seznamu, snažil jsem se překlad základních informací doplnit o vlastní poznámky a v poslední kapitole se krátce zmínil i o vývojovém software pro AVR a uvedl i ukázky praktických konstrukcí. Podrobnější informace o programování AVR, stejně jako praktické konstrukce s AVR, budou jistě obsahem dalších publikací.

426410

PŘÍLOHA 1

– INSTRUKČNÍ SOUBOR ŘADY AVR

Použité zkratky:

Stavový registr (SREG)

SREG: Stavový registr.

C: Příznak přenosu CARRY, indikuje přenos při poslední aritmetické operaci nebo posuvu.

Z: Příznak nuly ZERO, indikuje nulový výsledek aritmetické, logické a bitové operace, nebo přesunu, je rozdílný oproti příznaku ZERO u jiných procesorů, u kterých závisí jen na poslední instrukci, u AVR může záviset i na předchozích instrukcích.

N: Příznak NEGATIVE, indikuje negativní výsledek poslední aritmetické, logické a bitové operace, nebo posunu.

V: Příznak OVERFLOW, indikuje přetečení při aritmetické operaci.

S: $N \oplus V$.

H: Příznak HALF-CARRY, přenos mezi třetím a čtvrtým bitem, využíván pro BCD aritmetiku.

T: Příznak TRANSFER-BIT, využíván instrukcemi BLD a BST.

I: Příznak povolení všech přerušení (globální interrupt).

Registry a operandy

Rd: Cílový registr ze souboru registrů R0 až R31, do něhož je ukládán výsledek operace.

Rr: Zdrojový registr ze souboru registrů, který je zdrojem dat pro operace.

R: Výsledek po provedení instrukce.

K: Data – konstanta (8 bitů).

k: Adresa – konstanta.

b: Bit ze souboru registrů (3 bity).

s: Bit ve stavovém registru (3 bity).

X, Y, Z: Registry pro nepřímé adresování (dvojice registrů).

A: Adresa brány I/O.

q: Posun (offset) adresy (6 bitů) pro přímé adresování.

426410

I/O registry RAMPX, RAMPY, RAMPZ

Registry spojené s registry X, Y a Z umožňující přímé adresování celého datového prostoru MCU s více než 64Kbytovým datovým prostorem

RAMPD

Registr spojený s registrem Z umožňující přímé adresování celého datového prostoru MCU s více než 64 Kbytů datového prostoru.

EIND

Registr spojený s instrukčním slovem umožňující přímý skok a volání v celém programovém prostoru MCU s více než 64Kbytovým programovým prostorem.

STACK

STACK: Zásobník pro návratovou adresu a ukládání obsahu registrů.

SP: Ukazatel zásobníku na STACK.

Příznaky (FLAGs)

⇔: Příznak je ovlivněn instrukcí.

0: Příznak je vynulován instrukcí.

1: Příznak je nastaven instrukcí.

-: Příznak není ovlivněn instrukcí.

Podmínky větvení programu

Tab. P.1

Test	Booleovsky	Zkratka	Doplňkově	Booleovsky	Zkratka	Komentář
Rd>Rr	Z•(N⊕V) = 0	BRLT (1)	Rd≤Rr	Z+ (N⊕V) = 1	BRGE (1)	Se znaménkem
Rd≥Rr	(N⊕V) = 0	BRGE	Rd<Rr	(N⊕V) = 1	BRLT	Se znaménkem
Rd=Rr	Z = 1	BREQ	Rd≠Rr	Z = 0	BRNE	Se znaménkem
Rd≤Rr	Z+ (N⊕V) = 1	BRGE (1)	Rd>Rr	Z•(N⊕V) = 0	BRLT (1)	Se znaménkem
Rd<Rr	(N⊕V) = 1	BRLT	Rd≥Rr	(N⊕V) = 0	BRGE	Se znaménkem
Rd>Rr	C + Z = 0	BRLO (1)	Rd≤Rr	C + Z = 1	BRSH (1)	Bez znaménka
Rd≥Rr	C = 0	BRSH/BRCC	Rd<Rr	C = 1	BRLO/BRCS	Bez znaménka
Rd=Rr	Z = 1	BREQ	Rd≠Rr	Z = 0	BRNE	Bez znaménka
Rd≤Rr	C + Z = 1	BRSH (1)	Rd>Rr	C + Z = 0	BRLO (1)	Bez znaménka
Rd<Rr	C = 1	BRLO/BRCS	Rd≥Rr	C = 0	BRSH/BRCC	Bez znaménka
Carry	C = 1	BRCS	No Carry	C = 0	BRCC	Jednoduchý
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Jednoduchý
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Jednoduchý
Zero	Z = 1	BREQ	Not zero	Z = 0	BRNE	Jednoduchý

Poznámka: *1. přehodit Rd a Rr před testem tj. CP Rd, Rr → CP Rr, Rd*
• … logický součin
+ … logický součet
⊕ … exkluzivní or

Úplný instrukční soubor

Aritmetické a logické instrukce

Tab. P.2

Zkratka	Operandy	Popis	Operace	Příznaky	T
ADD	Rd, Rr	Součet bez přenosu	Rd ← Rd + Rr	Z, C, N, V, S, H	1
ADC	Rd, Rr	Součet s přenosem	Rd ← Rd + Rr + C	Z, C, N, V, S, H	1
ADIW	Rd, K	Přičtení konstanty ke slovu	Rd+1:Rd ← Rd+1:Rd+K	Z, N, V, S	2
SUB	Rd, Rr	Rozdíl bez přenosu	Rd ← Rd – Rr	Z, C, N, V, S, H	1
SUBI	Rd, K	Odečtení konstanty	Rd ← Rd – K	Z, C, N, V, S, H	1
SBC	Rd, Rr	Rozdíl s přenosem	Rd ← Rd – Rr – C	Z, C, N, V, S, H	1
SBCI	Rd, K	Odečtení konstanty s přenosem	Rd ← Rd – K – C	Z, C, N, V, S, H	1
SBIW	Rd, K	Odečtení konstanty od slova	Rd+1:Rd ← Rd+1:Rd – K	Z, C, N, V, S	1
AND	Rd, Rr	Logický součin	Rd ← Rd • Rr	Z, N, V, S	1
ANDI	Rd, K	Logický součin s konstantou	Rd ← Rd • K	Z, N, V, S	1
OR	Rd, Rr	Logický součet	Rd ← Rd ∨ Rr	Z, N, V, S	1
ORI	Rd, K	Logický součet s konstantou	Rd ← Rd ∨ K	Z, N, V, S	1
EOR	Rd, Rr	Nonekvivalence, výlučné OR	Rd ← Rd ⊕ Rr	Z, N, V, S	1
COM	Rd	Komplement Rd	Rd ← $FF – Rr	Z, C, N, V, S	1
NEG	Rd	Dvojkový doplněk Rd	Rd ← $00 – Rd	Z, C, N, V, S, H	1
SBR	Rd, K	Nastavení bitu v registru	Rd ← Rd ∨ K	Z, N, V, S	1
CBR	Rd, K	Vynulování bitu v registru	Rd ← Rd • ($FFh – K)	Z, N, V, S	1
INC	Rd	Zvýšení obsahu o 1	Rd ← Rd + 1	Z, N, V, S	1
DEC	Rd	Snížení obsahu o 1	Rd ← Rd – 1	Z, N, V, S	1
TST	Rd	Testování na nulu nebo minus	Rd ← Rd • Rd	Z, N, V, S	1
CLR	Rd	Nulování registru	Rd ← Rd ⊕ Rd	Z, N, V, S	1
SER	Rd	Naplnění registru FFh	Rd ← $FF	žádné	1
MUL	Rd, Rr	Násobení bez znaménka	R1:R0 ← Rd × Rr (UU)	Z, C	2
MULS	Rd, Rr	Násobení se znaménkem	R1:R0 ← Rd × Rr (SS)	Z, C	2
MULSU	Rd, Rr	Násobení se znaménkem s neznaménkovým činitelem	R1:R0 ← Rd × Rr (SU)	Z, C	2
FMUL	Rd, Rr	Násobení zlomků bez znaménka	R1:R0 ← (Rd × Rr) <<1 (UU)	Z, C	2
FMULS	Rd, Rr	Násobení zlomků se znaménkem	R1:R0 ← (Rd × Rr) <<1 (SS)	Z, C	2
FMULSU	Rd, Rr	Násobení zlomku se znaménkem s neznaménkovým zlomkem	R1:R0 ← (Rd × Rr) <<1 (SU)	Z, C	2

Skokové instrukce

Tab. P.3

Zkratka	Operandy	Popis	Operace	Příznaky	T
RJMP	k	Relativní skok	PC ← PC+k+1	žádný	2
IJMP		Nepřímý skok do (Z)	PC(15:0) ← Z, PC(21:16) ← 0	žádný	2
EIJMP		Rozšířený nepřímý skok do (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	žádný	2
JMP	k	Nepodmíněný skok	PC ← k	žádný	3
RCALL	k	Relativní volání podprogramu	PC ← PC+k1	žádný	3/4
ICALL		Nepřímé volání do Z	PC(15:0) ← Z, PC(21:16) ← 0	žádný	3/4
EICALL		Rozšíření nepřímé volání do Z	PC(15:0) ← Z, PC(21:16) ← EIND	žádný	4
CALL	k	Volání podprogramu	PC ← k	žádný	4/5
RET		Návrat z podprogramu	PC ← STACK	žádný	4/5
RETI		Návrat z obsluhy přerušení	PC ← STACK	I	4/5
CPSE	Rd, Rr	Skok při rovnosti	Když (Rd = Rr) tak PC ← PC + 2 nebo 3	žádný	1/2/3
CP	Rd, Rr	Porovnání	Rd – Rr	Z, C, N, V, S, H	1
CPC	Rd, Rr	Porovnání s přenosem	Rd – Rr – C	Z, C, N, V, S, H	1
CPI	Rd, K	Porovnání s konstantou	Rd – K	Z, C, N, V, S, H	1
SBRC	Rr, b	Skok při nulovém bitu v Rr	Když (Rr(b) = 0) tak PC ← PC+2 nebo 3	žádný	1/2/3
SBRS	Rr, b	Skok při nahozeném bitu v Rr	Když (Rr(b) = 1) tak PC ← PC+2 nebo 3	žádný	1/2/3
SBIC	A, b	Skok při nulovém bitu v I/O registru	Když (I/O(A, b) = 0) tak PC ← PC+2 nebo 3	žádný	1/2/3
SBIS	A, b	Skok při nahozeném bitu v I/O registru	Když (I/O(A,b) = 1) tak PC ← PC+2 nebo 3	žádný	1/2/3
BRBS	s, k	Skok při s = 1	Když (SREG(s) = 1) tak PC ← PC+k+1	žádný	1/2
BRRC	s, k	Skok při s = 0	Když (STRG(s) = 0) tak PC ← PC+k+1	žádný	1/2
BREQ	k	Skok při Z = 1, rovná se	Když (Z = 1) tak PC ← PC+k+1	žádný	1/2
BRNE	k	Skok při Z = 0, nerovná se	Když (Z = 0) tak PC ← PC+k+1	žádný	1/2
BRCS	k	Skok při C = 1, C je nastaveno	Když (C = 1) tak PC ← PC+k+1	žádný	1/2
BRCC	k	Skok při C = 0, C je vynulováno	Když (C = 0) tak PC ← PC+k+1	žádný	1/2
BRSH	k	Skok při C = 0 ,je stejné nebo větší	Když (C = 0) tak PC ← PC+k+1	žádný	1/2
BRLO	k	Skok při C = 1 ,je menší	Když (C = 1) tak PC ← PC+k+1	žádný	1/2
BRMI	k	Skok při N = 1	Když (N = 1) tak PC ← PC+k+1	žádný	1/2
BRPL	k	Skok při N = 0	Když (N = 0) tak PC ← PC+k+1	žádný	1/2
BRGE	k	Skok při N = V	Když (N⊕V = 0) tak PC ← PC+k+1	žádný	1/2
BRLT	k	Skok při N<>V	Když (N⊕V = 1) tak PC ← PC+k+1	žádný	1/2
BRHS	k	Skok při H = 1	Když (H = 1) tak PC ← PC+k+1	žádný	1/2
BRHC	k	Skok při H = 0	Když (H = 0) tak PC ← PC+k+1	žádný	1/2
BRTS	k	Skok při T = 1	Když (T = 1) tak PC ← PC+k+1	žádný	1/2
BRTC	k	Skok při T = 0	Když (T = 0) tak PC ← PC+k+1	žádný	1/2
BRVS	k	Skok při V = 1	Když (V = 1) tak PC ← PC+k+1	žádný	1/2
BRVC	k	Skok při V = 0	Když (V = 0 tak PC ← PC+k+1	žádný	1/2
BRIE	k	Skok při I = 1	Když (I = 1) tak PC ← PC+k+1	žádný	1/2
BRID	k	Skok při I = 0	Když (I = 0) tak PC ← PC+k+1	žádný	1/2

Instrukce pro přenos dat

Tab. P.4

Zkratka	Operandy	Popis	Operace	Příznaky	T
MOV	Rd, Rr	Přesun obsahu registru	Rd ← Rr	žádný	1
MOVW	Rd, Rr	Přesun obsahu dvojice registrů	Rd+1:Rd ← Rr+1:Rr	žádný	1
LDI	Rd, K	Přesun konstanty do registru	Rd ← K	žádný	1
LDS	Rd, k	Přímý přesun z datového prostoru	Rd ← (k)	žádný	2
LD	Rd, X	Přesun dat z adresy v X	Rd ← (X)	žádný	2
LD	Rd, X+	Přesun a zvýšení X o 1	Rd ← (X), X ← X + 1	žádný	2
LD	Rd, –X	Snížení X o 1 a přesun	X ←X – 1, Rd ← (X)	žádný	2
LD	Rd, Y	Přesun dat z adresy v Y	Rd ← (Y)	žádný	2
LD	Rd, Y+	Přesun a zvýšení Y o 1	Rd ← (Y), Y ← Y + 1	žádný	2
LD	Rd, –Y	Snížení Y o 1 a přesun	Y ←Y – 1, Rd ← (Y)	žádný	2
LDD	Rd, Y+q	Přesun z adresy Y+q	Rd ← (Y + q)	žádný	2
LD	Rd, Z	Přesun dat z adresy v Z	Rd ← (Z)	žádný	2
LD	Rd, Z+	Přesun a zvýšení Z o 1	Rd ← (Z), Z ← Z + 1	žádný	2
LD	Rd, –Z	Snížení Z o 1 a přesun	Z ←Z – 1, Rd ← (Z)	žádný	2
LDD	Rd, Z+q	Přesun z adresy Z+q	Rd ← (Z + q)	žádný	2
STS	k, Rr	Přímý přesun do datového prostoru	Rd ← (k)	žádný	2
ST	X, Rr	Přesun dat na adresu v X	(X) ← Rr	žádný	2
ST	X+, Rr	Přesun a zvýšení X o 1	(X) ← Rr, X ← X + 1	žádný	2
ST	–X, Rr	Snížení X o 1 a přesun	X ← X – 1, (X) ← Rr	žádný	2
ST	Y, Rr	Přesun dat na adresu v Y	(Y) ← Rr	žádný	2
ST	Y+, Rr	Přesun a zvýšení Y o 1	(Y) ← Rr, Y ← Y + 1	žádný	2
ST	–Y,Rr	Snížení Y o 1 a přesun	Y ← Y – 1, (Y) ← Rr	žádný	2
STD	Y+q, Rr	Přesun na adresu Y+q	(Y + q) ← Rr	žádný	2
ST	Z, Rr	Přesun dat na adresu v Z	(Z) ← Rr	žádný	2
ST	Z+, Rr	Přesun a zvýšení Z o 1	(Z) ← Rr, Z ← Z + 1	žádný	2
ST	–Z, Rr	Snížení Z o 1 a přesun	Z ← Z – 1, (Z) ← Rr	žádný	2
STD	Z+q,Rr	Přesun na adresu Z+q	(Z + q) ← Rr	žádný	2
LPM		Přesun do R0 z adresy v Z	R0 ← (Z)	žádný	3
LPM	Rd,Z	Přesun do Rd z adresy v Z	Rd ← (Z)	žádný	3
LPM	Rd,Z+	Přesun do Rd z adresy v Z a inkrementace Z	Rd ← (Z), Z ← Z + 1	žádný	3
ELPM		Přesun do R0 z adresy v rozšířené programové paměti	R0 ← (RAMPZ:Z)	žádný	3
ELPM	Rd,Z	Přesun do Rd z adresy v rozšířené programové paměti	Rd ← (RAMPZ:Z)	žádný	3
ELPM	Rd,Z+	Přesun do Rd z adresy v rozšířené programové paměti a inkrementace Z	Rd ← (RAMPZ:Z), Z ← Z + 1	žádný	3
SPM		Přesun na adresu v Z	(Z) ← R1:R0	žádný	-
ESPM		Přesun na adresu v rozšíření programové paměti	(RAMPZ:Z) ← R1:R0	žádný	-
IN	Rd,A	Čtení z brány	Rd ← I/O(A)	žádný	1
OUT	A, Rr	Výstup na bránu	I/O(A) ← Rd	žádný	1
PUSH	Rr	Uložení do zásobníku	STACK ← Rr	žádný	2
POP	Rd	Čtení ze zásobníku	Rr ← STACK	žádný	2

Bitové operace a instrukce testování bitů

Tab. P.5

Zkratka	Operandy	Popis	Operace	Příznaky	T
LSL	Rd	Posuv vlevo (do C)	Rd (n+1) ← Rd (n), Rd (0) ← 0,C ← Rd (7)	Z, C, N, V, H	1
LSR	Rd	Posuv vpravo (do C)	Rd (n) ← Rd (n+1), Rd (7) ← 0,C ← Rd (0)	Z, C, N, V	1
ROL	Rd	Rotace vlevo (přes C)	Rd (0) ← C, Rd (n+1) ← Rd (n),C ← Rd (7)	Z, C, N, V, H	1
ROR	Rd	Rotace vpravo (přes C)	Rd (7) ← C, Rd (n) ← Rd (n+1),C ← Rd (0)	Z, C, N, V	1
ASR	Rd	Aritmetický posuv vpravo	Rd (n) ← Rd (n+1), n = 0..6	Z, C, N, V	1
SWAP	Rd	Záměna slabik	Rd (3..0) ← Rd (7..4)	žádný	1
BSET	s	Nastavení stavového bitu	SREG(s) ← 1	SREG(s)	1
BCLR	s	Shození stavového bitu	SREG(s) ← 0	SREG(s)	1
SBI	A, b	Nastavení bitu v I/O registru	I/O(A, b) ← 1	žádný	2
CBI	A, b	Shození bitu v I/O registru	I/O(A, b) ← 0	žádný	2
BST	Rr, b	Přesun bitu z Rr do T	T ← Rr(b)	T	1
BLD	Rd, b	Přesun bitu T do Rd	Rd(b) ← T	žádný	1
SEC		Nastavení příznaku C v SREG	C ← 1	C	1
CLC		Nulování příznaku C v SREG	C ← 0	C	1
SEN		Nastavení příznaku N v SREG	N ← 1	N	1
CLN		Nulování příznaku N v SREG	N ← 0	N	1
SEZ		Nastavení příznaku Z v SREG	Z ← 1	Z	1
CLZ		Nulování příznaku Z v SREG	Z ← 0	Z	1
SEI		Nastavení příznaku I v SREG	I ← 1	I	1
CLI		Nulování příznaku I v SREG	I ← 0	I	1
SES		Nastavení příznaku S v SREG	S ← 1	S	1
CLS		Nulování příznaku S v SREG	S ← 0	S	1
SEV		Nastavení příznaku V v SREG	V ← 1	V	1
CLV		Nulování příznaku V v SREG	V ← 0	V	1
SET		Nastavení příznaku T v SREG	T ← 1	T	1
CLT		Nulování příznaku T v SREG	T ← 0	T	1
SEH		Nastavení příznaku H v SREG	H ← 1	H	1
CLH		Nulování příznaku H v SREG	H ← 0	H	1
NOP		Prázdná operace		žádný	1
SLEEP		Zapnutí úsporného režimu	Viz specifikaci popisu Sleep	žádný	1
WDR		Reset watchdog časovače	Viz specifikaci popisu WDR	žádný	1

ADC

ADD WITH CARRY

Součet s přenosem

Popis:

Sečte obsah dvou registrů a obsah příznaku C a výsledek umístí do cílového registru Rd.

Syntaxe instrukce:

ADC Rd, Rr

Interval operandu:

$0 \le d \le 31$, $0 \le r \le 31$

Operace:

Rd ← Rd + Rr + C

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0001	11rd	dddd	rrrr

Stavový registr (SREG)

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3 • Rr3 + Rr3 • $\overline{R3}$ + $\overline{R3}$ • Rd3
nastaví se, když byl přenos z bitu 3, v opačném případě se vynuluje.

S: $N \oplus V$
pro znaménkové testy.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
nastaví se při přeteční dvojkového doplňku výsledku, jinak se vynuluje.

N: R7
nastaví se, když MSB výsledku je nastaveno, jinak se vynuluje.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaví se, když výsledek je $00, jinak se vynuluje.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
nastaví se, když je přenos z MSB výsledku, jinak se vynuluje.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
            ; přičti R1:R0 k R3:R2
add r2,r0   ; sečte dolní byte
adc r3,r1   ; sečte s přenosem horní byte
```

ADD

ADD WITHOUT CARRY

Součet bez přenosu

Popis:

Sečte obsah dvou registrů bez příznaku C a výsledek umístí do cílového registru Rd.

Syntaxe instrukce:

ADD Rd, Rr

Interval operandu:

$0 \leq d \leq 31$, $0 \leq r \leq 31$

Operace:

Rd ← Rd + Rr

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0000	11rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
nastaví se, když byl přenos z bitu 3, v opačném případě se vynuluje.

S: $N \oplus V$
pro znaménkové testy.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} + \overline{Rr7} \bullet R7$
nastaví se při přetečení dvojkového doplňku výsledku, jinak se vynuluje.

N: R7
nastaví se, když MSB výsledku je nastaveno, jinak se vynuluje.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaví se, když výsledek je $00, jinak se vynuluje.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
nastaví se, když je přenos z MSB výsledku, jinak se vynuluje.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add r1,r2      ; přičte r2 k r1 (r1=r1+r2)
add r28,r28    ; přičte r28 k sobě sama (r28=r28+r28), násobení
               ; dvěma
```

ADIW

ADD IMMEDIATE TO WORD

Přičtení konstanty ke slovu

Popis:

Přičte konstantní hodnotu (0–63) k dvojici registrů a výsledek umístí do dvojice registrů. Instrukce pracuje s horními čtyřmi dvojicemi registrů a je velmi vhodná pro operace s ukazatelem.

Syntaxe instrukce:

ADIW Rd, K

Interval operandu:

$d \in (24, 26, 28, 30)$, $0 \leq K \leq 63$

Operace:

Rd + 1 : Rd ← Rd + 1 : Rd + K

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0110	KKdd	KKKK

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$
pro znaménkové testy.

426410

V: $\overline{Rdh7} \bullet R15$
nastaví se při přetečení dvojkového doplňku výsledku operace, jinak se vynuluje.

N: R15
nastaví se, když MSB výsledku je nastaveno, jinak se vynuluje.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaví se, když výsledek je $0000, jinak se vynuluje.

C: $\overline{R15} \bullet Rdh7$
nastaví se, když byl přenos z MSB výsledku, jinak se vynuluje

R (výsledek) se rovná Rdh:Rdl po skončení operace (Rdh7 – Rdh0 = R15 – R8, Rdl7 – Rdl0 = R7 – R0)

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
adiw r24,1     ; přičte 1 k r25:r24
adiw r30,63    ; přičte 63 do ukazatele Z (r31:r30)
```

AND

Logical AND

Logický součin

Popis:

Vytváří logický součin mezi obsahy registrů Rd a Rr a výsledek umístí do cílového registru Rd.

Syntaxe instrukce:

AND Rd, Rr

Interval operandu:

$0 \le d \le 31$; $0 \le r \le 31$

Operace:

Rd ← Rd • Rr

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0010	00rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: N ⊕ V
pro znaménkové testy.

426410

V: 0
vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
and r2,r3   ; bitový součet mezi r2 a r3, výsledek je v r2
ldi r16,1   ; nastaví bitovou masku 0000 0001 v r16
and r2,r16  ; vymaskuje bit 0 v r2
```

ANDI

LOGICAL AND WITH IMMEDIATE

Logický součin s konstantou

Popis:

Vytváří logický součin mezi obsahem registru Rd a konstantou a výsledek umístí do cílového registru Rd.

Syntaxe instrukce:

ANDI Rd, K

Interval operandu:

16 ≤ **d** ≤ **31**; 0 ≤ K ≤ 255

Operace:

Rd ← Rd • K

Programový čítač:

PC ← PC + 1

Šestnáctibitový bit kód instrukce:

0011	KKKK	dddd	KKKK

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: N ⊕ V
pro znaménkové testy.

426410

V: 0
vždy vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
andi r17,$0F     ; vynuluje horní polovinu r17
andi r18,$10     ; vymaskuje bit 4 v r18
andi r19,$AA     ; vynuluje liché bity r19
```

ASR

ARITHMETIC SHIFT RIGHT

Aritmetický posuv vpravo

Popis:

Posouvá všechny bity v Rd o jedno místo doprava. Bit 7 zůstává nezměněn. Bit 0 je přesunut do příznaku C v SREG. Tato operace provádí rychlé dělení dvěma znaménkové hodnoty bez změny znaménka. Příznak C může být využit k zaokrouhlení výsledku.

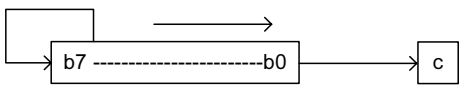
Syntaxe instrukce:

ASR Rd

Interval operandu:

$0 \leq d \leq 31$

Operace:



Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	010d	dddd	0101

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$
pro znaménkové testy.

V: $N \oplus C$
(pro N a C po posunu).

N: R7
nastaví se, když MSB výsledku je nastaveno, jinak se vynuluje.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaví se, když výsledek je $00, jinak se vynuluje.

C: Rd0
nastaví se, když před posunem bylo LSB v Rd nastaveno, jinak se vynuluje.

R (výsledek) se rovná Rd po operaci.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
ldi r16,$10      ; vloží dekadické číslo 16 do r16
asr r16          ; r16=r16 / 2
ldi r17,$FC      ; vloží -4 do r17
asr r17          ; r17=r17 / 2
```

BCLR

BIT CLEAR IN SREG

Vynulování bitu v SREG

Popis:

Nuluje jeden příznak v SREG.

Syntaxe instrukce:

BCLR s

Interval operandu:

$0 \leq s \leq 7$

Operace:

SREG(s) $\leftarrow$ 0

Programový čítač:

PC $\leftarrow$ PC + 1

Šestnáctibitový kód instrukce:

1001	0100	1sss	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I: 0 když s = 7, jinak zůstává nezměněn.

T: 0 když s = 6, jinak zůstává nezměněn.

H: 0 když s = 5, jinak zůstává nezměněn.

S: 0 když s = 4, jinak zůstává nezměněn.

V: 0 když s = 3, jinak zůstává nezměněn.

N: 0 když s = 2, jinak zůstává nezměněn.

Z: 0 když s = 1, jinak zůstává nezměněn.

C: 0 když s = 0, jinak zůstává nezměněn.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
bclr 0    ; vynuluje příznak přenosu
bclr 7    ; zakazuje přerušení
```

BLD

BIT LOAD FROM THE T FLAG IN SREG TO A BIT IN REGISTER

Přesun bitu z příznaku T v SREG do bitu v registru

Popis:

Kopíruje příznak T z SREG do bitu b v registru Rd.

Syntaxe instrukce:

BLD Rd, b

Interval operandu:

$0 \leq d \leq 31$, $0 \leq b \leq 7$

Operace:

$Rd(b) \leftarrow T$

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

1111	100d	dddd	0bbb

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
            ; kopírování bitu
bst r1,2    ; přenos bitu 2 z r1 do příznaku T
bld r0,4    ; přenos příznaku T do bitu 4 r0
```

BRBC

Branch if Bit in SREG is Cleared

Skok, když bit v SREG je vynulován

Popis:

Podmíněný relativní skok. Testuje jeden bit v SREG a provádí relativní skok vzhledem k PC, když je bit vynulován. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC + 64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru

Syntaxe instrukce:

BRBC s, k

Interval operandu:

$0 \le s \le 7$, $-64 \le k \le +63$

Operace:

Když SREG(s) = 0 pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	01kk	kkkk	ksss

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když je splněna podmínka.

Příklad:

```
          cpi  r20, 5       ; srovnává r20 s hodnotou 5
          brbc 1, noteq     ; skok když příznak Z je vynulován
          .....
noteq:    nop               ; cíl skoku (nic nedělá)
```

BRBS

BRANCH IF BIT IN SREG IS SET

Skok, když bit v SREG je nastaven

Popis:

Podmíněný relativní skok. Testuje jeden bit v SREG a provádí relativní skok vzhledem k PC, když je bit nastaven. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC+64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru.

Syntaxe instrukce:

BRBS s, k

Interval operandu:

0 ≤ s ≤ 7, –64 ≤ k ≤ +63

Operace:

Když SREG(s) = 1 pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	ksss

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když je splněna podmínka.

Příklad:

```
          bst r0,3          ; naplní bit T bitem 3 z r0
          brbs6, bitset     ; skok, když bit T byl nastaven
          .....
bitset:   nop               ; cíl skoku (nic nedělá)
```

BRCC

BRANCH IF CARRY CLEARED

Skok, když přenos je vynulován

Popis:

Podmíněný relativní skok. Testuje příznak přenosu (C) a provádí relativní skok vzhledem k PC, když je C vynulován. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC – 63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBC 0, k).

Syntaxe instrukce:

BRCC k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	01kk	kkkk	K000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          add     r22, r23        ; přičte r23 k r22
          brcc    nocarry         ; skok když přenos je vynulován
          .....
nocarry:  nop                     ; cíl skoku (nedělá nic)
```

BRCS

BRANCH IF CARRY SET

Skok, když přenos je nastaven

Popis:

Podmíněný relativní skok. Testuje příznak přenosu (C) a provádí relativní skok vzhledem k PC, když je C nastaven. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBS 0, k)

Syntaxe instrukce:

BRCS k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když C = 1 pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	K000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
        cpi  r26,$56     ; porovnává r26 s $56
        brcs carry       ; skok když je přenos nastaven
        .....
carry:  nop              ; cíl skoku (nedělá nic)
```

BREQ

BRANCH IF EQUAL

Skok při rovnosti

Popis:

Podmíněný relativní skok. Testuje příznak Zero (Z) a provádí relativní skok vzhledem k PC, když je Z nastaven. Je-li instrukce provedena bezprostředně po instrukcích CP, CPI, SUB nebo SUBI, skok nastane právě tehdy, když znaménkové nebo neznaménkové binární číslo, reprezentované Rd, se rovnalo znaménkovému nebo neznaménkovému binárnímu číslu reprezentovaném Rr. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBS 1, k).

Syntaxe instrukce:

BREQ k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když Rd = Rr (Z = 1), tak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	K001

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          cp r1,r0          ; porovnává registry r1 a r0
          breq equal        ; skok když registry se rovnají
          .....
equal:    nop               ; cíl skoku (nedělá nic)
```

BRGE

BRANCH IF GREATER OR EQUAL (SIGNED)

Skok při větší nebo rovno (znaménkově)

Popis:

Podmíněný relativní skok. Testuje znaménkový příznak (S) a provádí relativní skok vzhledem k PC, když je S vynulován. Je-li instrukce provedena bezprostředně po instrukcích CP, CPI, SUB nebo SUBI, skok nastane právě tehdy, když znaménkové binární číslo reprezentované Rd bylo větší nebo rovné binárnímu číslu reprezentovaném Rr. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBC 4, k).

Syntaxe instrukce:

BRGE k

Interval operandu:

$-64 \leq k \leq +63$

Operace:

Když $Rd \geq Rr (N \oplus V = 0)$ pak $PC \leftarrow PC + k + 1$, jinak $PC \leftarrow PC + 1$

Programový čítač:

$PC \leftarrow PC + k + 1$
$PC \leftarrow PC + 1$, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	01kk	kkkk	K100

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          cp r11,r12     ; porovnává registry r11 a r12
          brge greateq   ; provádí skok při r11 ≥ r12 (znaménkově)
          .....
greateq:  nop            ; cíl skoku (nedělá nic)
```

BRHC

BRANCH IF HALF CARRY FLAG IS CLEARED

Skok při vynulovaném příznaku Half Carry (přenos mezi třetím a čtvrtým bitem)

Popis:

Podmíněný relativní skok. Testuje příznak přenosu mezi třetím a čtvrtým bitem (H) a provádí relativní skok vzhledem k PC, když je H vynulován. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBC 5, k).

Syntaxe instrukce:

BRHC k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když H = 0, pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	01kk	kkkk	K101

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          brhc hclear      ; skok při vynulovaném half carry
          .....
hclear:   nop              ; cíl skoku (nedělá nic)
```

BRHS

BRANCH IF HALF CARRY FLAG IS SET

Skok při nastaveném příznaku Half Carry (přenos mezi třetím a čtvrtým bitem)

Popis:

Podmíněný relativní skok. Testuje příznak přenosu mezi třetím a čtvrtým bitem (H) a provádí relativní skok vzhledem k PC, když je H nastaven.Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBS 5, k).

Syntaxe instrukce:

BRHS k

Interval operandu:

$-64 \leq k \leq +63$

Operace:

Když H = 1, pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	K101

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
        brhs hset       ; skok při nastaveném příznaku half carry
        .....
hset:   nop             ; cíl skoku (nedělá nic)
```

BRID

BRANCH IF GLOBAL INTERRUPT IS DISABLED

Skok při zakázeném přerušení (všech)

Popis:

Podmíněný relativní skok. Testuje povolení přerušení (I) a provádí relativní skok vzhledem k PC, když je I vynulován. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBC 7, k).

Syntaxe instrukce:

BRID k

Interval operandu:

$-64 \leq k \leq +63$

Operace:

Když I = 0, pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	01kk	kkkk	k111

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          brid intdis    ; skok při zakázaném přerušení
          .....
intdis:   nop            ; cíl skoku (nedělá nic)
```

BRIE

BRANCH IF GLOBAL INTERRUPT IS ENABLED

Skok při povoleném přerušení (všech)

Popis:

Podmíněný relativní skok. Testuje povolení přerušení (I) a provádí relativní skok skok vzhledem k PC, když je I povolen. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBS 7, k).

Syntaxe instrukce:

BRIE k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když I = 1, pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	k111

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          brie inten       ; skok při povoleném přerušení
          .....
inten:    nop              ; cíl skoku (nedělá nic)
```

BRLO

BRANCH IF LOVER (UNSIGNED)

Skok při nerovnosti, menší než

Popis:

Podmíněný relativní skok. Testuje příznak přenosu (C) a provádí relativní skok vzhledem k PC, když je C nastaven. Je-li instrukce provedena bezprostředně po instrukcích CP, CPI, SUB nebo SUBI, skok nastane právě tehdy, když neznaménkové binární číslo reprezentované Rd bylo menší, než neznaménkové binární číslo reprezentované Rr. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrucí BRBS 0, k).

Syntaxe instrukce:

BRLO k

Interval operandu:

$-64 \leq k \leq +63$

Operace:

Když Rd ← Rr (C = 1), tak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	k000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          eor r19, r19       ; vynuluje r19
loop:     inc r19            ; zvětší r19 o 1
          .....
          cpi r19, $10       ; porovnává r19 s $10
          brlo loop          ; skok když r19<$10 (neznaménkově)
          nop                ; výstup ze smyčky (nedělá nic)
```

BRLT

BRANCH IF LESS THEN (SIGNED)

Skok při menší než

Popis:

Podmíněný relativní skok. Testuje příznak přenosu (S) a provádí relativní skok vzhledem k PC, když je S nastaven. Je-li instrukce provedena bezprostředně po instrukcích CP, CPI, SUB nebo SUBI, skok nastane právě tehdy, když znaménkové binární číslo reprezentované Rd bylo menší, než znaménkové binární číslo reprezentované Rr. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech ($PC - 63 \leq cíl \leq PC + 64$). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBS 4, k).

Syntaxe instrukce:

BRLT k

Interval operandu:

$-64 \leq k \leq +63$

Operace:

Když $Rd \leftarrow Rr$ $(N \oplus V = 1)$, tak $PC \leftarrow PC + k + 1$, jinak $PC \leftarrow PC + 1$

Programový čítač:

$PC \leftarrow PC + k + 1$
$PC \leftarrow PC + 1$, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	K100

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
        cp r16, r1      ; porovnává r16 s r1
        brlt less       ; skok když r16 < r1 (znaménkově)
        .....
less:   nop             ; cíl skoku (nedělá nic)
```

BRMI

BRANCH IF MINUS

Skok při záporné hodnotě

Popis:

Podmíněný relativní skok. Testuje příznak Negative (N) a provádí relativní skok vzhledem k PC, když je N nastaven. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBS 2, k).

Syntaxe instrukce:

BRMI k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když N = 1, pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	K010

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
            subi r18, 4         ; odečte 4 od r18
            brmi negative       ; skok při záporném výsledku
            .....
negative:   nop                 ; cíl skoku (nedělá nic)
```

BRNE

BRANCH IF NOT EQUAL

Skok při nerovnosti

Popis:

Podmíněný relativní skok. Testuje příznak Zero (Z) a provádí relativní skok vzhledem k PC, když je Z vynulován. Je-li instrukce provedena bezprostředně po instrukcích CP, CPI, SUB nebo SUBI, skok nastane právě tehdy, když znaménkové nebo neznaménkové binární číslo reprezentované Rd se nerovnalo znaménkovému nebo neznaménkovému binárnímu číslu reprezentovaném Rr. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrucí BRBC 1, k).

Syntaxe instrukce:

BRNE k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když Rd ≠ Rr (Z = 0), tak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	01kk	kkkk	K001

426410

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
       eor r27, r27     ; nuluje r27
loop:  inc r27          ; zvětšuje r27 o 1
       .....
       cpi r27, 5       ; srovnává r27 s 5
       brne loop        ; skok když r27<>5
       nop              ; výstup ze smyčky (nedělá nic)
```

BRPL

BRANCH IF PLUS

Skok při kladné hodnotě

Popis:

Podmíněný relativní skok. Testuje příznak Negative (N) a provádí relativní skok vzhledem k PC, když je N vynulován. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBC 2, k).

Syntaxe instrukce:

BRPL k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když N = 0, pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	01kk	kkkk	K010

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
            subi    r26, $50        ; odečte $50 od r26
            brpl    positive        ; skok při r26 kladném
            .....
positive:   nop                     ; cíl skoku (nedělá nic)
```

BRSH

BRANCH IF SAME OR HIGHER (UNSIGNED)

Skok když je rovný nebo větší

Popis:

Podmíněný relativní skok. Testuje příznak přenosu (C) a provádí relativní skok vzhledem k PC, když je C vynulován. Je-li instrukce provedena bezprostředně po instrukcích CP, CPI, SUB nebo SUBI, skok nastane právě tehdy, když neznaménkové binární číslo reprezentované Rd bylo větší nebo rovno, než neznaménkové binární číslo reprezentované Rr. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBC 0, k).

Syntaxe instrukce:

BRSH k

Interval operandu:

$-64 \leq k \leq +63$

Operace:

Když Rd ≥ Rr (C = 0), tak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	01kk	kkkk	k000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
            subi    r19, 4    ; odečte 4 od r19
            brsh    hishsm    ; skok když r19 ≥ 4
            .....
highsm:     nop               ; cíl skoku (nedělá nic)
```

BRTC

Branch if the T Flag is Cleared

Skok, když příznak T je vynulován

Popis:

Podmíněný relativní skok. Testuje příznak T a provádí relativní skok vzhledem k PC, když je T vynulován. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBC 6, k).

Syntaxe instrukce:

BRTC k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když T = 0 tak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

0001	11rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          bst     r3,5          ; přenáší bit 5 z r3 do příznaku T
          brtc    tclear        ; skok když bit byl vynulován
          .....
tclear:   nop                   ; cíl skoku (nedělá nic)
```

BRTS

BRANCH IF THE T FLAG IS SET

Skok, když příznak T je nastaven

Popis:

Podmíněný relativní skok. Testuje příznak T a provádí relativní skok vzhledem k PC, když je T nastaven. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBS 6, k).

Syntaxe instrukce:

BRTS k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když T = 1 tak, PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	k110

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
        bst r3, 5       ; přenese bit 5 z r3 do příznaku T
        brts tset       ; skok když bit byl nastaven
        .....
tset:   nop             ; cíl skoku (nedělá nic)
```

BRVC

BRANCH IF OVERFLOW CLEARED

Skok při vynulovaném příznaku přetečení

Popis:

Podmíněný relativní skok. Testuje příznak přetečení (V) a provádí relativní skok skok vzhledem k PC, když je V vynulován. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBC 3, k).

Syntaxe instrukce:

BRVC k

Interval operandu:

$-64 \leq k \leq +63$

Operace:

Když V = 0, pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	01kk	kkkk	k011

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          add     r3, r4        ; přičte r4 k r3
          brvc    noover        ; skok když není přetečení
          .....
noover;   nop                   ; cíl skoku (nedělá nic)
```

BRVS

BRANCH IF OVERFLOW SET

Skok při nastaveném příznaku přetečení

Popis:

Podmíněný relativní skok. Testuje příznak přetečení (V) a provádí relativní skok. Skok vzhledem k PC, když je V nastaven. Tato instrukce provádí relativní skok vzhledem k PC v obou směrech (PC –63 ≤ cíl ≤ PC +64). Parametr k je ofset vzhledem k PC a je vyjádřen v dvojkovém doplňkovém tvaru (je ekvivalentní s instrukcí BRBS 3, k).

Syntaxe instrukce:

BRVS k

Interval operandu:

–64 ≤ k ≤ +63

Operace:

Když V = 1, pak PC ← PC + k + 1, jinak PC ← PC + 1

Programový čítač:

PC ← PC + k + 1
PC ← PC + 1, když podmínka není splněna

Šestnáctibitový kód instrukce:

1111	00kk	kkkk	k011

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna.
2 když podmínka je splněna.

Příklad:

```
          add     r3, r4     ; přičte r4 k r3
          brvs    overfl     ; skok při přetečení
          .....
overfl:   nop                ; cíl skoku (nedělá nic)
```

BSET

BIT SET IN SREG

Nastavení bitu v SREG

Popis:

Nastavuje jeden příznak v SREG.

Syntaxe instrukce:

BSET s

Interval operandu:

$0 \leq s \leq 7$

Operace:

SREG(s) $\leftarrow$ 1

Programový čítač:

PC $\leftarrow$ PC + 1

Šestnáctibitový kód instrukce:

1001	0100	0sss	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
⇔	⇔	⇔	⇔	⇔	⇔	⇔	⇔

I: 1 když s = 7, jinak zůstává nezměněn.

T: 1 když s = 6, jinak zůstává nezměněn.

H: 1 když s = 5, jinak zůstává nezměněn.

S: 1 když s = 4, jinak zůstává nezměněn.

V: 1 když s = 3, jinak zůstává nezměněn.

N: 1 když s = 2, jinak zůstává nezměněn.

Z: 1 když s = 1, jinak zůstává nezměněn.

C: 1 když s = 0, jinak zůstává nezměněn.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
bset 6      ; nastavení příznaku T
bset 7      ; povolení všech přerušení
```

BST

BIT STORE FROM REGISTER TO T FLAG IN SREG

Přenesení bitu z registru do příznaku T v SREG

Popis:

Přenáší bit b z Rd do příznaku T v SREG (stavový registr)

Syntaxe instrukce:

BST Rd, b

Interval operandu:

$0 \leq d \leq 31$, $0 \leq b \leq 7$

Operace:

$T \leftarrow Rd(b)$

Programový čítač:

$PC \leftarrow PC + 1$

16 bit kód instrukce:

1111	101d	dddd	0bbb

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	⇔	-	-	-	-	-	-

T: 0, když bit b v Rd je vynulován, v opačném případě je nastaven na 1.

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
                    ; kopírování bitu
bst     r1, 2       ; přenáší bit 2 z r1 do příznaku T
bld     r0, 4       ; přenáší T do bitu 4 v r0
```


426410

CALL

LONG CALL TO A SUBROUTINE

(Vzdálené) volání podprogramu – jen u řady ATmega

Popis:

Volání podprogramu v dosahu dostupné paměti. Návratová adresa (pro instrukci následující po CALL) bude uložena do zásobníku (viz také RCALL). Ukazatel na zásobník užívá postdekrementální model volání.

Syntaxe instrukce:

CALL k

Interval operandu:

$0 \leq k \leq 64k$ v MCU s 16bitovým PC a programovou pamětí max. 128k.
$0 \leq k \leq 4M$ v MCU s 32bitovým PC a programovou pamětí max. 8M.

Operace:

PC ← k

Programový čítač:

PC ← k

Zásobník:

V MCU s 16bitovým PC a programovou pamětí max. 128k
STACK ← PC + 2
SP ← SP – 2, (2 byty, 16 bitů)

V MCU s 32bitovým PC a programovou pamětí max. 8M
STACK ← PC +2
SP ← SP – 3, (3 byty, 22 bitů)

426410

Třicetidvoubitový kód instrukce:

1001	010k	kkkk	111k
kkkk	kkkk	kkkk	kkkk

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

2 (4 byte)

Počet cyklů instrukce:

4 – v MCU s 16bitovým PC.
5 – v MCU s 22bitovým PC.

Příklad:

```
        mov  r16, r0   ; kopírování r0 do r16
        call check     ; volání podprogramu
        nop            ; pokračování (nedělá nic)
        .....
check:  cpi  r16, $42  ; kontrola, zda r16 má specifikovanou hodnotu
        breq error     ; skok při rovnosti
        ret            ; návrat z podprogramu
        .....
error:  rjmp error     ; nekonečná smyčka
```

CBI

CLEAR BIT IN I/O REGISTER

Nuluje bit v I/O registru

Popis:

Nuluje zadaný bit v I/O registru. Tato operace pracuje s dolními 32 I/O registry na adresách 0–31.

Syntaxe instrukce:

CBI A, b

Interval operandu:

$0 \leq A \leq 31$, $0 \leq b \leq 7$

Operace:

I/O(A, b) $\leftarrow$ 0

Programový čítač:

PC $\leftarrow$ PC + 1

Šestnáctibitový kód instrukce:

1001	1000	AAAA	Abbb

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
cbi  $12, 7    ; nuluje bit 7 v Portu D
```

CBR

CLEAR BITS IN REGISTER

Nuluje bity v registru

Popis:

Nuluje určené bity v registru Rd. Provádí logický součet AND mezi obsahy registru Rd a doplňkem ke konstantní masce K. Výsledek bude umístěn v registru Rd.

Syntaxe instrukce:

CBR Rd, K

Interval operandu:

16 ≤ **d** ≤ **31**, 0 ≤ K ≤ 255

Operace:

Rd ← Rd • ($FF – K)

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

(Viz ANDI s K doplňkem neboť CBR je jen jiné označení instrukce ANDI.)

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: N ⊕ V
pro znaménkové testy.

V: 0
vynulován.

426410

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
je nastaven, když výsledek je $00, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
cbr     r16, $F0        ; nuluje horní polovinu r16
cbr     r18, 1          ; nuluje bit 0 c r18
```

CLC

CLEAR CARRY FLAG

Nuluje příznak přenosu

Popis:

Nuluje příznak přenosu (C) v SREG (stavový registr).

Syntaxe instrukce:

CLC

Interval operandu:

Žádný

Operace:

C ← 0

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	1000	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	0

C: 0
příznak přenosu je vynulován.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add     r0, r0          ; přičte r0 samo k sobě
clc                     ; nuluje příznak přenosu
```

CLH

CLEAR HALF CARRY FLAG

Nuluje příznak přenosu mezi třetím a čtvrtým bitem

Popis:

Nuluje příznak přenosu mezi třetím a čtvrtým bitem (H) v SREG (stavový registr).

Syntaxe instrukce:

CLH

Interval operandu:

Žádný

Operace:

H ← 0

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	1000	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	0	-	-	-	-	-

H: 0
příznak přenosu mezi třetím a čtvrtým bitem je vynulován.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
clh       ; nuluje příznak přenosu mezi čtvrtým a pátým bitem
```

CLI

CLEAR GLOBAL INTERRUPT FLAG

Nuluje příznak povolení přerušení

Popis:

Zakazuje přerušení (I) v SREG (stavový registr).

Syntaxe instrukce:

CLI

Interval operandu:

Žádný

Operace:

$I \leftarrow 0$

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

1001	0100	1111	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
0	-	-	-	-	-	-	-

I: 0
příznak povolení přerušení je vynulován.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
cli                 ; zakazuje přerušení
in   r11, $16       ; čte port B
sei                 ; povoluje přerušení
```

CLN

CLEAR NEGATIVE FLAG

Nuluje příznak záporného výsledku

Popis:

Nuluje příznak záporného výsledku (N) v SREG (stavový registr).

Syntaxe instrukce:

CLN

Interval operandu:

Žádný

Operace:

N ← 0

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	1010	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	0	-	-

N: 0
příznak záporného výsledku je vynulován.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add r2, r3       ; přičte r3 k r2
cln              ; nuluje příznak záporného výsledku
```

CLR

CLEAR REGISTER

Nuluje registr

Popis:

Nuluje registr. Tato instrukce provádí exluzivní or mezi registrem a sama sebou. To vynuluje všechny bity v registru.

Syntaxe instrukce:

CLR Rd

Interval operandu:

$0 \leq d \leq 31$

Operace:

$Rd \leftarrow Rd \oplus Rd$

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

0010	01dd	dddd	dddd

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	0	0	0	1	-

S: 0
vynulován.

V: 0
vynulován.

N: 0
vynulován.

Z: 1
nahozen.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
          clr r18        ; nuluje r18
loop:     inc r18        ; zvětšuje r18 o 1
          .....
          cpi r18, $50   ; srovnání r18 s $50
          brne loop
```

CLS

CLEAR SIGNED FLAG

Nuluje znaménkový příznak

Popis:

Nuluje znaménkový příznak (S) v SREG (stavový registr).

Syntaxe instrukce:

CLS

Interval operandu:

Žádný

Operace:

S ← 0

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	1100	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	0	-	-	-	-

S: 0
znaménkový příznak je vynulován.

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add r2, r3       ; přičte r3 k r2
cls              ; nuluje znaménkový bit
```

CLT

CLEAR T FLAG

Nuluje příznak T

Popis:

Nuluje příznak T v SREG (stavový registr).

Syntaxe instrukce:

CLT

Interval operandu:

Žádný

Operace:

T ← 0

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	1100	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	0	-	-	-	-	-	-

T: 0
příznak T je vynulován.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
clt          ; nuluje příznak T
```

CLV

CLEAR OVERFLOW FLAG

Nuluje příznak přetečení

Popis:

Nuluje příznak přetečení (V) v SREG (stavový registr).

Syntaxe instrukce:

CLV

Interval operandu:

Žádný

Operace:

V ← 0

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	1011	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	0	-	-	-

V: 0
nuluje příznak přetečení.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add r2, r3       ; přičte r3 k r2
clv              ; nuluje příznak přetečení
```

CLZ

CLEAR ZERO FLAG

Nuluje příznak nuly

Popis:

Nuluje příznak nuly (Z) v SREG (stavový registr).

Syntaxe instrukce:

CLZ

Interval operandu:

Žádný

Operace:

Z ← 0

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	1001	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	0	-

Z: 0
příznak nuly je vynulován.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add  r2, r3          ; přičte r3 k r2
clz                  ; nuluje příznak nuly
```

COM

ONE'S COMPLEMENT

Jednotkový doplněk

Popis:

Tato instrukce vytváří jedničkový doplněk registru Rd.

Syntaxe instrukce:

COM Rd

Interval operandu:

$0 \leq d \leq 31$

Operace:

Rd ← $FF – Rd

Programový čítač:

PC ← PC+1

Šestnáctibitový kód instrukce:

1001	010d	dddd	0000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	1

S: $N \oplus V$
pro znaménkové testy.

V: 0
vynulován.

426410

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: 1
nastaven.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
        com     r4      ; provede jednotkový doplněk r4
        breq    zero    ; skok při nule
        .....
zero:   nop             ; cíl skoku (nic nedělá)
```

CP

COMPARE

Srovnání

Popis:

Tato instrukce provádí srovnání dvou registrů Rd a Rr. Obsah žádného z obou registrů není změněn. Všechny podmíněné skoky mohou být použity po skončení této instrukce.

Syntaxe instrukce:

CP Rd, Rr

Interval operandu:

$0 \leq d \leq 31$, $0 \leq r \leq 31$

Operace:

Rd – Rr

Programový čítač:

PC ← PC + 1

16 bit kód instrukce:

0001	01rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
nastaven, když byl přenos ze třetího bitu, jinak je vynulován.

S: $N \oplus V$
pro znaménkové testy.

V: $Rd7 \bullet \overline{Rd7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
nastaven, když výsledkem operace je přetečení dvojkového doplňku, jinak je vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
nastaven, když absolutní hodnota obsahu Rr je větší než absolutní hodnota obsahu Rd, jinak je vynulován.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
          cp r4 , r19      ; srovnává r4 s r19
          brne noteq       ; skok když r4 <> r19
          .....
noteq:    nop              ; cíl skoku (nedělá nic)
```

CPC

COMPARE WITH CARRY

Srovnání s přenosem

Popis:

Tato instrukce provádí srovnání dvou registrů Rd a Rr a také bere v úvahu předchozí přenos. Obsah žádného z obou registrů není změněn. Všechny podmínečné skoky mohou být použity po skončení této instrukce.

Syntaxe instrukce:

CPC Rd, Rr

Interval operandu:

$0 \le d \le 31$, $0 \le r \le 31$

Operace:

Rd – Rr – C

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0000	01rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
nastaven, když byl přenos ze třetího bitu, jinak je vynulován.

426410

S: $N \oplus V$
pro znaménkové testy.

V: $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
nastaven, když výsledkem operace je přetečení dvojkového doplňku, jinak je vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
předchozí hodnota zůstává zachována, když výsledek je nula, jinak je vynulován. Výsledek Zero bitu závisí i na předešlém stavu tohoto bitu. Tím se MCU AVR liší např. od řady x51.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
nastaven, když absolutní hodnota obsahu Rd je větší než absolutní hodnota obsahu Rd, jinak je vynulován.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
                    ; srovnává r3:r2 s r1:r0
       cp   r2, r0  ; srovnává dolní byty
       cpc  r3, r1  ; srovnává horní byty
       brne noteq   ; skok při nerovnání se
       .....
noteq: nop          ; cíl skoku (nedělá nic)
```

CPI

COMPARE WITH IMMEDIATE

Srovnání s konstantou

Popis:

Tato instrukce provádí srovnání registru Rd a konstanty. Obsah registru není měněn. Všechny podmíněné skoky mohou být použity po skončení této instrukce.

Syntaxe instrukce:

CPI Rd, K

Interval operandu:

16 ≤ **d** ≤ **31**, 0 ≤ K ≤ 255

Operace:

Rd – K

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0011	KKKK	dddd	KKKK

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$
nastaven, když byl přenos ze třetího bitu, jinak je vynulován.

S: $N \oplus V$
pro znaménkové testy.

V: $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$
nastaven, když výsledkem operace je přetečení dvojkového doplňku, jinak je vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$
nastaven, když absolutní hodnota obsahu K je větší než absolutní hodnota obsahu Rd, jinak je vynulován.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
          cpi     r19, 3    ; porovnání r19 s 3
          brne    error     ; skok když r19<>3
          .....
error:    nop               ; cíl skoku (nedělá nic)
```

CPSE

COMPARE SKIP IF EQUAL

Přeskakuje po porovnání na rovnost

Popis:

Tato instrukce provádí porovnání mezi dvěma registry Rd a Rr, přeskakuje následující instrukci, když Rd = Rr.

Syntaxe instrukce:

CSPE Rd, Rr

Interval operandu:

$0 \leq d \leq 31$, $0 \leq r \leq 31$

Operace:

Když Rd = Rr tak PC ← PC + 2 (nebo 3), jinak PC ← PC + 1

Programový čítač:

PC ← PC + 1	podmínka není splněna, neprovádí se přeskok.
PC ← PC + 2	přeskakuje jednoslovovou instrukci.
PC ← PC + 3	přeskakuje dvouslovovou instrukci.

Šestnáctibitový kód instrukce:

0001	00rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna a nepřeskakuje se.
2 když podmínka je splněna a přeskakuje se instrukce délky 1 slova.
3 když podmínka je splněna a přeskakuje se instrukce délky 2 slova.

Příklad:

```
inc r4              ; inkrementuje r4
cpse r4, r0         ; srovnává r4 a r0
neg r4              ; provádí se pouze když r4<>r0
nop                 ; pokračování (nedělá nic)
```

DEC

DECREMENT

Dekrementace, odečtení 1

Popis:

Odečítá jedničku – 1 od obsahu registru Rd a výsledek ukládá do cílového registru Rd. Příznak C z SREG se neúčastní operace, toto dovoluje instrukci DEC být užita coby čítač smyčky při výpočtech s násobnou přesností. Když pracuje s bezznaménkovými hodnotami, pak lze považovat za korektní pouze skoky BREQ a BRNE.

Když pracuje s dvojkově komplementárními hodnotami, tak všechny skoky závislé na znaménku jsou přístupné.

Syntaxe instrukce:

DEC Rd

Interval operandu:

$0 \leq d \leq 31$

Operace:

$Rd \leftarrow Rd - 1$

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

1001	010d	dddd	1010

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	-

426410

S: $N \oplus V$
pro znaménkové testy.

V: $\overline{R7} \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0$
nastaven, když výsledek operace způsobí přetečení dvojkového doplňku operace, jinak je vynulován. Přetečení dvojkového doplňku nastane právě když Rd bylo $80 před operací.

N: R7
nastaven, když MSB výsledku je nastaveno, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

R (výsledek) se rovná Rd po operaci.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
          ldi  r17, $10     ; přesouvá konstantu do r17
loop:     add  r1, r2       ; přičítá r2 k r1
          dec  r17          ; dekrement r17
          brne loop         ; skok když r17<>0
          nop               ; pokračuje (nic nedělá)
```

EICALL

EXTENDED INDIRECT CALL TO SUBROUTINE

Rozšířené nepřímé volání podprogramu

Popis:

Nepřímé volání podprogramu, na který ukazuje pointer registr Z (16 bitů) ze souboru registrů. Tato instrukce dovoluje nepřímé volání v celém prostoru programové paměti. Tato instrukce není implementována v MCU s 2bytovým PC, viz ICALL. Ukazatel zásobníku užívá postdekrementační model v průběhu EICALL.

Syntaxe instrukce:

EICALL

Interval operandu:

Žádný

Operace:

PC (15 : 0) ← Z (15 : 0)
PC (21 : 16) ← EIND

Programový čítač:

Viz operace.

Zásobník:

STACK ← PC + 1
SP ← SP – 3 (3 byte, 22 bitů)

Šestnáctibitový kód instrukce:

1001	0101	0001	1001

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

4 (je implementováno pouze u MCU s 22bitovým PC).

Příklad:

```
ldi  r16, $05          ; nastaví EIND a Z pointer
out  EIND, r16
ldi  r30, $00
ldi  r31, $10
eicall                 ; volání na $051000
```

EIJMP

EXTENDED INDIRECT JUMP

Rozšířený nepřímý skok

Popis:

Nepřímý skok na adresu, na kterou ukazuje registrový ukazatel Z (16 bitů) v souboru registrů a registr EIND v I/O prostoru. Tato instrukce dovoluje nepřímý skok v celém prostoru programové paměti.

Syntaxe instrukce:

EIJMP

Interval operandu:

Žádný

Operace:

PC (15:0) ← Z (15:0)
PC (21:16) ← EIND

Programový čítač:

Viz operace.

Zásobník:

Nepoužívá se.

Šestnáctibitový kód instrukce:

1001	0100	0001	1001

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
ldi  r16, $05       ; nastaví EIND a Z pointer
out  EIND, r16
ldi  r30, $00
ldi  r31, $10
eijmp               ; skok na $051000
```

ELPM

EXTENDED LOAD PROGRAM MEMORY

Přesun z rozšířené programové paměti

Popis:

Přenese jeden byte, na který ukazuje registr Z a registr RAMPZ v I/O prostoru, a umisťuje tento byte v cílovém registru Rd. Tato instrukce provádí přenos konstanty z programové paměti na kterou ukazuje RAMPZ:Z. Programová paměť je organizována v 16bitových slovech a nejméně významný bit v pointeru Z vybírá buď dolní byte (0) nebo horní byte (1). Tato instrukce může adresovat celý programový paměťový prostor. Registrový ukazatel Z může být buď ponechán nezměněn operací, nebo může být inkrementován. Inkrementace se aplikuje na celé 24bitové spojení RAMPZ a registrového ukazatele Z.

Není definován výsledek těchto operací:

ELPM r30, Z+

ELPM r31, Z+

Operace:

(i) R0 ← (RAMPZ:Z)

(ii) Rd ← (RAMPZ:Z)

(iii) Rd ← (RAMPZ:Z) (RAMPZ:Z) ← (RAMPZ:Z) + 1

Komentář:

RAMPZ:Z: nezměněn, R0 implicitně cílový registr

RAMPZ:Z: nezměněn

RAMPZ:Z: post inkrementovaný

Syntaxe:

(i) **ELPM**

(ii) **ELPM** Rd, Z

(iii) **ELPM** Rd, Z+

426410

Operandy:

Žádný, R0 implicitně

$0 \leq d \leq 31$

$0 \leq d \leq 31$

Programový čítač:

PC ← PC + 1

PC ← PC + 1

PC ← PC + 1

Šestnáctibitový kód instrukce:

(i)	1001	0101	1101	1000
(ii)	1001	000d	dddd	0110
(iii)	1001	000d	dddd	0111

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

3

Příklad:

```
clr  r16          ; nuluje RAMPZ
out  RAMPZ, r16
clr  r31          ; nuluje horní byte Z
ldi  r30, $F0     ; nastavuje dolní byte Z
elpm r16, z+      ; přenáší konstantu z programové paměti na kterou
                  ; ukazuje RAMPZ:Z (r31:r30)
```

EOR

EXCLUSIVE OR

Výlučné OR

Popis:

Vytváří logický EOR mezi obsahy registrů Rd a Rr a výsledek umisťuje do cílového registru Rd.

Syntaxe instrukce:

EOR Rd, Rr

Interval operandu:

$0 \le d \le 31$, $0 \le r \le 31$

Operace:

$Rd \leftarrow Rd \oplus Rr$

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

0010	01rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: $N \oplus V$
pro znaménkové testy.

426410

V: 0
vynulován.

N: R7
nastaven, když je výsledek nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
eor  r4, r4         ; nuluje r4
eor  r0, r22        ; bitové výlučné or mezi r0 a r22
```

ESPM

EXTENDED STORE PROGRAM MEMORY

Přesun do rozšířené programové paměti

Popis:

ESPM může být užito k vymazání stránky v programové paměti, k zápisu stránky do programové paměti (která je již vymazána) a nastaví boot loader lock bity.

U některých MCU může být do programové paměti zapsáno najednou jen jedno slovo, u jiných MCU celá stránka může být programována současně po prvním naplnění dočasného stránkového bufferu.

Ve všech případech, programová paměť musí být vymazana tak, že se najednou vymaže celá stránka. Při mazání programové paměti, registry RAMPZ a Z jsou použity jako adresa stránky.

Když se provádí zápis do programové paměti, registry RAMPZ a Z jsou užity jako stránkové nebo slovové adresy a registrová dvojice R1 : R0 je použita jako data. Když se nastavují boot loader lock bity, registrová dvojice R1 : R0 je užita jako data. Detailní popis užití ESPM najdete v dokumentaci. Tato instrukce může adresovat celou programovou paměť.

Syntaxe instrukce:

ESPM

Interval operandu:

Žádný

Operace:

(i) (RAMPZ:Z) ← $FFFF

(ii) (RAMPZ:Z) ← R1:R0

(iii) (RAMPZ:Z) ← R1:R0

(iv) (RAMPZ:Z) ← TEMP

(v) BLBITS ← R1:R0

Komentář:

Maže stránku programové paměti.
Zapisuje slovo programové paměti.
Zapisuje dočasný stránkový buffer.

426410

Zapisuje dočasný strankový buffer do programové paměti.
Nastavuje boot loader lock bity.

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0101	1111	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

Závisí na operaci.

Příklad:

```
; tento příklad ukazuje ESPM zápis jednoho slova do zařízení
; se stránkovým zápisem
clr  r31            ; nuluje horní byte Z
clr  r30            ; nuluje dolní byte Z
ldi  r16,$F0        ; naplňuje registr RAMPZ
out  RAMPZ, r16
ldi  r16, $CF       ; naplňuje data pro zápis
mov  r1, r16 ldi r16, $FF
mov  r0, r16
ldi  r16, 03        ; povoluje ESPM, maže stránku
out  SPMCR, r16
espm                ; maže stránku začínající na $F00000
ldi  r16, $01       ; povoluje ESPM, zapisuje R1:R0 do dočasného
                    ; bufferu na $F00000
out  SPMCR, r16
espm                ; provádí ESPM, zapisuje R1:R0 do dočasného
                    ; bufferu na $F00000
ldi  r16, $05       ; povoluje ESPM, zapisuje stránku
out  SPMCR, r16
espm                ; provádí SPM, zapisuje dočasný buffer do
                    ; stránky programové paměti od $F00000
```

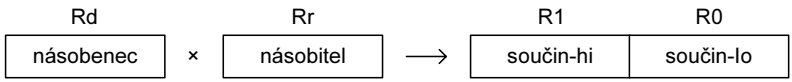
FMUL

FRACTIONAL MULTIPLY UNSIGNED

Násobení neznaménkových necelých čísel – jen u řady ATmega

Popis:

Tato instrukce provádí neznaménkové 8bit × 8bit → 16bit násobení a posouvá výsledek o jeden bit vlevo.



Nechť (N.Q) značí necelé číslo s N binárními číslicemi nalevo od tečky a Q binárními číslicemi napravo od tečky. Násobek mezi dvěma čísly ve tvaru (N1.Q1) a (N2.Q2) má výsledek ve tvaru (N1 + N2).(Q1 + Q2). Pro aplikace signálových procesorů je široce používán formát (1.7) pro vstupy, což dává pro výsledný součin formát (2.14). Proto se provádí posun výsledku (součinu) o jeden bit vlevo a tím se docílí stejný formát součinu jako mají násobenec a násobitel. Instrukce FMUL provádí operaci posunu se stejným počtem cyklů jako MUL. Násobenec Rd a násobitel Rr jsou dva registry obsahující neznaménková necelá čísla, kde implicitně je tečka mezi bitem 6 a bitem 7. Šestnáctibitový neznaménkový součin (necelý) má implicitně tečku mezi 14. a 15. bitem a je umístěn v R1 (horní byte) a R0 (dolní byte).

Syntaxe instrukce:

FMUL Rd, Rr

Interval operandu:

$16 \leq d \leq 23$, $16 \leq r \leq 23$

Operace:

R1 : R0 ← Rd × Rr (neznaménkový (1.15) ← neznaménkový (1.7) × nzn. (1.7))

Programový čítač:

PC ← PC + 1

426410

Šestnáctibitový kód instrukce:

0000	0011	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	⇔	⇔

Z: R16
nastaven, když bit 15 výsledku před posunem vlevo je nastaven, jinak je vynulován.

C: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $0000, jinak je vynulován.

R (výsledek) se rovná R0, R1 po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
fmul r23, r22  ; násobí beznaménkový r23 a r24 ve formátu (1.7),
movw r22, r0   ; kopíruje výsledek nazpět do r23:r22
               ; výsledek je ve formátu (1.15)
```

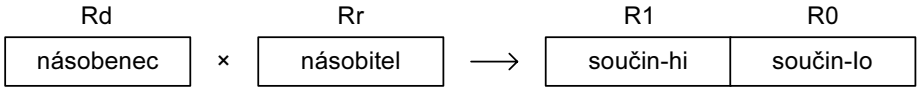
FMULS

FRACTIONAL MULTIPLY SIGNED

Násobení znaménkových necelých čísel – jen u řady ATmega

Popis:

Tato instrukce provádí znaménkové 8bit × 8bit → 16bit násobení a posouvá výsledek o jeden bit vlevo.



Nechť (N.Q) značí necelé číslo s N binárními číslicemi nalevo od tečky a Q binárními číslicemi napravo od tečky. Násobek mezi dvěma čísly ve tvaru (N1.Q1) a (N2.Q2) má výsledek ve tvaru (N1 + N2).(Q1 + Q2). Pro aplikace signálových procesorů je široce používán formát (1.7) pro vstupy, což dává pro výsledný součin formát (2.14). Proto se posouvá výsledek (součin) o jeden bit vlevo a tím se docílí stejný formát součinu a násobence a násobitele. Instrukce FMULS provádí operaci posunu se stejným počtem cyklů jako MULS. Násobenec Rd a násobitel Rr jsou dva registry obsahující znaménková necelá čísla, kde implicitně je tečka mezi bitem 6 a bitem 7. Šestnáctibitový znaménkový součin (necelý) má implicitně tečku mezi 14. a 15. bitem a je umístěn v R1 (horní byte) a R0 (dolní byte).

Syntaxe instrukce:

FMULS Rd, Rr

Interval operandu:

$16 \le d \le 23$, $16 \le r \le 23$

Operace:

R1 : R0 ← Rd × Rr (znaménkový (1.15) ← znaménkový (1.7) × zn. (1.7))

Programový čítač:

PC ← PC + 1

426410

Šestnáctibitový kód instrukce:

0000	0011	1ddd	0rrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	⇔	⇔

Z: R16
nastaven, když bit 15 výsledku před posunem vlevo je nastaven, jinak je vynulován.

C: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $0000, jinak je vynulován.

R (výsledek) se rovná R0, R1 po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
fmuls   r23, r22    ; násobí znaménkový r23 a r24 ve formátu
                    ; (1.7),
                    ; výsledek je ve formátu (1.15)
movw    r22, r0     ; kopíruje výsledek nazpět do r23:r22
```

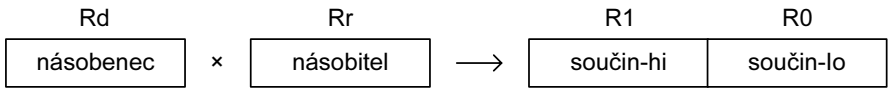
FMULSU

FRACTIONAL MULTIPLY SIGNED WITH UNSIGNED

Násobení znaménkových a neznaménkových necelých čísel – jen u řady ATmega

Popis:

Tato instrukce provádí znaménkové 8bit × 8bit → 16bit násobení a posouvá výsledek o jeden bit vlevo.



Nechť (N.Q) značí necelé číslo s N binárními číslicemi nalevo od tečky a Q binárními číslicemi napravo od tečky. Násobek mezi dvěma čísly ve tvaru (N1.Q1) a (N2.Q2) má výsledek ve tvaru (N1 + N2).(Q1 + Q2). Pro aplikace signálových procesorů je široce používán formát (1.7) pro vstupy, což dává pro výsledný součin formát (2.14). Proto se posouvá výsledek (součin) o jeden bit vlevo a tím se docílí stejný formát součinu a násobence a násobitele. Instrukce FMULSU provádí operaci posunu se stejným počtem cyklů jako MULSU. Násobenec Rd a násobitel Rr jsou dva registry obsahující necelá čísla, kde implicitně je tečka mezi bitem 6 a bitem 7. Násobenec Rd je znaménkové necelé číslo, a násobitel Rd je neznaménkové necelé číslo. Šestnáctibitový znaménkový součin (necelý) má implicitně tečku mezi 14. a 15. bitem a je umístěn v R1 (horní byte) a R0 (dolní byte).

Syntaxe instrukce:

FMULSU Rd, Rr

Interval operandu:

$16 \leq d \leq 23$, $16 \leq r \leq 23$

Operace:

R1 : R0 ← Rd × Rr (znaménkový (1.15) ← znaménkový (1.7) × zn. (1.7))

426410

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0000	0011	1ddd	1rrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	⇔	⇔

Z: R16
nastaven, když bit 15 výsledku před posunem vlevo je nastaven, jinak je vynulován.

C: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $0000, jinak je vynulován.

R (výsledek) se rovná R0, R1 po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
fmulSU r23, r22    ; násobí znaménkový r23 s neznaménkovým r24
                   ; ve formátu (1.7),
                   ; znaménkový výsledek je ve formátu (1.15)
movw   r22, r0     ; kopíruje výsledek nazpět do r23:r22
```

ICALL

INDIRECT CALL TO SUBROUTINE

Nepřímé volání podprogramu

Popis:

Nepřímé volání podprogramu, na který ukazuje registrový ukazatel Z (16bitový) ze souboru registrů. Registrový ukazatel Z má šířku 16 bitů a dovoluje volat podprogram v dolních 64K slovech (128K bytů) programového paměťového prostoru. Ukazatel zásobníku užívá postdekrementační model během ICALL.

Syntaxe instrukce:

ICALL

Interval operandu:

Žádný

Operace:

(i) PC (15:0) ← Z (15:0)
MCU s 16bitovým PC, max. 128K byty programové paměti.

(ii) PC (15:0) ← Z (15:0)
MCU s 22bitovým PC, max. 8M programovou pamětí.

PC (21:16) ← 0

Programový čítač:

Viz popis operace.

Zásobník:

(i) STACK ← PC + 1
SP ← SP – 2 (2 byty, 16 bitů)

(ii) STACK ← PC + 1
SP ← SP – 3 (3 byty, 22 bitů)

Šestnáctibitový kód instrukce:

1001	0101	0000	1001

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

3 – u MCU s 16bitovým PC.

4 – u MCU s 22bitovým PC.

Příklad:

```
mov r30, r0      ; nastavuje ofset do tabulky volání
icall            ; volá podprogram na který ukazuje r31:r30
```

IJMP

INDIRECT JUMP

Nepřímý skok

Popis:

Nepřímý skok na adresu, na kterou ukazuje registrový ukazatel Z (16bitů) ze souboru registrů. Registrový ukazatel Z má šířku 16 bitů a dovoluje skok v dolních 64K slovech (128K bytech) programové paměti.

Syntaxe instrukce:

IJMP

Interval operandu:

Žádný

Operace:

(i) PC (15:0) ← Z (15:0)
MCU s 16bitovým PC, max. 128K byty programové paměti.

(ii) PC (15:0) ← Z (15:0)
MCU s 22bitovým PC, max. 8M programovou pamětí.

PC (21:16) ← 0

Programový čítač:

Viz popis operace.

Šestnáctibitový kód instrukce:

1001	0100	0000	1001

426410

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
mov r30, r0    ; nastavení ofsetu v tabulce skoků
ijmp           ; skok na podprogram odkazovaný ukazatelem r31:r30
```

IN

LOAD AN I/O LOCATION TO REGISTER

Přesun dat z I/O prostoru do registru

Popis:

Přesouvá data z I/O prostoru (porty, časovače, konfigurační registry atd.) do registru Rd ze souboru registrů.

Syntaxe instrukce:

IN Rd, A

Interval operandu:

$0 \leq d \leq 31$, $0 \leq A \leq 63$

Operace:

Rd ← I/O (A)

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1011	0AAd	dddd	AAAA

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
        in   r25, $16     ; čte Port B
        cpi  r25, 4       ; srovnává čtenou hodnotu s konstantou
        breq exit         ; skok, když r25=4
        .....
exit:   nop               ; cíl skoku (nedělá nic)
```

INC

INCREMENT

Inkremenrace, zvýšení o 1

Popis:

Přičítá jedničku 1 k obsahu registru Rd a výsledek umisťuje v cílovém registru Rd. Příznak C v SREG není ovlivňován touto operací, což dovoluje instrukci INC použít pro čítač smyčky při výpočtech se zvýšenou přesností. Když pracuje s neznaménkovými čísly, tak pouze skoky BREQ a BRNE jsou korektní, když pracuje s dvojkově doplňkovými hodnotami, tak všechny skoky, včetně skoků závislých na znaménku, jsou přípustné.

Syntaxe instrukce:

INC Rd

Interval operandu:

$0 \leq d \leq 31$

Operace:

$Rd \leftarrow Rd + 1$

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

1001	010d	dddd	0011

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	-

426410

S: $N \oplus V$
pro znaménkové testy.

V: $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když přeteče dvojkový doplněk výsledku operace, v opačném případě je vynulován. Přetečení dvojkového doplňku nastane právě když Rd byl $7F před operací.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
       clr r22        ; nuluje r22
loop:  inc r22        ; inkrementuje r22
       .....
       cpi r22, $4F   ; srovnává r22 a $4F
       brne loop      ; skok, když se nerovná
       nop            ; pokračovat (nedělá nic)
```

JMP

JUMP

Skok – jen u řady ATmega

Popis:

Provádí skok na adresu v celé 4M (slovové) programové paměti. Viz také RJMP.

Syntaxe instrukce:

JMP k

Interval operandu:

$0 \leq k \leq 4M$

Operace:

PC ← k

Programový čítač:

PC ← k

Třicetidvoubitový kód instrukce:

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

2 (4 byte)

Počet cyklů instrukce:

3

Příklad:

```
        mov r1, r0      ; kopíruje r0 do r1
        jmp farplc      ; nepodmíněný skok
        .....
farplc: nop             ; cíl skoku (nedělá nic)
```

LD

LOAD INDIRECT FROM DATA SPACE TO REGISTER USING INDEX X

Přesun dat z adresy X do registru

Popis:

Přesouvá jeden byte nepřímým způsobem z datového prostoru do registru. U MCU s SRAM se datový prostor skládá ze souboru registrů (obecné pracovní registry a I/O registry), vnitřní paměti SRAM a vnější paměti SRAM (je-li použita). U MCU bez SRAM se datový prostor skládá pouze ze souboru registrů. Paměť EEPROM má oddělený adresový prostor.

Umístění dat je ukazováno pomocí registrového ukazatele X (16 bitů). Přístup k paměti je omezen na normální datový segment 64K bytů. Pro přístup k dalšímu datovému segmentu u MCU s více než 64Kbytovým datovým prostorem je potřeba změnit RAMPX v registrech v souboru I/O registrů.

Registrový ukazatel X může být touto operací buď nezměněn, nebo může být post inkrementován nebo pre dekrementován. Tyto způsoby nepřímé adresace pomocí registru X (pointeru) se využívají pro přístup k polím, tabulkám a ukazateli na zásobník. U MCU, jejichž paměťový prostor SRAM nepřesáhne 256 byte se využívá (mění obsah) jen dolní byte registru X, horní byte tato instrukce nepoužívá a tak ho lze používat pro jiné účely. U MCU s paměťovým prostorem SRAM větším než 64K byte se využívá registr RAMPX v I/O.

Není definován výsledek pro tyto kombinace:

LD r26, X+

LD r27, X+

LD r26, –X

LD r27, –X

Syntaxe instrukce:

(i) LD Rd, X

(ii) LD Rd, X+

(iii) LD Rd, –X

Interval operandu:

$0 \leq d \leq 31$

426410

Operace:

(i) Rd ← (X)

(ii) Rd ← (X) X ← X + 1

(iii) X ← X – 1 Rd ← (X)

Komentář:

X: nezměněn

X: post inkrementován

X: pre dekrementován

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

(i)	1001	000d	dddd	1100
(ii)	1001	000d	dddd	1101
(iii)	1001	000d	dddd	1110

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

2

Příklad:

```
clr  r27       ; nuluje horní byte X
ldi  r26, $60  ; nastavuje dolní byte X na $60
ld   r0, X+    ; naplňuje r0 daty z datového prostoru umístěného
               ; na $60 (X post inkr.)
ld   r1, X     ; naplňuje r1 daty z $61
ldi  r26, $63  ; nastavuje dolní byte X na $63
ld   r2, X     ; plní r2 daty z $63
ld   r3, -X    ; plní r3 daty z $62 (X pre dekr.)
```

LD (LDD)

LOAD INDIRECT FROM DATA SPACE TO REGISTER USING INDEX Y

Přesun dat z adresy Y do registru

Popis:

Přesouvá jeden byte nepřímým způsobem s posunem či bez posunu, datového prostoru do registru. U MCU s SRAM se datový prostor skládá ze souboru registrů (obecné pracovní registry a I/O registry), vnitřní paměti SRAM a vnější paměti SRAM (je-li použita). U MCU bez SRAM se datový prostor skládá pouze ze souboru registrů. Paměť EEPROM má oddělený adresový prostor.

Umístění dat je ukazováno pomocí registrového ukazatele Y (16 bitů). Přístup k paměti je omezen na normální datový segment 64K bytů. Pro přístup k dalšímu datovému segmentu u MCU s více než 64Kbytovým datovým prostorem je potřeba změnit RAMPY v registrech v souboru I/O registrů.

Registrový ukazatel Y může být touto operací buď nezměněn, nebo může být post inkrementován nebo pre dekrementován. Tyto způsoby nepřímé adresace pomocí registru Y (pointeru) se využívají pro přístup k polím, tabulkám a ukazateli na zásobník. U MCU, jejichž paměťový prostor SRAM nepřesáhne 256 byte se využívá (mění obsah) jen dolní byte registru Y, horní byte tato instrukce nepoužívá a tak ho lze používat pro jiné účely. U MCU s paměťovým prostorem SRAM větším než 64K byte se využívá registr RAMPY v I/O prostoru, k celkové 24bitové adrese takového MCU se přičítá posun.

Není definován výsledek pro tyto kombinace:

LD r28, Y+

LD r29, Y+

LD r28, –Y

LD r29, –Y

Syntaxe instrukce:

(i) LD Rd, Y

(ii) LD Rd, Y+

(iii) LD Rd, –Y

(iiii) LDD Rd, Y + q

426410

Interval operandu:

$0 \le d \le 31$

$0 \le q \le 63$

Operace:

(i) Rd ← (Y)

(ii) Rd ← (Y) Y ← Y + 1

(iii) Y ← Y – 1 Rd ← (Y)

(iiii) Rd ← (Y + q)

Komentář:

Y: nezměněn.

Y: post inkrementován.

Y: pre dekrementován.

Y: nezměněn, q: nahrazen.

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

(i)	1000	000d	dddd	1000
(ii)	1001	000d	dddd	1001
(iii)	1001	000d	dddd	1010
(iiii)	10q0	qq0d	dddd	1qqq

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
clr  r29       ; nuluje horní byte Y
ldi  r28, $60  ; nastavuje dolní byte Y na $60
ld   r0, Y+    ; naplňuje r0 daty z datového prostoru umístěného
               ; na $60 (Y post inkr.)
ld   r1, Y     ; naplňuje r1 daty z $61
ldi  r28, $63  ; nastavuje dolní byte Y na $63
ld   r2, Y     ; plní r2 daty z $63
ld   r3, -Y    ; plní r3 daty z $62 (Y pre dekr.)
ldd  r4, Y+2   ; plní r3 daty z $64
```

LD (LDD)

LOAD INDIRECT FROM DATA SPACE TO REGISTER USING INDEX Z

Přesun dat z adresy Z do registru

Popis:

Přesouvá jeden byte nepřímým způsobem, s posunem či bez posunu, z datového prostoru do registru. U MCU s SRAM se datový prostor skládá ze souboru registrů (obecné pracovní registry a I/O registry), vnitřní paměti SRAM a vnější paměti SRAM (je-li použita). U MCU bez SRAM se datový prostor skládá pouze ze souboru registrů. Paměť EEPROM má oddělený adresový prostor.

Umístění dat je ukazováno pomocí registrového ukazatele Z (16 bitů). Přístup k paměti je omezen na normální datový segment 64K bytů. Pro přístup k dalšímu datovému segmentu u MCU s více než 64Kbytovým datovým prostorem je potřeba změnit RAMPZ v registrech v souboru I/O registrů.

Registrový ukazatel Z může být touto operací buď nezměněn, nebo může být post inkrementován nebo pre dekrementován. Tyto způsoby nepřímé adresace pomocí registru Z (pointeru) se využívají pro přístup k ukazateli na zásobník. Protože registr Z může být použit pro nepřímé volání podprogramu či nepřímé skoky, je často výhodné použít jako ukazatel na zásobník pointer X nebo Y. U MCU, jejichž paměťový prostor SRAM nepřesáhne 256 byte se využívá (mění obsah) jen dolní byte registru Z, horní byte tato instrukce nepoužívá a tak ho lze používat pro jiné účely. U MCU s paměťovým prostorem SRAM větším než 64K byte se využívá registr RAMPZ v I/O prostoru, k celkové 24bitové adrese takového MCU se přičítá posun. U MCU s více než 64Kbytovou programovou pamětí a více než 64Kbytovou datovou pamětí je registr RAMPZ používán v instrukcích ELPM a ESPM (a není naopak použit v instrukci ST).

Není definován výsledek pro tyto kombinace:

LD r30, Z+

LD r31, Z+

LD r30, –Z

LD r31, –Z

Syntaxe instrukce:

(i) LD Rd, Z
(ii) LD Rd, Z+
(iii) LD Rd, –Z
(iiii) LDD Rd, Z + q

Interval operandu:

$0 \leq d \leq 31$
$0 \leq q \leq 63$

Operace:

(i) Rd ← (Z)
(ii) Rd ← (Z) Z ← Z + 1
(iii) Z ← Z – 1 Rd ← (Z)
(iiii) Rd ← (Z + q)

Komentář:

Z: nezměněn.
Z: post inkrementován.
Z: pre dekrementován.
Z: nezměněn, q: nahrazen.

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

(i)	1000	000d	dddd	0000
(ii)	1001	000d	dddd	0001
(iii)	1001	000d	dddd	0010
(iiii)	10q0	qq0d	dddd	0qqq

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
clr  r31       ; nuluje horní byte Z
ldi  r30, $60  ; nastavuje dolní byte Z na $60
ld   r0, Z+    ; naplňuje r0 daty z datového prostoru umístěného
               ; na $60 (Z post inkr.)
ld   r1, Z     ; naplňuje r1 daty z $61
ldi  r30, $63  ; nastavuje dolní byte Z na $63
ld   r2, Z     ; plní r2 daty z $63
ld   r3, -Z    ; plní r3 daty z $62 (Z pre dekr.)
ldd  r4, Z+2   ; plní r3 daty z $64
```

LDI

LOAD IMMEDIATE

Přesun konstanty do registru

Popis:

Přesouvá 8bitovou konstantu přímo do registru 16 až 31.

Syntaxe instrukce:

LDI Rd, K

Interval operandu:

16 ≤ **d** ≤ **31**, 0 ≤ K ≤ 255

Operace:

Rd ← K

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1110	KKKK	dddd	KKKK

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
clr  r31          ; nuluje horní byte Z
ldi  r30, $F0     ; nastavuje dolní byte Z na $F0
lpm               ; přesouvá konstantu z programové
                  ; paměti odkazované v Z
```

LDS

LOAD DIRECT FROM DATA SPACE

Přímý přesun z datového prostoru

Popis:

Naplňuje registr jedním bytem z datového prostoru. Pro MCU s SRAM, datový prostor obsahuje soubor registrů, I/O paměť a vnitřní SRAM (vnější SRAM, je-li použita). Pro MCU bez SRAM, datový prostor sestává pouze z souboru registrů. Paměť EEPROM má oddělený adresový prostor.

Musí být použity 16bitové adresy. Přístup do paměti je omezen na aktuální datový segment 64K bytů. Instrukce LDS užívá RAMPD registr k přístupu do paměti nad 64K. K přístupu k dalším datovým segmentům u zařízení s více než 64K byte datového prostoru, je registr RAMPD v I/O prostoru měněn.

Syntaxe instrukce:

LDS Rd, k

Interval operandu:

$0 \le d \le 31$, $0 \le k \le 65535$

Operace:

Rd ← (k)

Programový čítač:

PC ← PC + 2

Třicetidvoubitový kód instrukce:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

2 (4 byte)

Počet cyklů instrukce:

2

Příklad:

```
lds  r2, $FF00        ; naplní r2 konstantou z $FF00
add  r2, r1           ; přičte r1 k r2
sts  $FF00, r2        ; zapíše nazpátek
```

LPM

LOAD PROGRAM MEMORY

Přesun z programové paměti

Popis:

Přenáší jeden byte odkazovaný registrem Z do cílového registru Rd. Tato instrukce umožňuje naplnění 100 % prostoru inicializační či datovou konstantou. Programová paměť je organizována v 16bitových slovech a nejméně významný bit ukazatele Z vybírá buď dolní byte (0) nebo horní byte (1). Tato instrukce může adresovat prvních 64K bytů (32 K slov) programové paměti. Ukazatel Z může být nezměněn operací, nebo může být inkrementován. Inkrementace se neprovádí s registrem RAMPZ.

Výsledky těchto kombinací nejsou definovány:

LPM r30, Z+

LPM r31, Z+

Syntaxe instrukce:

(i) LPM

(ii) LPM Rd, Z

(iii) LPM Rd, Z+

Operandy:

Žádné, implicitně R0

$0 \leq d \leq 31$

$0 \leq d \leq 31$

Operace:

(i) R0 ← (Z)

(ii) Rd ← (Z)

(iii) Rd ← (Z) Z ← Z+1

426410

Komentář:

Z: nezměněn, R0 implicitní cílový registr.

Z: nezměněn.

Z: post inkrementován.

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

(i)	1001	0101	1100	1000
(ii)	1001	000d	dddd	0100
(iii)	1001	000d	dddd	0101

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

3

Příklad:

```
clr  r31            ; nuluje horní byte Z
ldi  r30, $F0       ; nastavuje dolní byte Z
lpm                 ; přenáší konstantu z programové paměti
                    ; odkazované Z (r31:r30)
```

LSL

LOGICAL SHIFT LEFT

Logický posun vlevo

Popis:

Posouvá všechny bity v Rd o jedno místo vlevo. Bit 0 je vynulován. Bit 7 je přesunut do příznaku C v SREG. Tato operace efektivně provádí násobení dvěma znaménkových i neznaménkových hodnot.

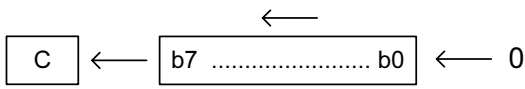
Syntaxe instrukce:

LSL Rd

Interval operandu:

$0 \leq d \leq 31$

Operace:



Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0000	11dd	dddd	dddd

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: $N \oplus V$
pro znaménkové testy.

V: $N \oplus C$
(pro N a C po posunu).

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: Rd7
nastaven, když před posunem byl MSB Rd nastaven, jinak je vynulován.

R (výsledek) se rovná Rd po skončemí operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add  r0, r4          ; přičte r4 k r0
lsl  r0              ; vynásobí r0 dvěma
```

LSR

LOGICAL SHIFT RIGHT

Logický posuv vpravo

Popis:

Posouvá všechny bity doprava. Bit 7 je vynulován. Bit 0 je přesunut do příznaku C v SREG. Tato operace efektivně dělí neznaménkové hodnoty dvěma. Příznak C může být užit k zaokrouhlení výsledku.

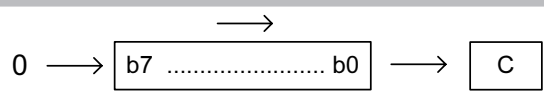
Syntaxe instrukce:

LSR Rd

Interval operandu:

$0 \leq d \leq 31$

Operace:



Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	010d	dddd	0110

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	0	⇔	⇔

426410

S: $N \oplus V$
pro znaménkové testy.

V: $N \oplus C$
(pro N a C po posunu).

N: 0

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: Rd0
nastaven, když před posunem byl LSB Rd nastaven, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add  r0, r4          ; přičte r4 k r0
lsr  r0              ; dělí r0 dvěma
```

MOV

COPY REGISTER

Překopírovává obsah registru

Popis:

Instrukce kopíruje obsah jednoho registru do druhého registru. Zdrojový registr Rr zůstává nezměněn, zatímto cílový registr Rd je naplněn kopií Rr.

Syntaxe instrukce:

MOV Rd, Rr

Interval operandu:

$0 \leq d \leq 31$, $0 \leq r \leq 31$

Operace:

Rd ← Rr

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0010	11rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
        mov r16, r0         ; kopíruje r0 do r16
        call check          ; volá podprogram
        .....
check:  cpi r16, $11        ; srovnává r16 s $11
        .....
        ret                 ; návrat z podprogramu
```

MOVW

COPY REGISTER WORD

Kopíruje obsah dvojice registrů – jen u řady ATmega

Popis:

Tato instrukce kopíruje jednu registrovou dvojici do druhé dvojice registrů. Zdrojová registrová dvojice Rr + 1 : Rr zůstáva nezměněna, přičemž do cílové dvojice registrů Rd + 1 : Rd se překopíruje obsah z Rr + 1 : Rr.

Syntaxe instrukce:

MOVW Rd, Rr

Operandy:

d ∈ (0,2 ... 30), r ∈ (0,2 ... 30)

Operace:

Rd + 1 : Rd ← Rr + 1, Rr

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0000	0001	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
        movw r16, r0     ; kopíruje r1:r0 do r17:r16
        call check       ; volá podprogram
        .....
check:  cpi r16, $11     ; porovnává r16 s $11
        .....
        cpi r17, $32     ; porovnává r17 s $32
        .....
        ret              ; návrat z podprogramu
```

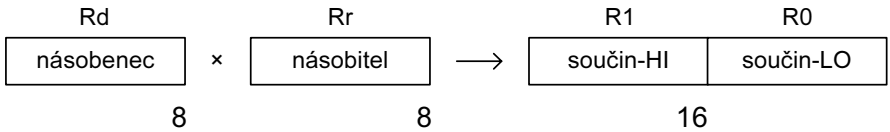
MUL

MULTIPLY UNSIGNED

Násobení neznaménkových čísel – jen u řady ATmega

Popis:

Tato instrukce vytváří 8bit × 8bit neznaménkové násobení.



Násobenec Rd a násobitel Rr jsou dva registry, obsahující neznaménková čísla. Šestnáctibitový součin je umístěn do R1 (horní byte) a R0 (dolní byte). Povšimněte si, že když násobenec a násobitel jsou vybrány z R0 a R1, výsledek je zapsán zpět do registrů R0 a R1, tj. původní obsah těchto registrů je přepsán výsledkem.

Syntaxe instrukce:

MUL Rd, Rr

Operandy:

$0 \le d \le 31$, $0 \le r \le 31$

Operace:

R1 : R0 ← Rd × Rr (neznaménkové ← neznaménkové × neznaménkové)

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	11rd	dddd	rrrr

426410

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	⇔	⇔

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

nastaven, když výsledek je $0000, jinak je vynulován.

C: R15

nastaven, když bit 15 výsledku je nastaven, jinak je vynulován.

R (výsledek) se rovná R1, R0 po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
mul  r5, r4          ; násobí neznaménkové r5 a r4
movw r4, r0          ; kopíruje výsledek nazpátky do r5:r4
```

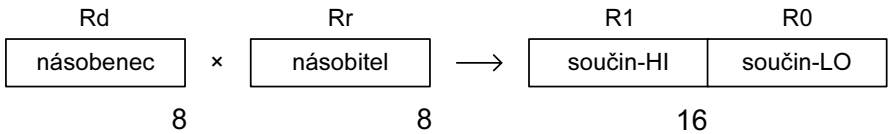
MULS

MULTIPLY SIGNED

Násobení znaménkových čísel – jen u řady ATmega

Popis:

Tato instrukce vytváří 8bit × 8bit znaménkové násobení.



Násobenec Rd a násobitel Rr jsou dva registry obsahující znaménková čísla. Šestnáctibitový součin je umístěn do R1 (horní byte) a R0 (dolní byte).

Syntaxe instrukce:

MULS Rd, Rr

Operandy:

$0 \le d \le 31$, $0 \le r \le 31$

Operace:

R1 : R0 ← Rd × Rr (znaménkové ← znaménkové × znaménkové)

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	⇔	⇔

Stavový registr (SREG):

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $0000, jinak je vynulován.

C: R15
nastaven, když bit 15 výsledku je nastaven, jinak je vynulován.

R (výsledek) se rovná R1, R0 po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
muls r21, r20          ; násobí neznaménkové r21 a r20
movw r20, r0           ; kopíruje výsledek nazpátky do r21:r20
```

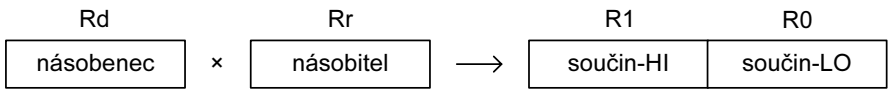
MULSU

MULTIPLY SIGNED WITH UNSIGNED

Násobení znaménkových a neznaménkových čísel – jen u řady ATmega

Popis:

Tato instrukce vytváří 8bit × 8bit násobení znaménkového a neznaménkového čísla



Násobenec Rd a násobitel Rr jsou dva registry. Násobenec Rd je znaménkové číslo a násobitel Rr neznaménkové číslo.

Šestnáctibitový součin je umístěn do R1 (horní byte) a R0 (dolní byte).

Syntaxe instrukce:

MULSU Rd, Rr

Operandy:

$0 \le d \le 31$, $0 \le r \le 31$

Operace:

R1 : R0 ← Rd × Rr (znaménkové ← znaménkové × neznaménkové)

Programový čítač:

PC ← PC + 1

426410

Šestnáctibitový kód instrukce:

0000	0011	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	⇔	⇔

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $0000, jinak je vynulován.

C: R15
nastaven, když bit 15 výsledku je nastaven, jinak je vynulován.

R (výsledek) se rovná R1, R0 po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
mulsu   r21, r20        ; násobí neznaménkové r21 a r20
movw    r20, r0         ; kopíruje výsledek nazpátky do r21:r20
```

NEG

TWO'S COMPLEMENT

Dvojkový doplněk

Popis:

Naplňuje obsah registru Rd jeho dvojkovým doplňkem, hodnota $80 se nezmění.

Syntaxe instrukce:

NEG Rd

Interval operandu:

$0 \leq d \leq 31$

Operace:

Rd ← $00 – Rd

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	010d	dddd	1,00E+00

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: R3 + Rd3
nastaven, když byla „výpůjčka“ z bitu 3, jinak je vynulován.

426410

S: $N \oplus V$
pro znaménkové testy.

V: $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven při přetečení dvojkového doplňku z implicitního odečítání od nuly, jinak je vynulován. Přetečení dvojkového doplňku nastane právě tehdy, když obsah registru po operaci (výsledek) je $80.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0
je nastaven, když je „výpůjčka“ v implicitním rozdílu od nuly, jinak je vynulován. Příznak C bude nastaven ve všech případech s výjimkou, kdy obsah registru po operaci je $00.

R (výsledek) se rovná Rd po operaci.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
            sub  r11, r0   ; odečte r0 od r11
            brpl positive  ; skok, když výsledek je kladný
            neg  r11       ; bere dvojkový doplněk r11
positive:   nop            ; cíl skoku (nedělá nic)
```

NOP

NO OPERATION

Prázdná instrukce

Popis:

Tato instrukce trvá jeden cyklus. Jinak neprovádí žádnou činnost.

Syntaxe instrukce:

NOP

Operand:

Žádný

Operace:

Žádná

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0000	0000	0000	0000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
clr  r16              ; nuluje r16
ser  r17              ; nastavuje r17
out  $18, r16         ; zapisuje nulu do portu B
nop                   ; čeká (nedělá nic)
out  $18, r17         ; zapisuje jedničku do portu B
```

ORI

LOGICAL OR WITH IMMEDIATE

Logické OR s konstantou

Popis:

Vytvoří logický součet OR mezi obsahem registru Rd a konstantou a výsledek ukládá do cílového registru Rd.

Syntaxe instrukce:

ORI Rd, K

Operandy:

16 ≤ **d** ≤ **31**, 0 ≤ K ≤ 255

Operace:

Rd ← Rd ∨ K

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0110	KKKK	dddd	KKKK

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: N ⊕ V
pro znaménkové testy.

V: 0
vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
ori  r16, $F0          ; nastavuje horní polovinu r16
ori  r17, 1            ; nastavuje bit 0 v r17
```

OR

Logical OR

Logické OR

Popis:

Vytvoří logický součet OR mezi obsahy registru Rd a registru Rr a výsledek umisťuje do cílového registru Rd.

Syntaxe instrukce:

OR Rd, Rr

Operandy:

$0 \leq d \leq 31$, $0 \leq r \leq 31$

Operace:

$Rd \leftarrow Rd \vee Rr$

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

0010	10rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: $N \oplus V$
pro znaménkové testy.

426410

V: 0
vynulován.

N: R7
nastaven když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
        or r15, r16      ; bitové OR mezi registry
        bst r15, 6       ; přesouvá bit 6 z r15 do příznaku T
        brts ok          ; skok, když je příznak T nastaven
        .....
ok:     nop              ; cíl skoku (nedělá nic)
```

OUT

STORE REGISTER TO I/O LOCATION

Přesouvá obsah registru do I/O

Popis:

Přesouvá data z registru Rr ze souboru registrů do I/O prostoru (porty, časovače, konfigurační registry atd.).

Syntaxe instrukce:

OUT A, Rr

Operandy:

$0 \leq r \leq 31$, $0 \leq A \leq 63$

Operace:

I/O(A) $\leftarrow$ Rr

Programový čítač:

PC $\leftarrow$ PC + 1

Šestnáctibitový kód instrukce:

1011	1AAr	rrrr	AAAA

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
clr  r16              ; nuluje r16
ser  r17              ; nastavuje r17
out  $18, r16         ; zapisuje nuly do portu B
nop                   ; čeká (nedělá nic)
out  $18, r17         ; zapisuje jedničky do portu B
```

POP

POP REGISTER FROM STACK

Čtení ze zásobníku

Popis:

Tato instrukce naplňuje registr Rd bytem ze zásobníku. Ukazetel na zásobník je pre inkrementován o jedničku před provedením POP.

Syntaxe instrukce:

POP Rd

Operandy:

$0 \leq d \leq 31$

Operace:

Rd ← STACK

Programový čítač:

PC ← PC + 1

Ukazatel zásobníku:

SP ← SP + 1

Šestnáctibitový kód instrukce:

1001	000d	dddd	1111

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
          call routine       ; volání podprogramu
          .....
routine:  push r14           ; ukládá r14 do zásobníku
          push r13           ; ukládá r13 do zásobníku
          .....
          pop r13            ; obnovuje r13
          pop r14            ; obnovuje r14
          ret                ; návrat z podprogramu
```

PUSH

PUSH REGISTER ON STACK

Uložení do zásobníku

Popis:

Tato instrukce ukládá obsah registru Rd do zásobníku. Ukazatel na zásobník je post dekrementován o jedničkou po provedení PUSH.

Syntaxe instrukce:

PUSH Rd

Operandy:

$0 \leq d \leq 31$

Operace:

STACK ← Rd

Programový čítač:

PC ← PC + 1

Ukazatel zásobníku:

SP ← SP – 1

Šestnáctibitový kód instrukce:

1001	001d	dddd	1111

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
          call routine     ; volání podprogramu
          .....
routine:  push r14         ; ukládá r14 do zásobníku
          push r13         ; ukládá r13 do zásobníku
          .....
          pop r13          ; obnovuje r13
          pop r14          ; obnovuje r14
          ret              ; návrat z podprogramu
```

RCALL

RELATIVE CALL TO SUBROUTINE

Relativní volání podprogramu

Popis:

Relativní volání na adresu v rozsahu PC –2K + 1 a PC + 2K (slov). Návratová adresa (instrukce po RCALL) je zapamatována v zásobníku (viz také CALL). V assembleru, jsou labely (návěští) užity místo relativních operandů. Pro AVR mikrokontroléry s programovou pamětí nepřekračující 4K slov (8K bytů) může tato instrukce adresovat celou paměť z libovolné adresové pozice. Ukazatel zásobníku užívá post dekrementační postup během RCALL.

Syntaxe instrukce:

RCALL k

Operandy:

$-2K \leq k \leq 2K$

Operace:

(i) PC ← PC + k + 1 MCU s 16bitovým PC, max. 128K bytů paměti.

(ii) PC ← PC + k + 1 MCU s 22bitovým PC, max. 8M bytů paměti.

Zásobník:

(i) STACK ← PC + 1 SP ← SP – 2 (2 byty, 16 bitů)

(ii) STACK ← PC + 1 SP ← SP – 3 (3 byty, 22 bitů)

Programový čítač:

PC ← PC + k + 1

Šestnáctibitový kód instrukce:

1101	kkkk	kkkk	kkkk

426410

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

3 – MCU s 16bitovým PC.

4 – MCU s 22bitovým PC.

Příklad:

```
            rcall routine    ; volání podprogramu
            .....
routine:    push r14         ; uložení r14 do zásobníku
            .....
            pop r14          ; obnovení r14
            ret              ; návrat z podprogramu
```

RET

RETURN FROM SUBROUTINE

Návrat z podprogramu

Popis:

Návrat z podprogramu. Návratová adresa je přesunuta ze zásobníku. Ukazatel na zásobník používá pre inkrementační postup během RET.

Syntaxe instrukce:

RET

Operandy:

Žádné

Operace:

(i)	PC (15:0) ← STACK	MCU s 16bitovým PC, max. 128K programové paměti
(ii)	PC (21:0) ← STACK	MCU s 22bitovým PC, max. 8M programové paměti

Programový čítač:

Viz operace.

Zásobník:

(i)	SP ← SP + 2	(2 byty, 16 bitů)
(ii)	SP ← SP + 3	(3 byty, 22 bitů)

Šestnáctibitový kód instrukce:

1001	0101	0000	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

4 – u MCU s 16bitovým PC.

5 – u MCU s 22bitovým PC.

Příklad:

```
          call routine  ; volání podprogramu
          .....
routine:  push r14      ; uloží r14 do zásobníku
          .....
          pop r14       ; obnoví r14
          ret           ; návrat z podprogramu
```

RETI

RETURN FROM INTERRUPT

Návrat z přerušení

Popis:

Návrat z přerušení. Návratová adresa je přesunuta ze zásobníku (příznak povolení všech přerušení je nastaven). Všimněte si, že stavový registr není automaticky ukládán, když probíhá obslužný podprogram přerušení, ani není automaticky obnovován po návratu z přerušení. To musí být provedeno aplikačním programem. Ukazatel zásobníku užívá pre inkrementační postup během RETI.

Syntaxe instrukce:

RETI

Operandy:

Žádný

Operace:

(i) PC (15 : 0) ← STACK u MCU s 16bitovým PC, max.128K programové paměti.

(ii) PC (21 : 0) ← STACK u MCU s 22bitovým PC, max. 8M programové paměti.

Programový čítač:

Viz operace.

Šestnáctibitová kód instrukce:

1001	0101	0001	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
1	-	-	-	-	-	-	-

I: 1
příznak I je nastaven.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

4 – u MCU s 16bitovým PC.

5 – u MCU s 22bitovým PC.

Příklad:

```
          .....
extint:   push r0      ; ukládá r0 do zásobníku
          .....
          pop r0       ; obnovuje r0
          reti         ; návrat a povolení přerušení
```


426410

RJMP

RELATIVE JUMP

Relativní skok

Popis:

Relativní skok na adresu v rozsahu PC – 2K + 1 až PC + 2K (slov). V assembleru, návěští (labels) se používají místo relativních operací. Pro AVR mikrokontroléry s programovou pamětí nepřesahující 4K slova (8K bytů) může tato instrukce adresovat celou paměť z libovolné adresové pozice.

Syntaxe instrukce:

RJMP k

Operandy:

$-2K \leq k \leq 2K$

Operace:

PC ← PC + k + 1

Programový čítač:

PC ← PC + k + 1

Šestnáctibitový kód instrukce:

1100	kkkk	kkkk	kkkk

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
          cpi  r16, $42     ; srovnává r16 s $42
          brne error        ; skok když r16<>$42
          rjmp ok           ; napodmínkový skok
error:    add  r16, r17     ; přičte r17 k r16
          inc  r16          ; inkrementace r16
ok:       nop               ; cíl pro rjmp (nedělá nic)
```

ROL

ROTATE LEFT TROUGH CARRY

Rotace vlevo s přenosem

Popis:

Posouvá všechny bity v Rd o jednu pozici doleva. Příznak C je posunut do bitu 0 v Rd. Bit 7 je přesunut do příznaku C. Tato operace v kombinaci s LSL, efektivně násobí vícebytové znaménkové i neznaménkové hodnoty dvěma.

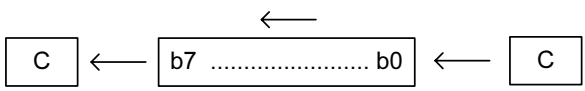
Syntaxe instrukce:

ROL Rd

Operandy:

$0 \leq d \leq 31$

Operace:



Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0001	11dd	dddd	dddd

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: $N \oplus V$
pro znaménkové testy.

V: $N \oplus C$
(pro N a C po posunu).

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: Rd7
nastaven, když před posunem byl MSB Rd nastaven, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

Počet cyklů instrukce:

1 (2 byte)

Příklad:

```
          1
          lsl r18     ; násobí r19:r18 dvěma
          rol r19     ; r19:r18 je dvoubytové celé číslo
                      ; s nebo bez znaménka
          brcs oneenc ; skok když přenos je nastaven
          .....
oneenc:   nop         ; cíl skoku (nedělá nic)
```

ROR

ROTATE RIGHT TROUGH CARRY

Rotace vpravo s přenosem

Popis:

Posouvá všechny bity v Rd o jednu pozici doprava. Příznak C je posunut do bitu 7 v Rd. Bit 0 je přesunut do příznaku C. Tato operace v kombinaci s ASR či LSR, efektivně dělí vícebytové znaménkové i neznaménkové hodnoty dvěma. Příznak přenosu může být využit k zaokrouhlování výsledku.

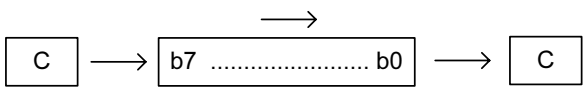
Syntaxe instrukce:

ROR Rd.

Operandy:

$0 \leq d \leq 31$

Operace:



Programový čítač:

PC ← PC + 1

Šestnáctibitové kód instrukce:

1001	010d	dddd	0111

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	⇔

426410

S: $N \oplus V$
pro znaménkové testy.

V: $N \oplus C$
(pro N a C po posunu).

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: Rd0
nastaven, když před posunem byl LSB Rd nastaven, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

2

Počet cyklů instrukce:

1 (2 byte)

Příklad:

```
            lsr  r19       ; dělí r19:r18 dvěma
            ror  r18       ; r19:r18 je neznaménkové 2bytové
                           ; celé číslo
            brcc zeroenc1  ; skok při vynulovaném přenosu
            asr  r17       ; dělí r17:r16 dvěma
            ror  r16       ; r17:r16 je znaménkové dvou bytové
                           ; celé číslo
            brcc zeroenc2  ; skok při vynulovaném přenosu
            .....
zeroenc1:   nop            ; cíl skoku (nedělá nic)
            .....
zeroenc2:   nop            ; cíl skoku (nedělá nic)
```

SBC

SUBTRACT WITH CARRY

Odečítání s přenosem

Popis:

Odečítá obsah dvou registrů a odčítá příznak přenosu C a výsledek umisťuje do cílového registru Rd.

Syntaxe instrukce:

SBC Rd, Rr

Operandy:

$0 \leq d \leq 31$, $0 \leq r \leq 31$

Operace:

$Rd \leftarrow Rd - Rr - C$

Programový čítač:

$Pc \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

0000	10rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
nastaven, když byla „výpůjčka“ z bitu 3, jinak je vynulován.

S: $N \oplus V$
pro znaménkové testy.

V: $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
nastaven při přetečení dvojkového doplňku výsledku operace, jinak je vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
předchozí hodnota zůstává nezměněna, když výsledek je nula, jinak je vynulován. Tím se liší řada AVR od některých jiných MCU, např. řady x51.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
nastaven, když absolutní hodnota obsahu Rr plus předchozí přenos je větší, než absolutní hodnota Rd, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
                ; odčítá r1:r0 od r3:r2
sub  r2, r0     ; odčítá dolní byty
subc r3, r1     ; odčítá s přenosem horní byty
```

SBCI

SUBTRACT IMMEDIATE WITH CARRY

Odečítání konstanty s přenosem

Popis:

Odečítá konstantu od registru a odčítá příznak přenosu C a výsledek umisťuje do cílového registru Rd.

Syntaxe instrukce:

SBCI Rd, K

Operandy:

$16 \leq d \leq 31$, $0 \leq K \leq 255$

Operace:

$Rd \leftarrow Rd - K - C$

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

0100	KKKK	dddd	KKKK

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$
nastaven, když byla „výpůjčka“ z bitu 3, jinak je vynulován.

426410

S: $N \oplus V$
pro znaménkové testy.

V: $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$
nastaven při přetečení dvojkového doplňku výsledku operace, jinak je vynulová.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0} \bullet Z$
předchozí hodnota zůstává nezměněna, když výsledek je nula, jinak je vynulován. To je rozdíl oproti některým jiným MCU, např. řady x51.

C: $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$
nastaven, když absolutní hodnota konstanty plus předchozí přenos je větší, než absolutní hodnota Rd, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
                        ; odčítá $4F23 od r17:r16
subi r16, $23           ; odčítá dolní byty
sbci r17, $4F           ; odčítá s přenosem horní byty
```

SBI

SET BIT IN I/O REGISTER

Nastavení bitu v I/O registru

Popis:

Nastavuje specifikovaný bit v I/O registru. Tato instrukce pracuje s dolními 32 I/O registry – adresy 0–31.

Syntaxe instrukce:

SBI A, b

Operandy:

$0 \leq A \leq 31$, $0 \leq b \leq 7$

Operace:

I/O (A, b) $\leftarrow$ 1

Programový čítač:

PC $\leftarrow$ PC + 1

Šestnáctibitový kód instrukce:

1001	1010	AAAA	Abbb

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
out  $1E, r0        ; zapisuje EEPROM adresu
sbi  $1C, 0         ; nastavuje bit čtení v EECR
in   r1, $1D        ; čte data z EEPROM
```

SBIC

SKIP IF BIT IN I/O REGISTER IS CLEARED

Přeskok při vynulovaném bitu v I/O registru

Popis:

Tato instrukce testuje jeden bit v I/O registru a v případě, když je testovaný bit vynulován, přeskakuje následující instrukci. Tato instrukce pracuje s dolními 32 I/O registry – adresa 0–31.

Syntaxe instrukce:

SBIC A, b

Operandy:

$0 \leq A \leq 31$, $0 \leq b \leq 7$

Operace:

Když I/O (A, b) = 0 tak PC ← PC + 2 (nebo 3) jinak PC ← PC + 1

Programový čítač:

PC ← PC + 1	podmínka není splněna, neprovádí se skok.
PC ← PC + 2	přeskočení jednoslovové instrukce.
PC ← PC + 3	přeskočení dvouslovové instrukce.

Šestnáctibitový kód instrukce:

1001	1001	AAAA	Abbb

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna (neprovádi se skok).

2 při splněné podmínce, přeskakovaná instrukce je 1 slovová.

3 při splněné podmínce, přeskakovaná instrukce je 2 slovová.

Příklad:

```
e2wait:   sbic $1C, 1    ; přeskakuje násl. instrukci když
                         ; je EEWE vynulován
          rjmp e2wait    ; zápis do EEPROM není skončen
          nop            ; pokračovat (nedělá nic)
```

SBIS

SKIP IF BIT IN I/O REGISTER IS SET

Přeskok při nastaveném bitu v I/O registru

Popis:

Tato instrukce testuje jeden bit v I/O registru a v případě, kdy testovaný bit je nastaven, přeskakuje následující instrukci. Tato instrukce pracuje s dolními 32 I/O registry – adresa 0–31.

Syntaxe instrukce:

SBIS A, b

Operandy:

$0 \leq A \leq 31$, $0 \leq b \leq 7$

Operace:

Když I/O (A, b) = 1 tak PC ← PC + 2 (nebo 3) jinak PC ← PC + 1

Programový čítač:

PC ← PC + 1	podmínka není splněna, neprovádí se skok.
PC ← PC + 2	přeskočení jednoslovové instrukce.
PC ← PC + 3	přeskočení dvouslovové instrukce.

Šestnáctibitový kód instrukce:

1001	1011	AAAA	Abbb

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna (neprovádi se skok).

2 při splněné podmínce, přeskakovaná instrukce je 1 slovová.

3 při splněné podmínce, přeskakovaná instrukce je 2 slovová.

Příklad:

```
waitset:  sbis $10, 0    ; přeskakuje násl.instr. Když bit 0
                         ; v portu D je nastaven
          rjmp waitset   ; bit není nastaven
          nop            ; pokračuje (nedělá nic)
```

SBIW

SUBTRACT IMMEDIATE FROM WORD

Odečítá konstantu od slova

Popis:

Odečítá konstantu (0–63) od dvojice registrů a výsledek umisťuje do registrové dvojice. Tato instrukce pracuje s horními čtyřmi dvojicemi registrů a je velmi vhodná pro práci s registrovými ukazateli.

Syntaxe instrukce:

SBIW Rd, K

Operandy:

$d \in (24, 26, 28, 30)$, $0 \leq K \leq 63$

Operace:

Rd + 1 : Rd ← Rd + 1 : Rd – K

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0111	KKdd	KKKK

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$
pro znaménkové testy.

426410

V: Rdh7 • $\overline{R15}$
nastaven, když přeteče dvojkový doplněk výsledku operace, jinak je vynulován.

N: R15
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: R15 • $\overline{Rdh7}$
nastaven, když absolutní hodnota K je větší, než absolutní hodnota Rd, jinak je vynulován.

R (výsledek) se rovná Rdh:Rdl po operaci (Rdh7 – Rdh0 = R15 – R8, Rdl7 – Rdl0 = = R7– R0).

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
sbiw r24, 1      ; odčítá 1 od r25:r24
sbiw r28, 63     ; odčítá 63 od ukazetele Y (r29:r28)
```

SBR

SET BITS IN REGISTER

Nastavuje bity v registru

Popis:

Nastavuje specifikované bity v registru Rd. Vytváří logický ORI mezi obsahem registru Rd a obsahem konstantní masky K a umisťuje výsledek do cílového registru Rd. SBR je jen jiné značení instrukce ORI – obě mají stejný kód instrukce.

Syntaxe instrukce:

SBR Rd, K

Operandy:

16 ≤ **d** ≤ **31**, 0 ≤ K ≤ 255

Operace:

Rd ← Rd ∨ K

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0110	KKKK	dddd	KKKK

tento kód je stejný jako kód instrukce ORI.

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: $N \oplus V$
pro znaménkové testy.

V: 0.
vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

R: (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
sbr  r16, 3      ; nastavuje bity 0 a 1 v r16
sbr  r17, $F0    ; nastavuje 4 MSB v r17
```

SBRC

SKIP IF BIT IN REGISTER IS CLEARED

Přeskok, když bit v registru je vynulován

Popis:

Tato instrukce testuje jeden bit v registru a přeskakuje následující instrukci, když bit je vynulován.

Syntaxe instrukce:

SBRC Rr, b

Operandy:

$0 \leq r \leq 31$, $0 \leq b \leq 7$

Operace:

Když R (b) = 0, pak PC ← PC + 2 (nebo 3), jinak PC ← PC + 1

Programový čítač:

PC ← PC + 1	podmínka není splněna, skok se neprovádí.
PC ← PC + 2	přeskakuje jednoslovovou instrukci.
PC ← PC + 3	přeskakuje dvouslovovou instrukci.

Šestnáctibitový kód instrukce:

1111	110r	rrrr	0bbb

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna (neprovádí se skok).

2 při splněné podmínce, přeskakovaná instrukce je jednoslovová.

3 při splněné podmínce, přeskakovaná instrukce je dvouslovová.

Příklad:

```
sub  r0, r1      ; odčítá r1 od r0
sbrc r0, 7       ; přeskakuje, když bit 7 v r0 je nastaven
sub  r0, r1      ; provádí se pouze když bit 7 v r0 není vynulován
nop              ; pokračování (nedělá nic)
```

SBRS

SKIP IF BIT IN REGISTER IS SET

Přeskok, když bit v registru je nastaven

Popis:

Tato instrukce testuje jeden bit v registru a přeskakuje následující instrukci, když bit je nastaven.

Syntaxe instrukce:

SBRS Rr, b

Operandy:

$0 \le r \le 31$, $0 \le b \le 7$

Operace:

Když R (b) = 1, pak PC ← PC + 2 (nebo 3), jinak PC ← PC + 1

Programový čítač:

PC ← PC + 1	podmínka není splněna, skok se neprovádí.
PC ← PC + 2	přeskakuje jednoslovovou instrukci.
PC ← PC + 3	přeskakuje dvouslovovou instrukci.

Šestnáctibitový kód instrukce:

1111	111r	rrrr	0bbb

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1 když podmínka není splněna (neprovádí se skok).

2 při splněné podmínce, přeskakovaná instrukce je jednoslovová.

3 při splněné podmínce, přeskakovaná instrukce je dvouslovová.

Příklad:

```
sub  r0, r1       ; odčítá r1 od r0
sbrs r0, 7        ; přeskakuje, když bit 7 v r0 je vynulován
neg  r0           ; provádí se pouze když bit 7 v r0 není vynulován
nop               ; pokračování (nedělá nic)
```

SEC

SET CARRY FLAG

Nastavení příznaku přenosu

Popis:

Nastavuje příznak přenosu (C) v SREG (stavový registr).

Syntaxe instrukce:

SEC

Operandy:

Žádné

Operace:

C ← 1

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	0000	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	1

C: 1
příznak přenosu je nastaven.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
sec              ; nastavuje příznak přenosu
adc r0, r1       ; r0=r0+r1+1
```

SEH

SET HALF CARRY FLAG

Nastavení příznaku Half Carry (přenos mezi třetím a čtvrtým bitem)

Popis:

Nastavuje příznak Half Carry (H) v SREG (stavový registr).

Syntaxe instrukce:

SEH

Operandy:

Žádné

Operace:

H ← 1

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	0101	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	1	-	-	-	-	-

H: 1
příznak HALF-CARRY je nastaven.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
seh             ; nastavuje příznak Half Carry
```

SEI

SET GLOBAL INTERRUPT FLAG

Nastavení příznaku povolení všech přerušení

Popis:

Nastavuje příznak povolení všech přerušení (I) v SREG (stavový registr).

Syntaxe instrukce:

SEI

Operandy:

Žádné

Operace:

I ← 1

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	0111	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
1	-	-	-	-	-	-	-

I: 1
nastavení příznaku povolení všech přerušení.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
cli                 ; zákaz přerušení
in r13, $16         ; čte port B
sei                 ; povolení přerušení
```

SEN

SET NEGATIVE FLAG

Nastavení příznaku záporného výsledku

Popis:

Nastavuje příznak záporného výsledku (N) v SREG (stavový registr).

Syntaxe instrukce:

SEN

Operandy:

Žádné

Operace:

N ← 1

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	0010	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	1	-	-

N: 1
nastavení příznaku záporného výsledku.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add  r2, r19          ; přičte r19 k r2
sen                   ; nastavuje příznak záporného výsledku
```

SER

SET ALL BITS IN REGISTER

Nastavuje všechny bity v registru

Popis:

Přesouvá $FF přímo do registru Rd.

Syntaxe instrukce:

SER Rd

Operandy:

$16 \leq d \leq 31$

Operace:

Rd ← $FF

Programový čítač:

PC ← PC+1

Šestnáctibitový kód instrukce:

1110	1111	dddd	1111

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
clr  r16            ; nuluje r16
ser  r17            ; nastavuje r17
out  $18, r16       ; zapisuje nuly do portu B
nop                 ; prodleva (nedělá nic)
out $18, r17        ; zapisuje jedničky do portu B
```

SES

SET SIGNED FLAG

Nastavuje znaménkový příznak

Popis:

Nastavuje znaménkový příznak (S) v SREG (stavový registr).

Syntaxe instrukce:

SES

Operandy:

Žádný

Operace:

S ← 1

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	0100	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	1	-	-	-	-

S: 1
nastavení příznaku znaménka.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add  r2, r19        ; přičítá r19 k r2
ses                 ; nastavuje příznak záporného výsledku
```

SET

SET T FLAG

Nastavuje příznak T

Popis:

Nastavuje příznak T v SREG (stavový registr).

Syntaxe instrukce:

SET

Operandy:

Žádný

Operace:

T ← 1

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	0110	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	1	-	-	-	-	-	-

T: 1
nastavuje příznak T.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
set          ; nastavuje příznak T
```

SEV

SET OVERFLOW FLAG

Nastavuje příznak přetečení

Popis:

Nastavuje příznak přetečení (V) v SREG (stavový registr).

Syntaxe instrukce:

SEV

Operandy:

Žádný

Operace:

V ← 1

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	0011	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	1	-	-	-

V: 1
nastavuje příznak přetečení.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add  r2, r19          ; přičítá r19 k r2
sev                   ; nastavuje příznak přetečení
```

SEZ

SET ZERO FLAG

Nastavuje příznak nuly

Popis:

Nastavuje příznak nuly (Z) v SREG (stavový registr).

Syntaxe instrukce:

SEZ

Operandy:

Žádný

Operace:

Z ← 1

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0100	0001	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	1	-

Z: 1
nastavuje příznak nuly.

426410

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
add  r2, r19          ; přičítá r19 k r2
sez                   ; nastavuje příznak nuly
```

SLEEP

Zapnutí úsporného režimu

Popis:

Tato instrukce nastavuje obvody do úsporného módu definovaného v MCU řídicím registru.

Syntaxe instrukce:

SLEEP

Operandy:

Žádný

Operace:

Viz detailní popis v dokumentaci k MCU.

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0101	1000	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
mov  r0, r11        ; kopíruje r11 do r0
ldi  r16, (1<<SE)   ; povoluje sleep mod
out  MCUCR, r16
sleep               ; nastavuje MCU do úsporného režimu (sleep módu)
```

SPM

STORE PROGRAM MEMORY

Zápis do programové paměti – jen u ATmega

Popis:

SPM může být užit k vymazání programové paměti, k zápisu stránky v programové paměti (která je již vymazána), a nastavuje lock bity boot loaderu, do programové paměti může být najednou zapsáno jedno slovo, v jiných MCU může být celá stránka programována najednou po prvním naplnění dočasného stránkového bufferu. Ve všech případech musí být programová paměť mazána po jedné stránce najednou. Když se maže programová paměť, registr Z je použit jako stránková adresa. Když zapisujeme do programové paměti, registr Z je užit jako stránková nebo slovová adresa, a jako data je použita registrová dvojice R1:R0. Jsou-li nastaveny lock bity boot loaderu, registrová dvojice R1:R0 je užita jako data. Pro detailnější popis užití SPM je detailněji popsána v dokumentaci k MCU. Tato instrukce může adresovat 64K bytů (32 K slov) programové paměti.

Syntaxe instrukce:

SPM

Operandy:

Žádné

Operace:

(i) (Z) ← $FFFF

(ii) (Z) ← R1 : R0

(iii) (Z) ← R1 : R0

(iv) (Z) ← TEMP

(v) BLBITS ← R1 : R0

Komentář:

Maže stránku programové paměti.

Zapisuje slovo do programové paměti.

426410

Zapisuje do dočasného stránkového buferu.
Zapisuje dočasný stránkový bufer do programové stránky.
Nastavuje lock bity boot loaderu.

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0101	1110	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

Závisí na operaci.

Příklad:

```
; tento příklad ukazuje SPM zápis jednoho slova pro MCU
; se slovovým zápisem
ldi  r31,   $F0    ; nastavuje horní byte Z
clr  r30           ; nuluje dolní byte Z
ldi  r16,   $CF    ; přenáší data do paměti
mov  r1,    r16
ldi  r16,   $FF
mov  r1,    r16
ldi  r16,   $03    ; povoluje EPM mazat stránku
out  SPMCR, r16    ;
spm                ; maže stránku začínající na $F000
ldi  r16,   $01    ; povoluje EPM zapisovat do programové paměti
out  SPMCR, r16    ;
spm                ; provádí SPM zapisuje R1:R0 do programové
                   ; paměti na $F000
```

ST

STORE INDIRECT FROM REGISTER TO DATA SPACE USING INDEX X

Nepřímý zápis z registru do datového prostoru s použitím indexu X

Popis:

Přesun jednoho bytu z registru do datového prostoru. Pro MCU s SRAM datový prostor sestává z souboru registrů, I/O paměti a vnitřní SRAM (a externí SRAM je-li použita). Pro MCU bez SRAM, datový prostor se skládá pouze z registrů. EEPROM má vlastní oddělený adresový prostor. Místo v paměti je odkazováno ukazatelovým registrem X (16 bitů) v souboru registrů.

Přístup k paměti je omezen na aktuální datový segment 64K bytů. K přístupu k dalším datovým segmentům v MCU s více než 64 K byty datového prostoru je třeba změnit registr RAMPX v I/O prostoru. Registrový ukazetel X může být buď nezměněn operací, nebo může být post inkrementován nebo pre dekrementován. Tyto vlastnosti jsou vhodné pro přístup k polím, tabulkám, a užití ukazatele X jako zásobníkového ukazatele. Stojí za povšimnutí, že pouze dolní byte ukazatele X je měněn v MCU s nejvýše 256 byty datového prostoru. Pro taková MCU horní byte ukazatele není použit touto instrukcí a může být využit pro jiné účely. Registr RAMPX v I/O prostoru je aktualizován v obvodech s více než 64Kbytovým datovým prostorem.

Není definován výsledek pro tyto kombinace:

ST X+, r26

ST X+, r27

ST –X, r26

ST –X, r27

Syntaxe instrukce:

(i) ST X, Rr

(ii) ST X+, Rr

(iii) ST –X, Rr

426410

Operandy:

$0 \le r \le 31$

Operace:

(i) $(X) \leftarrow Rr$

(ii) $(X) \leftarrow Rr \quad X \leftarrow X + 1$

(iii) $X \leftarrow X - 1 \quad (X) \leftarrow Rr$

Komentář:

X: nezměněn.

X: post inkrementován.

X: pre dekrementován.

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

(i)	1001	001r	rrrr	1100
(ii)	1001	001r	rrrr	1101
(iii)	1001	001r	rrrr	1110

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
clr  r27            ; nuluje horní byte X
ldi  r26, $60       ; nastavuje dolní byte X na $60
st   X+, r0         ; přenáší obsah r0 do datového prostoru na $60
                    ; (X post inkrem.)
st   X, r1          ; přenáší obsah r1 do datového prostoru na $63
ldi  r26, $63       ; nastavuje dolní byte X na $63
st   X, r2          ; přenáší obsah r2 do datového prostoru na $63
st   -X, r3         ; přenáší obsah r3 do datového prostoru na $62
                    ; (X pre dekr.)
```


426410

STS

STOREC DIRECT TO DATA SPACE

Přímý přesun do datového prostoru

Popis:

Přesouvá jeden byte z registru do datového prostoru. Pro MCU se SRAM datový prostor sestává ze souboru registrů, I/O paměti a vnitřní SRAM (externi SRAM, je-li použita). Pro MCU bez SRAM datový prostor je složen jen ze souboru registrů. EEPROM má oddělený adresový prostor. Musí být použity 16bitové adresy. Přístup do paměti je omezen na aktuální datový segment 64K bytů. Instrukce STS užívá RAMPD registr k přístupu do paměti nad 64K. K přístupu k dalším datovým segmentům u MCU s více než 64K byte datového prostoru, je registr RAMPD v I/O prostoru měněn.

Syntaxe instrukce:

STS k, Rr

Operandy:

$0 \leq r \leq 31$, $0 \leq k \leq 65535$

Operace:

(k) ← Rr

Programový čítač:

PC ← PC + 2

Šestnáctibitový kód instrukce:

1001	01dd	dddd	0000
kkkk	kkkk	kkkk	kkkk

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

2 (4 byte)

Počet cyklů instrukce:

2

Příklad:

```
lds  r2, $FF00    ; naplňuje r2 obsahem datového prostoru na $FF00
add  r2, r1       ; přičte r1 k r2
sts  $FF00, r2    ; zapisuje nazpět
```

ST (STD)

STORE INDIRECT FROM REGISTER TO DATA SPACE USING INDEX Y

Nepřímý zápis z registru do datového prostoru s použitím indexu Y

Popis:

Přesun jednoho byte z registru do datového prostoru. Pro MCU s SRAM datový prostor sestává z souboru registrů, I/O paměti a vnitřní SRAM (a externí SRAM, je-li použita). Pro MCU bez SRAM, datový prostor se skládá pouze z registrů. EEPROM má vlastní oddělený adresový prostor.

Na místo v paměti je odkazováno ukazatelovým registrem Y (16 bitů) v souboru registrů. Přístup k paměti je omezen na aktuální datový segment 64K bytů. K přístupu k dalším datovým segmentům v MCU s více než 64 K byty datového prostoru je třeba změnit registr RAMPY v I/O prostoru.

Registrový ukazatel Y může být buď nezměněn operací, nebo může být post inkrementován nebo pre dekrementován. Tyto vlastnosti jsou speciálně vhodné pro přístup k polím, tabulkám a užití ukazatele Y jako zásobníkového ukazatele. Stojí za povšimnutí, že pouze dolní byte ukazatele X je měněn v MCU s nejvýše 256 byty datového prostoru. Pro taková MCU horní byte ukazatele není použit touto instrukcí a může být využit pro jiné účely. Registr RAMPY v I/O prostoru je aktualizován v obvodech s více než 64Kbytovým datovým prostorem.

Není definován výsledek pro tyto kombinace:

ST Y+, r28

ST Y+, r29

ST –Y, r28

ST –Y, r28

Syntaxe instrukce:

(i) ST Y, Rr

(ii) ST Y+, Rr

(iii) ST –Y, Rr

(iiii) STD Y+q, Rr

Operandy:

$0 \le r \le 31, 0 \le q \le 63$

Operace:

(i) (Y) ← Rr

(ii) (Y) ← Rr Y ← Y + 1

(iii) Y ← Y – 1 (Y) ← Rr

(iiii) (Y + q) ← Rr

Komentář:

Y: nezměněn.

Y: post inkrementován.

Y: pre dekrementován.

Y: nezměněn, q: přemístěn.

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

(i)	1001	001r	rrrr	1100
(ii)	1001	001r	rrrr	1101
(iii)	1001	001r	rrrr	1110
(iiii)	10q0	qq1r	rrrr	1qqq

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
clr  r29       ; nuluje horní byte Y
ldi  r28, $60  ; nastavuje dolní byte Y na $60
st   Y+, r0    ; přenáší obsah r0 do datového prostoru na $60
               ; (Y post inkrem.)
st   Y, r1     ; přenáší obsah r1 do datového prostoru na $63
ldi  r28, $63  ; nastavuje dolní byte Y na $63
st   Y, r2     ; přenáší obsah r2 do datového prostoru na $63
st   -Y, r3    ; přenáší obsah r3 do datového prostoru na $62
               ; (Y pre dekr.)
std  Y+2, r4   ; přenáší obsah r4 do datového prostoru na $64
```

ST (STD)

STORE INDIRECT FROM REGISTER TO DATA SPACE USING INDEX Z

Nepřímý zápis z registru do datového prostoru s použitím indexu Z

Popis:

Přesun jednoho byte z registru do datového prostoru. Pro MCU s SRAM datový prostor sestává ze souboru registrů, I/O paměti a vnitřní SRAM (a externí SRAM, je-li použita). Pro MCU bez SRAM, datový prostor se skládá pouze z registrů. EEPROM má vlastní oddělený adresový prostor.

Na místo v paměti je odkazováno ukazatelovým registrem Z (16 bitů) v souboru registrů. Přístup k paměti je omezen na aktuální datový segment 64K bytů. K přístupu k dalším datovým segmentům v MCU s více než 64 K byty datového prostoru je třeba změnit registr RAMPZ v I/O prostoru.

Registrový ukazatel Z může být operací buď nezměněn, nebo může být post inkrementován nebo pre dekrementován. Tyto vlastnosti jsou vhodné pro přístup k ukazateli na zásobník užitím ukazatelového registru Z, avšak protože registrový ukazatel Z může být použit pro nepřímé volání podprogramů či nepřímé skoky a uzamčení tabulek, je výhodnější pro ukazatele zásobníku vyhradit X nebo Y.

Stojí za povšimnutí, že v MCU, které nemají více než 256 bytů paměti SRAM je aktualizován pouze dolní byte ukazatele Z. V takových MCU instrukce nepoužívá horní byte ukazatele a může být použit pro jiné účely. Registr RAMPZ v I/O prostoru je aktualizován v MCU s více než 64Kbytovým datovým prostorem a je přidáno rozšíření na celou 24bitovou adresu v takovém MCU. V MCU s více než 64K byty programového paměťového prostoru a více než 64K byty datové paměti, RAMPZ registr je používán pouze instrukcemi ELPM a ESPM. Tudíž RAMPZ není používán instrukcí ST.

Není definován výsledek pro tyto kombinace:

ST Z+, r30

ST Z+, r31

ST –Z, r30

ST –Z, r31

426410

Syntaxe instrukce:

(i) ST Z, Rr
(ii) ST Z+, Rr
(iii) ST –Z, Rr
(iiii) STD Z + q, Rr

Operandy:

$0 \leq r \leq 31$ $0 \leq q \leq 63$

Operace:

(i) (Z) ← Rr
(ii) (Z) ← Rr Z ← Z + 1
(iii) Z ← Z – 1 (Z) ← Rr
(iiii) (Z + q) ← Rr

Komentář:

Z: nezměněn.
Z: post inkrementován.
Z: pre dekrementován.
Z: nezměněn, q: přemístěn.

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

(i)	1001	001r	rrrr	0000
(ii)	1001	001r	rrrr	0001
(iii)	1001	001r	rrrr	0010
(iiii)	10q0	qq1r	rrrr	0qqq

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

2

Příklad:

```
clr  r31       ; nuluje horní byte Z
ldi  r30, $60  ; nastavuje dolní byte Z na $60
st   Z+, r0    ; přenáší obsah r0 do datového prostoru na $60
               ; (Z post inkrem.)
st   Z, r1     ; přenáší obsah r1 do datového prostoru na $63
ldi  r30, $63  ; nastavuje dolní byte Z na $63
st   Z, r2     ; přenáší obsah r2 do datového prostoru na $63
st   -Z, r3    ; přenáší obsah r3 do datového prostoru na $62
               ; (Z pre dekr.)
std  Z+2, r4   ; přenáší obsah r4 do datového prostoru na $64
```

SUB

SUBTRACT WITHOUT CARRY

Odčítání bez přenosu

Popis:

Odčítá obsah dvou registrů a výsledek umisťuje do cílového registru Rd.

Syntaxe instrukce:

SUB Rd, Rr

Operandy:

$0 \le d \le 31$, $0 \le r \le 31$

Operace:

Rd ← Rd ← Rr

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0001	10rd	dddd	rrrr

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet Rr3 + Rr3 \bullet R3 + R3 \bullet \overline{Rd3}$
nastaven, když je „výpůjčka“ z bitu 3, jinak je vynulován.

S: $N \oplus V$
pro znaménkové testy.

V: $Rd7 \bullet \overline{Rr7} \bullet \overline{R7} + \overline{Rd7} \bullet Rr7 \bullet R7$
nastaven, když přeteče dvojkový doplněk výsledku operace, jinak je vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: $\overline{Rd7} \bullet Rr7 + Rr7 \bullet R7 + R7 \bullet \overline{Rd7}$
nastaven, když absolutní hodnota obsahu Rr je větší než absolutní hodnota Rd, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
          sub  r13, r12     ; odečte r12 od r13
          brne noreq        ; skok když r12<>r13
          .....
noteq:    nop               ; cíl skoku (nedělá nic)
```

SUBI

SUBTRACT IMMEDIATE

Odčítání konstanty

Popis:

Odčítá obsah registru a konstanty a výsledek umisťuje do cílového registru Rd. Takto pracuje s registry R16 až R31 a je velmi vhodná pro operace s ukazateli X, Y a Z.

Syntaxe instrukce:

SUBI Rd, K

Operandy:

16 ≤ **d** ≤ **31**, 0 ≤ K ≤ 255

Operace:

Rd ← Rd – K

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

0101	KKKK	dddd	KKKK

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	⇔	⇔	⇔	⇔	⇔	⇔

H: $\overline{Rd3} \bullet K3 + K3 \bullet R3 + R3 \bullet \overline{Rd3}$
nastaven, když je „výpůjčka“ z bitu 3, jinak je vynulován.

S: $N \oplus V$
pro znaménkové testy.

V: $Rd7 \bullet \overline{K7} \bullet \overline{R7} + \overline{Rd7} \bullet K7 \bullet R7$
nastaven, když přeteče dvojkový doplněk výsledku operace, jinak je vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

C: $\overline{Rd7} \bullet K7 + K7 \bullet R7 + R7 \bullet \overline{Rd7}$
nastaven, když absolutní hodnota obsahu K je větší, než absolutní hodnota Rd, jinak je vynulován.

R (výsledek) se rovná Rd po skončení operace.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
          subi r22, $11  ; odečte $11 od r22
          brne noreq     ; skok když r22<>K
          .....
noteq:    nop            ; cíl skoku (nedělá nic)
```

SWAP

SWAP NIBBLES

Přehození částí bytů

Popis:

Přehazuje horní a dolní polovinu bytu v registru.

Syntaxe instrukce:

SWAP Rd

Operandy:

$0 \leq d \leq 7$

R (7:4) ← Rd (3:0), R (3:0) ← Rd (7:4)

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	010d	dddd	0010

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
inc     r1      ; inkrementuje r1
swap    r1      ; přehodí horní a dolní část r1
inc     r1      ; inkrementuje horní část r1
swap    r1      ; přehození nazpátek
```


426410

TST

TEST FOR ZERO OR MINUS

Testování na nulu nebo zápornou hodnotu

Popis:

Testuje zda v registru je nula či záporný výsledek. Vytváří logické AND mezi registrem a sebou sama. Registr zůstává nezměněn.

Syntaxe instrukce:

TST Rd

Operandy:

$0 \leq d \leq 31$

Operace:

$Rd \leftarrow Rd \bullet Rd$

Programový čítač:

$PC \leftarrow PC + 1$

Šestnáctibitový kód instrukce:

0010	00dd	dddd	dddd

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	⇔	0	⇔	⇔	-

S: $N \oplus V$
pro znaménkové testy.

V: 0
vynulován.

N: R7
nastaven, když MSB výsledku je nastaven, jinak je vynulován.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
nastaven, když výsledek je $00, jinak je vynulován.

R (výsledek) se rovná Rd.

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
        tst  r0         ; testuje r0
        breq zero       ; skok když r0=0
        .....
zero:   nop             ; cíl skoku (nedělá nic)
```

WDR

WATCHDOG RESET

Reset watchdog časovače

Popis:

Tato instrukce resetuje časovač Watchdog. Tato instrukce musí být provedena v čase daném WD předděličkou. Viz specifikaci hardware Watchdog časovače.

Syntaxe instrukce:

WDR

Operandy:

Žádné

Operace:

Reset WD časovače.

Programový čítač:

PC ← PC + 1

Šestnáctibitový kód instrukce:

1001	0101	1010	1000

Stavový registr (SREG):

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Počet slov instrukce:

1 (2 byte)

Počet cyklů instrukce:

1

Příklad:

```
wdr              ; reset časovače watchdog
```

LITERATURA

Seznam literatury a internetových adres s informacemi o AVR:

[1] http://www.atmel.com

[2] CD Atmel Products, May 2002

[3] http://avr-forum.com

[4] http://www.avr-freaks.net

[5] http://www.hpinfotech.ro

[6] http://www.LancOS.com

[7] http://www.mcselec.com

[8] http://www.e-lab.de

[9] http://www.iar.com

[10] http://www.mjolner.com

[11] http://www.hw.cz

[12] Janeček J.: „Projektování mikropočítačových systémů“, ČVUT Praha 1999

[13] Šubrt V.: „Mikrokontroléry ATMEL AVR vývojové prostředí“, BEN – technická literatura, Praha 2002

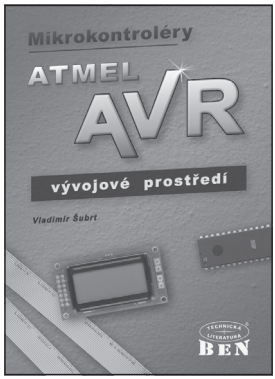
[14] Pytlík J.: „Jednočipové mikropočítače AT90S...“, Příloha Praktické elektroniky = ELECTUS ´97

[15] Kopelent J.: „ATMEL mikroprocesory AVR“, seriál, Rádio Plus KTE, 2002

426410

Knihy nakladatelství BEN – technická literatura:

MIKROKONTROLÉRY ATMEL



96 stran B5 + CD ROM
Vladimír Šubrt
obj. číslo 121099
MC 199 Kč

Mikrokontroléry Atmel AVR – vývojové prostředí

Publikace si klade za cíl seznámit čtenáře s novou řadou mikroprocesorů RISC AT90.. firmy Atmel, které jsou, sice zatím nepravidelně, dodávány i do maloobchodní sítě. Je tedy možné využít těchto součástek při realizaci některých amatérských konstrukcí a nahradit tak velmi oblíbenou řadu '51. Proto je celá publikace zaměřena jako praktický návod jak získat cenově dostupné vývojové prostředky a jak je prakticky použít. Celý výklad problematiky je veden tak, aby čtenář mohl veškeré ukázky programů prakticky vyzkoušet na některých „start kitech“, které lze v přijatelné ceně zakoupit buď kompletované nebo jako stavebnice. Lze samozřejmě realizovat celé zapojení také na univerzální desce plošných spojů.

Jedná o praktickou knihu, která umožnuje v interakci s podklady z Internetu přímo vývoj jednoduššího přístroje. Je však určena čtenářům, kteří již mají nějaké zkušenosti s programováním mikrokontrolérů. Předpokládá se, že čtenář disponuje některým ze „start kitů“, připojitelným přes sériovou linku k počítači PC.

Knihu doplňuje CD ROM, který obsahuje některé vývojové prostředky, na něž je v publikaci kladen důraz. Autor předpokládá, že ostatní prostředky, především překladače assembleru a „C“ a veškeré katalogové údaje má čtenář možnost získat z internetu. Z toho důvodu publikace také obsahuje pouze ty informace o součástce, které mají bezprostřední vztah k popisované aplikaci. Součástí publikace je soubor odkazů na stránky internetu, které mají přímou i nepřímou vazbu na mikroprocesory Atmel řady AT90.

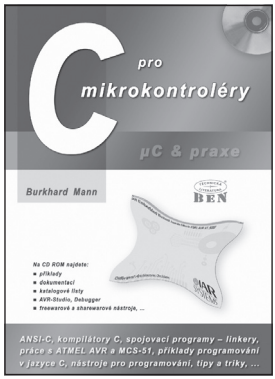
Publikace se zaobírá následujícími okruhy:
- způsoby ladění vyvíjené aplikace
- jednoduchý víceúlohový operační systém
- vizualizace zpracovávaných údajů především na LCD displeji
- upgrade aplikačního programu po sériové lince

Na doprovodném CD je obsažen software pro mikroprocesory AVR, který se na uvedenou problematiku řeší. Jedná se tedy především o monitor, základní funkce operačního systému a soubor podprogramů obsluhy LCD displeje včetně funkcí pro tvorbu menu. Součástí CD je také soubor školních programů na PC pro Windows. Důraz je kladen především na program „DebuggerAVR“, který je nadstavbou monitoru. Umožňuje sledovat běh programu aplikace, zastavit běh programu nebo modifikovat proměnné. Dalším programem je jednoduchý nástroj pro automatickou editaci tabulek operačního systému podle předem stanovených kritérií. V celé řadě aplikací se zvýšily nároky na interpretaci dat. Jednoduchá signalizace prostřednictvím LED diod nebo použití jednoho nebo i více sedmisegmentového displeje se používá pouze u velmi jednoduchých přístrojů. S poklesem cen se velmi často používají textové jedno a dvouřádkové displeje LED, v omezených případech potom i víceřádkové. Zároveň ale stoupají nároky na programové řešení, předpokládá se realizace co nejjednoduššího ale zároveň efektivního ovládání. Samozřejmostí je uživatelské menu ovládané pouze několika tlačítky. Právě tvorba menu při vývoji zařízení podléhá časté modifikaci v co nejkratší době. Součástí CD je také účinný nástroj pro automatickou tvorbu menu LCD displeje.

Z obsahu: 1. O knize; 2. Stručný popis základních vlastností mikroprocesorů Atmel RISC AT90Sxxxx; 3. Začínáme "Hello,World"; 4. Ladění aplikací; 5. Operační systém; 6. Ladění OS s použitím programu DebuggerAVR; 7. Zobrazení údajů aplikace; 8. Periferie; 9. Řízení PWM signálu z Debuggeru; 10. Upgrade aplikace.

Knihy nakladatelství BEN – technická literatura:

MIKROKONTROLÉRY



300 stran B5 + CD ROM
vázané
Burkhard Mann
obj. číslo 121120
MC 499 Kč

C pro mikrokontroléry

Jazyk ANSI-C v posledních letech dokázal, že je pro požadavky oboru mikrokontrolérů velmi vhodný. C je v současné době nejčastěji používaný programovací jazyk, a to nejen při programování mikrokontrolérů. Jeho blízkost hardwaru včetně efektivní a pružné práce s pamětmi a přídavnými zařízeními uspokojuje zejména požadavky na embedded systémy.

Mnohým připadá přechod z assembleru na jazyk C obtížný. Pochybnosti týkající se velikosti kódu a rychlosti zpracování však již u moderních mikrokontrolérů a kompilátorů C nejsou na místě. Firmy Atmel a IAR Systems dávají skvělý příklad v podobě úspěšné rodiny mikrokontrolérů AVR. Výše uvedené požadavky moderního softwarového inženýrství jsou do jazyka C podstatně snáze přenositelné.

Proto vznikla i tato kniha, která ve stručném úvodu omezujícím se na podstatné věci, vysvětluje základy ANSI-C relevantní pro mikrokontroléry. V popředí stojí tvorba efektivního programového kódu. Podrobněji jsou například uváděny zvláštnosti jazyka C pro rodiny mikrořadičů AVR a MCS-51. Na doprovodném CD jsou vedle všech příkladů a tabulek jazyka C k dispozici také kompilátor jazyka C a simulátory MCU od firmy IAR Systems pro rodiny AVR a 8051 AT89 firmy Atmel, takže všechny příklady je možno způsobem blízkým praxi napodobit v simulátoru.

Je také ukázáno, jak je možno si vývojovou práci zjednodušit v rámci vývojového systému Embedded Workbench. A přirozeně také mnoho tipů a triků, které mají začátečníkům v C usnadnit začátky a zkušeným poskytnout podněty k zamyšlení.)

Jako příloha byl do českého vydání knihy doplněn popis vývojového kitu RD2. Pochází z dílny českých vývojářů a rozhodně ulehčí laborování s mikrokontroléry. Je určen především pro rychlý vývoj nových aplikací a výuku programování v jazyce C.

Z obsahu: 1. Stručný úvod do jazyka ANSI-C pro mikrokontroléry; 2. Kompilátor C pro mikrokontroléry; 3. Příklady programů v jazyce C pro AVR; 4. Tipy a triky v jazyce C; 5. Systémy pracující v reálném čase; 6. Dobrý programovací styl v jazyce C; Dodatky – ASCII tabulky apod.; Příloha – popis vývojového kitu RD2.

DOPORUČUJEME

- USB – měření, řízení a regulace pomocí sběrnice USB (obj. č. 121116, 399 Kč)
- Práce se sběrnicí USB na bázi obvodů FTDI (obj. č. 121136, připravujeme)

Knihy nakladatelství BEN – technická literatura:

ČÍSLICOVÁ TECHNIKA

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

Udělejte si z PC – generátor, čítač, převodník, programátor…

Měření, řízení a regulace pomocí sériového portu PC a sběrnice I²C

Kniha je určena čtenářům, kteří mají alespoň základní znalosti číslicové techniky a rámcové představy o programování. Jádrem knihy je totiž popis konstrukce a ovládání šesti elektronických přístrojů, které lze využít v amatérské praxi. Všechna tato zařízení se připojují k sériovému portu počítače a jsou ovládána programy, které běží na operačních systémech: Windows 95, Windows 98, Windows NT, Windows 2000 nebo Windows Me. Ovládací programy byly vytvořeny ve vývojovém prostředí C++ Builder verze 1.0.

První kapitola probírá teorii spojenou s ovládáním sériového portu jak pod Windows, tak i na úrovni operačního systému MS-DOS. Krátce je věnována pozornost i práci s paralelním portem.

Kapitoly 2, 3, 4 a 5 popisují stavbu několika jednodušších zařízení. Jedná se impulzní generátor do 1 MHz, čítač do 16 MHz, programátor obvodů GAL a programovatelný generátor do 100 kHz.

Šestá kapitola nejdříve popisuje chování sběrnice I²C a poté se věnuje výkladu tří obvodů pracujících s touto sběrnicí. Jedná se o obvody: TDA8444 (8násobný 6bitový D/A převodník), PCF8574 (8bitový vstupně/výstupní port) a PCF8591 (jednoduchý 8bitový D/A převodník a 4kanálový 8bitový A/D převodník).

Sedmá kapitola používá obvody popsané v kapitole 6 pro konstrukci měřicí desky vybavené převodníky A/D a D/A a číslicovými vstupy a výstupy. Jsou ukázány dvě aplikace této desky.

Osmá kapitola popisuje stavbu programovatelného generátoru do 100 kHz vycházejícího z konstrukce popsané v kapitole 5. Dochází k rozšíření možných nastavení parametrů signálu a s tím je spojená i úprava ovládacího programu.

V příloze jsou výkresy konstrukce desek plošných spojů pro výrobu všech uvedených zařízení v amatérských podmínkách.

Z obsahu: 1. Porty PC; 2. Impulzní generátor do 1 MHz; 3. Čítač do 16 MHz; 4. Programátor obvodů GAL; 5. Programovatelný generátor do 100 kHz; 6. Obvody se sběrnicí I²C; 7. Měřicí deska k sériovému portu počítače; 8. Vylepšený programovatelný generátor; Příloha.

426410

Knihy nakladatelství BEN – technická literatura:

ČÍSLICOVÁ TECHNIKA

224 stran B5 + CD ROM
Ing. David Matoušek
obj. číslo 121114
MC 299 Kč

Udělejte si z PC – 2. díl (Připravujeme)

Měření, řízení a regulace pomocí portů PC prostřednictvím několika jednoduchých přípravků

Komunikace PC s aplikacemi mikrořadičů řady AT89C2051

Stavba jednoduchého programátoru mikrořadiče AT89C2051

Kniha volně navazuje na první díl. Je určena čtenářům, kteří jsou obeznámeni ze základy číslicové techniky a programováním mikrokontrolérů řady 8051 a AVR. Jádrem knihy je popis několika elektronických přístrojů, které jsou řízeny sériovým nebo paralelním portem počítače a ovládány programy, které běží na operačních systémech: Windows 95/98/NT/2000/Me. Ovládací programy jsou vytvořeny ve vývojovém prostředí C++ Builder verze 5.0.

První kapitola uvádí možné způsoby ovládání portů v operačním systému Windows. Kromě klasických možností poskytovaných funkcemi Win API je ukázána možnost přímého přístupu na porty (a to i pod Windows NT/2000).

Druhá kapitola uvádí základní parametry mikrokontrolérů AT89C2051, AT89S8252 a AT90S2313. Tyto mikrokontroléry jsou pak použity v dalších konstrukcích.

Třetí kapitola je zaměřena na jednotlivé standardy paralelních portů (SPP/EPP/ECP) a ukazuje realizaci tří jednoduchých desek ovládaných paralelním portem na úrovni SPP resp. EPP standardu.

Čtvrtá kapitola uvádí dvě aplikace přímého řízení sériového portu. Jedná se jednoduché přípravky ve funkci dvoukanálového A/D převodníku (s obvodem MCP3002) a programátoru sériových E2PROM typu 93Cx6. Oba přípravky jsou napájeny přímo ze sériového portu.

Pátá kapitola uvádí nejjednodušší aplikaci mikrokontroléru AT89C2051 ve spojení se sériovým portem. Jedná se o jednoduchou 8bitovou vstupně/výstupní desku (může ovládat různá zařízení). V kapitole 7 je tato konstrukce výrazně rozšířena.

Šestá kapitola popisuje konstrukci programátoru mikrokontroléru AT89C2051, který je nazván ATPROG2.1. V opravené konstrukci autor také reagoval na připomínky čtenářů.

Další kapitoly (7 a 8) uvádějí dvě aplikace mikrokontrolérů AT89C2051. Jedná se o měřicí desku vybavenou dvěma A/D a D/A převodníky a osmi vstupy/výstupy a zdroj řízený počítačem pracující v rozsahu 0 až 20 V.

Devátá kapitola popisuje stavbu čítače s mikrokontrolérem AT90S2313.

Desátá kapitola uvádí značně přepracovanou konstrukci programovatelného generátoru z prvního dílu. Jádrem konstrukce je mikrokontrolér AT89S8252. Konstrukce je jednodušší a vede na plošný spoj menších rozměrů.

V poslední kapitole se dotkneme USB sběrnice tím, že jsou uvedeny převodníky od firmy FTDI, které konvertují signály USB sběrnice na signály sériového portu. To dává stávajícím aplikacím (mimo jiné) možnost používat vyšší počet sériových portů.

V knize jsou rovněž publikovány výkresy plošných spojů všech uvedených konstrukcí.

2. vydání
256 stran B5 + CD ROM
Ing. David Matoušek
obj. číslo 121093
MC 349 Kč

Práce s mikrokontroléry Atmel AT89C2051

Měření, řízení a regulace pomocí několika jednoduchých přípravků

1. díl – edice µP & praxe

Kniha podrobně vysvětluje jednotlivé rysy mikrořadičů typu AT89C2051 a ukazuje jejich použití jak v klasických příkladech, tak i v dosud nepublikovaných konstrukcích. Velký důraz je kladen na srozumitelnost a postupné vysvětlování jednotlivých pojmů.

V úvodu je čtenář seznámen se základními pojmy mikroprocesorové techniky, následuje velmi stručný popis základních schopností řadičů ATMEL, který je zakončen popisem konstrukce programátoru a testovací desky (vývojového kitu).

Následuje popis programátorského modelu, instrukcí a assembleru mikrořadičů ATMEL. Tento výklad je doplněn praktickými příklady použití.

Po nezbytném úvodu je věnována pozornost popisu portů P1 a P3 včetně základních aplikací, které jsou určeny především začátečníkům. Poté se seznámíme s méně tradičním použitím portů P1 a P3, kromě jiného zde nalezneme realizaci nízkofrekvenčního generátoru, popis zmnožování vstupů a výstupů, konstrukci a ovládání vícesegmentových displejů, použití zabudovaného komparátoru pro měření kapacity, odporu a napětí (tedy jako A/D převodník).

Přerušovací systém je vysvětlen na příkladech realizace nízkofrekvenčního generátoru s volitelným průběhem (tedy generátoru funkcí). Podobně sériová komunikace ukazuje velmi jednoduchou realizaci programovatelného generátoru nízkofrekvenčního signálu do 10 kHz. Čítače a časovače jsou předvedeny v aplikacích vytvářejících zvukové efekty, PWM regulátory, D/A převodníky a měřiče kmitočtu a přeneseně i odporu či kapacity.

Poslední dvě kapitoly jsou věnovány použití perspektivních periferních obvodů vybavených sběrnicí MicroWire (93C66, M5451B7, TLC549) nebo I^2C (PCF8591), převodníků napětí nebo teploty na kmitočet realizovaných obvodem LM331 a hlídacích obvodů TL77xxA a MAX690A.

Na doprovodném CD-ROM naleznete zdrojové texty všech 23 publikovaných příkladů a klišé plošných spojů všech 16 realizovaných přípravků.

426410

Knihy nakladatelství BEN – technická literatura:

MIKROKONTROLÉRY ATMEL

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

Práce s mikrokontroléry Atmel AT89S8252

Měření, řízení a regulace pomocí několika jednoduchých přípravků

2. díl – edice µP & praxe

Kniha je zaměřena na popis mikrokontroléru **AT89S8252** včetně tří desítek zajímavých aplikací. Řada informací je použitelná především nejen pro mikrokontrolér AT89C2051, ale i pro jiné typy.

V úvodu jsou krátce vysvětleny základní pojmy mikroprocesorové techniky. Následuje druhá kapitola, která uvádí základní vlastnosti mikrokontroléru **AT89S8252** včetně popisu sériového downloadu (programování přímo v navrhovaném systému). Tyto poznatky jsou zužitkovány ve třetí kapitole, která popisuje konstrukci programátoru spojeného s vývojovým kitem (pro programování a testování postačí jediná deska plošných spojů). Součástí knihy je i komplexní "oživovací" program tohoto programátoru. Takže oživení zvládne i začátečník!

Čtvrtá kapitola vysvětluje pojmy spojené s vnitřní a vnější pamětí programu resp. dat a popisuje základní registry mikrokontroléru. Pátá kapitola uvádí instrukční soubor a šestá kapitola uvádí možnosti assembleru. Je zde uveden i popis programu **AT8252.EXE**, který slouží k pohodlnému vývoji a programování aplikací pro mikrokontrolér **AT89S8252**.

Sedmá kapitola popisuje chování portů P0 až P3 a uvádí základní aplikace (připojení osmi LED, připojení osmi spínačů, připojení 16 LED a 16 spínačů pomocí sériové sběrnice Microware).

Osmá kapitola je věnována obvodům se sběrnicí **I²C**. Pro popis byly vybrány obvody: **SAA1064** (budič 4místného LED displeje s regulací jasu), **PCD3312** (DTMF generátor), **TDA8444** (8kanálový 6bitový D/A převodník) a **PCF8591** (4kanálový 8bitový A/D převodník spojený s 8bitovým D/A převodníkem). V této kapitole je uvedeno mnoho příkladů použití.

Devátá kapitola popisuje přerušovací systém mikrokontroléru. Jeho použití je ukázáno na dvou příkladech připojení klasické klávesnice IBM PC k mikrokontroléru **AT89S8252**.

Desátá kapitola uvádí vlastnosti čítačů/časovačů 0 až 2 a doplňuje velmi zajímavé příklady jejich použití (ovládání 4místného displeje s časovým multiplexem pomocí časovače, elektronické stopky, levný D/A převodník, přesný přeladitelný zdroj kmitočtu, PWM regulátor, dvě varianty levného A/D převodníku).

Jedenáctá kapitola se věnuje použití zabudovaného sériového portu. Je uvedeno jednoduché a levné připojení mikrokontroléru k sériovému portu počítače (bez nutnosti použít obvod MAX232). Nejdříve je uvedena zajímavá konstrukce počítačem řízeného stabilizovaného zdroje s regulací napětí v rozsahu 0 až 10 V, následuje impulzní generátor pracující do 600 kHz se střídou nastavitelnou v rozsahu 1 : 9 až 9 : 1. Nakonec je uveden čítač pracující do kmitočtu 16 MHz.

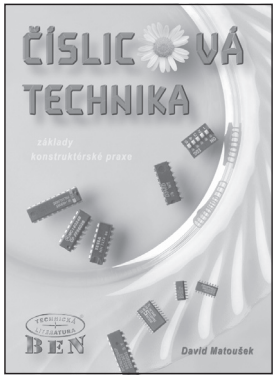
Dvanáctá kapitola uvádí pokročilé schopnosti mikrokontroléru **AT89S8252** jako jsou: řízení spotřeby, SPI sběrnice a ovládání zabudované paměti E2PROM a použití obvodu Watchdog.

Kniha rovněž obsahuje popis konstrukce přípravků (včetně desek plošných spojů) pro všech 30 publikovaných příkladů. Přiložené CD ROM obsahuje klišé plošných spojů přípravků a zdrojové kódy všech publikovaných příkladů.

426410

Knihy nakladatelství BEN – technická literatura:

ČÍSLICOVÁ TECHNIKA



208 stran B5
Ing. David Matoušek
obj. číslo 121060
MC 249 Kč

Číslicová technika

Kniha velice podrobně popisuje druhy číslicových obvodů a jejich použití. Je určena nejen začátečníkům, protože i pokročilí „bastlíři" zde naleznou řadu dosud nepublikovaných konstrukcí.

První tři kapitoly jsou určeny především začátečníkům. Jsou zde uvedeny a vysvětleny základní pojmy číslicové techniky včetně aplikací kombinačních obvodů (hradel) i sekvenčních obvodů (klopných obvodů, čítačů a posuvných registrů).

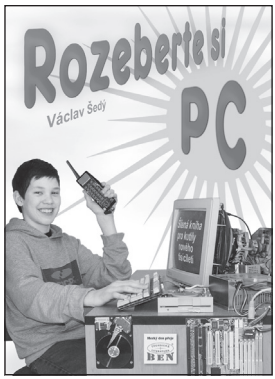
Čtvrtá kapitola je věnována zajímavým aplikacím číslicových obvodů. Nejprve jsou popsány pokročilejší obvody (číslicové komparátory, dekodéry, budiče sběrnice). Následují aplikace hradel se vstupy vybavenými Schmittovými klopnými obvody a příklady zapojení generátorů. Nakonec je uvedeno pět zajímavých konstrukcí displejů se 7segmentovkami.

Pátá kapitola diskutuje problémy spojené s používáním logických obvodů (zákmity, hazardy, vysokofrekvenční rušení) a ukazuje možné způsoby jejich řešení.

Šestá kapitola seznamuje čtenáře s obvody GAL. Pozornost je věnována běžně dostupným obvodům GAL16V8 a GAL22V10. Kromě detailního popisu obvodů a způsobu programování jsou uvedeny dva jednoduché příklady použití.

Sedmá kapitola popisuje dva významné obvody: PWM (pulzněšířkový modulátor) a kmitočtový interpolátor (násobičku sledu impulzů). Jsou předvedena základní použití těchto obvodů pro řízení výkonu resp. napětí.

Osmá kapitola ukazuje různé užitečné konstrukce vytvořené na bázi číslicových obvodů. Jedná se o zajímavá použití ovládacích tlačítek (například zapnutí a vypnutím jedním tlačítkem), časovací obvody, řízení výkonových obvodů (stejnosměrné i střídavé obvody) včetně použití PWM.



Rozeberte si PC

Je to kniha pro opravdové kutily nového tisíciletí. O pájení, pájkách, páječkách, odpájení, antistatické ochraně, … Povídání o různých součástkách, ale tak trochu jinak než obvykle. Viděli jste již někdy sektory vašeho hard disku? … a jiné bláznivé návody. Až budete číst tuto knihu nebudete věřit vlastním očím!

Příručka je určena všem, kteří se chtějí snadno, rychle a bez horentních finančních nákladů dozvědět něco o elektronice, avšak bez zbytečných a složitých teorií. A aby pak to, co se dozví, mohli nějak prakticky využít. Populárně technicky psaná kniha s humorně komentovanými příklady z praxe, které se skutečně udály, určená především pro začínající elektroniky, badatele a kutily.

Autor Václav Šedý, vydalo nakladatelství BEN – technická literatura, 208 stran B5, obj. číslo 121051, MC 199 Kč.

426410

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/121125

Vladimír Váňa

Mikrokontroléry ATMEL AVR – popis procesoru a instrukční soubor

Vydalo nakladatelství BEN – technická literatura, Praha 2003
1. vydání

Recenzent Jiří Kopelent, David Matoušek

Vedoucí nakladatelství Libor Kubica
Vedoucí redakce a DTP Martin Havlák
Odpovědná a technická redaktorka Katka Hrubá

Sazba	Katka Hrubá
Obálka	Libor Kubica
Rozsah	336 stran
Tisk	Marten

objednací číslo	121125
EAN	9788073000837
ISBN	978-80-7300-083-0 (tištěná kniha)
	978-80-7300-380-7 (elektronická kniha v PDF)

TECHNICKÁ LITERATURA BEN®

426410