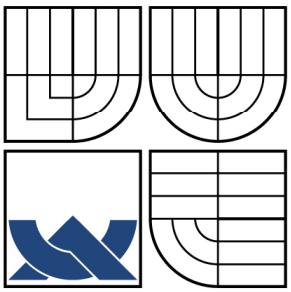
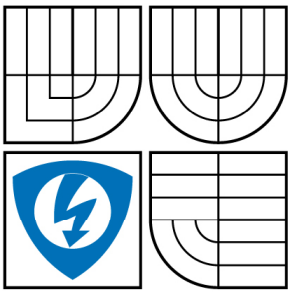


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV ELEKTROTECHNOLOGIE

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF ELECTRICAL AND ELECTRONIC TECHNOLOGY

MULTIMEDIÁLNÍ VÝUKOVÉ ROZHRANNÍ PRO PROGRAMOVANÍ MIKROPROCESORŮ AVR

MULTIMEDIAL EDUCATION INTERFACE FOR PROGRAMMING AND DEBUGGING SOFTWARE FOR AVR MICROCONTROLLERS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

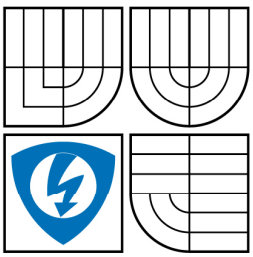
AUTOR PRÁCE — Bc. JOSEF ŠRAJLA

AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL PAVLÍK

SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav elektrotechnologie

Diplomová práce

magisterský navazující studijní obor
Elektrotechnická výroba a management

Student:	Bc. Josef Šrajla	***ID:***	54522
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Multimediální výukové rozhranní pro programovaní mikroprocesorů AVR

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s instrukční sadou mikrokontrolerů typu AVR a se skriptovacím jazykem PHP. Vytvořte vývojový diagram webové aplikace a otestujte jej.
Naprogramujte dynamickou webovou aplikaci, která umožní multimediální výuku programování mikroprocesorů řady AVR. Součástí webové aplikace bude také překladač instrukcí do binárního kódu.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání:	9.2.2009	***Termín odevzdání:***	29.5.2009

Vedoucí práce: Ing. Michal Pavlík

prof. Ing. Jiří Kazelle, CSc.
Předseda oborové rady

Abstrakt:

Diplomová práce se zabývá problematikou programování mikroprocesorů AVR. Zaměřuje se na vytvoření uceleného prostředí, ve kterém si bude moci uživatel osvojit metody potřebné k úspěšnému zvládnutí naprogramování mikroprocesoru. K tomu bylo zapotřebí vytvoření aplikace umožňující interaktivní simulaci procesoru. Předností této aplikace je jednoduché ovládání, přehlednost a hardwarová nenáročnost. K jejímu používání stačí uživateli pouze počítač s připojením na internet, odpadá veškerá instalace, která je u podobných produktů běžná.

Abstract:

Diploma thesis is concerned with programming of AVR microprocessors. Main task was development of compact environment where user can learn methods necessary for successful programming of microprocessor. It was necessary to make an application enabling interactive simulation of processor to reach this goal. User-friendly control, simplicity and low hardware demands are main advantages of this application. Only computer with internet connection is demanded. Complicated installation, usual for similar products, is avoided.

Klíčová slova:

AVR, mikroprocesor, programovaní, výuka, simulace, interaktivní

Keywords:

AVR, microcontrollers, programming, education, simulation, interactive

Bibliografická citace díla:

ŠRAJLA, J. Multimediální výukové rozhranní pro programovaní mikroprocesorů AVR. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. XY s. Vedoucí diplomové práce Ing. Michal Pavlík.

Prohlášení autora o původnosti díla:

Prohlašuji, že jsem tuto vysokoškolskou kvalifikační práci vypracoval samostatně pod vedením vedoucího diplomové práce, s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 29. 5. 2009

…….

Poděkování:

Děkuji vedoucímu diplomové práce Ing. Michalu Pavlíkovi za metodické a cíleně orientované vedení při plnění úkolů realizovaných v návaznosti na diplomovou práci.

OBSAH

1 ÚVOD 7

1.1 ÚVOD DO PROBLEMATIKY 7
1.2 PŘEHLED EXISTUJÍCÍCH APLIKACÍ, VÝHODY A NEVÝHODY 8
1.3 VÝHODY A NEVÝHODY MÉ WEBOVÉ APLIKACE 10

2 POPIS A OBSAH WEBOVÉ APLIKACE 11

2.1 ZÁKLADNÍ CHARAKTERISTIKA MÉ WEBOVÉ APLIKACE 12
2.2 TEORIE 12
2.2.1 Mikroprocesor 12
2.2.2 Klávesnice 15
2.2.3 Modul LED-diod 15
2.2.4 Modul LCD displeje 16
2.2.5 Maticová klávesnice 17
2.3 PROGRAMY 17
2.3.1 Test 17
2.3.2 Vrata 17
2.4 SIMULACE 18
2.5 HARDWARE 18
2.5.1 Programátor s rozhranním JTAG 18
2.5.2 Programátor s rozhranním ISP LPT 19
2.5.3 Vrata 20

3 SIMULACE 21

3.1 EDITOR 21
3.2 POLE ZNAKŮ 21
3.3 ŘÁDEK 21
3.4 PERIFERIE MIKROPROCESORU 22
3.4.1 Záložka Mfile 23
3.4.2 Záložka Simul 24
3.4.3 Záložka I/O 24
3.5 PRACOVNÍ REGISTRY 24
3.6 SLEDOVANÉ REGISTRY 25
3.7 CHYBOVÁ HLÁŠENÍ 26
3.8 FORMÁTOVÁNÍ 27
3.8.1 Tlačítko Bez syntaxe 27
3.8.2 Tlačítko Zvýraznění syntaxe 27
3.8.3 Tlačítko Opravy 28
3.8.4 Tlačítka Instrukce, Návěští a Direktivy 29
3.8.5 Zarovnání 29
3.9 NÁSTROJE 30
3.9.1 Příklad 30

3.9.2 Vložení hlavičky 30
3.9.3 Vytvoření mfile 31
3.9.4 Krokování 31
4 SOFTWAROVÉ ŘEŠENÍ 32
4.1 ROZPOZNÁVÁNÍ KÓDU 33
4.2 DEKÓDOVÁNÍ INSTRUKCE 33
4.3 STRUČNÝ POPIS JEDNOTLIVÝCH FUNKCÍ 35
4.1 PROMĚNNÉ 45
5 ZÁVĚR A VÝHLED DO BUDOUCNA 48
5.1 SOUČASNÝ STAV 48
5.2 MOŽNÉ SMĚRY DALŠÍHO VÝVOJE 48
5.3 POUŽITÉ ZNALOSTI A DOVEDNOSTI 49
6 POUŽITÁ LITERATURA 50
7 SEZNAM POUŽITÝCH ZKRATEK A SYMBOLŮ. 51
8 SEZNAM PŘÍLOH. 52

1 Úvod

1.1 Úvod do problematiky

Elektronika jako obor se vyvíjí velmi dynamicky. Nikoho dnes nepřekvapí mluvicí kávovar nebo auto, které umí samo zaparkovat. Aby však mohla zařízení takto pracovat, je zapotřebí jim požadované chování předem naprogramovat. První logické obvody pracovaly na mechanickém principu (např. soustavy ozubených kol). Postupně se prosadily součástky elektronické. Nejprve se využívali tranzistory, které později vystřídaly integrované logické obvody (nejznámější se stala řada 74XX). Všechny zmíněné prvky však mají jeden společný nedostatek. V případě nutnosti provést změnu v chování zařízení, je nutné zasáhnout do jeho konstrukce. Takové řešení označujeme jako hardwarové. Jeho použití může být výhodné jen u velmi jednoduchých zařízení s velkou sériovou výrobou. Častěji ale požadujeme větší univerzálnost. Z výrobního hlediska se vyplatí produkovat jeden typ výrobku, který bude, třeba jen s malou úpravou, možno použít ve více aplikacích.

V tomto ohledu bylo průlomem sestrojení součástky, do které je možno snadno a opakovaně uložit požadované chování (naprogramovat posloupnost logických kroků) a kterou je možno vyrábět ve velkých sériích za přijatelnou cenu. Takovou součástkou je mikroprocesor.

Základní vlastnosti:

- univerzálnost použití,
- malé rozměry,
- velké množství variant vstupně výstupních portů,
- nízká spotřeba oproti jiným řešením,
- snadná změna funkce stávající aplikace,
- příznivá cena.

Funkce mikroprocesoru tedy můžeme měnit použitím obslužného softwarového vybavení. Abychom mohli plně této vlastnosti využít, musíme zvládnout nový samostatný obor – programování.

Dostupné webové informace o programování mikroprocesorů

Byly vyhledány webové stránky, které se zabývají problematikou mikroprocesorů AVR a jejich programováním. Nalezl jsem více internetových prezentací, ale žádná nenabízí ucelený postup, který by provedl čtenáře od teorie (popis mikroprocesorů, začátky programování), přes simulaci chování mikroprocesoru až po návrh reálné aplikace. Blíže jsem procházel a testoval webové stránky psané v českém jazyce. Navrhovaná zapojeni a uvedené programové vybavení nemusí uživatele dovést k úspěšnému zprovoznění mikroprocesoru. Velký problém nastává v momentě, kdy mikroprocesor programván. Je třeba správně určit zdroj hodinového kmitočtu a další parametry procesoru, které se však nastavují v různých programech odlišně. Mnoho začínajících uživatelů může mít problém s naprogramováním mikroprocesoru. Nechtějí investovat do zakoupení příslušného hardwarového vybavení, ani neumí postavit si vlastní programátor, který by nahrání napsaného kódu do mikroprocesoru zajistil. Komplikací v začátcích může být i volba příslušného softwarového vybavení. V dnešní době jich je na trhu větší množství a pro uživatele, který nemá s programováním žádné zkušenosti, může být volba velmi obtížná. Pro lepší orientaci uvádím přehled dostupných aplikací, jejich výhody a nevýhody.

1.2 Přehled existujících aplikací, výhody a nevýhody

V této kapitole je uveden výpis jednotlivých aplikací včetně jejich stručného popisu.

AVR Studio

Tento program je zatím nejvěrnějším simulačním prostředím na trhu pro mikroprocesory AVR. Jedná se o produkt firmy Atmel, která vyrábí i samotné procesory. Program může být použit na širokou škálu mikroprocesorů. Programem lze simulovat reálný mikroprocesor, ale také ladit aplikaci v reálném prostředí pomocí rozhraní JTAG.

Výhody:

- Podporuje velké množství mikroprocesorů,
- umožňuje simulovat i ladit přímo na mikroprocesoru,
- dostupný zdarma z webových stránek www.atmel.com.

Nevýhody:

- Nutná instalace,
- program nepodporuje češtinu.

BASCOM-AVR

K programování se používá vyšší programovací jazyk, kterým lze snadno naprogramovat jednoduché aplikace. Program umožňuje jednoduchou simulaci. Ta však kromě zobrazení na virtuálních perifériích neumožňuje podrobnější zobrazení vnitřních registrů. Výsledný kód nemá uživatel pod kontrolou, protože jej z vyššího programovacího jazyka generuje program a může tak být méně čitelný, než pokud bychom jej psali rovnou v Assembleru.

Výhody:

- rychlý vývoj jednoduchých aplikací.

Nevýhody:

- není dostupný zdarma,
- nutná instalace,
- nutné naučit se syntaxi programovacího jazyka,
- velký a nepřehledný výstupní binární kód,
- obtížná kontrola běhu programu,
- není možné nasadit do náročných aplikací, kde je požadován maximální výpočetní výkon.

CodeVisionAVR

Podobně jako BASCOM umožňuje snadno a rychle zadat kód. Jedná se o kombinaci Assembleru a jazyka C. Program podporuje periferie jako LCD, I^2C, a další. Kód napsaný v tomto prostředí má některé výhody jazyka C a zachovává si dobrou přehlednost.

Výhody:

- jde o kompiler jazyka ANSI C,
- automaticky vkládané části kódu v závislosti na vybraných periferiích,
- přehledný kód.

Nevýhody:

- licence stojí 5tis. Kč,
- při vkládání kódu nebere v úvahu integrovaná rozhraní (místo připravených hardwarových řešení vkládá své vlastní softwarová řešení, což vede ke zpomalení výsledné aplikace),
- nutné naučit se syntaxi programovacího jazyka.

WinAVR

Jde o kompiler ANSI C s GCC licencí využívající AVR Studio jako editoru s možností zvolit zvýraznění syntaxe kódu. Optimální řešení pro uživatele, kteří již mají zkušenosti s psaním programů v jazyku C.

Výhody:

- jednoduché psaní kódu,
- není nutno se učit speciální syntaxi, postačí příkazy jazyka C,
- je možné vygenerovat velké množství výstupních souborů.

Nevýhody:

- není možné simulovat napsaný kód přímo v programu,
- obtížné nastavení,
- komplikovaná instalace a shánění knihoven.

1.3 Výhody a nevýhody mé webové aplikace

Zde jsou stručně zmíněny výhody a nevýhody mé webové aplikace v porovnání s konkurenčními produkty.

Výhody:

- není nutná instalace,
- je dostupná zdarma,
- nenáročnost na rychlost připojení,
- nízké požadavky na hardwarové vybavení počítače,
- velmi jednoduchá obsluha,
- možnost nadstandardního formátovaní kódu psaného v Assembleru (viz 3.8.5),
- česká lokalizace,
- možnost simulovat napsaný kód,
- snadná rozšiřitelnost o další periférie,

Nevýhody:

- nutnost internetového připojení,
- mikroprocesor není možné programovat přímo z této aplikace,
- zatím nenaprogramovány všechny instrukce,
- zatím omezená podpora Mikroprocesorů.

2 Popis a obsah webové aplikace

Hlavní menu stránek obsahuje položky ***Úvod***, ***Teorie***, ***Programy***, ***Simulace***, ***Hardware***, ***Diskuze***, ***Ukládaní***, ***Odkazy***. Ty jsou vždy zobrazeny v horní části obrazovky. Po vybrání požadované položky se pole zabarví a tím je znázorněno, kde se uživatel v aplikaci nachází. Některé položky mají podmenu, které se nachází v liště pod hlavním menu. Konkrétní podobu stránek je možné vidět na www.avr.poruce.com.

Obr. 1: Rozvržení WWW stránky.

Stránky mají jednoduchý jednotný vzhled. Do HTML (z anglického HyperText Markup Language) šablony je pomocí PHP podle zadaných parametrů vkládán obsah. Vzhled jednotlivých komponent je definován pomocí CSS stylu. Toto řešení je výhodné nejen pro udržení jednotného vzhledu, ale i pro snadnou změnu designu celých stránek. Složky a soubory na webovém úložišti jsou uspořádány stejně jako položky v menu. Toto členění pomáhá udržet uložený obsah přehledný. Složky hlavního menu mohou obsahovat soubor „podmenu.php“. V tomto souboru je definována podoba podmenu příslušné kategorie a složky pojmenované podle položek v podmenu. Ty pak obsahují soubor, který se zobrazuje v hlavním okně.

2.1 Základní charakteristika mé webové aplikace

Vytvořená webová aplikace, je určena jak úplným začátečníkům, tak již pokročilým uživatelům, kteří se v programování mikroprocesorů chtějí zdokonalit. Obsahuje ucelené informace týkající se problematiky mikroprocesorů AVR. Uživatele vede krok po kroku celým procesem vytváření vlastní aplikace od nastudování teoretických podkladů (položka ***Teorie***) přes napsání programu a jeho simulaci (položka ***Simulace***) po samotnou realizaci a oživení vlastní aplikace na mikroprocesoru (položka ***Hardware***).

Nejdůležitější částí aplikace je interaktivní simulace chování mikroprocesoru (realizována interaktivní simulace mikrokontrolér ATMega16). Uživatel nemusí vlastnit skutečný mikroprocesor, přesto dostává po každé napsané instrukci zpětnou vazbu, jak by se mikroprocesor zachoval. Má tak kontrolu, zda správně zadaný instrukci pochopil. Aplikace obsahuje rozhraní vstupů, výstupů a přehled o obsahu vnitřních registrů. Uživatel má možnost k jednotlivým portům připojovat periférie, jako jsou signalizační LED-diody, klávesnice, krokový motor, LCD, potenciometr a další. Běh programu může být řízen na základě podnětů z těchto periférií. Zároveň lze sledovat stavy jednotlivých proměnných a stav registrů mikroprocesoru. K lepšímu pochopení problematiky programování aplikací mikroprocesoru přispívá i animovaný průvodce v podobě flashové prezentace.

Uživatelé mohou testovat uložené programy (položka ***Programy***), vytvářet programy nové a ty zde ukládat (položka ***Ukládání***) pro inspiraci ostatním uživatelům.

Uživatelé si budou moci na webu vytvořit svůj vlastní profil a po přihlášení tak mít k dispozici své ukončené i rozpracované projekty. Na svých projektech tak budou moci pracovat na kterémkoliv počítači připojeném k internetu, třeba v internetové kavárně.

2.2 Teorie

Podrobné informace o hardwaru, pro který je naprogramována simulace.

2.2.1 Mikroprocesor

V současné době existuje celá řada mikroprocesorů. Tato práce je zaměřena na řadu AVR. Všechny příklady jsou psány pro konkrétní mikroprocesor ATMega16 od firmy Atmel.

Parametry ATmega16 podle katalogového listu:

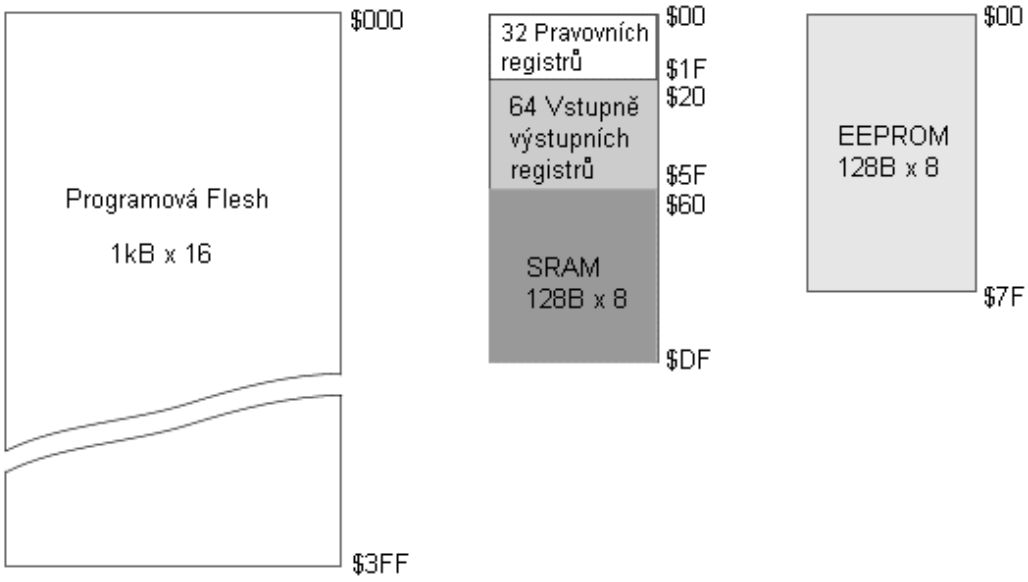
- instrukční soubor obsahuje 131 instrukcí,
- 32 registrů délky 8 bitů,
- čtyři 8bitové vstupně/ výstupní porty (celkem tedy 32 vstupů/ výstupů),
- hodinový kmitočet 0 až 16 MHz, maximální výpočetní výkon až 16 MIPS,
- paměť programu je tvořena zabudovanou Flash, kapacita je 16 kB, počet přeprogramování je 1000 cyklů,
- datová paměť RAM kapacity 1 kB,
- datová paměť E^2PROM kapacity 512 B, počet přeprogramování je 100 000 cyklů,
- Flash a E^2PROM jsou programovatelné přímo v systému pomocí rozhraní SPI, nebo JTAG,
- dva 8bitové čítače/ časovače, jeden 16bitový (dokonalejší) čítač/ časovač,
- čtyři PWM kanály,
- analogový komparátor, 10bitový A/D převodník,
- jednotky USART, SPI, TWI (podpora I^2C),
- jednotky WDT, Power-on reset,
- zabudovaný RC oscilátor,
- napájení integrovaného obvodu se může pohybovat v rozsahu 4,5 až 5,5 V a zajišťuje jej stabilizovaný zdroj 5 V s obvodem 78S05. Proudový odběr se pohybuje od 1 μA,
- pouzdra DIP 40, TQFP 44.

Podrobnější popis:

Mikroprocesor ATmega16 je vybaven jednoduchým 8bitovým čítačem/časovačem. K dispozici je také další 8bitový asynchronní čítač/časovač (jeho hodinový kmitočet není odvozen od kmitočtu jádra, ale z „hodinkového" krystalu 32,768 kHz). Nejdokonalejší je 16bitový čítač/časovač (režimy Output Compare, Input Capture, PWM a další). [1]

Mikroprocesor ATmega16 je vybaven obvodem WDT (Watch-Dog Timer, hlídač správného běhu programu) a analogovým komparátorem. Také je k dispozici zabudovaný synchronní a asynchronní sériový kanál USART a 10bitový A/D převodník (3 funkční režimy: 8 kanálů SE, 7 diferenčních kanálů, 2 diferenční kanály s volitelným ziskem 1x, 10x a 200x). SPI kanál a TWI rozhraní zajišťuje komunikaci s periferními obvody buď ve stylu sběrnice Microware, nebo I^2C. Rozhraní SPI se také používá pro programování mikroprocesoru. Zajímavostí je zabudovaný kalibrovaný RC oscilátor. V případě jeho použití není třeba připojovat krystal nebo vnější zdroj hodinového signálu. Relativní odchylka generovaného kmitočtu je při napájecím napětí 5 V a teplotě 25 °C, pouze ±1 %. Pomocí zabudované děličky lze vybrat kmitočty 1, 2, 4 nebo 8 MHz. Rozhraní JTAG zajišťuje především podporu pro ladění aplikace přímo na čipu (není nutno použít speciální emulační čipy).[1]

Mikroprocesory řady AVR využívají harvardskou architekturu, která se vyznačuje odděleným paměťovým prostorem pro program a data. Jednotlivé datové segmenty jsou tvořeny pamětí typu FLASH, RAM, E^2PROM. Mapa vnitřní paměti včetně adres segmentů je na obr. 2.

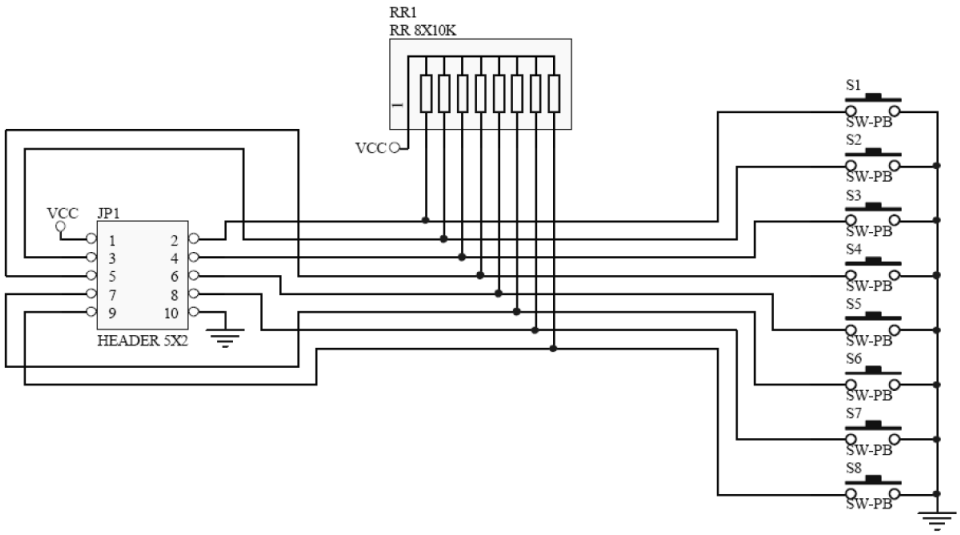


Obr. 2: Struktura vnitřní paměti ATmega16.

2.2.2 Klávesnice

Stav logické jedničky na výstupu v klidovém stavu zajišťuje rezistorová síť. Je tvořená odpory 10 kΩ, které jsou připojeny ke kladnému potenciálu. V případě, že dojde ke stisknutí tlačítka, výstup se uzemní. Jedná se o odporový dělič tvořený stisknutým tlačítkem a odporem 10 kΩ. Jelikož odpor stisknutého tlačítka je zanedbatelný v porovnání s odporem rezistoru s hodnotou 10 kΩ, je téměř celé napájecí napětí na odporu a na výstupu se objeví logická nula. Jednotlivé signály jsou vyvedeny na standardizovaný desetipinový konektor, který lze připojit k libovolnému portu mikroprocesoru.

Pomocí klávesnice se simulují stavy externích periférií, které jsou později použity v reálné aplikaci. Tyto periférie jsou dvoustavová čidla tlaku, nebo teploty, optické závory, bezpečnostní spínače, magnetické kontakty, senzor osvětlení a mnoho dalších. Dvoustavových veličin je dnes v digitálním věku nepřeberné množství a setkáme se s nimi i u většiny senzorů.

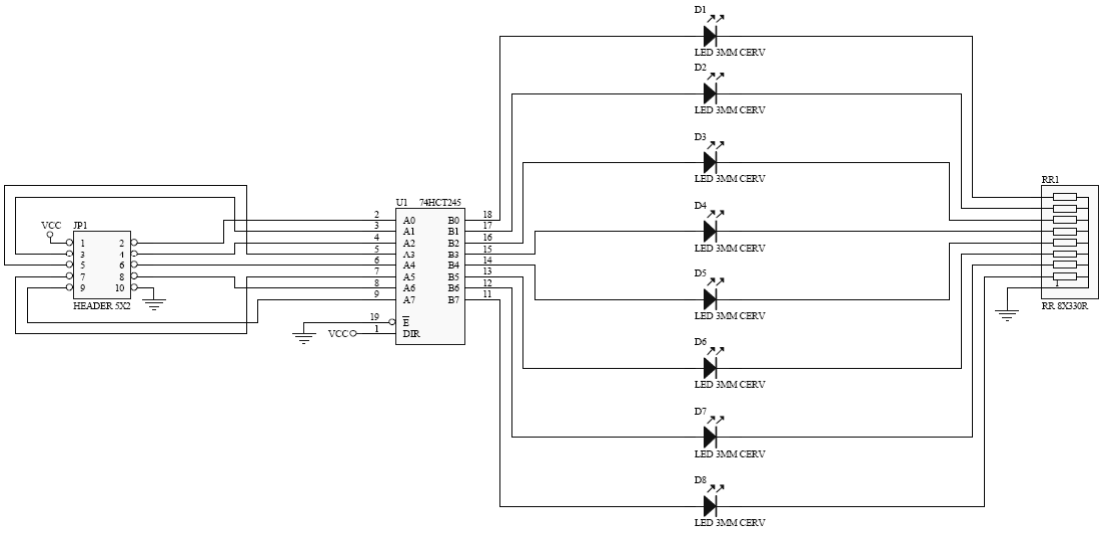


Obr. 3: Zapojení klávesnice

2.2.3 Modul LED-diod

K zobrazování stavů portů dat je využito osmi na sobě nezávislých LED-diod. Pro proudové přizpůsobení je použit budič sběrnice 74HCT245. Omezení proudu diodou zajišťuje rezistorová síť RR 8X330R. Jednotlivé signály jsou vyvedeny na univerzální sběrnici. To umožňuje připojit tento modul na libovolný port mikroprocesoru. Jelikož jsou diody zapojené přes rezistor na nízký potenciál (zemnící svorky), stav logické jedničky na výstupu mikroprocesoru rozsvítí diodou. Stav logické nuly na výstup způsobí zhasnutí LED-diody.

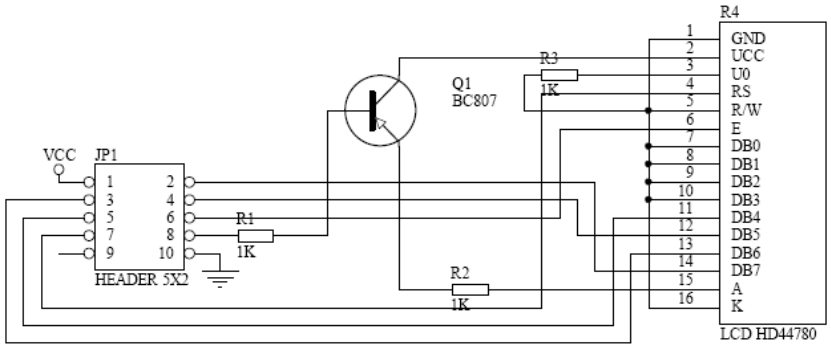
Předchozí modul klávesnice lze připojit k tomuto modulu paralelně. Což umožňuje na jednom portu mikroprocesoru sledovat výstupní signály na modulu ***LED-diod*** a současně i simulují stavy externích periférií pomocí modulu ***Klávesnice***. Proudové přizpůsobení pomocí budiče 74HCT245 není nutné používat při použití malé zátěže do odběru proudu 20 mA.



Obr. 4: Zapojení LED-diod

2.2.4 Modul LCD displeje

Cenově nejdostupnější varianta LCD (do 300 Kč). Toto zapojení je pro LCD řízené řadičem HD44780. Rezistorem R3 se nastavuje kontrast. Jeho hodnotu je nutno zvolit dle použitého typu displeje (řádově kΩ). To platí i o pro podsvětlení. To se nastavuje hodnotou odporu R2. U tohoto zapojení je možné podsvětlení řídit pomocí mikroprocesoru (osmý vývod na konektoru). Jsou přípustné dva stavy (vypnuto a zapnuto). V případě požadavku na plynulé rozsvícení či vypnutí LCD je možné použít PWM (Pulse Width Modulation) modulace.

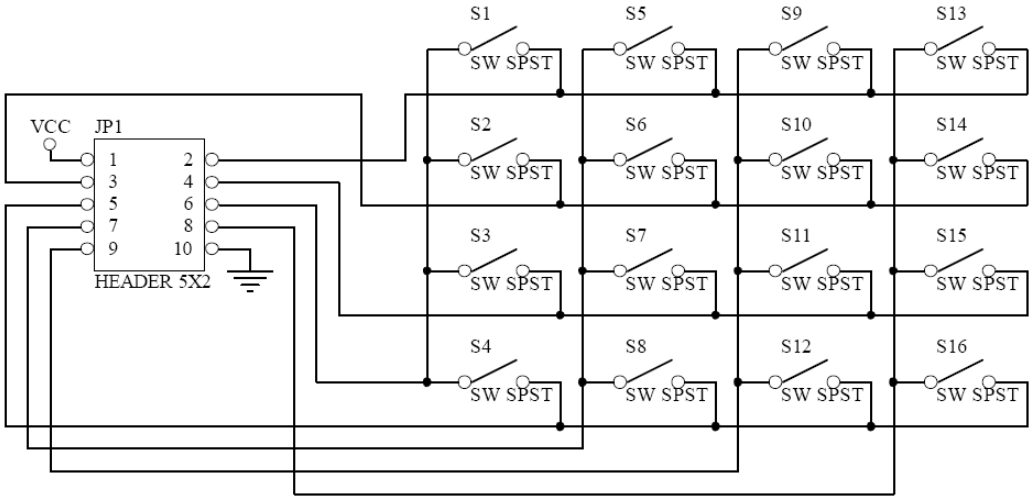


Obr. 5: Zapojení LCD displeje

2.2.5 Maticová klávesnice

Jednoduché zapojení maticové klávesnice lze zakoupit i jako celek. Výstupy je vhodné přizemnit odporem. Pro tyto účely je vhodné použít odporovou řadu. Hodnota odporů se volí v řádech kΩ. Toto zapojení umožní snímat stav šestnácti tlačítek, a to při použití osmi vodičů. Někdy je výhodné z důvodu prostorového uspořádání. využít jednoho mikroprocesoru pro sledování stavu klávesnice. Ten pak může data posílat do řídicího mikroprocesoru po sériové lince.

Uspořádání kláves do maticové struktury je v modifikované podobě použito i u počítačové klávesnice, kde se využívá dokonce i řešení komunikace po sériové lince.



Obr. 6: Maticová klávesnice

2.3 Programy

Příklady již naprogramovaných a otestovaných programů v Assembleru.

2.3.1 Test

Pro demonstraci již naprogramovaných instrukcí mikroprocesoru je vytvořen demonstrační program, který se nachází pod záložkou ***Test***. Pro vložení příkladu do simulace je určeno tlačítko ***Příklad***. Více vkládání testovacího příkladu je uvedeno v kap. 3.9.1.

2.3.2 Vrata

Kód v Assembleru, který je určen pro konstrukci modulu ***Vrata***, popsaného v kap. 2.5.3.

2.4 Simulace

Simulace reálného mikroprocesoru. V tomto případě je realizována interaktivní simulace mikroprocesoru ATMega16.

Simulace je vytvořena v programu Macromedia Flash 8 a je možné ji spouštět přímo v internetovém prohlížeči. Uživatel si tak bude moci lépe představit možnosti, které nabízí využití mikroprocesoru. Výhodou Flashové aplikace je také to, že neběží na serveru, ale na počítači uživatele. Při připojení na web dojde jen ke stažení méně rozsáhlých dat.

To umožňuje připojení velkého množství klientů současně. Současná podoba flashové aplikace obsahuje pouze 30,2 kB dat. Při zadávání kódu a jeho simulaci se již žádná data se serverem nevyměňují, což je výhodné v místech pomalejšího připojení. Uvítá to také uživatel při výpadku sítě. Nedojde tak ke ztrátě rozpracované práce a v práci je možné pokračovat i bez připojení k serweru. Podrobnějšímu popisu simulace je věnována kapitola 3.

2.5 Hardware

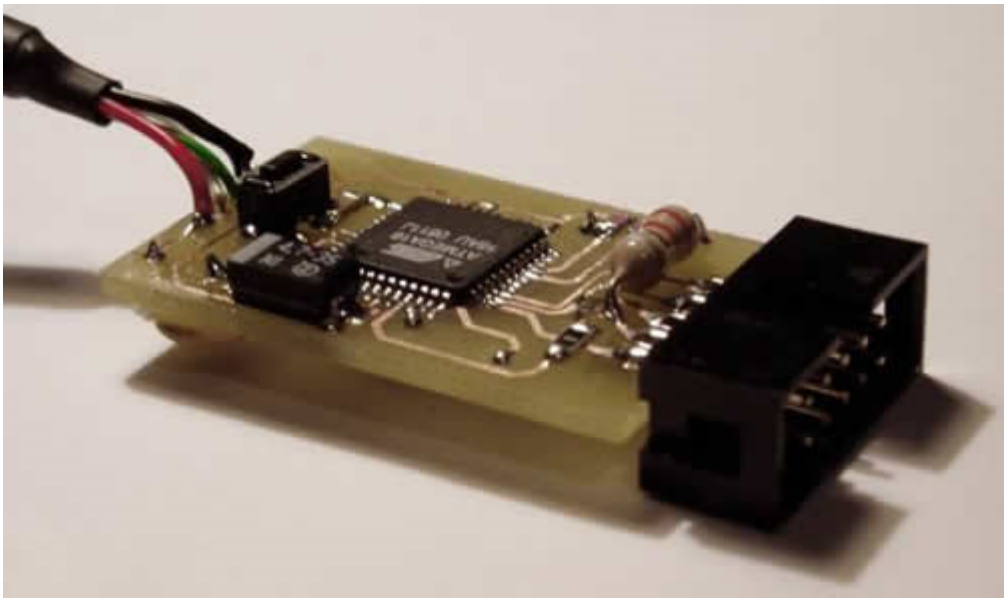
Realizovaná zařízení, která využívají mikroprocesor (aplikace mikroprocesoru), nebo slouží j jeho naprogramování.

2.5.1 Programátor s rozhranním JTAG

Jedná se o programátor mikroprocesorů AVR - rozhraní JTAG.

Parametry:

- napájení přímo s USB,
- proudová ochrana (0,5A),
- zpožděné zapnutí napájení,
- vyhlazené přes cívku,
- malé rozměry,
- kompatibilita a AVR studio,
- možnost ladění přímo v aplikaci,
- rychlost.



Obr. 7: JTAG konstrukce

2.5.2 Programátor s rozhranním ISP LPT

Jedná se o programátor mikroprocesorů AVR - rozhraní ISP.

Parametry:

- připojeno na LPT port PC,
- ochrana LPT proti zkratu na ISP,
- jednoduchá konstrukce a snadné oživení,
- malé rozměry.

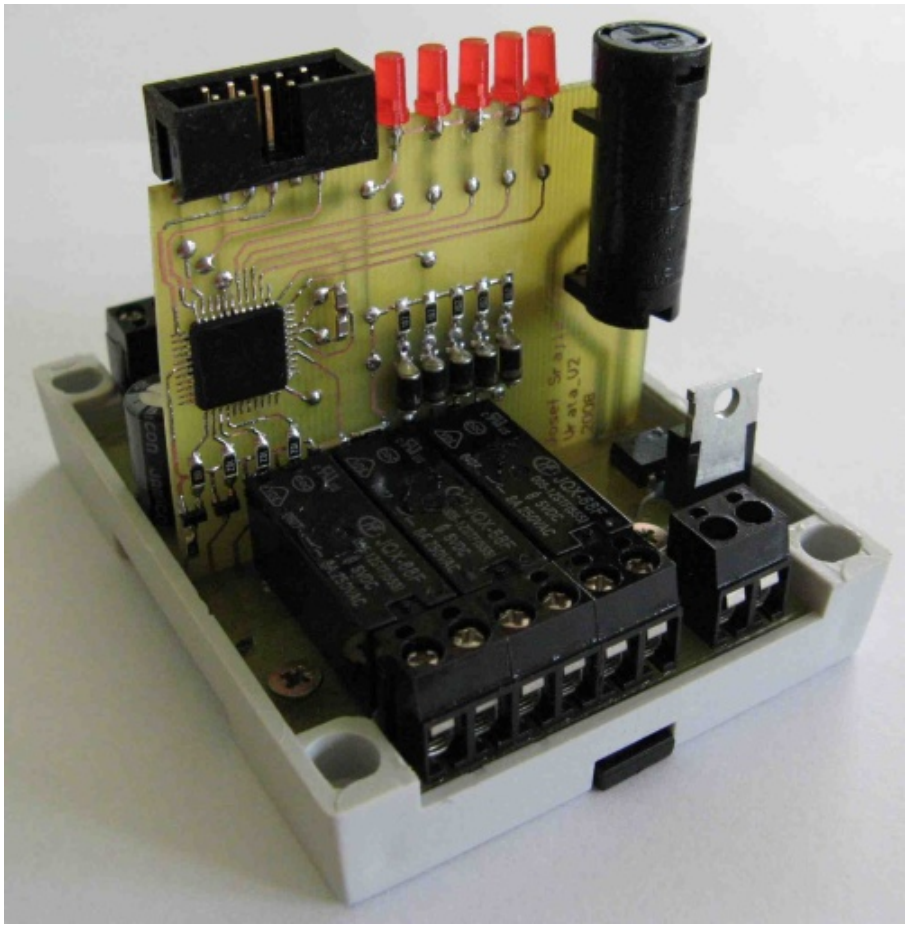
Obr. 8:ISP LPT programátor

2.5.3 Vrata

Realizovaná a zdokumentovaná konstrukce otevíraní vrat řízené mikroprocesorem.

Parametry:

- vstup dálkového ovládání,
- vstup z optické závory,
- vstup dvou koncových rozpínacích kontaktů,
- 3x spínací kontakt 250V/8A (relé),
- 1X spínací kontakt 250V/16A (triak),
- 5x signalizační LED dioda,
- pojistka,
- napájení ze sítě 230V AC,
- vyveden programovací konektor JTAG.



Obr. 9: Otevíraní vrat řízené mikroprocesorem

3 Simulace

Aby bylo možné simulovat reálný program, je nutno jej nejprve vložit do webového simulačního prostředí. K tomuto účelu slouží rozhraní pro jeho vkládání. Aplikace pro tento účel využívá textové pole, v němž lze zadávat kód Assembleru. Aby bylo zadávání kódu snazší a uživatelsky přívětivější, je možné využít některých funkcí pro barevné zvýraznění specifických částí kódu. Možnosti, které tato aplikace nabízí, jdou dále, než je tomu u jiných komerčních řešení. Jednotlivé funkce budou popsány níže.

3.1 Editor

Editor je textové pole, ve kterém se zobrazuje programový kód v Assembleru, který uživatel zadává, nebo předem napsaný uložený kód. Pole má rozsah 23 řádků a tento rozsah lze programově měnit. V případě, že zadaný kód překročí rozsah 23 řádků, objeví se po jeho pravé straně posuvná lišta. Ta umožní vertikální pohyb v programovém kódu. Obdobnou funkci zajistí lišta objevující se na spodní straně okna ve směru horizontálním.

Reálné rozměry okna jsou (560 x 516) px. Použitý font Courier, velikost 17 má shodné rozměry mezer s rozměry znaků. Zobrazovaný kód je pak přehlednější. Tato vlastnost je výhodná také u zobrazení obsahu registru a ostatních polí, kde je potřebné při vizualizaci dodržet sloupce.

3.2 Pole znaků

K ***Editoru*** je zleva těsně připojen jeden sloupec zabarvený do šedého odstínu, pro zobrazení jednoho vybraného znaku ke každému řádku programového kódu.

Použité znaky:

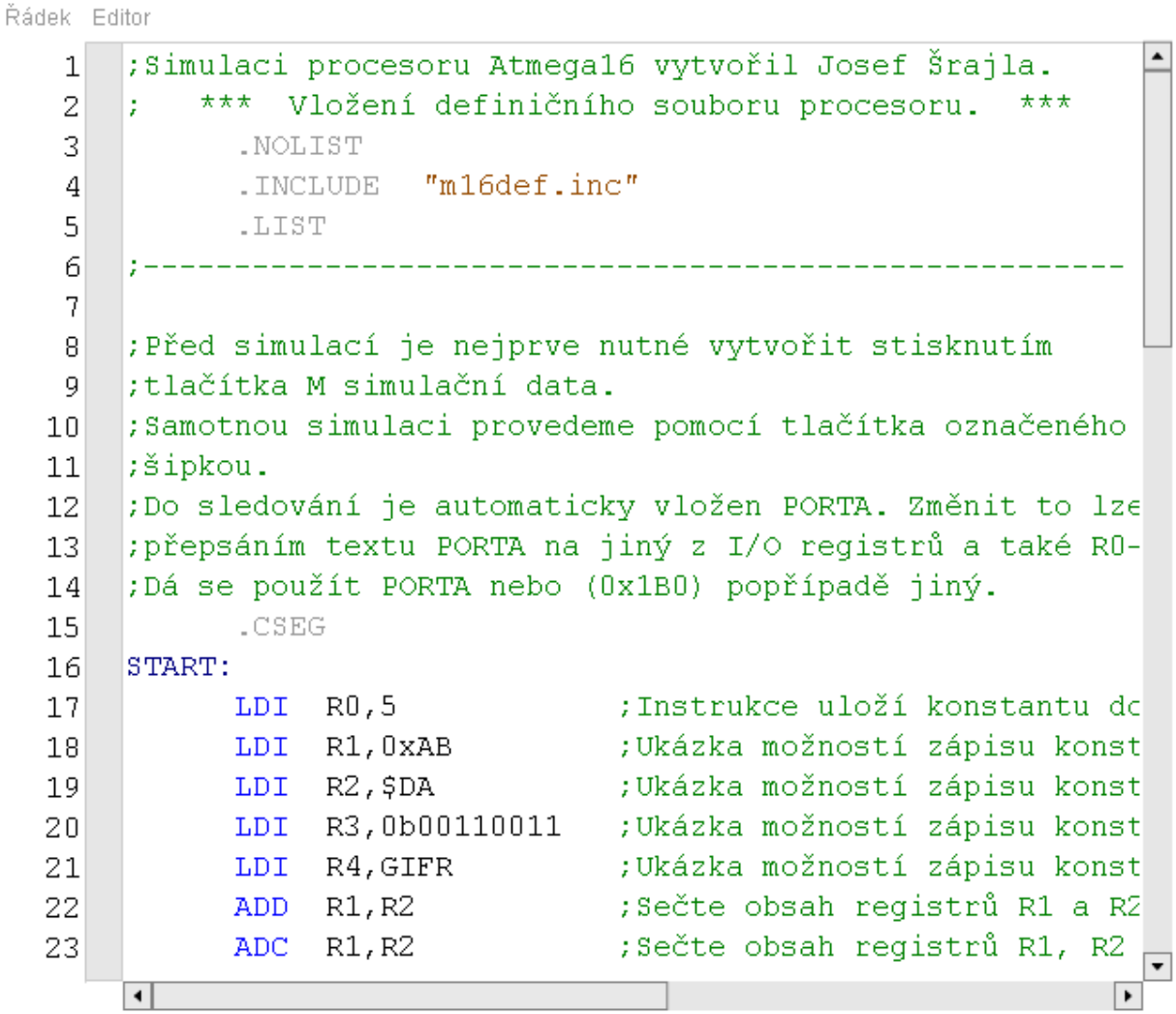
- ► Ukazuje, na jakém řádku se nachází simulace.
- ● Breakpoint. Zarážka, na které se má simulace při krokování zastavit.

3.3 Řádek

Ke sloupci ***Pole znaků*** je zleva těsně připojen sloupec s označením čísla řádku programového kódu zobrazeného v ***Editoru***. Generování čísla řádku v ***Editoru*** se ukázalo, jako poměrně výpočetně náročné. Počítání řádků je realizováno jinou metodou.

Celá řada čísel je uložena do textového pole a toto pole se jen vertikálně posunuje podle textu v ***Editoru***. Tato metoda má jednu nevýhodu a to, že je omezen počet zobrazených řádků.

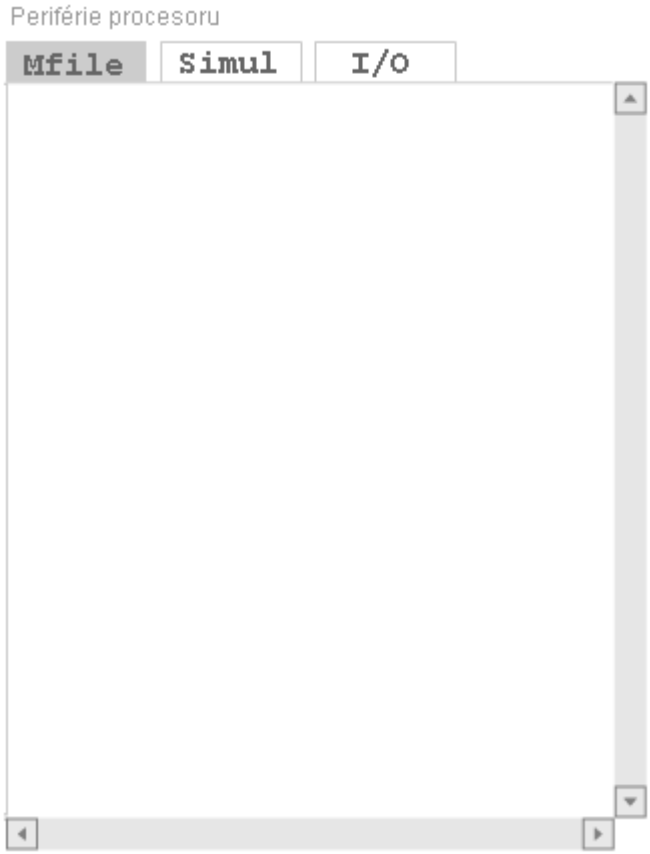
Maximální počet řádků zadaného kódu omezen není, ale zobrazení čísla řádku se zastaví na hodnotě 10000. Nepředpokládá se však překročení této hranice a tak je toto omezení nedůležité.



Obr. 10: Grafická podoba oka editoru.

3.4 Periferie mikroprocesoru

Jde o textové pole se záložkami ***Mfile***, ***Simul***, ***I/O***. Je vybaveno rolovacími lištami což umožňuje zobrazovat i rozměrnější textové řetězce.



Obr. 11: Grafická podoba okna Periférie mikroprocesoru

3.4.1 Záložka Mfile

Záložka ***Mfile*** zobrazuje obsah souboru mfile totožného s tím, který generuje program AVR Studio 4. V něm jsou zobrazeny všechny deklarace symbolů použitých v programovém kódu (mimo rezervovaná slova Assembleru). Jsou zde např. uvedeny adresy jednotlivých návěští, adresy vstupně/výstupních registrů.

Soubor mfile se vygeneruje při překladu programového kódu a je třeba jej vygenerovat před každou simulací.

Doporučené deklarace k mikroprocesoru ATmega16 jsou zapsány v souboru m16def.inc. Ten vložíme do programu pomocí příkazu `.INCLUDE "m16def.inc"` v hlavičce programu. Pokud žádný soubor s deklaracemi do programu nevložíme, bude v souboru mfile uloženo vše, co si uživatel deklaroval sám. Deklarace se provádí pomocí direktivy .EQU.

Příklad správného zápisu:

```
.EQU PORTC = 0x15
.EQU E = 2.
```

3.4.2 Záložka Simul

Záložka ***Simul*** obsahuje vybrané části kódu, které jsou pro simulaci zajímavé. Informace jsou dostupné až po vygenerování simulačních dat. Parametry instrukcí jsou zde zobrazeny buď čísly v hexadecimálním formátu, nebo registry (začínají písmenem R a za ním následuje označení registru). Údaj je dále doplněn číslem řádku, na kterém se v původním kódu instrukce nachází. To umožňuje při simulaci zvýraznit odpovídající řádek v programovém kódu. První řádek obsahuje hlavičku s popisem zobrazovaných sloupců.

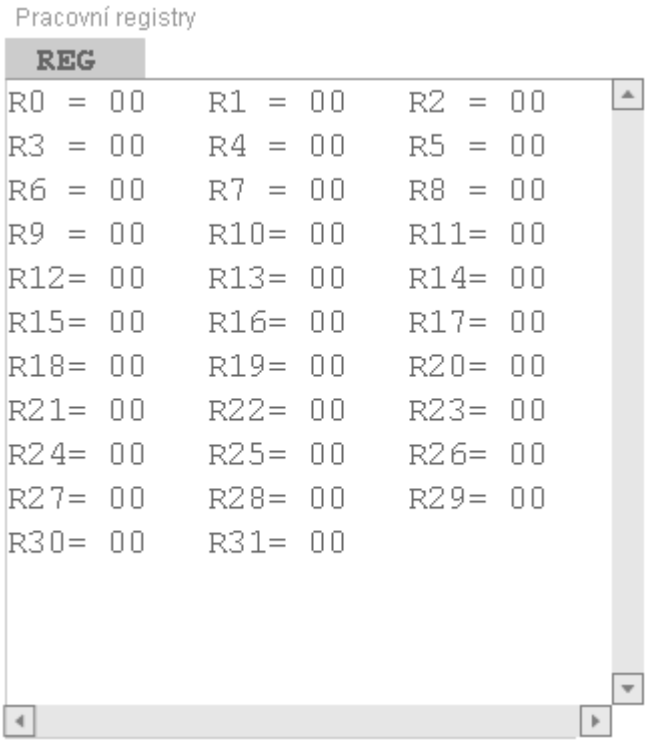
Informace v záložce ***Simul*** je výhodné před samotnou simulací zkontrolovat a zjistit, zda jsou načteny všechny parametry korektně.

3.4.3 Záložka I/O

Zde se zobrazují veškeré vstupně výstupní registry. Je možné sledovat jejich aktuální stav a to jak v hexadecimální, tak i binární podobě. Na prvním řádku se nachází popis jednotlivých sloupců.

3.5 Pracovní registry

Jde o textové pole, ve kterém je možno sledovat obsah pracovních registrů R0 až R31. Je vybaveno rolovacími lištami což umožňuje zobrazovat i rozměrnější textové řetězce.

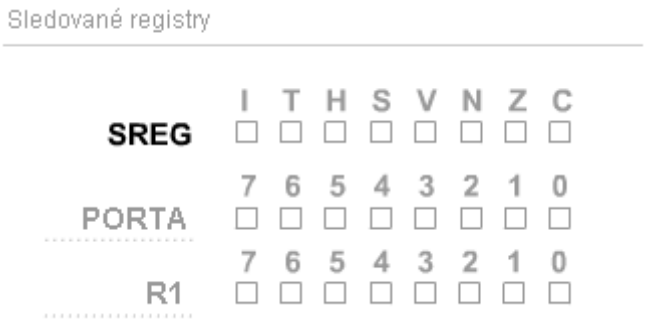


Obr. 12: Grafická podoba okna Pracovní registry

3.6 Sledované registry

Jde o grafické interaktivní rozhraní jednoho vybraného registru (pro svou důležitost vybrán registr SREG) a dvou volitelných registrů (počínaje R0-R31 až po vstupně/výstupní registry). Lze v něm sledovat aktuální stavy zvolených registrů a tyto stavy podle potřeby měnit. Stav sledovaných registrů lze měnit buď přímo kliknutím na příslušný bit, nebo při simulaci pomocí instrukcí v kódu Assembleru. Pole bitu má dva stavy.

V případě, zobrazení stavu logické úrovně „L“, je pole zabarveno bílou barvou. V opačném případě, kdy je zobrazen logický stav „H“, je pole zabarveno černě. Zajímavou vlastností tohoto zobrazovacího prvku je také jeho schopnost měnit stav již při zadávání textu sledovaného registru. Znamená to, že ihned po napsání je možné vidět stav registru bez nutnosti volbu potvrzovat. Tímto způsobem můžeme rychle navolit počáteční podmínky simulace. Na následujícím obrázku je vyobrazena grafická podoba rozhraní.



Obr. 13: Grafická podoba okna Sledované registry

Vrchní registr SREG je zobrazen permanentně. Spodní dva registry je možné změnit přepsáním názvu požadovaného registru. Název registru je jen reprezentací konstanty, která ukazuje na konkrétní místo v paměti mikroprocesoru. Má-li tato volba pracovat korektně, musí být zadanému názvu přiřazena konstanta. V případě, že nechceme nebo nemůžeme zadávat adresu tímto způsobem, můžeme zadat přímo její adresu. Jsou podporovány všechny typy číselných soustav.

Volba registru PORTA a R1 na obrázku 16 není náhodná. Souvisí s demonstračním programem, který je možno vložit pomocí tlačítka ***Příklad*** v panelu ***Nástroje*** (viz kapitola 3.9.1).

Rychlý náhled stavů registrů funguje také jako klávesnice.

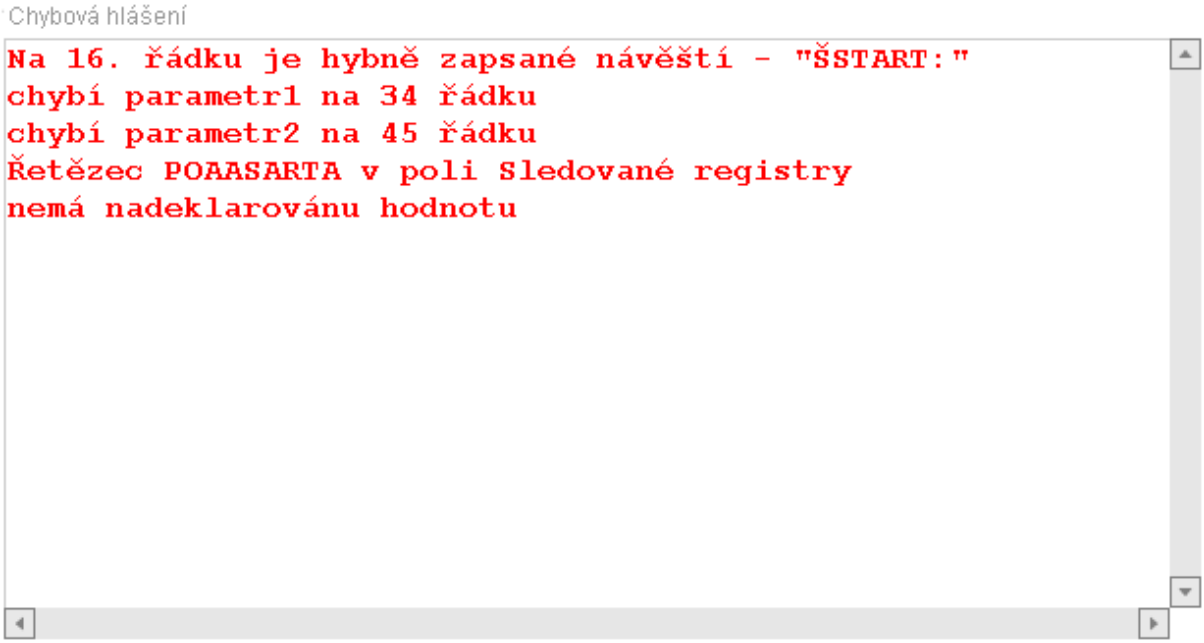
3.7 Chybová hlášení

Jde o textové pole, ve kterém je uživatel informován o chybách, které vznikly při kompilaci. Celý text zobrazený v tomto poli má červenou barvu, aby okamžitě upoutal pozornost uživatele. Toto pole má případě bezchybně zapsaného kódu zůstat prázdné. Jednotlivé případy, kdy dojde k vypsání chybového hlášení, jsou řazeny podle doby jejich vzniku, tedy podle výskytu chyby při kompilaci programu. Pro snazší orientaci je u chyby a jejího popisu přiřazena informace o čísle řádku, na kterém se chyba vyskytla.

Chybová hlášení:

- na řádku … není nadeklarovaná hodnota u … ,
- chybí parametr1 na … řádku,
- chybí parametr2 na ... řádku,
- chybný znak "," na' … řádku,
- na … řádku je druhé .INCLUDE -pro správnou funkci jej odstraňte,
- nebyl nalezen soubor pro vložení na .. .řádku -zkontrolujte: text v uvozovkách,
- není hodnota u ORG na … ,
- není hodnota nebo proměnná u DEF na … ,
- chybí "=" na … řádku,
- není deklarovaná hodnota nebo proměnná u EQU,
- chybí "=" na … řádku,
- na … řádku je nepovolený znak … ,
- chybný parametr u SBI na … řádku,
- chybný parametr u CBI na … řádku,
- řetězec … v poli Sledované registry nemá deklarovánu hodnotu,
- na … řádku je chybně zapsané návěští - "…".

Z výčtu těchto chybových hlášení je patrné, že software dovede uživatele podrobně informovat a pomůže mu odstranit případné konflikty. Užitečná je informace o poloze, kde se daný problém nalézá. Tato funkce je nezbytná a lze ji najít ve většině aplikací umožňujících psaní kódu.



Obr. 14: Grafická podoba okna Chybová hlášení

3.8 Formátování

Jde panel tlačítek s funkcemi pro úpravu a formátování zadávaného programového kódu v textovém poli ***Editor***.

3.8.1 Tlačítko Bez syntaxe

Je znázorněno pěti vodorovnými čarami v černé barvě. Při zvolení tohoto režimu je textové pole necháno bez jakýchkoliv zvýraznění. Je to výhodné hlavně při rozsáhlejších projektech, kdy zvýrazňování kódu je již náročnější na výkon procesoru počítače. Režim zadávání lze v průběhu měnit, a tak je možné využít jen občasnou kontrolu kódu.

Obr. 15:Grafická podoba tlačítka Bez syntaxe

3.8.2 Tlačítko Zvýraznění syntaxe

Je znázorněno pěti vodorovnými čarami v různých barvách. Při této volbě jsou viditelná tři tlačítka ***Instrukce***, ***Návěští*** a ***Direktivy***. V tomto režimu je aktivní zvýrazňování funkcí (modře), komentářů (zeleně), direktiv překladače (šedě), návěští (tmavě modře) a řetězců uzavřených v uvozovkách (hnědě). Ostatní text zůstává nezměněn.

Protože je testování prováděno při každé změně textového pole (každým úhozem do klávesnice při zadávání), je u rozsáhlejších projektů odezva pomalejší. K výraznějšímu zpomalení však dochází pouze při jednorázové změně 15 znaků a více. Je tak ošetřeno vkládání kódu, kdy je zapotřebí přebarvit celé okno.

Odezva je o mnoho rychlejší při běžném psaní, kdy dochází ke kontrole pouze v rozpětí aktivního řádku. Tato funkce je již integrována do poslední verze aplikace a bude podrobněji popsána dále v textu.

Obr. 16:Grafická podoba tlačítka Zvýraznění syntaxe

3.8.3 Tlačítko Opravy

Tlačítko ***Opravy*** je znázorněno pěti vodorovnými čarami v červené barvě. Při této volbě jsou viditelná tři tlačítka ***Instrukce***, ***Návěští*** a ***Direktivy***. Při zvolení tohoto režimu jsou v zadávaném kódu zvýrazněny chyby. Jsou detekovány chyby v zadávání funkcí, návěští a direktiv překladače.

U instrukcí je kontrolován počet výskytu na jednom řádku. V případě, že je na jeden řádek vloženo více instrukcí, je tento stav vyhodnocen jako chybový. V takovém případě jsou instrukce vyjma první, zabarveny červeně a podtrženy.

U návěští jsou kontrolovány jednotlivé znaky. Ty musí být A-Z, a-z, 0-9. Navíc se nesmí jako první znak objevit číslice. Další podmínkou při zadávání návěští je, aby v něm nebyly použity výrazy vyhrazené pro funkce a direktivy překladače. V případě, že se objeví jakákoliv nepatřičná syntaxe, je zabarvena červeně.

Jelikož je tato metoda opět o něco náročnější na výpočetní výkon počítače než předchozí metoda, můžeme tuto volbu použít na namátkovou kontrolu před kompilací a po zkontrolování se vrátit zpět k jednomu z předchozích režimů.

Obr. 17:Grafická podoba tlačítka Opravy

3.8.4 Tlačítka Instrukce, Návěští a Direktivy

Tlačítka ***Instrukce***, ***Návěští*** a ***Direktivy*** slouží k formátování psaného kódu. Tlačítka jsou viditelná pouze při volbě režimu psaní pomocí tlačítek ***Zvýraznění syntaxe***, nebo ***Oprava***. Jedná se o třístavová tlačítka.

Stav, ve kterém se nacházejí, je indikován textem na jejich pravé straně a zobrazením nápisu v požadované velikosti. Pomocí těchto tlačítek se dá sjednotit vzhled kódu.

Tlačítko ***Instrukce*** formátuje všechny instrukce v simulovaném kódu a to i aktuálně při zadávání. Volí se mezi stavy, kdy jsou všechny znaky velké nebo malé. Poslední možnost umožňuje ponechat jak velké tak malé znaky a neupravuje nijak jejich velikost. Tato možnost je nastavena jako výchozí. Obdobně pracují i následující tlačítka.

Zvolené formátování velikosti zůstává zachováno i v režimu ***Bez úprav***, ale nově zadávaný kód se již automaticky neformátuje.

Obr. 18: Grafická podoba tlačítka Instrukce, Návěští a Direktivy

3.8.5 Zarovnání

Aby bylo možné se v psaném kódu dobře orientovat, je dobré si zvolit jisté formátovaní. Vzhledem k tomu, že webová aplikace neumožňuje používat klávesu tabulátoru, je výhodné využít funkci automatického zarovnání kódu. Ta se provede jen jednou po stisknutí příslušného tlačítka. Poté se vrátí do režimu, z něhož byla zavolána (***Bez syntaxe***, ***Zvýraznění syntaxe***, ***Opravy***).

Obr. 19: Grafická podoba tlačítka

V tomto režimu se postupně vyhodnocují jednotlivé části kódu. Pří rozpoznání návěští a komentáře, jsou tyto prvky zarovnány k levé straně. Dále je otestováno, je-li návěští na řádku samotné. V případě že tomu tak není, je automaticky vloženo na samostatný řádek.

Zápis instrukce je při zarovnání odsazen od levého okraje šesti mezerami a mezi její parametry jsou vloženy mezery tak, aby parametry všech funkcí byli od levého okraje stejně vzdálené.

V případě detekování některé chyby v zápisu (viz. 3.7), je tato chyba opravena. Chybné návěští je převedeno na komentář s upozorněním. V případě, že instrukce nemá ani jeden ze svých parametrů, je za ní doplněna čárka, která na tuto chybu upozorní. Režim zarovnání je novinkou při psaní programu v Assembleru. Tuto možnost není možné nalézt ani u komerčních řešení.

3.9 Nástroje

Jde o panel tlačítek s nejdůležitějšími funkcemi aplikace.

3.9.1 Příklad

Tato funkce je určena pro testování chování napsaného kódu, kdy je nutné opakovaně vkládat testovací kód, na kterém jsou prováděny úpravy. Současná podoba vkládaného kódu ukazuje možnosti simulačního programu. Je zde uveden seznam instrukcí, které jsou již implementovány v jádru simulačního programu. Po vytvoření simulačních dat je možné vložený kód ihned krokovat.

Obr. 20:Grafická podoba tlačítka Příklad

3.9.2 Vložení hlavičky

Tato funkce je využita hlavně z důvodu použití jednoho typu mikroprocesoru (ATmega16). Usnadní uživateli vkládání kódu.

Obr. 21: Grafická podoba tlačítka Vložení hlavičky

Hlavička je vkládána vždy před napsaný programový kód a tak se uživatel nemusí obávat, že by při nechtěném stisku tlačítka přišel o svou práci.

Vložená hlavička vypadá následovně:

```
;Simulaci mikroprocesoru Atmega16 vytvořil Josef Šrajla.
;   ***  Vložení definičního souboru mikroprocesoru.  ***
     .NOLIST
     .INCLUDE "m16def.inc"
     .LIST
;-----------------------------------------------------
```

3.9.3 Vytvoření mfile

Toto tlačítko je jedno z nejdůležitějších vůbec. Vytváří simulační data pro vlastní simulaci. Po stisku tohoto tlačítka je vstupní kód zkopírován a dílčí úpravy se provádějí již jen na zkopírované verzi. Jednotlivé kroky tak jak jdou za sebou: Odstranění komentáře, načtení definičního souboru, získání simulačních dat.

Obr. 22: Grafická podoba tlačítka Vytvoř mfile

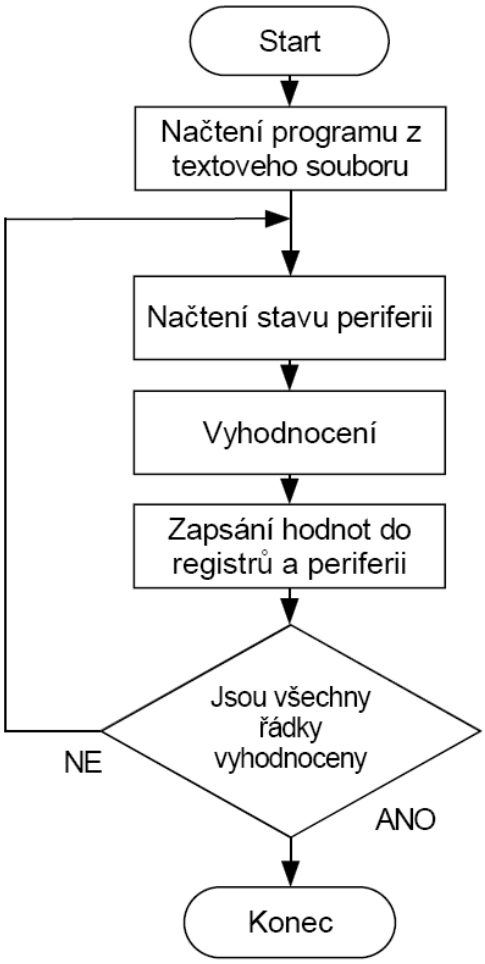
3.9.4 Krokování

Krokování umožní postupné procházení jednotlivých řádků a sledování co daná instrukce s jejími parametry pozměnila a jak ovlivnila chod celého programu. Je to stěžejní část nejen tohoto projektu, ale výuky programováni jako takového. Pří procházení kódu dochází i k jeho automatickému posouvání tak, aby uživatel viděl právě aktivní řádek.

Obr. 23: Grafická podoba tlačítka Krokování

4 Softwarové řešení

Všechny moduly uvedené v části ***2.2 Teorie*** mají své protějšky v simulačním programu. Účelem je, aby se chovaly co možná nejvěrněji a uživatel mohl měnit jejich stavy při běhu programu. Celá aplikace je umístěna na webu. Simulace je naprogramována ve flashovém prostředí, ve kterém bylo možné využít Actionscript (jedná se v podstatě o modifikaci jazyka ***C***) a předdefinované objekty jako jsou tlačítka a textová pole.



Obr. 24: Běh hlavního cyklu

Běh hlavní smyčky je znázorněn na obr. 24. Jednotlivé textové řetězce v poli ***Editor*** jsou postupně načtené a vyhodnocené. Na základě naprogramovaného algoritmu se nastaví stavové registry. Podle propojených periferií bude nastavení registrů doprovázet například rozsvícení LED diod či roztočení krokového motoru.

Je zde možné využít i integrovaného rozhraní v hlavním panelu, které umožní sledovat stavy jednotlivých I/O registrů a pomocných registrů R0 až R31. Běh programu je možné krokovat. Není možné, že by časové úseky při simulaci byly shodné s reálnou aplikací. Z tohoto důvodu bude běh programu přepočítáván na „skutečný“ čas. Vysvětlení na konkrétním příkladu je následující.

Program běží ve smyčce, ve které se po deseti cyklech rozsvítí LED a po dalších deseti cyklech zhasne. Na počítači se dá poměrně snadno zaznamenat pouhým okem zhasnutí a rozsvícení diody. Po nahrání do reálného mikroprocesoru ale žádné blikaní není pozorovatelné. Lidské oko vnímá pouze rychlé zhasínání a rozsvěcení jako tlumený svit. Simulace díky přepočtu na „skutečný“ čas dokáže uživatele na tuto věc upozornit.

4.1 Rozpoznávání kódu

Aby bylo možné rozpoznat specifické části napsaného kódu, je nutné použít na dekódování určitý algoritmus. V této kapitole bude funkce vysvětlena jen blokově. Kompletní řešení v podobě vývojového diagramu je přiloženo na konci této práce.

4.2 Dekódování instrukce

Dekódování probíhá po jednotlivých znacích. Ty jsou postupně načítány do zásobníku a ten je v zápětí vyhodnocován. Jednotlivé instrukce jsou od sebe odděleny mezerami a mají specifický tvar. Při určování, o jaký se jedná kód, je výhodné rozdělit segmenty kódu do pěti skupin. Rozdělení je určováno podle prvního znaku řetězce

- jestliže je na první pozici načten znak středník, potom je zbylý řádek zabarven zelenou barvou a další dekódování znaků na tomto řádku neprobíhá (jedná se o komentář),
- jestliže je na první pozici načten znak uvozovky, potom se do zásobníku ukládají všechny následující znaky, dokud není nalezen další znak uvozovek. Podaří-li se nalézt tato dvojce na jednom řádku, je rozpoznání úspěšné a prostor vymezený středníky je zabarven hnědou barvou. V opačném případě je ponechán bez zabarvení.
- jestliže je na první pozici načten znak tečka, potom se jedná o direktivu Assembleru. Ta je ukončena mezerou nebo koncem řádku. Zabarvení je realizováno šedou barvou. V případě, že se na řádku vyskytují dvě takovéto direktivy, nebo více funkcí, je příslušnou barvou zabarven pouze první řetězec, ostatní jsou zabarveny červeně.

- jestliže je na první pozici načten znak různý od středníku, uvozovek, tečky a řetězec není ukončen dvojtečkou, potom se jedná o funkci.
- jestliže je na první pozici načten znak různý od středníku, uvozovek, tečky a řetězec je ukončen dvojtečkou, potom se jedná o návěští.

Jelikož bylo účelem této práce se co nejvíce přiblížit reálnému překladači, jsou zde ošetřeny i stavy, které jsou pří zadávání kódu nepřípustné. Musí se předejít stavům, kdy

- návěští nesmí začínat číslicí,
- text návěští nesmí být shodný s názvem instrukce nebo direktivy assembleru,
- na řádku se může vyskytovat nanejvýš jedna instrukce nebo direktiva.

Všechny zmíněné stavy jsou signalizovány zabarvením přímo v textu červenou barvou. Všechny zmíněné kontrolní mechanizmy jsou aktivní při zadávání programového kódu v režimu ***Opravy*** (viz kap. 3.8.3).

Ukázka dekódování řetězce:

Obr. 25: Dekódování direktivy překladače.

Na obr. 25. Je příklad dekódování direktivy překladače ze zadaného kódu. Podobným způsobem jsou detekovány i ostatní části kódu.

Jestliže instrukce využívají dalších parametrů, potom jsou načteny i tyto parametry. Protože instrukce mohou mít různý počet parametrů, je nutné mít vytvořený seznam instrukcí, ve kterém jsou tyto požadavky uložené. Při načtení instrukce je pak možné se do takového seznamu nahlédnout a načíst požadovaný počet parametrů.

4.3 Stručný popis jednotlivých funkcí

V této kapitole jsou popsané jednotlivé funkce, které se využívají při simulaci. Struktura je následující: Název funkce; popis funkce; řádek programového kódu, kde je funkce definovaná. Závorka u funkce obsahuje vstupní hodnoty a jejich typ (Boolean, Number, String). Za závorkou je výstupní hodnota, kterou vrací funkce zpět. Jestliže funkce postrádá parametry, potom funkce pracuje s globálními proměnnými.

TextovePole_txt.onScroller

Tato funkce zajišťuje sjednocení scrollování polí Radek_txt, Znaky_txt a TextovePole_txt. Při rychlém pohybu se může stát, že ostatní pole za hlavním polem mírně zaostávají.

Zápis funkce:

```
//////////////////////////////////////////////////////////
//Sjednocení posouvání scroll podle hlavního okna
TextovePole_txt.onScroller = function() {...}
```

VlozitHlavicku

Funnkce vloží hlavičku do editoru. Vice v kap. 3.9.2 .

Zápis funkce:

```
//////////////////////////////////////////////////////////
//Vloží předefinovanou hlavičku
VlozitHlavicku = function () {
```

VlozitPriklad

Vloží testovací příklad z externího zdroje s příponou .txt. Princip této funkce bude použit v další verzi, která bude umožňovat ukládání a zpětné čtení vlastních prací. Zatím není rozhodnuto o přesné metodě, jakým způsobem bude ukládání na server probíhat a tak není důvod tuto funkci zatím aktivovat.

Zápis funkce:

```
//////////////////////////////////////////////////////////
//Vloží testovací příklad
VlozitPriklad = function () {...}
```

DoplnMezery

Vrátí řetězec složený z mezer. Využívá se především pro zarovnání kódu do sloupců.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Odstraní mezeru při načítání řetězce
DoplnMezery = function (Mezer:Number, Roztec:Number):String {
```

Mezera

Pomocí této funkce je možné přeskočit tu část kódu, která je vyplněna mezerami, nebo tabulátory. Postup vyhodnocení je následující.

V uzavřené smyčce probíhá postupně načítání znaku. Při každém průchodu je načtený znak porovnán se znakem tabulátoru, mezery a prázdným znakem. K opuštění smyčky dojde v případě rovnosti načteného znaku s porovnávaným.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Odstraní mezeru při načítání řetězce
Mezera = function () {...}
```

Dalsiznak

Načte další znak ze zpracovávaného vstupního textu. Jedná se o nejčastěji volanou funkci při kompilaci napsaného kódu. U často volaných funkcí je důležité, aby jejich vykonávání nebylo časově náročné. Funkce nemá žádné vstupní parametry a pracuje pouze s globálními proměnnými. Zápis celé funkce zabírá v paměti programu pouze devět řádků.

Zápis funkce:

```
////////////////////////////////////////////////////////
//načte další znak a zkontroluje jestli se nejedna o konec
řádku
Dalsiznak = function () {...}
```

DoplnNulu

Při zobrazování hodnoty v hexadecimálním formátu ošetřuje stav, kdy je zaplněna jen jedna pozice ze dvou možných. Při zobrazení jednomístného tvaru v textovém poli dochází k posunu celého řádku vlevo. Je-li doplněna na prázdnou pozici „0“, nezmění se hodnota čísla a zároveň zůstává zachována i délka řetězce.

Zápis funkce DoplnNulu:

```
////////////////////////////////////////////////////////
//do HEXa čísla doplní na první pozici "0", když je to třeba
DoplnNulu = function (Hex:String) :String {...}
```

PrevodnaBIN

Všechna čísla vyjádřena v různých soustavách jsou převedena na společný hexadecimální tvar. V této podobě je problematické s číslem pracovat. Programovací jazyk Actionscript chápe takové vyjádření jako řetězec. Stav je ošetřen obdobně jako u funkce „doplnNulu“ s tím rozdílem, že je zde kontrolováno, jestli výsledný kód má osm platných pozic.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Převod HEXa na BIN
PrevodnaBIN = function (Hex:String) :String {...}
```

NastavBitReg

Podle vstupních parametrů (jméno registru a číslo v rozmezí od 0 do 7) udává, jaký bit se má nastavit. Jedná se o velmi užitečnou funkci. V současné době využívá integrovaných matematických funkcí. Je zde ale i varianta, která nevyžaduje převod z textového řetězce na číslo. U metody bez převodu na číslo by bylo možné využít funkce pracující s řetězci.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Nastaví zvolený BIT ve zvoleném REG
NastavBitReg = function (Bit:Number, Reg:Number) {...}
```

NulujBitReg

Obdobná jako funkce „NastavBitReg“ s tím rozdílem, že dochází k nulování příslušného bitu. Obě tyto funkce jsou volány jak při vykonávání instrukcí, tak i při stisknutí tlačítek na vstupně/výstupním panelu.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Nuluje zvolený BIT ve zvoleném REG
NulujBitReg = function (Bit:Number, Reg:Number) {...}
```

TestBitReg

Tato funkce testuje podle zadaných parametrů, jestli je bit nastaven či nikoliv. Jestliže je testovaný bit nastaven, potom vrací logickou hodnotu „true“, v opačném případě hodnotu „false“.

Zápis funkce:

```
//////////////////////////////////////////////////////
//Testuje zvolený BIT ve zvoleném REG - vrací true nebo false
TestBitReg = function (Bit:Number, Reg: Number) :Boolean {...}
```

TestSreg

Stavový registr mikroprocesoru AVR je jedním z nejdůležitějších vůbec. Jsou v něm nastaveny hodnoty informující o výsledkách výpočtu, povolení globálního přerušení a kopírovací bit. Význam jednotlivých bitů: [1]

- **C – příznak přetečení.** Indikuje přetečení po aritmetické nebo logické operaci.
- **Z – Příznak nulového výsledku.** Indikuje nulový výsledek po aritmetické nebo logické operaci.
- **N – příznak negativního výsledku.** Indikuje záporný výsledek po aritmetické nebo logické operaci.
- V – příznak přeplnění čísla v druhém doplňku. Podporuje aritmetiku druhého doplňku.
- **S – znaménkový bit.** Indikuje znaménko čísla v druhém doplňku (S = N xor V).
- **H – pomocný příznak přetečení.** Indikuje přenos mezi dolní a horní polovinou výsledku (důležité zejména při práci s BCD čísly).
- **T – kopírovací bit.** Instrukce BLD a BST používají bit T jako zdrojový nebo cílový bit. Bit z registru registrového pole může být kopírován do bitu T instrukcí BST a zase uložen do bitu registru registrového pole instrukcí BST.
- **I – globální povolení přerušení.** Pro příjem přerušení musí být tento bit nastaven (I = 1). Individuální přerušení lze řídit dalšími registry. Je-li I = 0, je příjem přerušení zakázán. Po vstupu do obsluhy přerušení přejde I = 0, a tím je příjem dalších přerušení zablokován. Vyvoláním instrukce RETI z rutiny obsluhy přerušení, se zajistí zpětné nastavení I = 1.

Zápis funkce TestSreg:

```
////////////////////////////////////////////////////////
//Nastavuje příznaky ve stavovém registru dle zadané předlohy
a vrací HEX z poslední vstupní hodnoty
TestSreg = function (an:Number, bn:Number, cn:Number) :String
{...}
```

OrHex

Jedná se o součet dvou čísel v hexadecimálním formátu. Jak jsem již zmínil u funkce „PrevodnaBin“, v Actionscriptu není možné přímo pracovat s hexadecimální podobou čísla. Hodnota je zde totiž chápána jako řetězec. Před matematickou operací se tento řetězec musí nejprve převést na číselnou hodnotu a teprve poté s touto hodnotou pracovat.

Zápis funkce OrHex:

```
////////////////////////////////////////////////////////
//Součet v HEXa formátu, vrací HEXa
OrHex = function (a:String, b:String) :String {...}
```

AndHex

Obdobná jako funkce „OrHex“ jen s tím rozdílem, že se jedná o logický součin namísto součtu. Tyto a následující funkce by bylo možno přepsat do podoby, kde by nebylo nutné převádět vstupní řetězec na číslo a zpět. Tato úprava by vyžadovala vytvoření pomocných funkcí.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Součin v HEXa formátu, vrací HEXa
AndHex = function (a:String, b:String) :String {...}
```

XorHex

Exkluzívní logický součet vstupů.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Exkluzivní součin v HEXa formátu, vrací HEXa
XorHex = function (a:String, b:String) :String {...}
```

MinusHex

Mínus není klasická logická funkce, ale ukázalo se, že její vytvoření usnadní některé operace.

Zápis funkce MinusHex:

```
////////////////////////////////////////////////////////
//Rozdíl v HEXa formátu, vrací HEXa
MinusHex = function (a:String, b:String) :String {...}
```

NotHex

Negování čísla ve formátu HEX.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Negace v HEXa formátu, vrací HEXa
NotHex = function (a:String) :String {...}
```

	10111010		11100001		11101101		
and	00001111	*or*	00100100	*xor*	00000001	*not*	00010111
	00001010		11100101		11101110		11101000

Obr. 26: Ukázka výpočtů předešlých funkcí

HledaniVmfile

Tato funkce najde ve vytvořeném souboru mfile zadaný řetězec a vrátí jeho hodnotu v hexadecimálním tvaru.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Podle zadaného vstupního řetězce najde v souboru _Mfile
příslušnou hodnotu v hex a vrátí ji zpět
HledaniVmfile = function (Retezec:String):String {...}
```

PrevodNaHEX

Jedna z nejsofistikovanějších funkcí v programu. Na základě vstupního parametru, kterým může být libovolný formát čísla a dokonce i text, je vrácena hodnota v hexadecimálním tvaru. Detekované tvary jsou: R0 až R31, 0X…, $..., 0B…,LOW, HIGH, 0-9 a prostý text. V případě, že se jedná o pouhý text, je hledána hodnota k tomuto řetězci v proměnné _Mfile. Zápis funkce:

```
////////////////////////////////////////////////////////
//převede libovolný vstupní formát na hexadecimální tvar
PrevodNaHEX = function (VstupHEX:String):String {...}
```

NacteniParametru

Každá instrukce má dáno, s jakými parametry pracuje. Při nalezení instrukce je vyhodnoceno, jaké má daná instrukce požadavky. Na základě těchto požadavků tato funkce načte parametry a uloží je do pole. Výstupem této funkce je _PoleRadek, _PoleInstrukce, _PoleParametr1 a _PoleParametr2. Tyto uložené hodnoty jsou klíčové pro samotnou simulaci. Vzhledem k tomu, že dopředu víme, kolik má mít daná instrukce parametrů, lze kontrolovat správné zapsání uživatelem. V případě chyby v zápisu je uživatel aplikace informován a má možnost chybu odstranit.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Načte parametry za funkcí a vloží do pole řádek, název
instrukce, parametr1 a 2
NacteniParametru = function (TvarParamertu:String) {...}
```

OdstraneniKomentare

Kód v Assembleru může být opatřen komentářem. Je to výhodné pro zlepšení orientace kódu. Zvýší se také jeho srozumitelnost pro ostatní uživatele. Tento kód je ale pro samotnou simulaci nadbytečný a z tohoto důvodu musí být odstraněn. Odstranění probíhá, stejně jako mnoho jiných operací s kódem, na jeho kopii a uživatel si při kompilaci ničeho nevšimne.

Zápis funkce:

```
////////////////////////////////////////////////////////
//odstraní komentář z _VstupniTextZkopirovany
OdstraneniKomentare = function () {...}
```

NacteniInclude

Zdánlivě triviální záležitost, jako je vložení dílčí části kódu do stávajícího kódu, nevytváří dojem, že by na ní bylo zapotřebí zvláštní funkce. Komplikace s tímto úkonem spojené jsou dány hlavně tím, že ke spojení dvou kódů nedochází. Nejčastěji nastává varianta kdy dochází začlenění prvního kódu do druhého. Druhý text je prvním rozdělen na dva díly. Protože se toto děje na pozadí a uživatele by to nemělo nijak ovlivňovat, komplikuje se však počítání nových řádků. Protože vkládaný soubor neobsahuje žádné instrukce, není nutné jeho rozvinutí při krokování. Rozvinutí je možné drobnou úpravou kódu Actionscriptu. Ukázalo se ale, že se výrazně zhorší komfort uživatele při následném procházení kódu. Současná verze umožňuje vložení pouze jednoho souboru s příponou „.inc“. Jelikož při vkládání externích dat s příponou „.txt“ dochází k poměrně významnému časovému zpoždění, je do flashové aplikace vložen kompletní obsah include.

Toto řešení není příliš komfortní, ale ze všech možných obstálo nejlépe. V případě rozšíření na větší počet mikroprocesorů, s čímž se do budoucna počítá, je nutné tuto funkci upravit. Uživatele je třeba o těchto omezeních informovat. K tomu dochází ve stavovém okně, kde je detailně popsán problém v případě, že se při zadávání include vyskytne.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Vloží soubor INCLUDE podle zadaného textu v hlavičce
NacteniInclude = function () {...}
```

Zpracovani

Jedná se o nejrozsáhlejší funkci v celé simulaci. Na základě vstupního parametru provádí buď zabarvení textu, jeho formátování a generování souboru označeného jako mfile. Vývojový diagram této funkce viz příloha na CD této práce. V průběhu vývoje aplikace byly vytvořeny tyto funkce zvlášť, což značně navyšovalo velikost kódu programu. Protože se vzájemně velmi podobaly, bylo výhodné je sloučit v jeden celek. To částečně zhoršuje přehlednost kódu. Velmi zjednodušený model této funkce je vidět na obr. 24.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Klíčová funkce programu. Podle zadaného parametru buď
zvýrazní kód, nebo připraví mfile
Zpracovani = function (Parametr:String) {...}
```

ObsahPeriferie

Všechny signalizační prvky jsou umístěny pod touto funkcí. Po zavolání dojde k aktualizaci všech stavů najednou. Jedná se především o pole ***Periferie mikroprocesoru***, pole ***Pracovní registry*** a ***Sledované registry***. Tato funkce je zároveň připravená aktualizovat další periferie, které postupem času budou přibývat.

Zápis funkce:

```
////////////////////////////////////////////////////////
//funkce vypsáni obsahu do okna periferie a ostatních
stavových polí
ObsahPeriferie = function () {...}
```

TextovePole_txt.onChanged

Tato funkce je volána při změně obsahu v ***Editoru***. Důležitou součástí této funkce je vyhodnocení, na jakém místě ke změně došlo.

Poté se označí celý řádek a pošle se požadavek na jeho opětovné zabarvení. Tento způsob výrazně zrychluje odezvu na zadávaný kód. Naopak velké komplikace to přináší při zadávání komentáře pomocí znaku „/*”.

Tento způsob se používá u víceřádkových elementů a není možné jej správně vyhodnotit. Proto není tento způsob zadávání podporován. Jedinou alternativou je vytvoření databáze, kde by byla uložena čísla řádků, na kterých se komentář nalézá. Poté by už stačil jednoduchý algoritmus, který by vyhodnotil, jestli má být zadaný řádek vyhodnocen nebo pouze zeleně zbarven.

Zápis funkce:

```
////////////////////////////////////////////////////////
//funkce při změně obsahu okna
TextovePole_txt.onChanged = function() {...}
```

Simulace

Slouží k přepínání z režimu simulace do režimu editace. Zatím není využita. Uvažuje se o změně tlačítek při přepínání a uzamknutí hlavního okna pro editaci v režimu simulace.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Nepoužita, připravena na přepnutí ze stavu editování do
stavu simulování. Zatím není potřeba.
Simulace = function () {...}
```

CisloREG

Vstupem do této funkce je číslo registru. Protože vstupně/výstupní registry jsou v paměti programu až za registry R0-R31, ale jsou číslovány od nuly, je jim zde přičtena konstanta 32. V případě, že se jedná o registry R0-R31, je odtržen znak R a zbylý řetězec převeden na číselnou hodnotu v dekadickém formátu.

Zápis funkce:

```
////////////////////////////////////////////////////////
// určí registr v paměti a vrátí číselné vyjádření jeho pozice
CisloREG = function (REG:String):Number {...}
```

Instrukce

V této funkci jsou uloženy všechny instrukce Assembleru. Funkce je volána s parametry INSTRUKCE, PARAMETR1 a PARAMETR2.

Všechny vstupní hodnoty jsou ve formátu „String“. Na základě těchto vstupních parametrů jsou nastaveny parametry mikroprocesoru tak, jak by se choval v reálné aplikaci.

Zatím je naprogramováno zhruba čtyřicet instrukcí. Jelikož jsou připraveny podprogramy pro práce s čísli v hexadecimálním tvaru, jako je AndHex, OrHex, XorHex, NegHex a další, není již naprogramovaní chování instrukcí obtížné. Nejvíce času je vynaloženo na testování „správného“ chování instrukce v programu.

Zápis funkce Instrukce:

```
////////////////////////////////////////////////////////
//podle načtené instrukce provede jednotlivé operaci při
simulaci
Instrukce = function (instrukce:String, parametr1:String,
parametr2:String) {...}
```

OznacRadek

Vstupní parametr je číslo řádku, které se má označit výstupem je znak „►“ v poli znaku u příslušného řádku. Časem přibudou další parametry, které budou udávat řádky s programovou zarážkou, pro lepší odladění programu v Assembleru.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Označí řádek na kterém jsou prováděny výpočty při simulaci
OznacRadek = function (Radek:Number) {...}
```

Krok

Krokování aplikace. V prvním kroku je zavolána funkce označení řádku podle aktuální pozice, dále je vyslán požadavek na zpracování instrukce se zadanými parametry a poté je inkrementováno číslo zpracovávaného řádku. Je zde navíc ošetřen stav, kdy je krokování již na posledním řádku. V takovém případě se začíná opět od začátku. Tato vlastnost je zde jen dočasně. Bude odstraněna po zprovoznění instrukcí skoku.

Zápis funkce:

```
////////////////////////////////////////////////////////
//Krokování (simulace). Načítá jednotlivé řádky simulačních
dat a to pořád dokola
Krok = function () {...}
```

Prepinani

Zajišťuje změnu stavu bitu v daném registru. Vstupními parametry jsou číslo bitu v rozmezí od 0 do 7 a název registru. Funkce nevrací zpět žádnou hodnotu. Pouze mění globální proměnné.

Zápis funkce:

```
////////////////////////////////////////////////////////
//funkce na přepínání ze stavu 0 do 1 a obráceně
Prepinani = function (Bit:Number, Registr:String) {...}
```

4.1 Proměnné

Proměnné reprezentují datový prostor reálného procesoru. Jsou nedílnou součástí aplikace. Pro pochopení funkce aplikace je nezbytné pochopit, co jednotlivé proměnné znamenají. Následuje výpis důležitých proměnných.

_Mfile

_Mfile je globální proměnná typu String. Do této proměnné je ukládán obsah souboru mfile, tedy veškeré konstanty a adresy návěští, které se v kódu Assembleru vyskytují. Mezi nimi i šestnáctibitové registry X, Y a Z. Ty jsou zpravidla tvořeny některým registrovým párem z R0 až R31. Většina zde definovaných proměnných pochází z vloženého souboru „m16def.inc“.

Jestliže překladač narazí na řetězec v místě, kde je očekávána konstanta, potom se pokusí tento řetězec nalézt souboru mfile (v proměnné _Mfile). Jestliže se v souboru nalézá přiřazení numerické hodnoty tomuto řetězci, potom je touto hodnotou (pro potřeby překladu programu) původní řetězec nahrazen. V opačném případě program tuto chybu nahlásí a uživatel musí chybu opravit.

Strukturu a obsah souboru mfile byl volen tak, aby byl kompatibilní se souborem generovaným programem AVR Studio. To umožňuje kontrolní porovnání souborů z obou programů, ale i snazší přechod mezi oběma programy. Porovnání generovaných dat je možné nalézt na přiloženém CD.

Deklarace proměnné:

```
var _Mfile:String = "";
```

_PoleInstrukce

_PoleInstrukce je globální proměnná typu Array, tedy pole. V tomto poli jsou uloženy veškeré instrukce v tom pořadí, ve kterém se v programovém kódu vyskytují.

Instrukce mohou mít až dva parametry a spolu s instrukci je tedy potřeba uchovávat i hodnoty jejích parametrů. Výhodné by bylo využít vícerozměrné pole. To ale programovací jazyk Actionsctript neumožňuje. Z tohoto důvodu byla použita dvě pomocná pole (_PoleParametr1 a _PoleParametr2), která v součinnosti vytvářejí požadovanou vícerozměrnou strukturu.

Deklarace proměnné:

```
var _PoleInstrukce:Array = new Array();
```

_PoleParametr1

Globální proměnná typu Array. Pole prvních parametrů přiřazených k instrukci.

Deklarace proměnné:

```
var _PoleParametr1:Array = new Array();
```

_PoleParametr2

Globální proměnná typu Array. Pole druhých parametrů přiřazených k instrukci.

Deklarace proměnné:

```
var _PoleParametr2:Array = new Array();
```

_PoleRadek

Do polí _PoleInstrukce, _PoleParametr1 a _PoleParametr2 se ukládají pouze funkční řádky programového kódu (prázdné řádky a řádky obsahující komentář byly vynechány). Aby nedošlo ke ztrátě informace o původním umístění v programovém kódu, uchovávají se v poli _PoleRadek čísla řádku, na kterém se instrukce v programovém kódu nachází. Při krokování programu je pak možné označovat jednotlivé řádky, na kterých se aktuální instrukce nachází.

Deklarace proměnné:

```
var _PoleRadek:Array = new Array();
```

_DatovaPamet

_DatovaPamet je proměnná typu Array, tedy pole. Pole je potřeba na začátku inicializovat nulovými hodnotami.

Do pole _DatovaPamet se ukládají řetězce zobrazující číslo v hexadecimálním tvaru. Pole _DatovaPamet reprezentuje reálnou paměť mikroprocesoru. Její struktura je probrána v kap. 2.2.1.

Deklarace proměnné _DatovaPamet:

```
var _DatovaPamet:Array = new Array(…);
```

_DatovaPametBIN

Analogická proměnná k proměnné _DatovaPamet, jen hodnoty v tomto poli jsou uložené v binárním formátu. Tento formát je vhodnější pro zobrazení obsahu registru.

Deklarace proměnné:

```
var _DatovaPametBIN:Array = new Array(…);
```

5 Závěr a výhled do budoucna

Stručné shrnutí výstupu mé práce a možných vizí, kam by se práce v budoucnu mohla ubírat.

5.1 Současný stav

Vytvořil jsem webové stránky umístěné na adrese www.avr.poruce.com. Pod záložkou ***Simulace*** nalezneme webovou aplikaci, ve které je možné simulovat programování mikroprocesoru. Záložky Úvod, Teorie, Programy, Hardware a Odkazy jsou také zprovozněny a naplněny příslušnými daty.

Z možných 113. instrukcí jsou naprogramovány následující: LDI, ADD, ADC, SUB, SUBI, SBC, SBCI, COM, NEG, INC, DEC, CLR, SER, CP, CPI, MOV, OUT, SBI, CBI, SEC, CLC, SEN, CLN, SEZ, CLZ, SEI, CLI, SES, CLS, SEV, CLV, SET, CLT, SEH, CLH a NOP.

5.2 Možné směry dalšího vývoje

Přehled jednotlivých směrů, kde by bylo možné v práci pokračovat.

- zbývá naprogramovat některé instrukce,
- programování lze rozšířit i na další mikroprocesory z řady AVR (Mega8, Mega32, Mega64, Mega128, atd.), které se liší jen pouzdrem, počtem periferií a vnitřní pamětí,
- počítá se s možností k jednotlivým portům připojovat periférie, jako jsou signalizační LED-diody, klávesnice, krokový motor, stejnosměrný motor, LCD, teplotní čidlo, potenciometr, sklonoměr a další. Tak by uživatel mohl sledovat už reálné chování výsledné aplikace mikroprocesoru. Běh programu by pak měl být řízen na základě podnětů z těchto periférií,
- webová aplikace může být vybavena rozhraním pro ukládání, aby bylo možné archivovat pro inspiraci již otestované výtvory.

Webová aplikace je vytvořena tak, že se na jejím dalším vývoji může podílet nezávisle více lidí najednou.

Části, na kterých se takto může pracovat jsou:

- jednotlivé záložky psané v PHP,
- jamotný simulátor, což je Flashová aplikace.

Popřípadě je možné ještě dělení na jádro a periférie mikroprocesoru.

5.3 Použité znalosti a dovednosti

K vytvoření webové aplikace, jsem využil programovacích jazyků ASSEMBLER, ACTIONSCRIPT+FLASH, HTML, PHP, JAVA, CSS. Nutnou podmínkou bylo zvládnutí návrhu schémat vlastních elektronických zařízení a k nim určených desek plošných spojů. Desky plošných spojů jsou v takto malých sériích osazovány manuálně. Z tohoto důvodu byla vyžadována znalost procesu pájení a manuální zručnost.

6 Použitá literatura

[1] MATOUŠEK, D. *Práce s mikrokontroléry ATmega16.* BEN: technická literatura, Praha 2006. 317 stran. ISBN 80-7300-174-8.

[2] PINKER, J. *Mikroprocesory a mikropočítače.* BEN: technická literatura, Praha 2004. 156 stran. ISBN 80-7300-110-1.

[3] DEHAAN, J. *Macromedia Flash ®MX 2004 – Oficiální výukový kurz.* SoftPress s.r.o, Praha 2004. 531 stran. ISBN 80-86497-64-X.

[4] KADLEC, V. *Učíme se programovat v jazyce C.* CP Books, a.s, Brno 2005. 227 stran. ISBN 80-7226-715-9.

[5] FRENKLIN, D. / MAKAR, J. *Macromedia FLASH® MX 2004 ActionScript – oficiální výukový kurz.* SoftPres s.r.o, Praha 2005. 758 stran. ISBN 80-86497-75-5.

[6] RADKOVSKÝ, K. Stránky zabývající se mikroprocesorovou elektronikou. [cit. 2009-2-20]. Dostupné z http://www.dioda.cz/index.php .

[7] HW Server. Stránky zabývající se mikroprocesorovou elektronikou. [cit. 2008-1-24]. Dostupné z http://avr.hw.cz/.

[8] CHURÝ, L. Stránky zabývající se programováním. [cit. 2008-1-24]. Dostupné z http://programujte.com/. ISSN 1801-1586

7 Seznam použitých zkratek a symbolů.

A/D - Analogově digitální převodník.

AC - Střídavý proud.

ACTIONSCRIPT - objektově orientovaný programovací jazyk pro aplikace vyvíjené pomocí Macromedia Flash.

ASSEMBLER - Jazyk symbolických adres.

AVR - je označení pro rodinu 8-bitových RISC mikroprocesorů z produktů korporace Atmel.

BIN - Dvojková soustava.

C - nízkoúrovňový, kompilovaný, relativně minimalistický programovací jazyk.

CSS - jazyk pro formátování internetových stránek.

HEX - Hexadecimální soustava.

HTML - HyperText Markup Language je značkovací jazyk pro hypertext.

I/O - Vstup/výstup.

I^2C – je multi-masterová počítačová sériová sběrnice.

JTAG - je standard definovaný normou IEEE 1149.1.

LCD - Displej z tekutých krystalů.

LED - elektroluminiscenční dioda.

LPT - Paralelní port počítače.

PC - Osobní počítač (anglicky personal computer).

PHP - skriptovací programovací jazyk, určený především pro programování dynamických internetových stránek.

PWM - Pulsně šířková modulace.

px - nejmenší jednotka digitální rastrové (bitmapové) grafiky.

SPI - je sériové periferní rozhraní.

TWI – obdoba I^2C využitá u mikroprocesorů.

USART - Synchroní a asynchroní seriové rozhraní.

WDT - hlídač správného běhu programu.

8 Seznam příloh.

Na přiloženém CD naleznete:

- Kompletní webovou aplikaci zprovozněnou na www.avr.poruce.com.
- Vývojový diagram funkce ***Zpracovaní***
- Tabulku s přehledem registrů mikroprocesoru AVR včetně jejich adres.
- Převodní tabulku pro instrukce AVR do binárního kódu
- Porovnání dat generovaných mojí aplikací v porovnání s programem AVR Studio.