Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra Fyziky

Jednoduchý programovatelný logický automat

bakalářská práce

Autor: **Martin Vlna**

Vedoucí bakalářské práce: **Ing. Michal Šerý**

České Budějovice 2007

Anotace

Cílem práce „Jednoduchý programovatelný logický automat" bylo vytvořit jednoduchý systém obsahující několik digitálních vstupů a výstupů, nad kterými bude možno programovat základní logické operace pomocí displeje, klávesnice a programovacího rozhraní systému. Jedná se o přiblížení k průmyslovému systému zvanému programovatelný logický automat (PLC).

Synopsis

The objective of the work „Easy programable logical controller" was to create an easy system which contains some digital inputs and outputs, under which it will be possible to program the basic logical operations with the help of the display, keyboard and system interface. The point is to put near to the industrial system which is called programable logical controller.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v úpravě vzniklé vypuštěním vyznačených částí archivovaných pedagogickou fakultou elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

Děkuji vedoucímu mé bakalářské práce panu Ing. Michalovi Šerému za odborné vedení, cenné rady a připomínky při vypracování mé bakalářské práce.

Obsah

Úvod

Moderní automatizace vyžaduje nasazení mnoha různých řídících systémů. Často se pro zjednodušení, zkvalitnění a zrychlení návrhu využívá již hotových programátorsky přívětivých jednotek. Vývoj řídící elektroniky pro obráběcí centrum by kupříkladu značně prodražil finální výrobek, který by pravděpodobně přestal být konkurenceschopný. Proto existuje řada různých firem, které se zabývají právě vývojem těchto „univerzálních" systémů s názvem Programovatelný Logický Automat zkráceně PLC (z anglického Programmable Logic Controller).

Vyrábí se mnoho různých druhů lišících se velikostí, vlastnostmi, složitostí i cenou. Práce na projektu „Jednoduchý programovatelný logický automat" má za cíl přiblížit se k těmto profesionálním systémům podstatou programování, funkcí, využitelností atd.

1. Jednočipové počítače vlastnosti a využití

Mikropočítač či jednočipový mikropočítač se považuje za stavový sekvenční automat. Tím myšleno, že zpracování dat (signálu) je prováděno po nedělitelných krocích, v nichž prochází mikropočítač známou posloupností stavů, přičemž průchod těmito stavy závisí kromě momentálních hodnot datových signálů ještě na jejich předchozích hodnotách.

Pod pojmem „jednočipový mikropočítač" se rozumí spojení mikroprocesoru, paměti a vstupně/výstupních obvodů do jediného celku s konstrukcí na jediném čipu. Prioritně jsou tedy jednočipové mikropočítače konstruovány pro použití ve funkci samostatných řídicích jednotek. Vzhledem k integraci různorodých obvodů na jediném čipu jsou konstrukce jednočipových mikropočítačů výrazněji omezeny ve velikostech paměťových prostorů, rychlosti apod., než je tomu u prostých mikropočítačů, které obvykle nejsou vybaveny žádnou samostatně programově přístupnou vstupně/výstupní periferií.

1.1 Architektura jednočipového mikropočítače

Mikropočítače obecně, a pochopitelně i jednočipové mikropočítače, mohou být konstruovány v jednom ze dvou základních typů architektury. K dispozici je architektura typu von Neumann nebo architektura harvardská.

Mikropočítače konstruované v těchto architekturách se zásadně odlišují přístupem ke konstrukci paměti dat. Zatímco architektura typu von Neumann má pro data i program společnou paměť, architektura harvardská striktně odlišuje paměť programu a paměť dat. Obě řešení mají své výhody i nevýhody. Základní výhodou, která se může při nepozornosti programátora stát katastrofickou nevýhodou, je v architektuře von Neumann možnost vykonávat data jako program a program interpretovat jako data. V harvardské architektuře to možné není (alespoň nejednoduchým způsobem), neboť ze svého principu na to není mikropočítač s touto architekturou konstruován. Pravdou však je to, že obě architektury je možné s pomocnými obvody převádět z jedné na druhou. U architektury harvardské není výsledek převodu zcela rovnocenný a odpovídající architektuře von Neumann. Je tomu tak proto, že v harvardské architektuře jsou oba adresové prostoty, tj. prostor pro data i program, alokovány od adresy 0 a navíc není možné vkládat data do programu a naopak bez respektování těchto základních obvodových vlastností.

1.2 Vlastnosti a architektura obvodů periferií

V předchozím textu byla zmíněna paměť programu a paměti dat. Kromě těchto paměťových prostorů oplývají jednočipové mikropočítače i paměťovým prostorem pro ovládání periferií ať už integrovaných nebo externích. Pokud je paměťový prostor periferií ve společném paměťovém prostoru s pamětí dat, říkáme, že periferie jsou paměťově mapovány.

1.3 Sběrnice mikropočítače

Aby mohl mikropočítač pracovat s pamětí programu, dat či obsluhovat periferie, musí být schopen generovat pro tyto obvody řadu signálů, tj. musí být s nimi schopen komunikovat. K této komunikaci slouží sběrnice. Pojmem sběrnice se označuje skupina signálů sloužící pro řízení komunikace a komunikaci s okolním světem mikropočítače. Každý mikropočítač má k dispozici alespoň tři typy sběrnice. Adresová část sběrnice je u mikropočítačů, a zvláště pak jednočipových, obvykle jednosměrná s orientací od mikropočítače k okolí. Slouží k přenosu adresy dat či adresy kódu instrukce. Datová část sběrnice je konstruována jako obousměrná a slouží k přenosu dat, která jsou adresována adresovou sběrnicí. Poslední částí sběrnice je sběrnice řídicí. Tuto sběrnici tvoří signály určené k řízení přenosu dat po datové sběrnici ve spolupráci se sběrnicí adresovou. Řídicí signály obvykle označují platnost adresy a dat na příslušných sběrnicích a slouží tak k synchronizaci mikropočítače s okolním světem.

1.4 Časování mikropočítače

Každý mikropočítač (mějme na paměti, že jde o sekvenční logický automat) vykonává zadaný program krok po kroku. Tento základní stavební kámen programu se nazývá instrukcí. Každá instrukce je složena alespoň z jednoho strojového cyklu a strojový cyklus je složen alespoň z jedné fáze. Dříve byl místo termínu fáze používán termín takt, který je dnes v podstatě synonymem k pojmu strojového cyklu.

Strojovým cyklem se obvykle označuje doba, která uplyne mezi dvěma přístupy na adresovou sběrnici. Než se tak stane, musí mikropočítač projít alespoň jednou, zpravidla však výrazně větším počtem fází.

Mikropočítače pracují téměř výhradně synchronně, tj. jejich činnost je řízena tzv. hodinovým signálem. Hodinový signál (míníme tím skutečný řídicí signál mikropočítače) je signálem s nejkratší periodou (nejvyšším kmitočtem), se kterým mikropočítače pracují. Nemusí být totožný s kmitočtem krystalu oscilátoru, který se pro generování hodinového kmitočtu používá. Hodinový kmitočet mikropočítače může být i řádově větší či menší než kmitočet krystalu. Z tohoto důvodu je naprostým omylem názor, že čím větší kmitočet krystalu tím větší je výpočetní výkon. Toto porovnávání výkonu může být pravdou pouze tehdy, pokud se porovnávají mikropočítače v rámci jednoho typu. S daleko horším přiblížením toto platí mezi jednotlivými řadami mikropočítačů od jediného výrobce. Zmíněné porovnání už vůbec nelze použít napříč trhem mikropočítačů produkovaných různými výrobci s použitím odlišné konstrukce. Chceme-li přesto použít nějakého kmitočtu pro srovnání a odhad výpočetního výkonu daného mikropočítače vůči nám známému, musíme použít právě informaci o rychlosti strojového cyklu. Ta totiž vyjadřuje mezní kmitočet přenosu dat po sběrnici, jehož je mikropočítač schopen. Mikropočítač může být kupříkladu tisíckrát rychlejší než přenos dat po sběrnici, ale nebude schopen poskytovat smysluplný výsledek rychleji, než je rychlost přenosu dat. Nedodáme-li po sběrnici data, nemá mikropočítač s čím pracovat a jeho rychlost není nic platná. Dalším

úskalím je, že se v tomto porovnávání obvykle zapomene na skutečnost v rozdílu stavby a konstrukce ve smyslu, zda jde o mikropočítač typu CISC nebo typu RISC. V praxi totiž není podstatná rychlost výpočtu triviální operace, ale to, zda mikropočítač bude schopen dostatečně rychle zpracovat naši úlohu, tj. problém daleko složitější, než je obyčejná aritmetická operace.

1.5 Řadič a aritmeticko-logická jednotka

V odborné literatuře bývá často uváděna věta typu „Mikropočítač je vystavěn okolo osmibitové aritmeticko-logické jednotky". Tato věta ovšem nevystihuje fakt, že aritmeticko-logická jednotka je v mikropočítači obvod důležitý, nikoli však zcela hlavní. Složitost této jednotky a poměřujeme-li složitostí obvodu jeho význam v zařízení, ani zdaleka nedosahuje složitosti obvodu řadiče. Řadič se skládá, alespoň v hrubém dělení, z registru instrukce, dekodéru instrukce, řídicích a časovačích obvodů mikroprogramu. Řadič se stará o veškeré časování dějů v mikropočítači a o generování řídicích signálů pro datové přenosy po datové sběrnici. Aritmeticko-logická jednotka pak provádí na základě řídicích signálů řadiče aritmetické či logické operace.

Aritmetické operace jsou řešeny obvykle obvodem sčítačky doplněným přenosem z nižšího a do vyššího řádu a obvodem pro realizaci záporného čísla pomocí tzv. dvojkového doplňku.

Logické operace obvykle řeší posuvný registr s paralelními vstupy a výstupy a řídicí logikou posunu vlevo či vpravo. Ve složitějších případech bývá posuvný registr nahrazen obvodem typu „barel shifter", který umožňuje realizovat vícebitové posuny. Obvod typu „barel shifter" je možné poměrně snadno realizovat sadou multiplexerů.

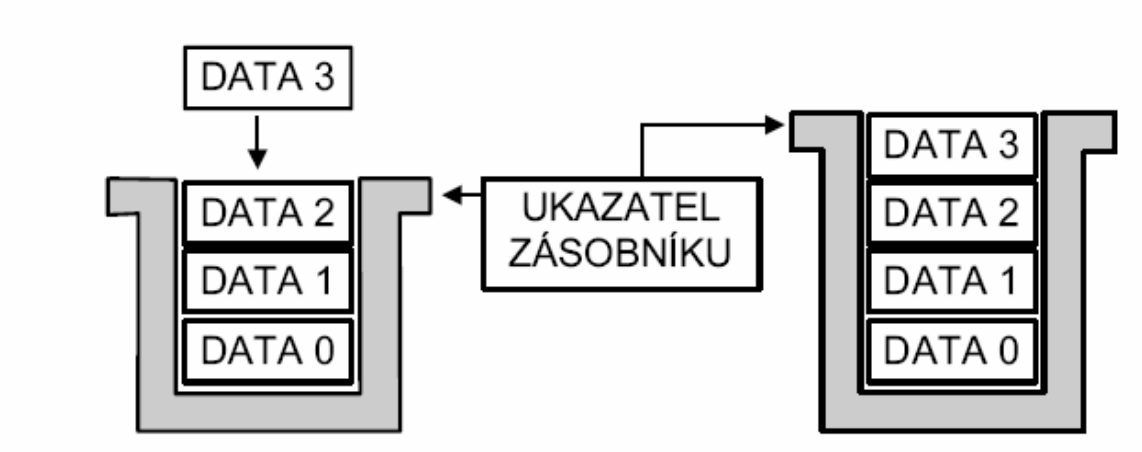
1.6 Paměť

Do obvodů aritmeticko-logické jednotky počítáme též vyrovnávací registry operandů a jeden nebo více registrů pro uložení výsledku výpočtu. Pro označení registru výsledku používáme pojem střadač.

Každý mikropočítač je určen pro zpracování dat a je tedy nutné ho vybavit pamětí pro jejich ukládání. Datovou paměť mikropočítače dělíme podle rychlosti přístupu k datům a podle způsobu přístupu k nim na zápisník, zásobník a obecnou datovou paměť. Paměti typu zápisník a zásobník jsou téměř výhradně integrovány společně s řadičem a aritmeticko-logickou jednotkou. Obecná paměť je mnohdy realizována alespoň z části vnějším paměťovým obvodem. Pravidlo o lokalizaci obecné datové paměti a někdy i zásobníku není možné chápat striktně a konkrétní řešení této lokalizace téměř vždy závisí na požadovaném rozsahu paměťových obvodů pro danou aplikaci.

Zápisník je paměť, která slouží pro dočasné uchování dat. Pojem dočasné nevyjadřuje fakt, že se snad data z této paměti ztrácejí, ale fakt, že k paměti typu zápisník má mikropočítač nejrychlejší přístup. Velikost paměti zápisníku je obvykle

velmi omezená a zápisník je tedy především určen k uchovávání mezivýsledků při složitějších výpočtech. Zápisník je konstruován jako paměť s libovolným přístupem (překlad slova „Random" slovem „náhodný" při označení paměti RAM - Random Access Memory je v tomto kontextu zavádějící).



obr.1.1 – Zápis dat na zásobník a modifikace ukazatele

Zásobník (Stack) je paměť typu LIFO. LIFO je zkratka z anglického označení Last-In First-Out vyjadřujícího skutečnost, že v tomto typu paměti je přístup k uloženým datům sekvenční, tj. poslední zapsaná hodnota je přečtena jako první. K zásobníku se ještě váže pojem vrchol zásobníku, tj. paměťová buňka, na niž ukazuje (adresuje ji) ukazatel zásobníku obr.1.1. Ukazatel zásobníku je inkrementován resp. dekrementován při zápisu resp. čtení dat v případě, že zásobník narůstá směrem k vyšším adresám. Používá se i řešení opačné, tj. zásobník narůstá směrem k nižším adresám. V tomto druhém typu řešení je ukazatel zásobníku při ukládání dat dekrementován a při jejich čtení inkrementován. Zásobník se obvykle využívá pro krátkodobé uložení dat především proto, že se o něj dělí program (uživatel) a CPU mikropočítače. CPU mikropočítače používá zásobník pro uložení návratové adresy v okamžiku volání podprogramu nebo přerušení. V případě přerušení pak mikropočítače Microchip řady PIC18 automaticky ukládají na zásobník též celý programovací model, tj. obsah všech interních registrů CPU. Vhodný soubor instrukcí pro práci se zásobníkem pak umožňuje jednoduché předávání parametrů při volání podprogramů ve vyšších programovacích jazycích. Ve velké většině případů totiž generátory kódu vyšších programovacích jazyků používají pro předávání parametrů podprogramů a funkcí právě zásobník jako obecnou strukturu dat.

Obecná datová paměť mikropočítače slouží k ukládání dat a větších datových struktur. Obvykle má k obecné datové paměti mikropočítač nejpomalejší přístup. Co do kapacity je však tato paměť největší.

Dělíme-li paměť mikropočítače z hlediska rychlosti přístupu, je nutné si uvědomit, v čem tento rozdíl spočívá. Rychlost přístupu k paměti v popisovaném dělení není dána obvodovou konstrukcí samotné paměti, ale určuje ji metoda, kterou používá řadič mikropočítače pro generování úplné adresy dat uložených v paměti.

1.7 Běh programu a zpracování dat

Program, který mikropočítač vykonává, tvoří jednotlivé instrukce mikropočítače. Pomocí základních instrukcí mikropočítače je tvořen výsledný algoritmus programu. Mikropočítač spouští program bezprostředně po ukončení sekvence základních nastavení. Tato sekvence probíhá v okamžiku resetu (inicializace) a kromě jiných úkolů řeší obvykle pomocí časového zpoždění ustálení kmitočtu oscilátoru, základní nastavení módu periferií a základní nastavení velikostí paměťových prostorů pro program data atd.

Program se spouští od zadané adresy programové paměti. Tato specifická adresa je zapsána řadičem do programového čítače. Tento specializovaný registr je určen k uchování adresy programové paměti a ukazuje na paměťovou buňku s instrukcí, která se bude vykonávat. V průběhu vykonávání instrukce je hodnota programového čítače inkrementována, čímž se zajišťuje přechod na další instrukci. Obsah programového čítače je zcela změněn, pokud se vykonává instrukce skoku. Dá se tedy říci, že instrukce skoku je speciální instrukce pro přenos dat do programového čítače. Z pohledu přenosu dat se tedy instrukce skoku nikterak neodlišuje od instrukcí, které používáme pro načítání dat např. při aritmetických operacích. Jediná odlišnost je v tom, že cílem přenosu dat při instrukci skoku není aritmeticko-logická jednotka, ale programový čítač.

Spouštění programu od specifické adresy může být prakticky řešeno dvěma způsoby. První řešení se označuje termínem „restart" adresa, druhé řešení pak termínem „reset" vektor (pozor! Reset vektor není totožný se stavem reset). V řešení typu „restart" je adresa pro spuštění přímo dána konstrukcí řadiče. V řešení typu „reset" vektor je adresa spuštění programu dána nepřímo. V tomto případě je programový čítač naplněn hodnotou, kterou řadič načte z adresy „reset" vektoru. Adresa „reset" vektoru je součástí tabulky vektorů přerušení. Jedná se o vyhrazenou, pevně určenou část programové paměti, která tvoří tabulku hodnot. Za naplnění těchto hodnot odpovídá programátor. Má-li se spustit program z adresy „reset" vektoru, pak první, co řadič udělá je přečtení hodnoty z příslušné položky tabulky přerušovacích vektorů.

V druhém kroku uloží přečtenou hodnotu do programového čítače, čímž vlastně provede instrukci skoku a spustí tak program.

Základní běh programu je možné ovlivnit pomocí tzv. přerušení. Přerušení je zásadní zásah do běhu programu, který spočívá v tom, že se dokončí rozpracovaná instrukce základního programu, uloží se návratová adresa a provede se volání specifického podprogramu, který nazýváme obslužný program přerušení či prostě obsluhou přerušení a začíná na vektoru přerušení.

V okamžiku volání obsluhy přerušení je nezbytné uložit obsah všech pracovních registrů mikropočítače (obvykle na zásobník). Dále se zakáže vyvolání přerušení od zdroje, který je právě obsluhován, a spustí se první instrukce obslužného programu přerušení. Popisované úkony při volání přerušení dělá obvykle mikropočítač automaticky. Pokud je automaticky nedělá, je nanejvýš vhodné úkony udělat v úvodních instrukcích obsluhy přerušení. Dodržením tohoto doporučení předejdeme mnoha chybám, které se jen velmi těžko odhalují.

Praxe ukazuje, že je vždy bezpečnější uložit pracovní registry (střadač apod.), než to neudělat. Nejčastější chybu, která při nesystematickém ukládání registrů

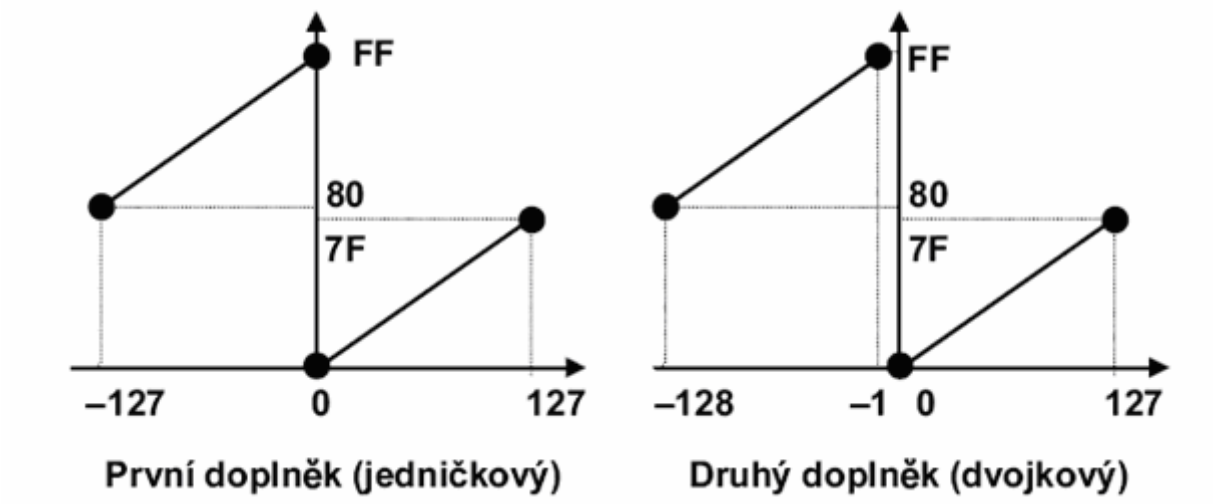
vznikne, můžeme označit jako chybu následnou či chybu druhé iterace. Tato chyba totiž vznikne při opravách programu, kdy se soustředíme na opravu chyby a opomeneme zkontrolovat, zda některý z pracovních registrů přepisujeme. Při velké smůle se chyba může projevit až za poměrně dlouhou dobu. Tato doba je dána zázněj vzniklým při běhu hlavního programu a periodickém volání přerušení. Zázněj vznikne tehdy, když doba potřebná k vykonání celé hlavní programové smyčky, včetně volání přerušení, je velmi blízká periodě volání přerušení.

Vzhledem k tomu, že mikropočítač na daný podnět zajišťuje volání přerušení automaticky, setkáváme se opět se dvěma způsoby konstrukce tohoto volání. Stejně jako v případě spouštění programu po inicializaci mikropočítače (resetu) jde v případě volání přerušení o načtení počáteční adresy obslužného programu přerušení do programového čítače. Mechanizmus načtení počáteční adresy se zajišťuje v mikropočítači konstrukcí řadiče a načtení adresy může být tedy přímé, typu „restart" nebo nepřímé, typu „vektor přerušení". Postup získání adresy se v obou případech kryje s postupem popsaným v případě spouštění programu (reset mikropočítače).

1.8 Čísla a jejich zobrazení

Mikropočítače pracují v číselné soustavě se základem dvě. V dvojkové soustavě tvoříme čísla vyšších hodnot nezjedná pomocí vyšších řádů. Jde o obdobu soustavy desítkové, v níž tvoříme pomocí vyšších řádů čísla s hodnotou větší než devět. Až potud je zcela jedno, v jaké soustavě počítáme. Odlišnost mezi oběma soustavami se objeví při práci se zápornými čísly. Zatímco v desítkové soustavě použijeme k označení záporného čísla prosté znaménko mínus, v soustavě dvojkové zobrazujeme záporná čísla odlišným způsobem, a to pomocí prvního nebo druhého (též dvojkového) doplňku. Důvod odlišného zobrazení záporných čísel je prostý a má kořeny v konstrukci aritmeticko-logické jednotky a vhodné obvodové interpretaci zmíněného znaménka mínus. Obvodově je aritmetická část této jednotky řešena sčítačkou. Aby bylo možné použít tuto sčítačku i pro realizaci rozdílu, bylo nutné zvolit vhodné zobrazení záporných čísel. Pro vhodné zobrazení záporných čísel je možné zvolit buď tzv. první doplněk nebo druhý doplněk (zobrazení se také označují termíny jedničkový a dvojkový doplněk).

První doplněk vznikne ze zadaného čísla záměnou jedniček za nuly a nul za jedničky. Jedná se tedy o prostou operaci negace bit po bitu. Podíváme-li se na možné výsledky, zjistíme, že záporná čísla mají vždy v nejvyšším řádu jedničku. Dále zjistíme že součet kladného a záporného čísla realizuje rozdíl. Patrně též objevíme, že dvojí aplikace prvního doplňku na dané číslo ponechá číslo beze změny nebo chceme-li, vytvoří číslo původní. Přijdeme však patrně i na jednu nevýhodu a tou je fakt, že v zobrazení čísel pomocí prvního doplňku máme k dispozici dva kódy pro nulu. První doplněk aplikovaný na hodnotu nula vrací na všech řádech jedničky. Protože však neexistuje záporná nula, můžeme toto zobrazení aplikovat pouze tehdy, jestliže provedeme dodatečnou korekci hodnoty, tj. když v případě, zeje výsledkem operace „záporná" nula, provedeme nad výsledkem dodatečnou operaci prvního doplňku



obr.1.2 – Zobrazování celých čísel v prvním a druhém doplňku

Nevýhodu se dvěma nulami odstraňuje zobrazení záporných čísel pomocí druhého (dvojkového) doplňku. Číslo s opačným znaménkem vytvoříme z čísla zadaného tak, že v prvním kroku provedeme jeho negaci a v druhém kroku přičteme k výsledku jedničku. Pro zobrazení v druhém doplňku platí stejná pravidla jako v zobrazení pomocí doplňku prvního, avšak s výhodou odstranění dvojí nuly. Za každou výhodu je ale nutné zaplatit. V případě druhého doplňku platíme tím, že rozsah zobrazení je pro záporná čísla vždy o jedničku větší než pro čísla kladná, tj. např. pro osmibitové číslo je rozsah zobrazení v druhém doplňku od -128 po 127.

Dalším problémem při zobrazování a práci s čísly je zobrazení a aritmetické operace s reálnými čísly. Reálná čísla můžeme zobrazovat buď v plovoucí řádové čárce nebo v pevné řádové čárce. Zobrazení v plovoucí řádové čárce je v technice mikropočítačů a především jednočipových mikropočítačů záležitostí matematických knihoven a mikropočítače s tímto zobrazením bez značné programové podpory právě knihovních funkcí nejsou schopny pracovat. Mikropočítače jsou však, alespoň v některých případech, schopny pracovat se zobrazením reálných čísel v pevné řádové čárce. Toto zobrazení nazýváme „fractional". Pro čísla v zobrazení „fractional" platí, že jsou vždy z intervalu (0, 1) tj. vždy menší než 1. V některých implementacích zobrazení „fractional" obsahují čísla v tomto zobrazení i znaménko a jsou tudíž z intervalu (-1,1). Pevná řádová čárka je vždy umístěna vlevo od nejvyššího řádu (bitu) a dekadický ekvivalent čísla vypočítáme podle vztahu:

$$A = a_n\, 1/2^1 + a_{n-1}\, 1/2^2 + \ldots + a_0\, 1/2^{n+1}$$

kde „A" je žádané číslo, „a" je bit řádu „n" a „n" označuje řád příslušného bitu čísla. Pro názornost uveďme příklad. Mějme osmibitové číslo vyjádřené v dvojkové soustavě takto:

$$11000001_B$$

Při výpočtu dekadického ekvivalentu zadaného čísla vyjdeme z výše uvedeného vztahu a výpočet bude tedy vypadat:

$$1.1/2^1 + 1.1/2^2 + \ldots + 1.1/2^{256} =$$
$$= 1.0{,}5 + 1.0{,}25 + \ldots 1.0{,}00390625 =$$
$$= 0{,}75390625$$

dekadický ekvivalent uvedeného čísla vyjde 0,75390625.

Zobrazení typu „fractional" můžeme doplnit celočíselnou aritmetikou pro práci s hodnotami nad 1, přičemž informaci o znaménku ponese pak celočíselná část zobrazení. Tímto triviálním postupem získáme poměrně slušnou aritmetiku v pevné řádové čárce.

1.9 Programovací model

Pojmem programovací model označujeme soupis vlastností a prostředků, jimiž je mikropočítač vybaven a které může při konstrukci programu využít programátor. Jedná se obvykle o popis vnitřní struktury mikropočítače v programátorském smyslu, tj. například kolik a jakých střadačů máme k dispozici, kolik a jakých ukazatelů zásobníku můžeme využít, jaké jsou rozsahy paměti, jaká je programová dostupnost periferií apod.

1.10 Vývojové prostředky

Vývojové prostředky pro programování a ladění programů jednočipových mikropočítačů jsou nezastupitelnými nástroji každého programátora. Mohou být různé kvality, různé ceny, mohou poskytovat více či méně komfortní nástroje pro programování a ladění, mohou být nezbytné nebo takové, bez nichž se lze obejít.

Vzhledem k tomu, že se mikropočítače, a zvláště pak jednočipové mikropočítače, programují ve značné míře pomocí zdrojového textu, považujeme vhodný textový editor za naprostý základ vývojových prostředků. Překladač zdrojového textu do operačních kódů je druhým nezbytným nástrojem, který potřebujeme pro ladění aplikací. Nejlepší variantou je, když překladač a editor spolupracují alespoň na úrovni hlášení chyb v průběhu překladu a na úrovni interaktivního vyhledávání chybových řádků ve zdrojovém textu.

Dalším nezbytným nástrojem je ladicí program (debugger) aplikace. Ladicí program může být v samostatné formě, tj. běží na hostitelském počítači a simuluje činnost reálného mikropočítače. Takový program pak označujeme termínem simulátor.

Druhou variantou je, že ladicí program běží na hostitelském počítači a přes tzv. emulační rozhraní spolupracuje přímo s mikropočítačem nebo s jeho hardwarovou obdobou. V tomto případě nazýváme celý komplex, tj. ladicí program a emulační zařízení, emulátor. Zásadní rozdíl mezi emulátorem a simulátorem spočívá v tom, že při práci na emulátoru pracujeme s reálným prostředím, které zabezpečuje maximální

míru kompatibility s reálným mikropočítačem, zatímco při práci na simulátoru je reálné prostředí pouze simulováno, tj. stupeň kompatibility se skutečným mikropočítačem bývá výrazně nižší. Simulátor se hodí a je ho možné doporučit pro konstrukci částí programu, které nezávisí na reálném čase. Jedná se např. o aritmetické výpočty, korekce a přepočítávání dat. Simulátor se naprosto nehodí k ladění komunikace mezi mikropočítačem a okolím.

Hlavní výhodou simulátoru je jeho cena. Bývá řádově levnější než emulátor. Jakýmsi mezistupněm, který se snaží o kompromis mezi cenou zařízení a jeho vlastnostmi a emulačními schopnostmi, jsou vývojové kyty či vývojové desky.

Oproti simulátoru mají výhodu v tom, že jsou konstruovány za pomoci mikropočítače, k jehož ladění jsou určeny. Kompatibilita s mikropočítačem ve skutečné aplikaci je zde téměř stoprocentní. Téměř je slovo použité v tomto případě záměrně. K tomu, abychom mohli ladit program aplikace na vývojové desce, potřebujeme obvykle vlastnosti, které mikropočítač nemá. Potřebujeme např. možnost krokování programem, zastavení programu na zadaném místě (breakpoint) atd. K využití těchto vlastností či schopností vývojové desky je nezbytné, aby byla deska vybavena základním komunikačním programem, který tyto ladicí funkce zajišťuje a který nazýváme obvykle „kernel". Právě vliv tohoto programu na mikropočítač způsobuje ne zcela stoprocentní kompatibilitu s výslednou aplikací mikropočítače.

Kernel obvykle využívá ke komunikaci s ladicím programem na hostitelském počítači některou z periferií (nejčastěji sériový asynchronní komunikační kanál) mikropočítače a ta je pak pro aplikaci nedostupná. I přes tato omezení se vývojové desky hojně používají a díky své ceně jsou dostupné širokému okruhu konstruktérů a programátorů.

2. Jednočipový mikropočítač Microchip PIC16F877A

Mikropočítač firmy Microchip řady PIC16F877A je postavený na RISC architektuře o 35 instrukcích s šířkou slova 14bitů. Napájecí napětí 5V, maximální taktovací frekvence 20MHz s výkonem 5Mips. Programová paměť (FLASH) o velikosti 8K slov x 14 bitů, operační paměť 368 x 8b a integrovaná paměť EEPROM 256 x 8b. Mikroprocesor disponuje řadou digitálních periferií třemi čítači/časovači, pulzně šířkovou modulací, synchronním sériovým porte (SSP) s SPI (master mód) a I2C (master/slave), USART, paralelní port PSP. Z analogových periferií 10bitový osmi kanálový A/D převodník, analogovými komparátory s programovatelnou vnitřní referencí.

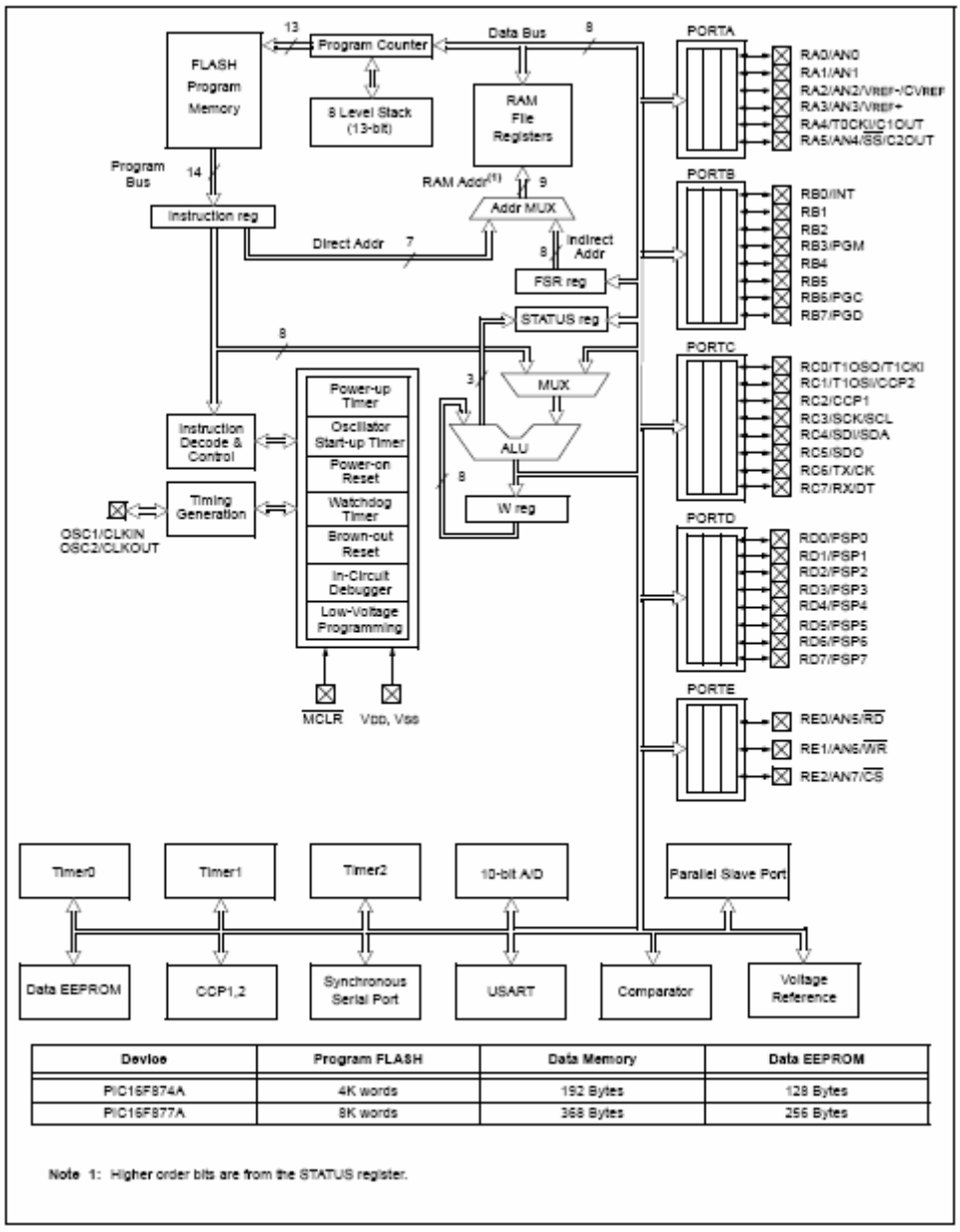
Procesor je konstruován ve třech typech pouzder PDIP (40 pin), PLCC nebo QFP (44 pin). K dispozici je 5 vstupně výstupních portů, 15 druhů přerušení od periferií. Stručné shrnutí procesorů 16F67xA na obr.2.1

Key Features	PIC16F873A	PIC16F874A	PIC16F876A	PIC16F877A
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory (bytes)	128	128	256	256
Interrupts	14	15	14	15
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Analog Comparators	2	2	2	2
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions
Packages	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin MLF	40-pin PDIP 44-pin PLCC 44-pin QFP	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin MLF	40-pin PDIP 44-pin PLCC 44-pin QFP

obr.2.1 – přehled vlastností mikroprocesorů PIC16F87XA

2.1 Struktura jednočipového mikropočítače PIC16F87XA

Na obrázku obr.2.2 je patrná vnitřní struktura mikropočítače PIC16F87XA. Programová paměť FLASH je adresovaná 13-bit programovým čítačem (PC), je čteno 14-bit dato instrukce, ve které je obvykle (při bitových a bytových operací) obsažena i přímá adresa na datovou paměť. Instrukce je zpracovávána v instrukčním dekodéru (řadiči), kde je generována skupina řídících signálů pro celý procesor. Požadovanou část kódu instrukce pak zpracovává aritmeticko-logická jednotka. Při aritmetických, nebo logických operacích je prvním operandem registr z paměti RAM a druhým střadač (W reg.). Důležitou roli zde pak dále hraje STATUS registr do kterého se generují příznaky jako je přetečení (7-bit -> 8-bit) Carry, podtečení Borrow, nulový výsledek Zero, poloviční přetečení (3-bit -> 4-bit) Digital Carry/Borrow. Dále se v tomto registru nacházejí i dva horní adresové bity paměti RAM RP1:RP0. Adresový multiplexer vybírá způsob adresování paměti RAM (přímý, nepřímý). Jsou zde patrné i vstupně/výstupní brány a veškeré periferie procesoru (Timer0, USART, atd.)



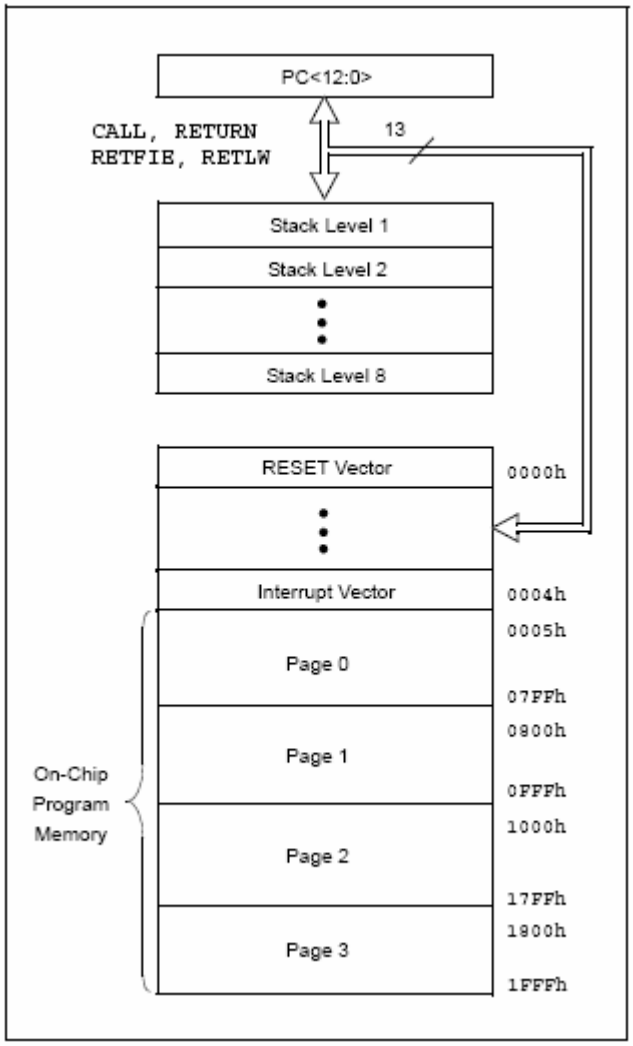
Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the STATUS register.

obr.2.2 – vnitřní struktura jednočipového mikropočítače PIC16

2.2 Organizace programové paměti mikroprocesoru

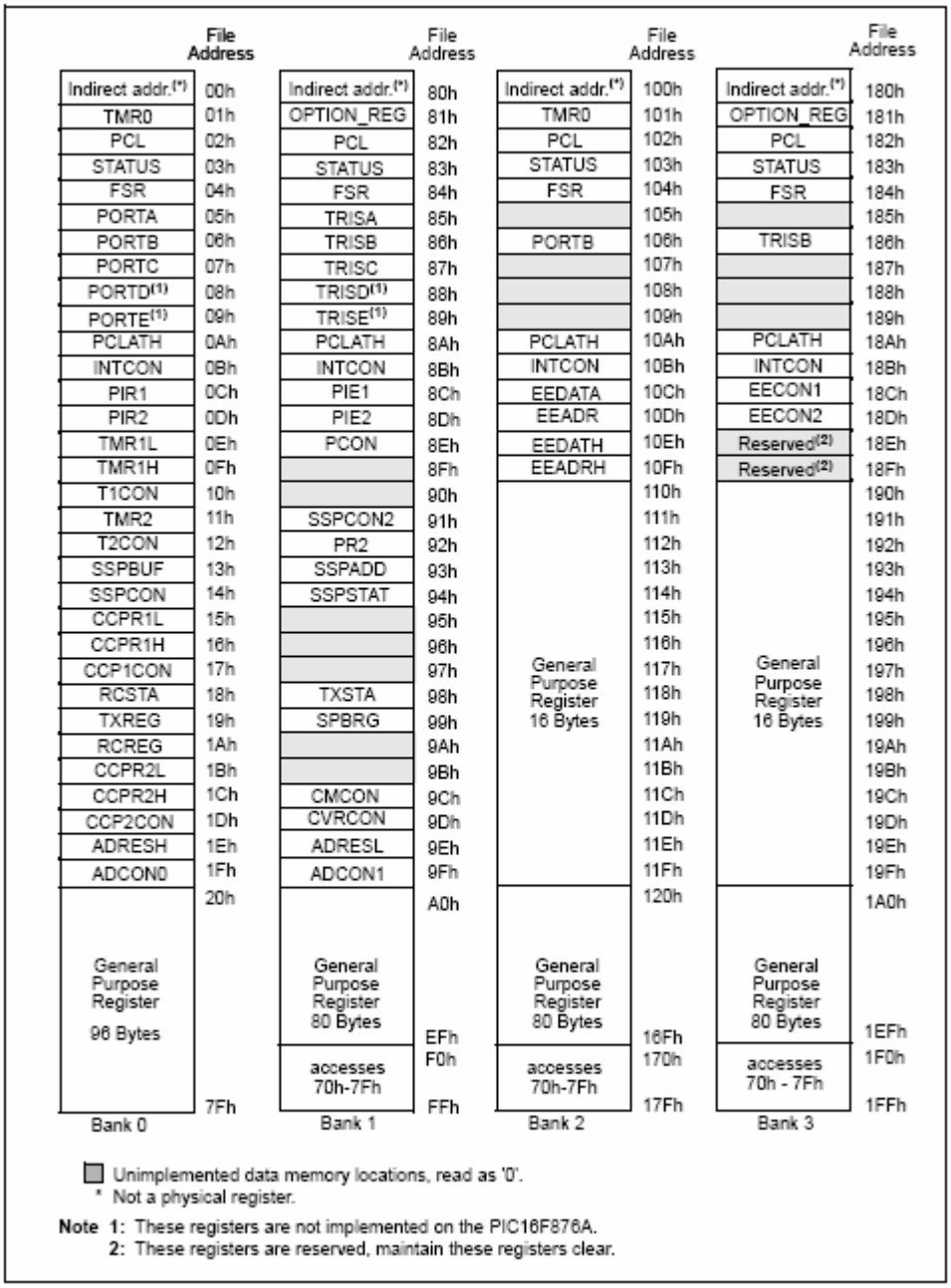
Součástí mikroprocesoru PIC16F877A je 13-bit programový čítač (PC), který dokáže adresovat 8K slov programové paměti. Osmi-úrovňový stack sloužící pro uložení návratové adresy při instrukcích skoku. Reset vektor se nachází na adrese 0000h a vektor přerušení od adresy 0004h. Programová paměť je dále dělena na stránky (každá 2048B) z důvodů omezení instrukcí skoku, které dokážou adresovat pouze v rámci jedné stránky.



obr.2.3 – Organizace paměti programu

2.3 Paměť dat a speciální funkční registry

Datová paměť procesoru PIC16F877A je rozdělena do čtyř bank. Součástí každé banky je několik speciálních funkčních registrů (SFR), jimiž je možno nastavit procesor a jeho veškeré periferie. Mimo SFR je v paměti dat k dispozici 368 volných bytů pro ukládání dat programu. Paměť dat je dělena do čtyř bank z důvodů omezení logických, aritmetických instrukcí a instrukcí přesunu, kdy je možné adresovat pouze v rámci jedné banky (128B). Horní 2b adresy jsou uložené v registru STATUS RP1:RP0. Při používání nepřímého adresování v paměti dat za použití File Select Registr (FSR) se lze pohybovat přímo v bance 0,1 nebo 2,3. FSR adresuje registr v rozmezí 0 – 255.



obr.2.4 – organizace paměti dat

2.4 Instrukční sada mikroprocesorů řady PIC16

Instrukční sadu dělíme do třech základních kategorií.

- Bytově-orientované operace
- Bitově-orientované operace
- Číselné a řídící operace

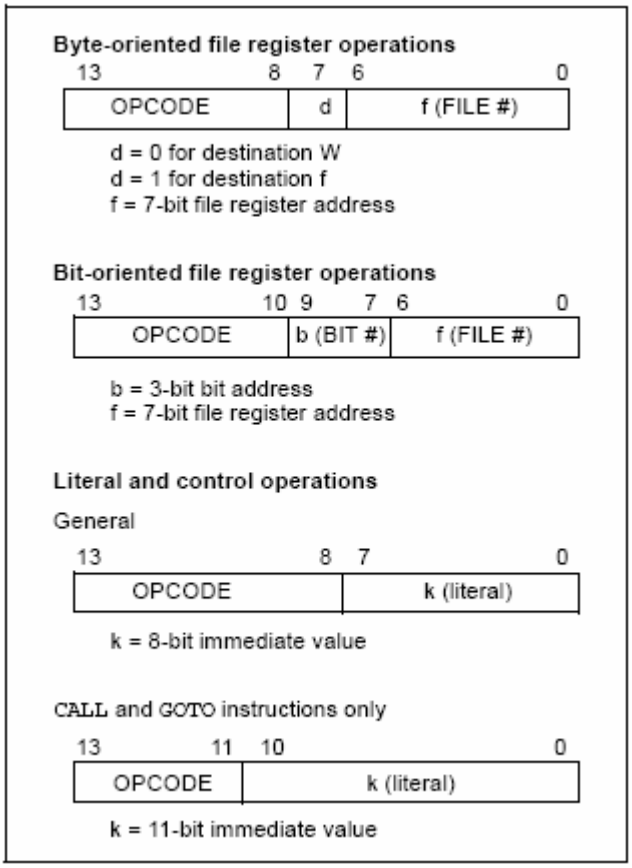
Jak již bylo zmíněno, procesory PIC16 disponují operacemi šířky 14-bit, které specifikují typ instrukce, zpracovávaný operand a obvykle i uložení výsledku operace. Formát operace je patrný na obr.2.6.

Pro bytově orientované operace reprezentuje „f" zpracovávaný registr a „d" uložení výsledku operace. Může být bud do f registru nebo do střadače.

Pro bitově orientované operace reprezentuje „b" bit z registru „f", který se má zpracovávat. Tato operace nemá výsledek, mohou to být operace nastavování nulování a operace skoku.

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1 .
PC	Program Counter
TO	Time-out bit
PD	Power-down bit

obr.2.5 – používané v instrukcích



obr.2.6 – přehled formátu instrukcí

U číselných a řídících operací figuruje pouze číselná konstanta „k". U číselných operací je to přesun konstanty do střadače W. Řídící operace jsou operace přesunu v paměti programu (skoky, volání podprogramů) zde má „k" šířku 11-bit a jedná se o dolních 11 bitů programového čítače, proto je možné adresovat 2048 bytů – instrukcí. Jak již bylo zmíněno, právě z tohoto důvodu je paměť programu dělena na stránky o velikosti 2048 bytů.

Na vykonání jednoho instrukčního cyklu u procesorů PIC je třeba čtyřech taktů oscilátoru. V těchto taktech je podle vnitřní struktury instrukce čtena a vykonávána. Příklad frekvence vykonávaného programu $F_{cy} = F_{osc}/4$ s dobou vykonávání jedné instrukce $T = 1/F_{cy}$

Redukovaná instrukční sada procesoru PIC16F87XA pro překladač MPASM™ Assembler je patrná z obrázku obr.2.7.

Za povšimnutí stojí fakt, že některé instrukce ke svému vykonání potřebují až tři instrukční cykly. Důvodem je, že programový čítač ukazuje na instrukci, která se bude vykonávat, ta už je tudíž částečně připravena k vykonání, avšak aktuální instrukce modifikuje programový čítač na jinou adresu, než bylo očekáváno. Následuje tedy další instrukční cyklus zpracování nové instrukce. Teprve poté se může vykonat. To má za následek prodloužení doby vykonávání jedné instrukce (goto, call, atd.).

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1,2
CLRF	f	Clear f	1	00 0001 1fff ffff		Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff			1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff			1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1,2
MOVWF	f	Move W to f	1	00 0000 1fff ffff			
NOP	-	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff			1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff			1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff			3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff			3
LITERAL AND CONTROL OPERATIONS							
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk		C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk		Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk			
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100		$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk			
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk		Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk			
RETFIE	-	Return from interrupt	2	00 0000 0000 1001			
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk			
RETURN	-	Return from Subroutine	2	00 0000 0000 1000			
SLEEP	-	Go into Standby mode	1	00 0000 0110 0011		$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk		C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk		Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

obr.2.7 – instrukční soubor mikroprocesoru PIC16

3. Návrh aplikace s jednočipovým počítačem

Cílem práce je vyvinout zařízení, do kterého je uživatel schopen přes jednoduché programovací rozhraní společně s klávesnicí a displejem, naprogramovat soustavu mezi sebou různě propojených základních logických operací s možností hardwarových vstupů, a několika hardwarových výstupů.

3.1 Výběr hardwaru, blokové schéma systému

Srdcem systému bude popisovaný jednočipový mikroprocesor PIC16F877A taktovaný frekvencí oscilátoru 8MHz.

Pro lepší grafické zobrazení bude vybrán Liquid Crystal Display Module firmy Elatec EL-12864A™ LCM o rozlišení 128x64 pixelů.

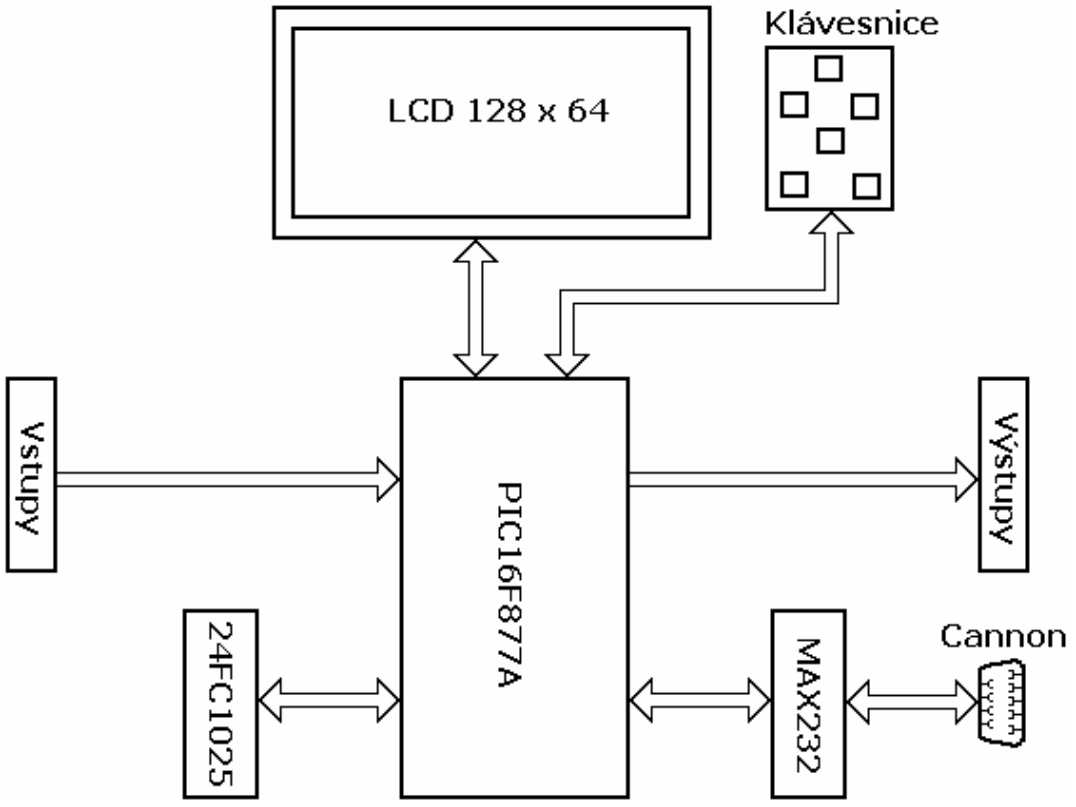
Pro ovládání zařízení je potřeba šesti kláves s funkcemi doprava, doleva, nahoru, dolu, OK, ESC. Klávesnici budou tvořit mikrospínače v zapojení do matice.

Externí uložiště dat bude tvořeno 1024Kbit I^2C™ CMOS sériovou EEPROM pamětí od firmy Microchip 24FC1025 taktovanou frekvencí 1MHz.

Pro komunikaci s osobním počítačem bude připravena sériová linka za použití převodníku úrovní MAX232.

Vstupy a výstupy budou vhodně zapojené vstupně/výstupní brány mikroprocesoru s ohledem na velikosti jejich logických úrovní a maximální proudové zatížení.

Blokové schéma zapojení systému je patrné na obr.3.1



obr.3.1 – blokové schéma zapojení aplikace „Jednoduchý programovatelný logický automat“

3.2 Grafický displej Elatec EL-12846A™ LCM

Do aplikace bude vybrán grafický displej o rozlišení 128 x 64 pixelů. Důvodem proč grafický, nikoli znakový displej, je flexibilita zobrazovaných znaků a velikost samotného displeje, což bude užitečné zejména v programovacím rozhranní zařízení.

Modul displeje obsahuje teplotně kompenzovaný generátor záporného napětí, které je pro funkci LCD nezbytné. Nevýhodou je poměrně velký počet ovládacích vodičů a absence znakové sady - jedná se o čistě grafický LCD a znaková sada bude nutné vytvořit externě v sériové paměti EEPROM. Komunikační sběrnice je paralelní 8-bitová, dále jsou nutné řídící signály (vývody E, RS, CS1, CS2, RW). Vývod RST není nutný zapojí se přímo na log. „1". Zapojení vývodů modulu je uvedeno v následující tabulce.

Vývod	Název	Funkce
1	Vss	napájení GND
2	Vcc	napájení +5V
3	Vee (Vo)	nastavení kontrastu
4	RS	volba mezi: RS=0 - instrukcí, RS=1 - daty
5	RW	volba mezi: RW=0 - zápisem, RW=1 - čtením
6	E	hodinový signál
7	DB0	data bit DB0
8	DB1	data bit DB1
9	DB2	data bit DB2
10	DB3	data bit DB3
11	DB4	data bit DB4
12	DB5	data bit DB5
13	DB6	data bit DB6
14	DB7	data bit DB7
15	CS1	výběr řadiče pro levou část LCD (CS1=1 - vybráno)
16	CS2	výběr řadiče pro pravou část LCD (CS2=1 - vybráno)
17	RST	reset modulu (RST=0 - reset, RST=1 - normalní funkce)
18	Vout	výstup záporného napětí pro LCD (asi -12V)
19	LED+	anoda podsvětlovacích LED
20	LED-	katoda podsvětlovacích LED

3.2.1 Komunikace s displejem

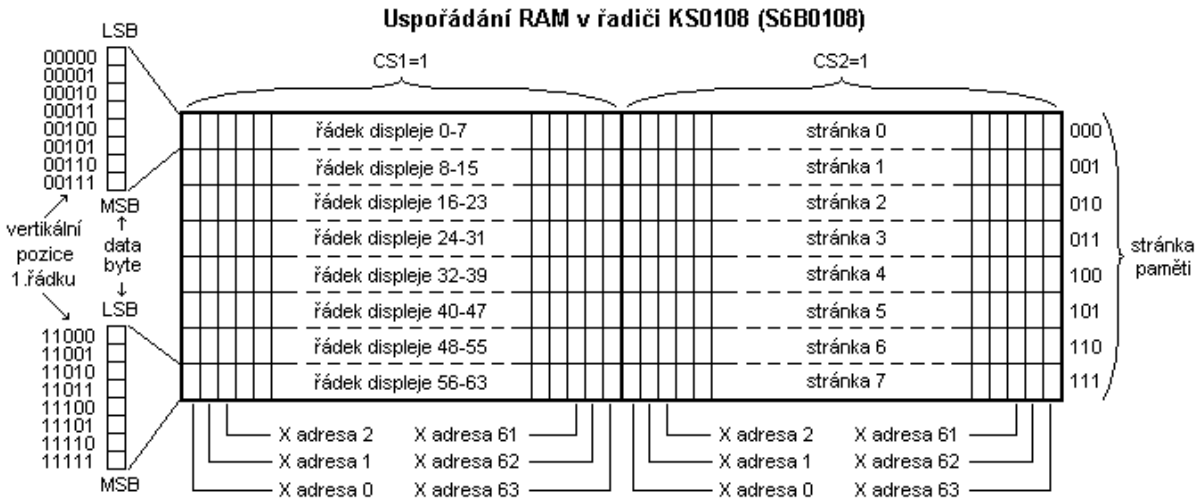
Celý displej je rozdělěn na 2 poloviny s rozlišením 64x64 pixelů. Každá z nich má svůj vlastní řadič (KS0108B). Všechny vývody mimo CS1 a CS2 jsou spojeny paralelně. Pomocí vývodů CS1 a CS2 se volí práce buď s levým, pravým nebo oběma řadiči (aktivní v log.1) viz obr.3.2.

Displej komunikuje po 8-bitové paralelní sběrnici. Volba mezi zápisem a čtením se provádí pomocí vývodu RW a volba mezi instrikcí a daty pomocí vývodu RS. Záporným pulzem na vývodu RST se LCD vypne a vynulují se interní registry. Obsah obrazové paměti se nezmění, takže po zapnutí LCD instrukcí se objeví původní obraz. Tento vývod není nutný a stačí ho připojit na +5V.

Při zápisu instrukce nebo dat je byte ze sběrnice načten při sestupné hraně kladného impulzu na vývodu E, který by měl trvat alespoň 450ns. Je důležité, aby byla data na sběrnici přivedena ještě před příchodem vzestupné hrany E a ne těsně před sestupnou hranou, jak naznačuje graf v dokumentaci na straně 15.

Stavové slovo obsahující "busy" příznak, stav resetu a zapnuto/vypnuto se čte následovně: Vývod RW na log.1, vývod RS na log.0 a následně E na log.1. V tomto stavu se objeví stavové slovo na sběrnici. Význam jeho jednotlivých bitů je uveden v tabulce instrukcí. Po přečtení slova vývod E zpět na log.0. Při čtení dat se postupuje stejně, pouze vývod RS musí být nastaven na log.1 a před vlastním čtením je třeba vygenerovat na E jeden kladný pulz (tzv. "dummy read").

Po zapnutí napájení není možné ihned vysílat instrukce. Je třeba buď generovat časovou prodlevu, nebo dokola číst příznaky "busy" a "reset" a čekat, dokud nebudou oba v log.0. Příznaky "busy" a "reset" je nutné kontrolovat i po resetu vývodem RST.



obr.3.2 – Uspořádání RAM v řadiči KS0108

3.2.2 Instrukce displeje

Viz. následující tabulka. Po instrukcích zápisu a čtení dat, se automaticky zvyšuje hodnota čítače adresy v X. Po posledním (63) sloupci se začíná opět zleva. Stránky se automaticky neinkrementují. Registr pro posun prvního řádku lze využít pro vertikální posouvání obsahu LCD.

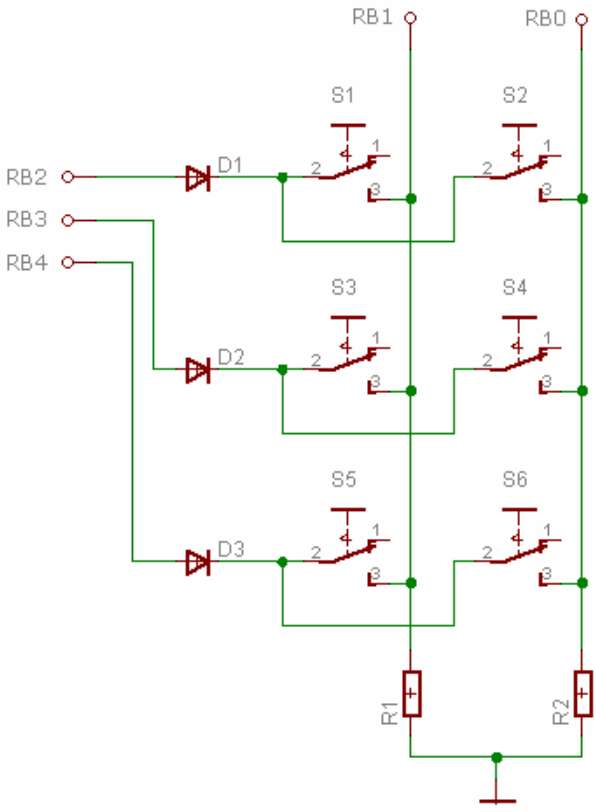
Význam instrukce	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
zap./vyp. displej, D=1 - zapnuto	0	0	0	0	1	1	1	1	1	D
nastaví stránku paměti (svislou osmici pixelů, bráno shora), rozmezí 0-7	0	0	1	0	1	1	1	stránka		
nastaví pozici v X (bráno zleva), rozmezí 0-63	0	0	0	1	X adresa					
nastaví vertikální pozici horního řádku, rozmezí 0-63	0	0	1	1	Y posuv					
zápis dat (LSB nahoře, MSB dole)	1	0	data byte							
čtení dat (LSB nahoře, MSB dole)	1	1	data byte							
čtení stavového slova B=1 - řadič zaneprázdněn, B=0 - připraven D=1 - LCD zapnut, D=0 - LCD vypnut R=1 - probáhá reset, R=0 - reset ukončen	0	1	B	0	D	R	0	0	0	0

3.3 2x3 maticová klávesnice

Z důvodů úspory vývodů vstupně/výstupní brány mikroprocesoru bylo zvoleno zapojení mikrospínačů do matice 2x3 podle obr.3.3. Usměrňovací diody (D1 : D3) zde plní funkci jištění proti zničení pinu brány, kdy při nevhodné kombinaci stisků kláves dochází k spojení výstupního pinu se vstupním nakrátko. Rezistory (R1, R2) jsou tzv. pull-down rezistory, které definují stav log. „0" ve stavu nestisknutých tlačítek.

3.3.1 Detekce stisknutých kláves

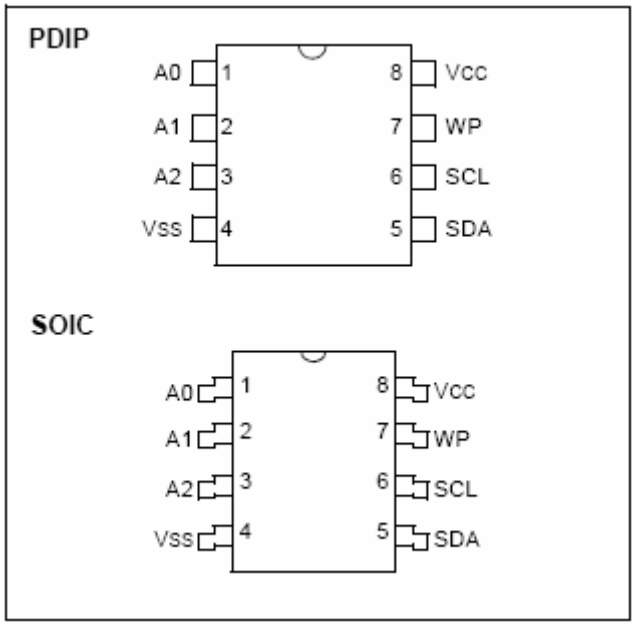
Detekování takovéto klávesnice spočívá v sekvenci buzení jednotlivých řádků a čtení sloupců. Způsob zpracování informace o stisknuté klávese je pak na kreativitě programátora.



obr.3.3 – zapojení klávesnice 2x3

3.4 Sériová I²C™ EEPROM paměť Microchip 24FC1025

Jako externí uložiště dat bude v aplikaci „Jednoduchý programovatelný logický automat" použita sériová paměť 24FC1025 s kapacitou 1024Kbit. Předpokládá se, že v této paměti bude uložena znaková sada grafického displeje (ascii tabulka) a vytvořený program PLC.



obr.3.4 - vývody pouzdra

3.4.1 Specifikace paměti

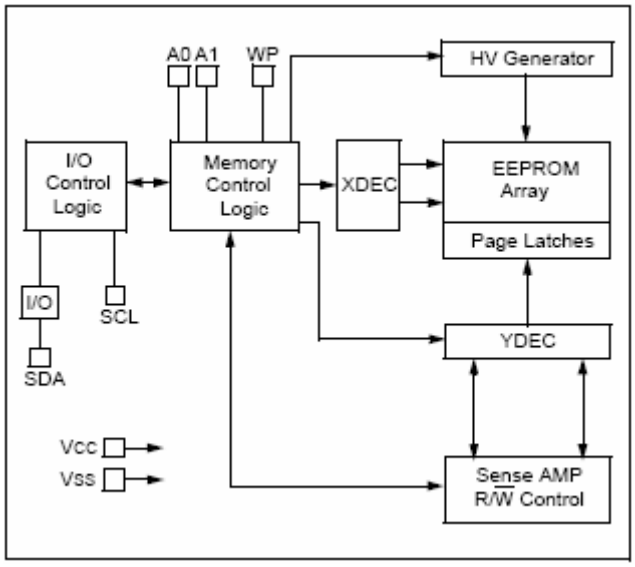
24FC1025 s napájecím napětím v rozsahu 2,5-5,5V může být taktována maximální frekvencí 1MHz.

Paměť je vytvořena CMOS technologií s proudovým odběrem při zápisu 5mA, při čtení 450µA, provozní (standby) odběr 100nA vše při napájecím napětí 5,5V.

Komunikace probíhá po 2-wire serial interface bus, I²C™ compatible. 128B stránkování při zápisu, doba zápisu 5ms, hardwarová ochrana proti přepsání WP , 1M cyklů zápis/čtení. Paměť může být v pouzdru 8-pin PDIP, SOIC obr.3.4

3.4.2 Popis paměti

24FC1025 je 128K x 8 (1024kbit) seriová Electrically Erasable PROM. Jako nízko příkonová může být použita v mnoha různých systémech. Dá se využít bytové nebo stránkové sekvenční zapisování (velikost stránky 128B). Čtení může být náhodné, nebo sekvenční díky vnitřnímu systému inkrementování adresy. Paměť je rozdělena do dvou bloků po 512Kbit, kdy výběr bloku je součástí kontrolního bytu. Čtení je tedy pouze od 0000h – FFFFh, nebo 10000h – 1FFFFh. Paměti se dají kaskádovitě propojit, čímž se dá získat paměťový prostor až 4Mbit.



b

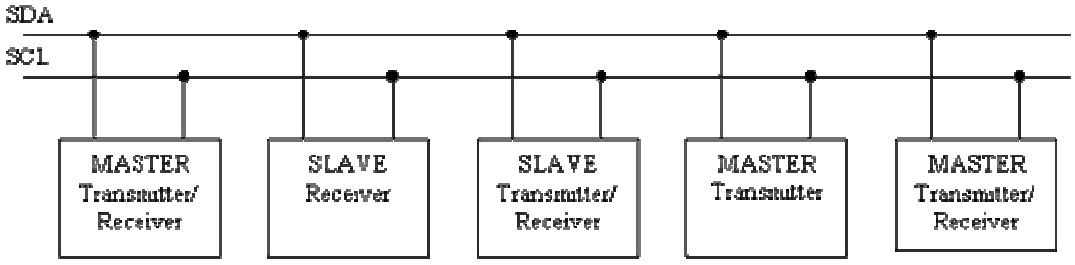
obr.3.5 – vnitřní struktura paměti

3.4.3 Popis komunikace po sběrnici I²C™

Sběrnice I²C™ je dvouvodičové datové propojení mezi jedním nebo několika procesory (Masters) a externími periferiemi - součástkami (Slaves). Všechny součástky jsou připojeny na téže sběrnici a jsou cíleně vybírány svými adresami. Adresy i data se přenášejí týmiž vodiči. Sběrnice umožňuje velmi jednoduché propojení mezi několika integrovanými obvody a bezproblémové dodatečné rozšiřování.

3.4.4 Protokol sběrnice

Mohou být připojeny všechny integrované obvody, které zvládají speciální protokol sběrnice. Mimo integrovaných obvodů RAM, EEPROM, obvodů pro rozšíření portů, A/D a D/A převodníků a obvodů hodinových signálů existuje ještě celá řada speciálních integrovaných obvodů, jako například budiče displejů nebo integrovaných obvodů pro televizní a audio techniku. Sběrnice I²C používá sériovou datovou linku SDA a linku hodinového signálu SCL. Data a adresy se přenášejí podobně jako v posuvných registrech společně s hodinovými impulsy. Obě linky je možno používat jako obousměrné. Jsou vybaveny zvyšovacími (pull-up) odpory a mohou být každým účastníkem sběrnice staženy na nízkou úroveň výstupem s otevřeným kolektorem nebo drainem.



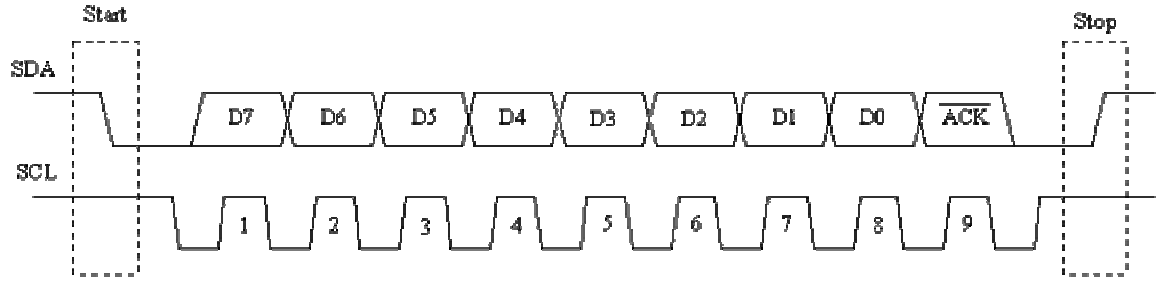
obr.3.6 – připojení na sběrnici

Obr.3.6 ukazuje princip propojení sběrnice. Neaktivní účastníci sběrnice mají vysokou impedanci, neustále však vyhodnocují signály na sběrnici. Je-li použit jen jeden master, vydává hodinový signál jen on. Data však může vysílat jak master, tak slave.

Protokol I²C rozeznává řadu přesně definovaných situací, které každému účastníkovi umožňují rozeznat začátek a konec přenosu a také své možné adresování:

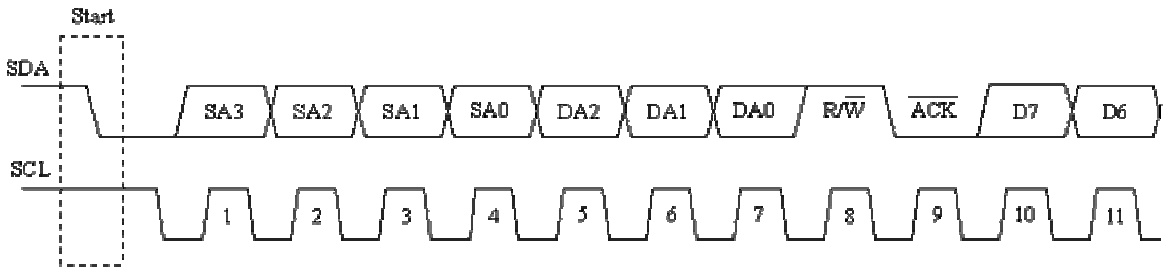
- Klidový stav: SDA i SCL jsou na vysoké úrovni (HIGH) a tím neaktivní.
- Podmínka startu: SDA je masterem stažena na nízkou úroveň, zatímco SCL zůstává na úrovni HIGH.
- Podmínka stopu: SDA přejde z LOW na HIGH, SCL zůstává na úrovni HIGH.
- Přenos dat: Příslušný vysílač přivede na datovou linku SDA osm datových bitů, které jsou hodinovými impulsy na lince SCL vysílanými masterem posouvány dále. Přenos začíná bitem s nejvyšší váhou.
- Potvrzení(acknowledge): Příslušný přijímač potvrzuje příjem bytu nízkou

úrovní na SDA, dokud master nevyšle devátý hodinový impuls na SCL. Potvrzení současně znamená, že se má přijímat další byte. Požadované ukončení přenosu se musí ohlásit neexistencí potvrzení. Vlastního ukončení přenosu se dosahuje podmínkou stopu.



obr.3.7 – Časování sběrnice

Přenos a potvrzování adres se provádí přesně stejně jako přenos dat. V nejjednodušším případě přenosu dat od mastera k podřízenému zařízení (slave), např. výstupnímu portu, probíhají následující děje: master vyrobí podmínku startu, v bitech 7 až 1 přenese adresu portu (součástky), v bitu 0 požadovaný směr přenosu dat, totiž 0 pro „zápis", 1 pro „čtení". Podřízené zařízení (slave) adresu potvrdí. Pak master vyšle datový byte, který rovněž bude potvrzen. Master nyní může spojení přerušit zasláním podmínky stopu (obr.3.7), nebo může témuž zařízení slave posílat další byty (obr.3.8)



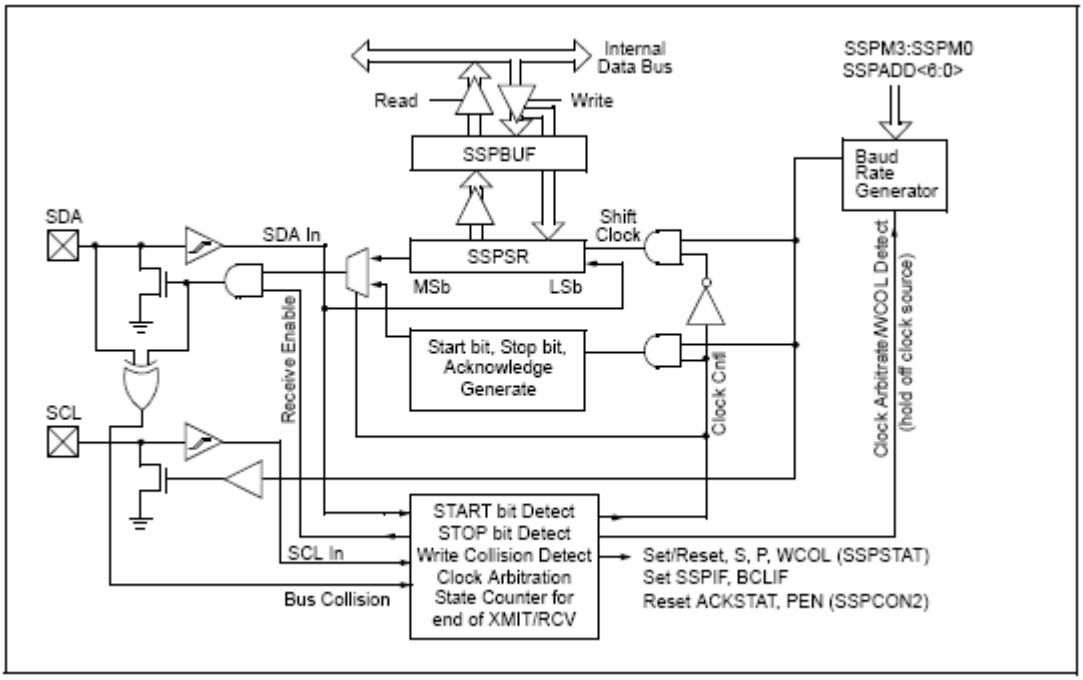
obr.3.8 – Časování sběrnice

Mají-li se číst data od zařízení slave, musí se adresa přenést s nastaveným bitem přenosu R/W. Master vždy vydá osm hodinových impulsů a dostane osm datových bitů. Potvrdí-li příjem vysláním devátého hodinového impulsu, může přijímat další byty. Přenos je nakonec masterem ukončen vynecháním potvrzení a podmínkou stopu. Každá součástka I^2C má stanovenou svoji adresu, která je zčásti pro daný typ specificky stanovená (SA0...SA3), zčásti proměnná (DA0...DA2). Při třech vyvedených adresových linkách může být na jedné sběrnici I^2C až osm součástek téhož typu. Maximální hodinový kmitočet pro sběrnici I^2C udává výrobce součástky.

3.4.5 Popis komunikace mikroprocesoru se seriovou I²C pamětí

Jak již bylo zmíněno jednou z periferií jednočipového mikroprocesoru PIC16F877A je Master Synchronous Serial Port (MSSP) module, který obsahuje mimo jiné i Inter-Integrate circuit (I^2C) programovatelného jako Master, Multi-Master, nebo Slave.

V aplikaci „Jednoduchý programovatelný logický automat" bude mikroprocesor nastaven do Master módu a každé další zařízení (seriová paměť) jako Slave. Veškerá nastavení se v mikroprocesoru provádějí ve čtyřech registrech SSPCON, SSPCON2, SSPADD, SSPSTAT vysílaná a přijímaná data přes registr SSPBUF. Blokové schéma je zřejmé podle obr.3.9.



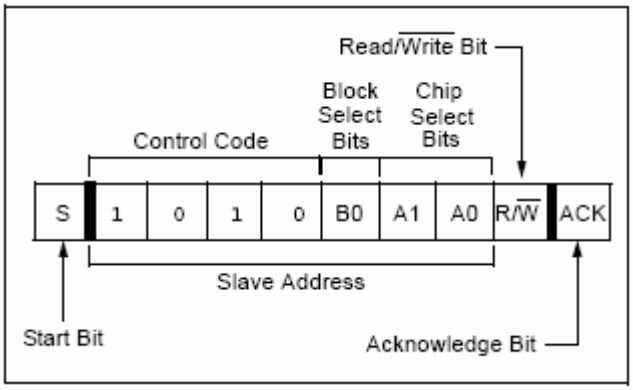
obr.3.9 – Blokové schéma I^2C v mikroprocesoru PIC16

3.4.6 Zápis dat do sériové I²C™ paměti 24FC1025

1. Procesor generuje START sekvenci nastavením bitu SEN v registru SSPCON2.

2. Čeká se na dokončení startovací sekvence, kterou ohlásí příznak SSPIF.

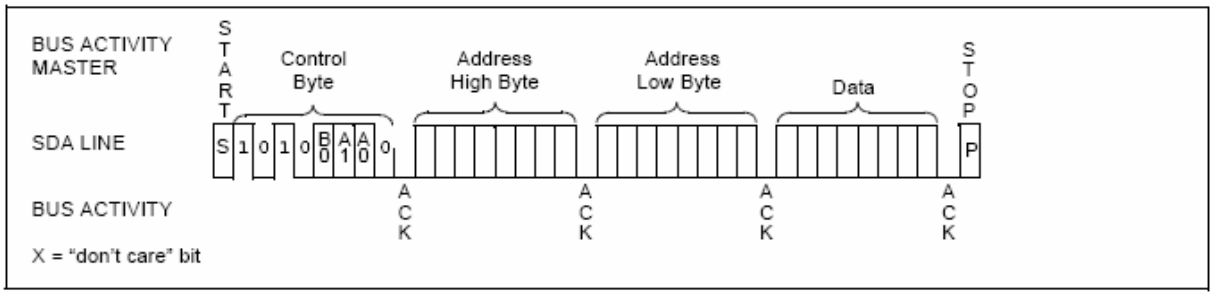
3. Program naplní registr SSPBUF prvním bytem (control byte, W = 1), který se bude po sběrnici vysílat hned po dokončení zápisu do registru začne automaticky probíhat vysílání. V případě paměti 24FC1025 bude kontrolní byte vypadat podle obr.3.10.



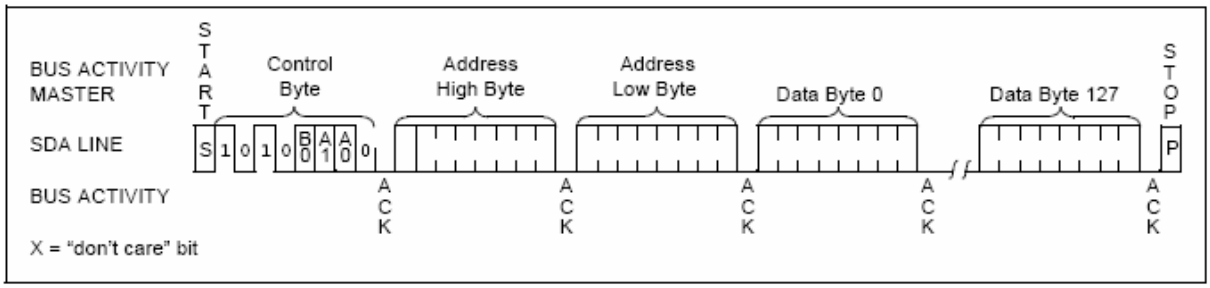
obr.3.10 – Control byte

4. MSSP přijímá ACK od sériové paměti, čímž dává paměť najevo, že zápis proběhl úspěšně, MSSP nastavuje příznak SSPIF.
5. Následuje zápis horních osmi z šestnácti bitů adresy paměti. Po zápisu generuje paměť ACK.
6. Probíhá zápis spodních osmi bitů adresy paměti. Paměť po zápisu generuje ACK.
7. V tuto chvíli je definována adresa paměti EEPROM, kam se bude zapisovat. Nyní může následovat bytové, nebo stánkové (128B) zapisování dat do paměti EEPROM.
8. Pro dokončení zápisu se po obdržení posledního ACK generuje STOP sekvence nastavením bitu PEN registru SSPCON2.

Zápis dat do paměti EEPROM je patrný z obr.3.11 (bytový zápis) a obr.3.11 (stránkový zápis).



obr.3.11 – Bytový zápis do paměti



obr.3.11 – Stránkový zápis do paměti

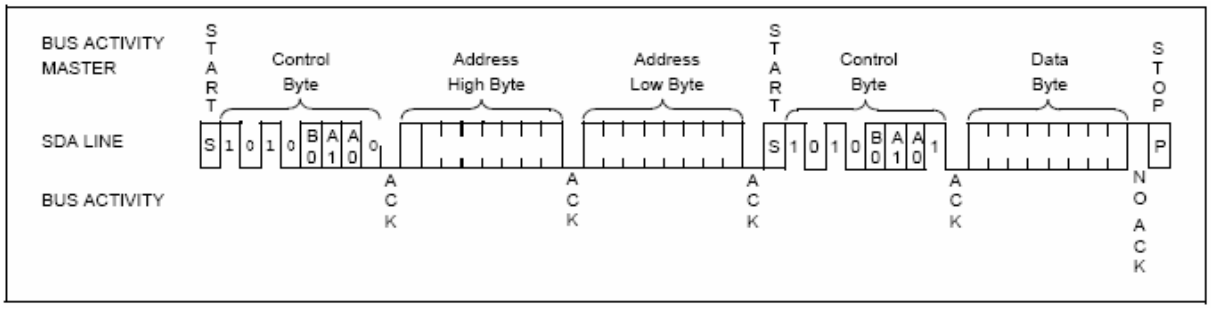
3.4.7 Čtení dat seriové I²C™ paměti 24FC1025

Při čtení dat z paměti EEPROM se obvykle jako první zapíše adresa, od které se budou data číst. Jako další následuje samotné čtení dat.

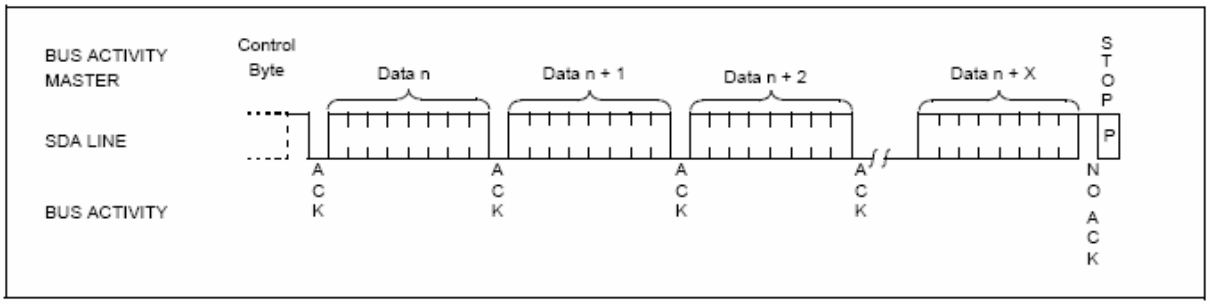
1. MSSP generuje START sekvenci, čeká na dokončení sekvence.
2. SSPBUF je naplněn Control bytem s nastaveným bitem W, čeká se na ACK.
3. SSPBUF je naplněn horními osmi bity paměti EEPROM, čeká se na ACK.
4. SSPBUF je naplněn spodními osmi bity paměti EEPROM, čeká se na ACK.
5. Generuje se STOP sekvence.
6. MSSP generuje START sekvenci,čeká na dokončení.
7. SSPBUF je naplněn Control bytem s nastaveným bitem R, čeká se na ACK.
8. Po SDA jsou přijímána data do SSPBUF, čeká se na dokončení příjmu.
9. MSSP nyní může generovat potvrzovací signál ACK, to znamená, že bude

následovat čtení dalšího bytu z adresy n + 1. MSSP, ale ACK generovat nemusí. Znamená to, že se bude přenos ukončovat a čeká se na započetí STOP sekvence.

Čtení z paměti EEPROM může probíhat opět bytově (obr.3.12), nebo sekvenčně (obr.3.13), kdy se může číst za sebou všech 512Kbit od adresy 0000h po adresu FFFFh. Pro přístup do druhé poloviny paměti je třeba nastavit šestnáctý bit adresy (B0), který je součástí control bytu.

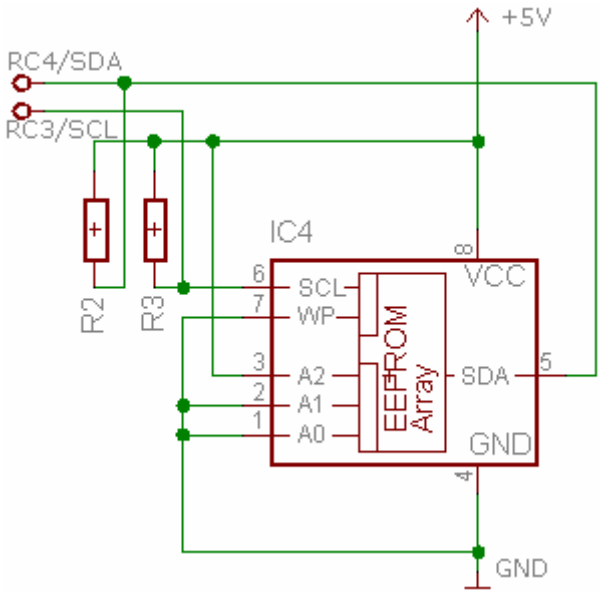


obr.3.12 – Bytové čtení z paměti



obr.3.13 – Stránkové čtení paměti

Obr.3.14 znázorňuje způsob hardwarového propojení paměti s jedno-čipovým mikroprocesorem PIC16F877A. Piny A0, A1 definují adresu paměti A2 je podle manuálu nutné připojit na +5V. Rezistory R2, R3 definují stav log. 1 na I^2C sběrnici (pull- up rezistory).



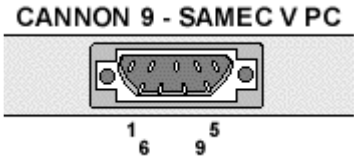
obr.3.14 – Typické zapojení paměti

3.5 Rozhranní RS-232

RS232 je rozhraní pro přenos informací vytvořené původně pro komunikaci dvou zařízení do vzdálenosti 20 m. Pro větší odolnost proti rušení je informace po propojovacích vodičích přenášena větším napětím, než je standardních 5 V. Přenos informací probíhá asynchronně, pomocí pevně nastavené přenosové rychlosti a synchronizace sestupnou hranou startovacího impulzu.

3.5.1 Zapojení konektorů RS - 232

Cannon 9			
PIN	NÁZEV	SMĚR	POPIS
1	CD	<--	Carrier Detect
2	RXD	<--	Receive Data
3	TXD	-->	Transmit Data
4	DTR	-->	Data Terminal Ready
5	GND	---	System Ground
6	DSR	<--	Data Set Ready
7	RTS	-->	Request to Send
8	CTS	<--	Clear to Send
9	RI	<--	Ring Indicator



obr.3.15 – počítačový konektor CANNON 9

3.5.2 Parametry RS-232

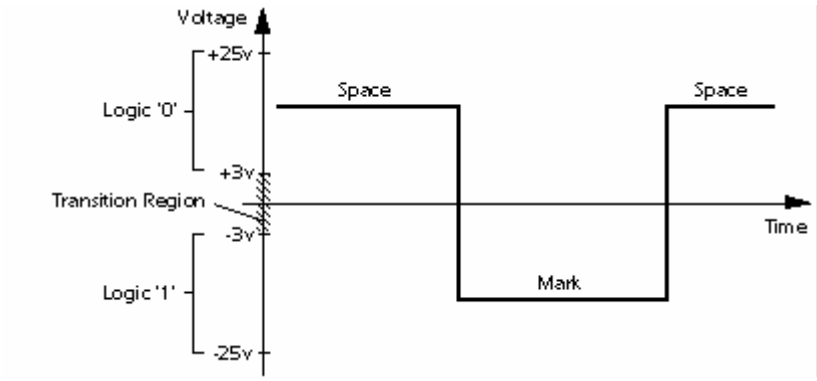
RS 232 používá dvě napěťové úrovně. Logickou 1 a 0. Log. 1 je někdy označována jako marking state, nebo také klidový stav, Log. 0 se nazývá space state.

Log. 1 je indikována zápornou úrovní, zatímco logická 0 je přenášena kladnou úrovní výstupních vodičů.

Povolené napěťové úrovně jsou uvedeny v následující tabulce. Nejběžněji se pro generování napětí používá napěťový zdvojovač z 5 V a invertor. Logické úrovně jsou potom přenášeny napětím +12 V pro log. 0 a –12 V pro log. 1.

Datové signály		
Úroveň	Vysílač	Přijímač
Log. L	+5 V to +15 V	+3 V to +25 V
Log. H	-5 V to -15 V	-3 V to -25 V
Nedefinovaný	-3 V to +3 V	

Řídící signály		
Signál	Driver	Terminátor
"Off"	-5 V to -15 V	-3 V to -25 V
"On"	5 V to 15 V	3 V to 25 V



obr.3.16 – Logické úrovně RS232

3.5.3 Parametry datového přenosu

Parita je nejjednodušší způsob, jak bez nároků na výpočetní výkon zabezpečit přenos dat. Ve vysílacím zařízení se sečte počet jedničkových bitů a doplní se paritním bitem tak, aby byla zachována předem dohodnutá podmínka sudého, nebo lichého počtu jedničkových bitů.

- SUDÁ PARITA – Počet jedničkových bitů + paritní bit = SUDÉ ČÍSLO
- LICHÁ PARITA – Počet jedničkových bitů + paritní bit = LICHÉ ČÍSLO
- SPACE PARITY – Tzv. nulová parita – paritní bit je vždy v log. 0, používá se například při komunikaci s 7bitového zařízení s 8bitovým, kdy paritní bit nahrazuje tvrdou log. 0 poslední bit v byte, tím je zachována kompatibilita s 8bitovým přenosem.
- MARK PARITY - Paritní bit je nastaven tvrdě na log. 1, při kompenzaci 7bitového provozu je třeba jej na přijímací straně nulovat, jinak není kompatibilní s ASCII.
- STOP BIT – Definuje ukončení rámce. Zároveň zajišťuje určitou prodlevu pro přijímač. Právě v době příjmu STOP bitu většina zařízení zpracovává přijatý BYTE.
- ZDVOJENÝ STOP BIT – Používá se u pomalejších zařízení pro doběh zpracování přijatého znaku. Jedná se o standard na 110 Bd

3.5.4 Synchronní přenos dat

Synchronní přenos informací znamená, že na nějakém vodiči nebo vodičích se nastaví určitá úroveň, která přenáší informaci a validita informace se potvrdí impulzem, nebo změnou úrovně synchronizačního signálu. Synchronizačním signálem se tedy informace kvantují.

3.5.5 Základní vlastnosti SYCHRONNÍHO přenosu

- Výhodné pro velké objemy dat, přenášené po více vodičích.
- Nutno jednoznačně určit, kdo vysílá synchronizační impulzy.
- Možno použít spojitě proměnnou rychlost přenosu, například podle poměru chybovosti.
- Nutnost synchronizačního vodiče „navíc" – v podstatě „nepřenáší žádnou informaci".
- Na straně zařízení nepotřebuje nijak složitou elektroniku.

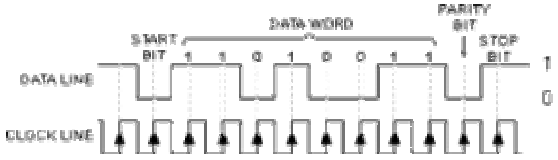
3.5.6 Asynchronní přenos dat

Asynchronní přenos dat přenáší data v určitých sekvencích. Data jsou přenášena přesně danou rychlostí a uvozena startovací sekvencí, na kterou se synchronizují všechna přijímací zařízení. Všechny strany obsahují vlastní přesný oscilátor, díky kterému odečítají data v přesně definovaných intervalech. Po ukončení sekvence je další příjem opět synchronizován startovní sekvencí.

3.5.7 Základní vlastnosti ASYNCHRONNÍHO přenosu

- Nevýhodné pro velké objemy dat, ale vhodné pro dlouhá vedení, na nichž by synchronizační vodič činil nezanedbatelné finanční náklady.
- Lze použít pro komunikaci mezi mnoha zařízeními.
- Nutno definovat jednoznačně přenosové rychlosti, změnu rychlosti je třeba ošetřit softwarovou sekvencí, která přiměje počítač změnit hardwarově přenosovou rychlost.
- Celkem složitá a drahá elektronika, nutno použít krystalové oscilátory.
- Až o 20% menší přenosová rychlost užitečných dat při stejné rychlosti komunikace, vzhledem k nutnosti startovacích a paritních bitů.

RS232 Používá asynchronní přenos informací. Každý přenesený byte konstantní rychlostí je proto třeba synchronizovat. K synchronizaci se používá sestupná hrana tzv. Start bitu. Za ní již následují posílaná data.



obr.3.17 – Asynchronní sériový přenos

obr.3.18 – Synchronizační hrana

3.5.8 Připojení RS 232 na TTL

Používají-li se v zařízení TTL nebo CMOS obvody, musí se jejich logická RS232 linka napěťově upravit před připojením do PC, protože napěťové úrovně RS-232 nejsou přímo slučitelné z žádnou logikou.

Pro toto upravení se standardně používaly obvody 1488 a 1489, které ale potřebovaly + 12V a –12V pro vytvoření výstupních úrovní. To bylo mimochodem jedním z důvodů, proč je v klasickém PC ze zdroje vyvedeno i –12V a –5V (dalším důvodem byla potřeba většího rozdílového napětí u historických dynamických pamětí pro zvýšení jejich rychlosti).

Průlom v tomto směru udělala firma MAXIM svým obvodem MAX232. Využila totiž svých znalostí ve vývoji spínaných nábojových měničů napětí a vyvinula obvod, který vystačil s +5 V a potřebné napětí si samostatně vyrobil pomocí 4 externích kondenzátorů. Obvod samozřejmě konvertuje log. 0 na +3,15V a log. 1 na –3,15V, jak je popsáno výše.

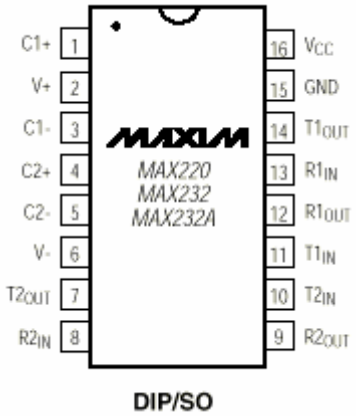
MAX232 se stal oblíbeným obvodem a dnes jeho obdobu najdete téměř ve všech komerčních zařízeních připojovaných k RS 232.

3.5.9 MAX 232 - Převodník RS-232/TTL MAX 232

Pro komunikaci zařízení s osobním počítačem po seriové lince bude připraven obvod převádějící napěťové úrovně RS-232 na TTL úrovně mikroprocesoru PIC16F877A.

MAX 232 je tedy převodník TTL na RS232. Obsahuje dvě dvojice oddělovačů konvertujících napěťové úrovně. Napětí pro RS232 se získává pomocí nábojové pumpy, a výstupní napětí proto značně závisí na kvalitě použitých kondenzátorů, která u elektrolytických kondenzátorů časem značně klesá. Napětí je možno získat na pinech 2 a 6 a použít pro další obvody.

Obvod funguje vždy na první zapojení. Maxim vyrábí i verze s minimální externí kapacitou – (MAX 232A – 0,1 µF) nebo verze pracující v rozsahu 7,5 – 13V (určeno pro bateriové aplikace) – MAX 201 a MAX 231. Specialitou firmy MAXIM jsou obvody MAX203 a MAX 233, které dokáží pracovat úplně bez potřeby vnějších kondenzátorů, a však pořízení je za podstatně vyšší cenu.



obr.3.19 – pouzdro DIP/SO

CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

obr.3.20 – použité koncenzátory

3.6 USART v mikroprocesoru PIC16F877A

Další významnou periferií jednočipového mikropočítače Microchip PIC16F877A je univerzální synchronní asynchronní receiver transmiter (USART). Jedná se o jeden ze dvou sériových vstupně výstupních modulů. USART může být konfigurován jako plně full-duplexní asynchronní systém, který je schopen komunikovat s různými zařízeními jako CRT terminály, osobní počítače atd.. Může být také konfigurován jako half-duplexní synchronní komunikační kanál při propojení se zařízeními, jako jsou A/D, D/A převodníky, sériové paměti EEPROM a podobně.

USART procesoru PIC16F877A může být konfigurován ve třech módech:

- Asynchronní (full duplexní)
- Synchronní – Master (half duplex)
- Synchronní – Slave (half duplex)

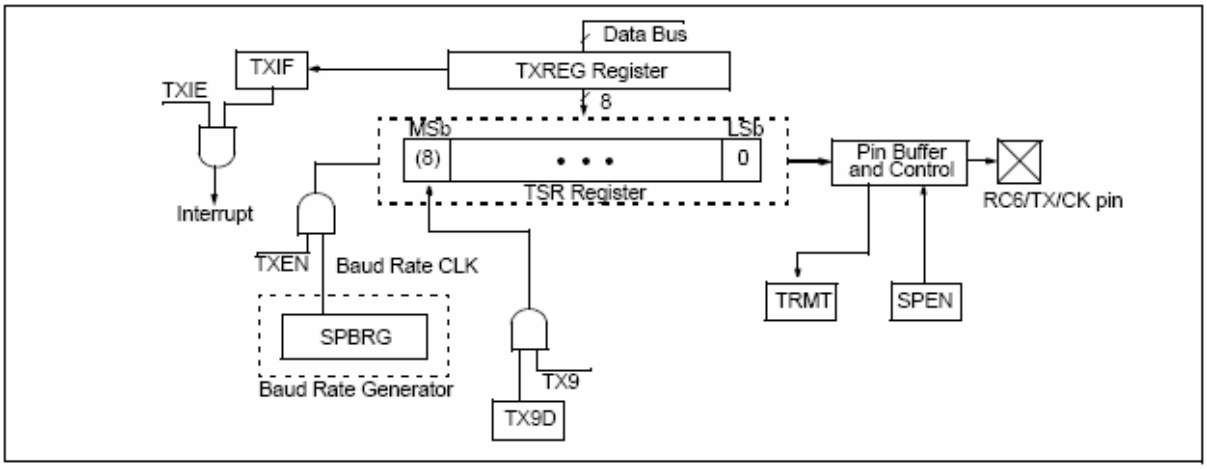
V aplikaci „Jednoduchý programovatelný logický automat" bude USART konfigurován jako Asynchronní s osmi-bitovou komunikací. Bude se tedy jednat o klasický formát s jedním start bitem osmi datovými bity a jedním stop bitem. Programově lze nastavit rychlost komunikace přes registry SPBRG a TXSTA. USART v mikroprocesoru PIC začíná vysílání i příjem od nejméně významného bitu LSb. Hardwarově zde není podporovaná parita, pokud je nutná, řeší se softwarově přes devátý bit přenosu.

3.6.1 USART jako asynchronní vysílač

Blokové schéma zapojení USARTu jako asynchronního vysílače je patrné z obr.3.21. Srdcem vysílače je sériový registr (TSR), který není přístupný v datové paměti. Tento registr přijímá data z pracovního registru TXREG, kam jsou data vkládána programem. V okamžiku příchodu STOP bitu (pokud probíhal předchozí přenos) se TXREG nahraje do TSR v jednom instrukčním cyklu. Po vyprázdnění TXREG se nastavuje příznak TXIF, který může vyvolat přerušení (pokud je povoleno). TXIF indikuje prázdný pracovní registr TXREG, jiný bit TRM (TXSTA<1>) indikuje prázdný registr TSR.

Vysílání je povoleno nastavením bitu TXEN v registru TXSTA<5>. Nový přenos nebude zahájen do té doby, než program nenaplní registr TXREG a nebude nastaven „baud rate generátor" BRG podle vzorce na obr.3.221.

Po nastavení BRG registru, povolení TXEN, nahrání dat do TXREG následně do TSR začíná vysílání na pinu RC6/TX/CK pinu mikroprocesoru. Při požadavku na devátý bit se povolí 9-bitová komunikace TX9 (TXSTA<6>) a devátý bit je nahrán do TX9D (TXSTA<0>). Devátý bit musí být nahrán před nahráním dat do TXREG.

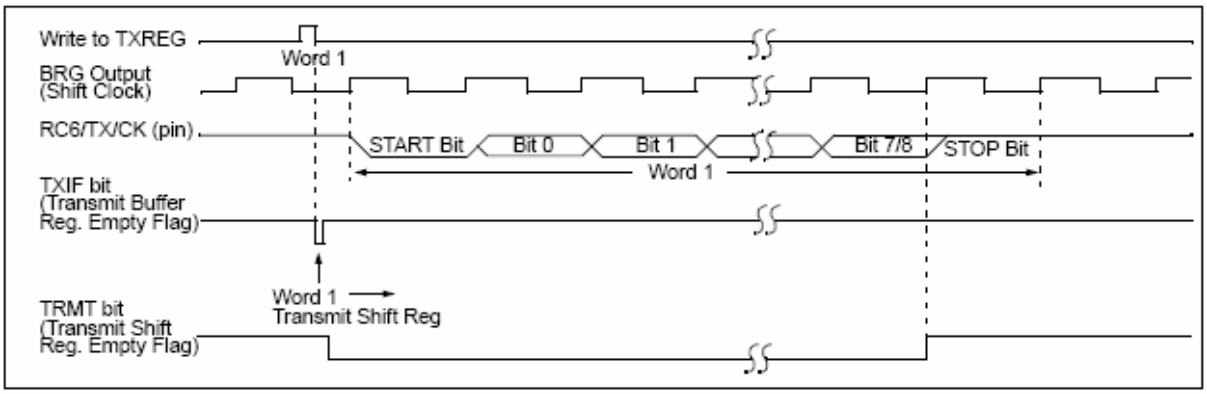


obr.3.21 – Vnitřní zapojení Transmitteru

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = Fosc/(64(X+1))	Baud Rate = Fosc/(16(X+1))
1	(Synchronous) Baud Rate = Fosc/(4(X+1))	N/A

X = value in SPBRG (0 to 255)

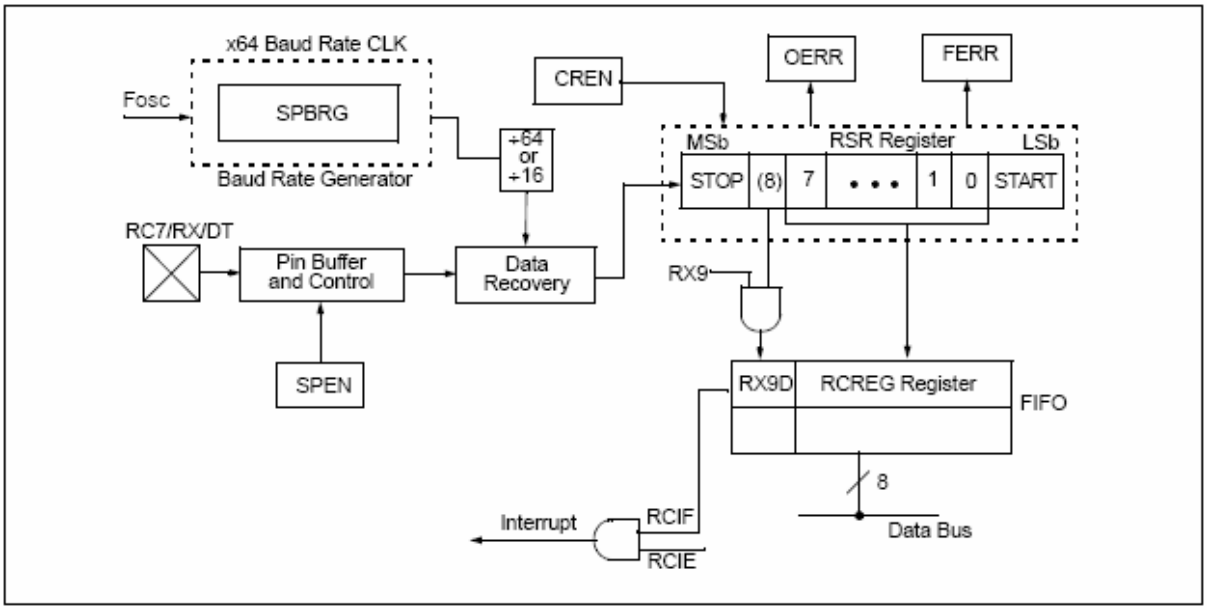
obr.3.22 – nastavení baud rate (časování)



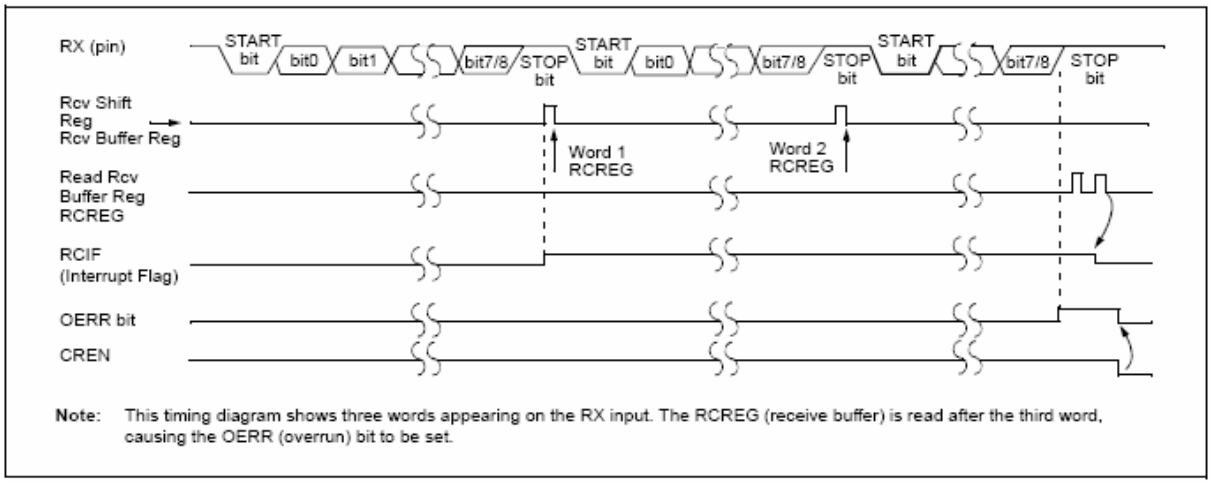
obr.3.23 – časování sériového vysílání

3.6.2 USART jako asynchronní přijímač

Blokové schéma zapojení asynchronního přijímače mikroprocesoru PIC16F877A je zřejmé z obr.3.23. Data jsou přijímána na pinu procesoru RC7/RX/DT a jsou dopravována přes „data recovery blok" do přijímacího registru RSR, podobně jako je tomu u vysílání. Po příchodu STOP bitu je RSR registr transportován do RCREG (pokud je prázdný). Když je aktuální přenos kompletní, nastavuje se příznak RCIF (PIR1<5>), který může vyvolat přerušení, pokud je povoleno. RCREG se vyprázdní hned po jeho přečtení. Jedná se o dvojitý „buffer registr" koncipovaný jako FIFO, který dokáže pojmout dva přenášené byty za sebou, třetí může být připraven v RSR. Pokud je při STOP bitu registr RCREG stále plný, generuje se příznak overrun OERR (RCSTA<1>) a přijatý byte v RSR je ztracen. V tomto případě je nutné číst RCREG dvakrát pro jeho úplné vyprázdnění. Při příznaku OERR je logika přijímače resetována a je nutné ji pro další přenos opět nastavit. FERR bit je tzv. frame error bit indikující chybu rámce, je nastaven při detekci STOP bitu v nule.



obr.3.23 – blokové schéma receiveru



obr.3.24 – časování sériového příjmu

3.7 Hardwarové vstupy/výstupy aplikace

Pro komunikaci s okolím bude „Jednoduchý programovatelný logický automat" vybaven čtyřmi hardwarovými digitálními vstupy a čtyřmi HW digitálními výstupy. Součástí jednočipového mikroprocesoru PIC16F877A jsou tři kompletní (8-bit) a dvě nekompletní vstupně výstupní brány (PORTy) A,B,C,D,E. Žádná z bran není jen čistě vstupně/výstupní, ale sdílí své vnější vývody (piny) s vstupy/výstupy některých periferií mikroprocesoru, jako jsou A/D převodník, komparátory, obě sériové linky, paralelní linka atd. I/O PORTy jsou mapovány do oblasti paměti dat, čtení a zápis se nijak neodlišuje od zápisu/čtení z kteréhokoliv jiného registru. Výběr, zda požadovaný pin v PORTu bude vstup nebo výstup, se provádí v přidruženém registru TRIS, případně v dalších registrech (ADCON1, TRISE) při sdílení PORTu s dalšími periferiemi. Při nastavení TRIS registru na samé jedničky nastavíme vstupně/výstupní PORT celý jako vstupní, nuly jsou výstupy. Příklad nastavení PORTu A na obr.3.25.

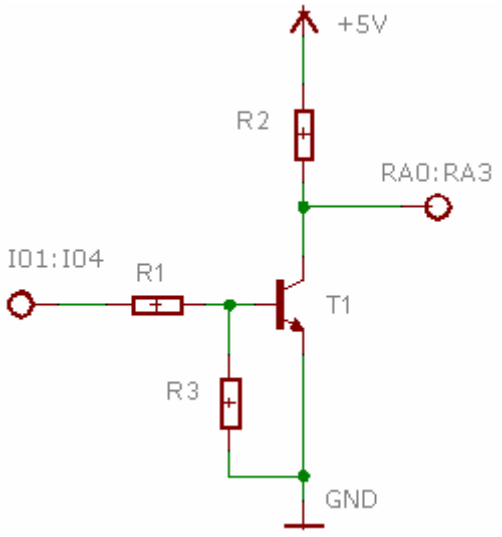
```
BCF     STATUS, RP0   ;
BCF     STATUS, RP1   ; Bank0
CLRF    PORTA         ; Initialize PORTA by
                      ; clearing output
                      ; data latches
BSF     STATUS, RP0   ; Select Bank 1
MOVLW   0x06          ; Configure all pins
MOVWF   ADCON1        ; as digital inputs
MOVLW   0xCF          ; Value used to
                      ; initialize data
                      ; direction
MOVWF   TRISA         ; Set RA<3:0> as inputs
                      ; RA<5:4> as outputs
                      ; TRISA<7:6>are always
                      ; read as '0'.
```

obr.3.25 – Příklad nastavení vstupně/výstupní brány

3.7.1 Zapojení vstupů

Vstupy aplikace, I01 až I04 jsou zapojené podle obr.3.26. Rezistor R1 na vstupu omezuje proud do báze I_{BE} tranzistoru T1, R3 plní funkci pull-down rezistoru při nezapojeném vstupu. Rezistor R2 zde omezuje proud I_{CE} při zavřeném stavu.

Toto zapojení invertuje vstup I01 až I04, znamená to tedy, že při logické „1" je úroveň na pinu RA0 až RA3 logická „0". Tento stav bude ošetřen softwarově.

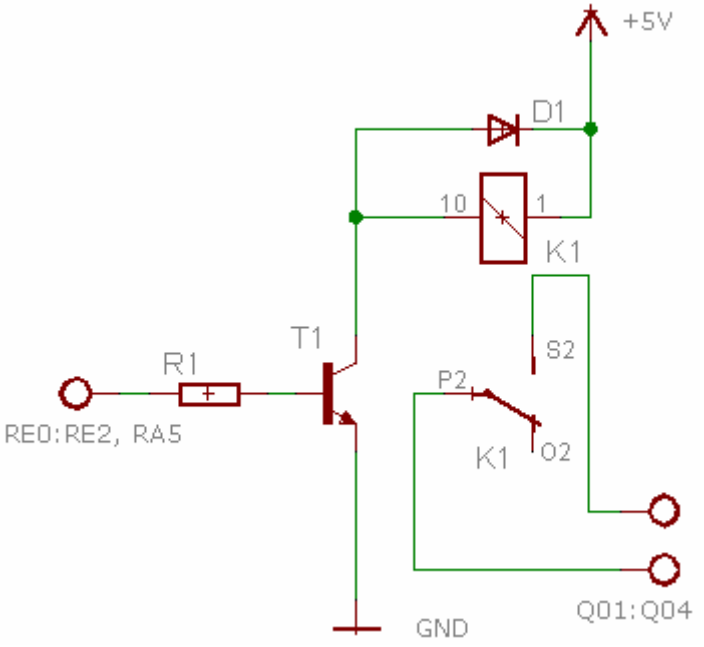


obr.3.26 – Hardwarové zapojení vstupu

3.7.1 Zapojení výstupů

Způsob zapojení hardwarových vstupů Q01 až Q04 bude v aplikaci „Jednoduchý programovatelný logický automat" podle obr.3.27. Proudové omezení celé výstupní brány je podle katalogového listu maximálně 200mA. Samotného pinu pak 25mA. Proud do báze tranzistoru je omezen rezistorem R1 tak, aby byl tranzistor v saturaci.

Při zavřeném tranzistoru začne protékat proud I_{CE} a sepne rele K1, které spojí kontakty P2 a S2, tím je uzavřen obvod vložený mezi svorky HW výstupu Q0x. Usměrňovací dioda D1 zde plní funkce přepěťové ochrany pro tranzistor T1. Přepětí vznikne při přerušení proudu do cívky relé K1 a mohlo by způsobit zničení přechodu tranzistoru.



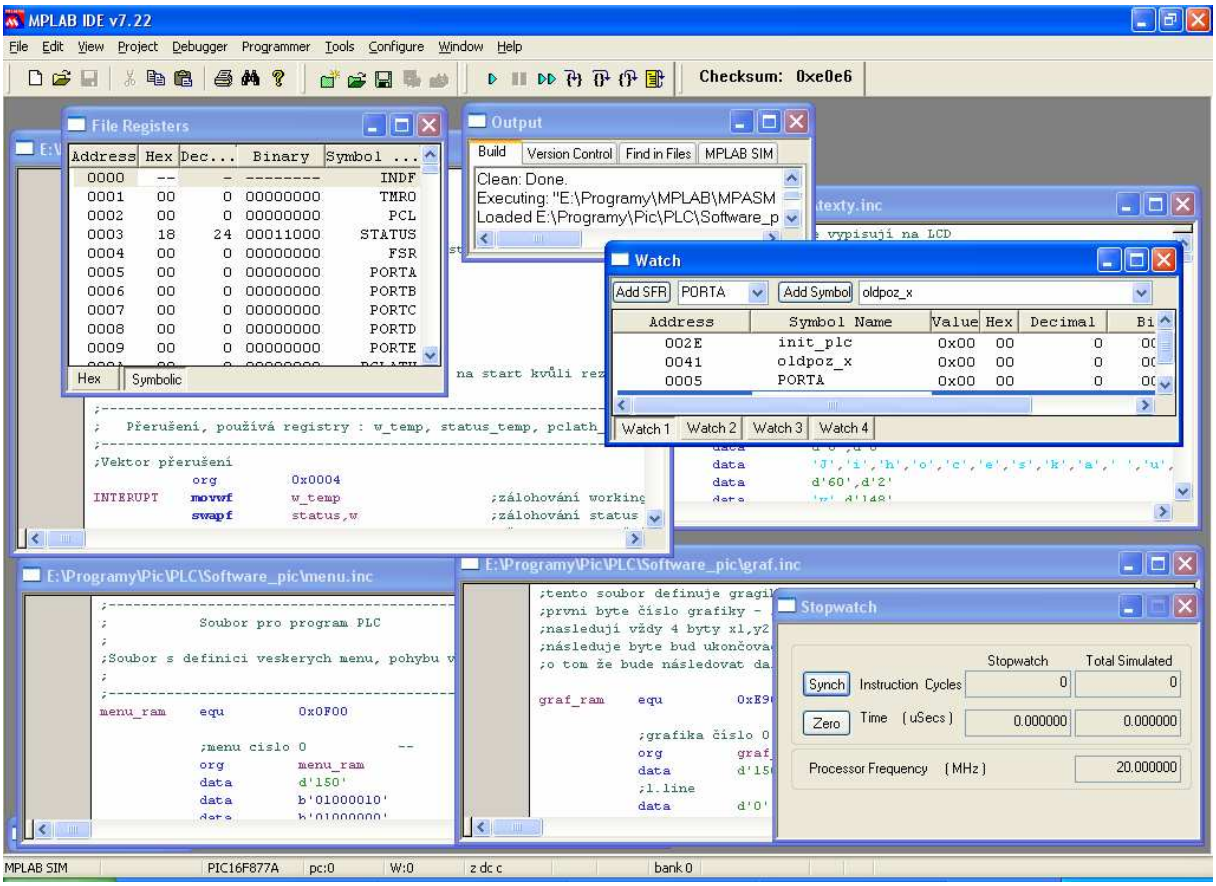
obr.3.27 – Hardwarové zapojení výstupu

4. Softwarové a Hardwarové vybavení

Při návrhu aplikace „Jednoduchý programovatelný logický automat" bylo použito několik softwarů a hardwarů. Stručný popis použitého SW/HW vybavení je shrnut v následujících odstavcích.

4.1 Vývojové prostředí Microchip MPLAB® IDE v7.22

Zdrojový kód programu pro jednočipový mikroprocesor PIC je samozřejmě možné psát v jakémkoliv textovém editoru. Také je ale dobré napsaný kód částečně (v rámci možností softwaru) vyzkoušet (simulovat), což poznámkový blok jen těžko umožní. Při tvorbě softwaru pro „Jednoduchý programovatelný logický automat" bylo použito vývojového prostředí MPLAB® IDE. Toto rozhranní od Microchipu dává programátorovi k dispozici řadu funkcí potřebných pro psaní a odladění vytvářené aplikace. V následujících několik řádcích bych se pokusil vysvětlit nejzákladnější funkce a práci s programem.



obr.4.1 – Vývojové prostředí MPLAB IDE

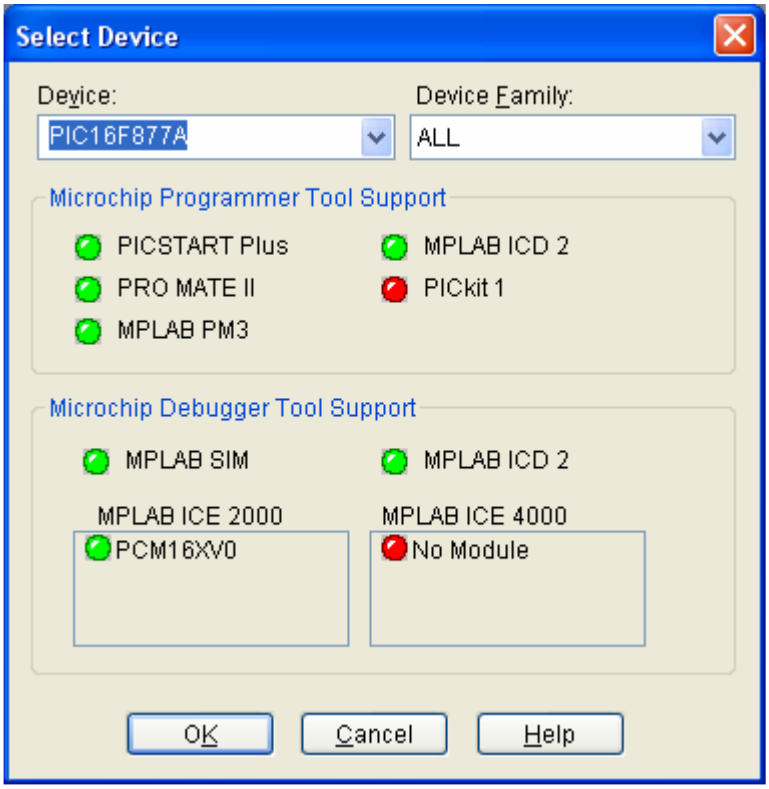
4.1.1 Popis rozhranní MPLAB® IDE

Program jako takový nevyžaduje žádné speciální znalosti. Měl by ho umět obsluhovat každý, kdo zvládne práci s operačním systémem Microsoft Windows.

Při začátku tvorby nového programu doporučuje výrobce pro větší efektivitu založit nový projekt pomocí *Project Wizard*, kde se dá definovat kupříkladu programovací jazyk, ve kterém chceme kód psát, definovat používané soubory, vybrat programovaný procesor atd.. Osobně jsem tuto možnost nevyužil z toho důvodu, že pracuji na vývoji pouze jedné aplikace a nepotřebuji tudíž při každém novém spuštění definovat jiné požadavky na prostředí (jazyk, výběr procesoru ...). Volím jednoduchou možnost pouze otevření souboru se zdrojovým kódem a práci s ním. I tak je ale na úplném začátku nutné vybrat některá nastavení programu, které si pak při zavření pamatuje.

- Výběr procesoru

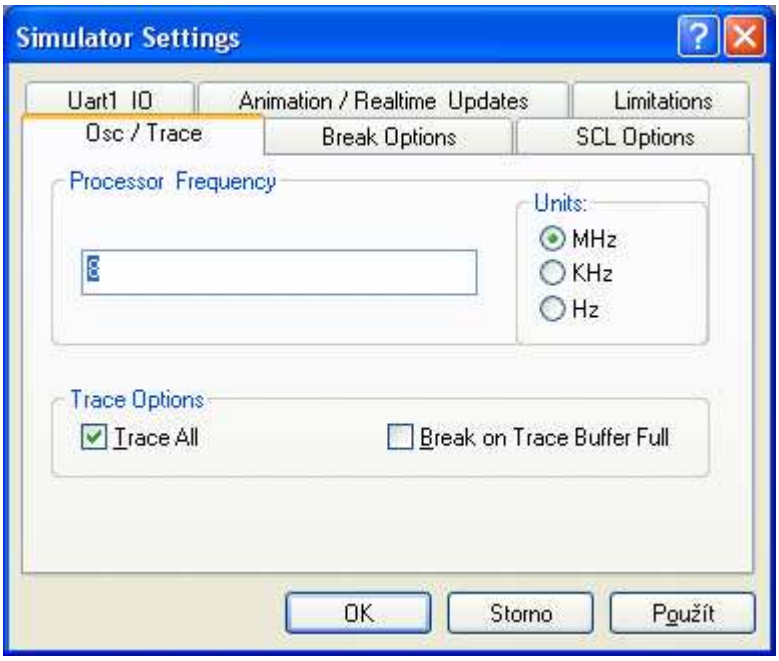
  Položka menu *Configure > Select Device* v položce Device je na výběr řada procesorů Microchip, v položce Device Family lze specifikovat výběr pouze s vybrané řady mikroprocesorů.
  Zeleně svítící body značí podporu programovacích a ladícího softwaru.



obr.4.3 – Výběr mikroprocesoru

- Zvolení pracovní frekvence procesoru (nutné před simulaci)

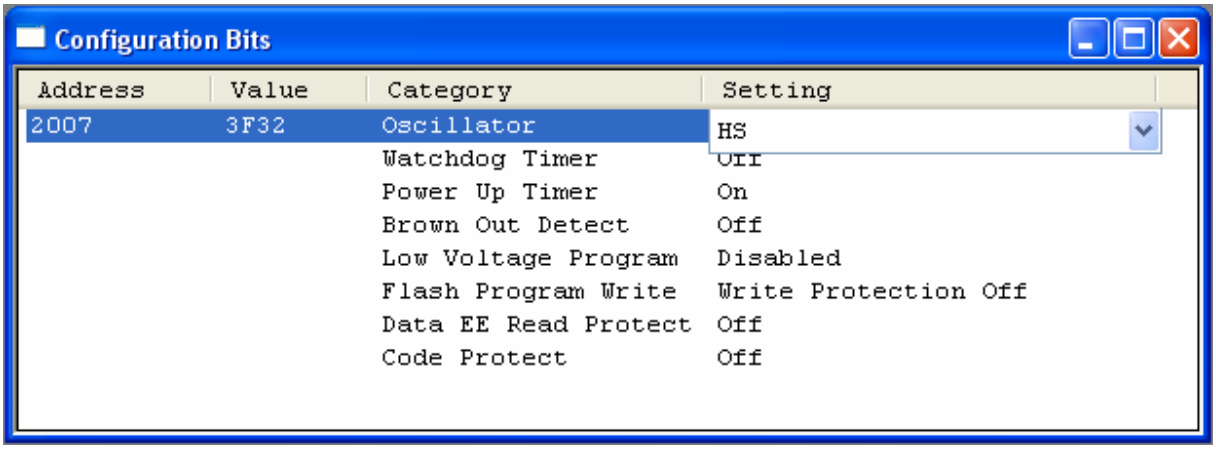
  Položka menu Debugger > Settings...



obr.4.4 – Nastavení pracovní frekvence

- Nastavení konfiguračního slova procesoru (z důvodů simulace, není nutné pokud je konfigurační slovo součástí zdrojového kódu).

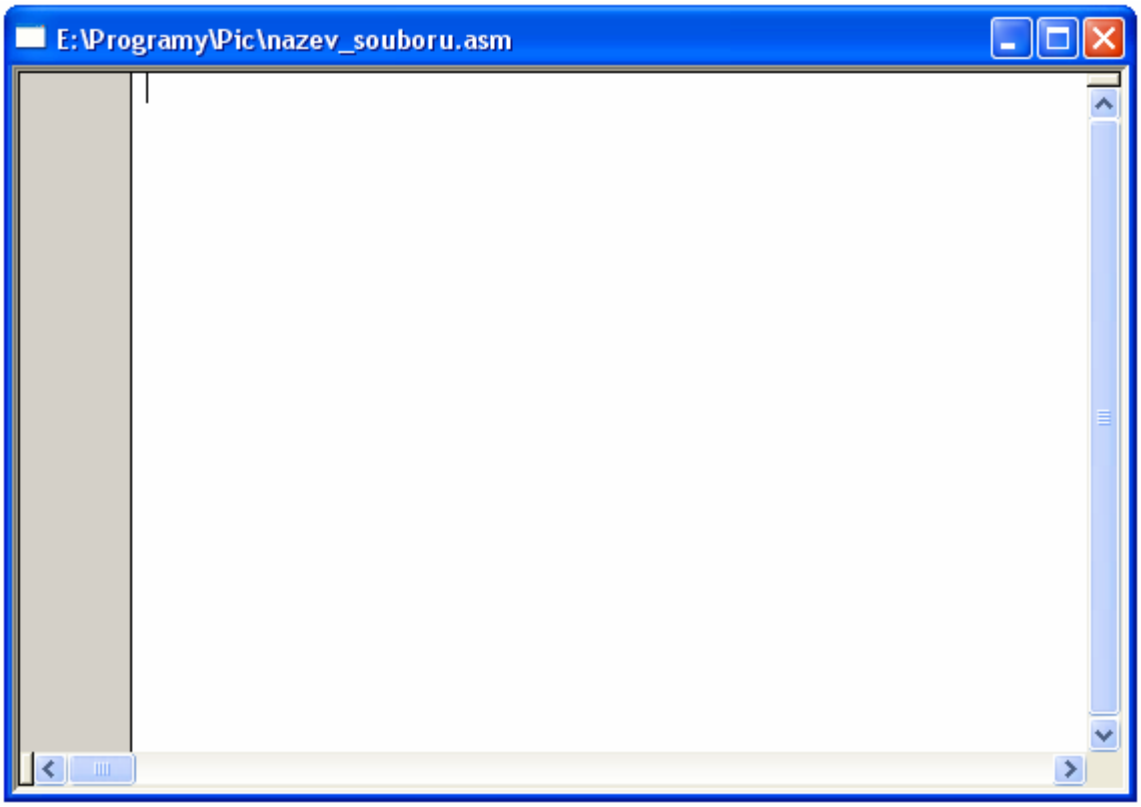
  Položka *menu Configure > Configuration Bits*



obr.4.5 - Nastavení konfiguračního slova

- Tvorba zdrojového kódu

  Položka menu File > New
  Položka menu File > Save As > nazev_souboru.asm

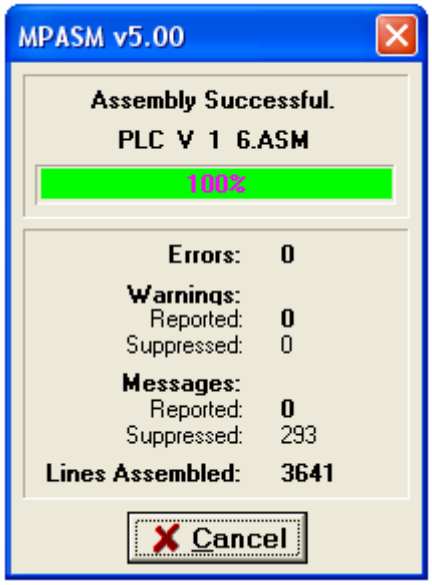


obr.4.6 – Editovací okno programu

Do prázdného okna můžeme začít psát zdrojový kód pro mikroprocesor. Po dokončení práce je nutné program zkompilovat do strojového kódu mikroprocesoru. K tomu nám poslouží program MPASM.

- Zkompilování vytvořeného *.asm zdrojového kódu

  Položka *Project > Quickbuild nazev_soburu.asm*
  Pokud vše proběhlo bez problémů objeví se okno obr.4.7.
  Zde si můžeme všimnout několika hlášení, kterými nás MPASM informuje o přeloženém programu. Message suppresed udává počet potlačených hlášení překladače, Lines Assembled celkový počet přeložených řádků programu.



obr.4.7 – Kompilování

Program MPASM generuje jakýsi log (output), ve kterém se můžeme dočíst o případných varováních, chybách, hlášení a podobně. Pokud byl program přeložen bez chyb MPASM, vytváří několik souborů:

*.err - výpis chybových hlášení
*.lst - listing programu
*.hex - strojový kód programu (nahrává se do procesoru)

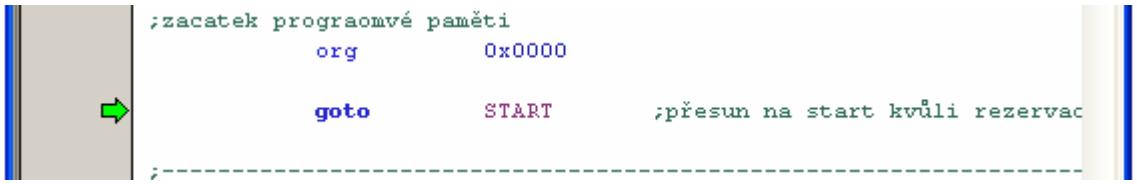
Po takto zkompilovaném programu je dobré provést alespoň částečně (v rámci možností vývojového prostředí) simulaci, kde zjistíme, zda program funguje jak by měl, zda pracuje se správnými registry, zda jsou skoky v programu na místa, kam mají být a podobně. Simulačním nástrojem v tomto vývojovém prostředí je MPLAB SIM.

- Simulace programu

  Položka menu *Debugger > Select Tool > 3 MPALB SIM*
  Do nástrojové lišty programu se přidá pole toolbar nástroje MPLAB SIM.

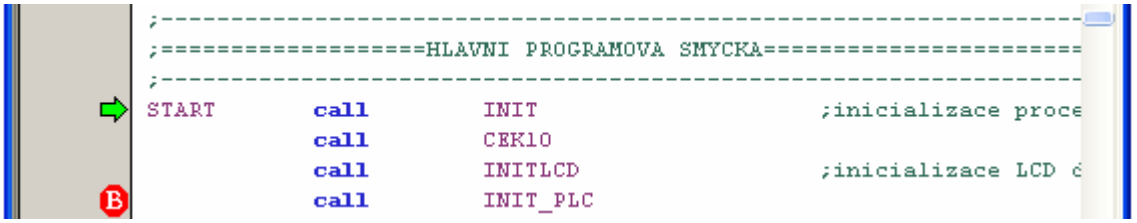
obr.4.8 – MPLAB SIM toolbar

Nyní se musí program znovu přeložit, po dokončení by se měla v okně editace programu objevit zelená šipka udávající informaci o pozici instrukce, která se vykoná po stisknutí běhu programu (krokování - step F7, běh programu – run F9).



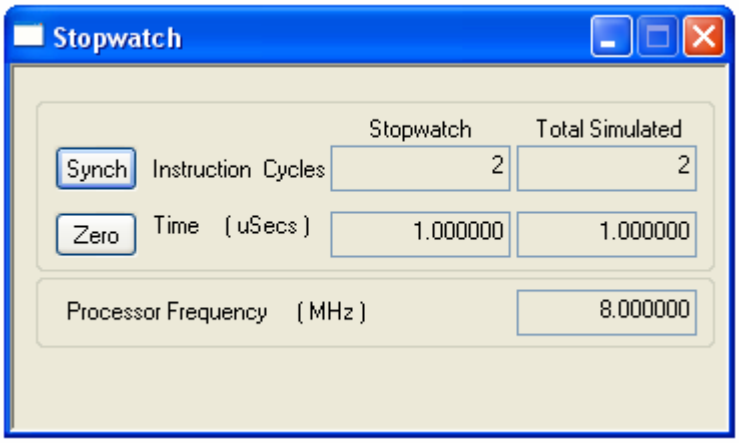
obr.4.9 – Šipka běhu programu

Do programu je možné vkládat Breakpointy (obr.4.10), na kterých se při simulaci běhu program zastaví.



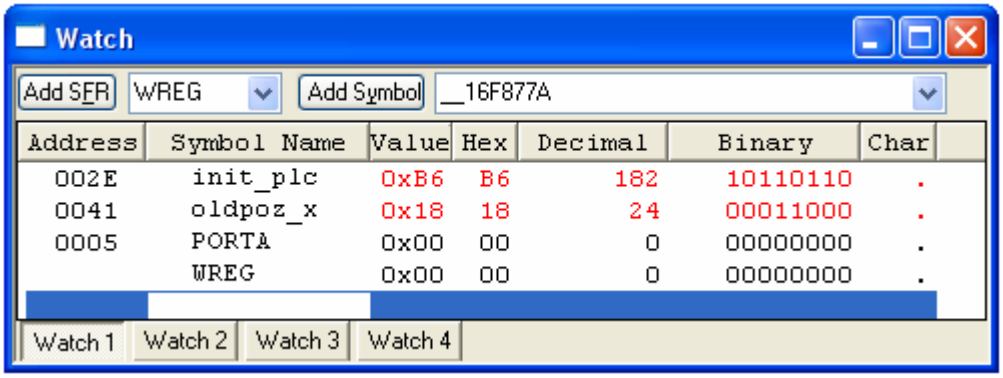
obr.4.10 - Breakpoint

Můžeme sledovat dobu běhu programu pomocí nástroje *Stopwatch*



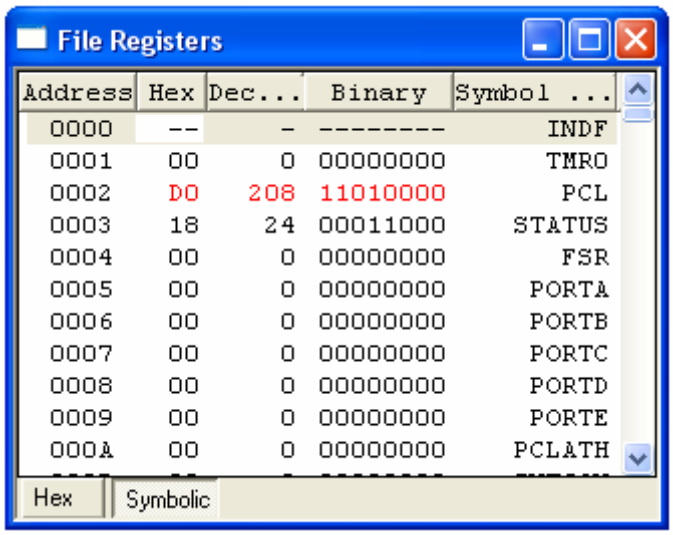
obr.4.11 – Nástroj Stopwatch

Kontrolování vybraných registrů pomocí nástroje *Watch*



obr.4.12 – Nástroj Watch

Sledovat chování v paměti RAM

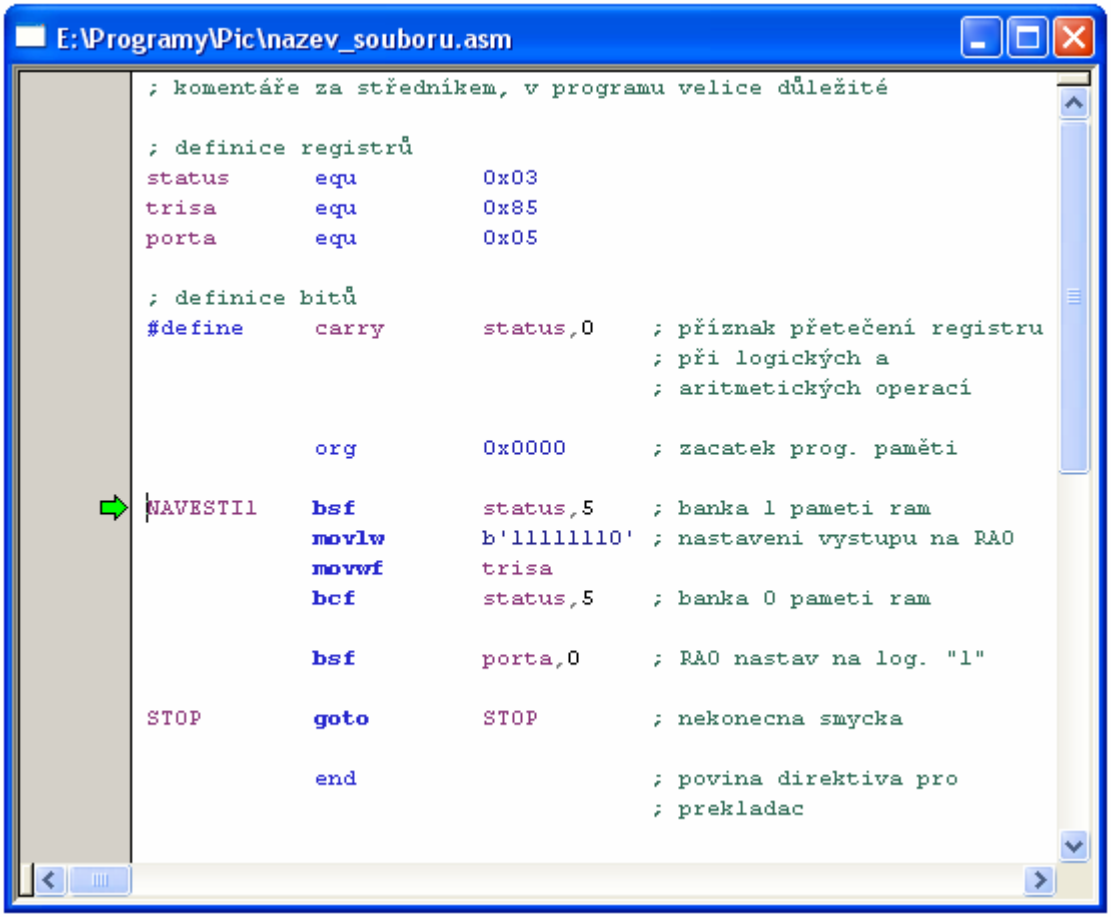


obr.4.13 – Sledování SFR

4.1.2 Způsob psaní a formátování zdrojového kódu

Při psaní zdrojového kódu programu musí dbát programátor, stejně jako u jiných programovacích jazyků, na předepsané formátování zdrojového kódu, se kterým pak pracuje překladač. Příklad zkušebního zdrojového kódu je na obr.4.14. Tento kód by v praxi jen nastavil vývod RA0 procesoru jako výstup, a dále ho nastavil na logickou úroveň „1".

- komentáře se píší za středníkem
- oddělovačem je Tabulátor (Tab)
- v prvním sloupci se píší návěští, které se adresují v instrukcích skoku
- ve druhém sloupci je instrukce pro procesor
- třetí sloupec je operand, se kterým se má pracovat (podle typu instrukce)
- za čárkou bývá zpravidla uložení výsledku nebo práce s konkrétním bitem registru (podle typu instrukce)
- v poslední řádce programu musí být povinná direktiva překladače end



obr.4.14 – Příklad způsobu psaní zdrojového kódu

4.2 Zápis strojového kódu do mikroprocesoru

Firmou vyrábějící velice povedené a spolehlivé víceúčelové programátory je ASIX s.r.o. Při práci na aplikaci „Jednoduchý programovatelný logický automat" byl využit programátor PICCOLO GRANDE od této firmy, který se již bohužel nevyrábí. Jeho nástupcem je PRESTO (obr.4.15). Jedná se o velmi rychlý a flexibilní USB programátor určený pro programování a testování velkého množství populárních obvodů (mikrokontroléry, sériové paměti, CPLD, FPGA a další).

Programátor PRESTO je určen pro programování a testování obvodů přímo v aplikaci. Mezi podporované součástky patří:

- Mikrokontroléry Microchip PIC - součástky se sériovým programováním (Flash, EPROM i OTP), tedy prakticky všechny PIC kromě některých zastaralých typů.

- Mikrokontroléry Atmel AVR - všechny součástky podporující programovací režim "SPI Low Voltage Serial Downloading", tj. například ATtiny12, AT90S8535 nebo ATmega128.

- Mikrokontroléry eCOG1 firmy Cyan Technology

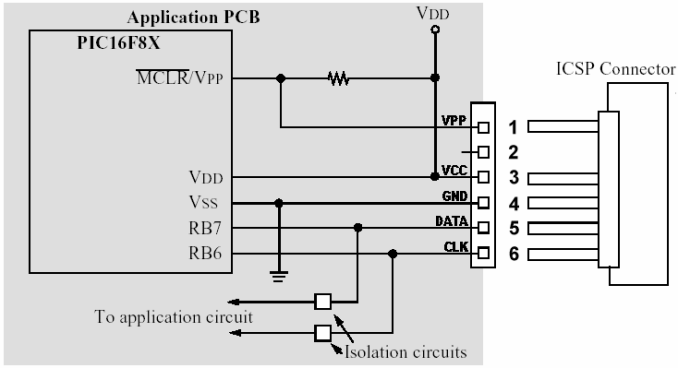
- Sériové EEPROM - rozhraní I2C (24LCxx), Microwire (93LCxx) a SPI (25Cxx)

- Součástky s rozhraním JTAG, jejichž vývojový software je schopen vytvořit soubor SVF nebo XSVF - jako jsou CPLD (např. Xilinx XC95xx a CoolRunner), konfigurační paměti pro FPGA (např. Xilinx XC18Vxx a XCFxxS), mikrokontroléry (např. ATmega128) a další.

Programátor je optimalizován na rychlost programování při současné minimální ceně. Je implementována nadproudová ochrana na Vpp a Vcc a přepěťová ochrana na Vcc. Programátor je napájen z USB, je schopen napájet aplikaci během programování nebo použít napájecí napětí z aplikace.

obr.4.15 – Programátor PRESTO

Způsob propojení aplikace a programátoru je zřejmý z obr.4.16, kdy se využívá tzv. ICSP (In-Circuit serial programing), neboli programování již osazeného mikroprocesoru přímo v aplikaci.



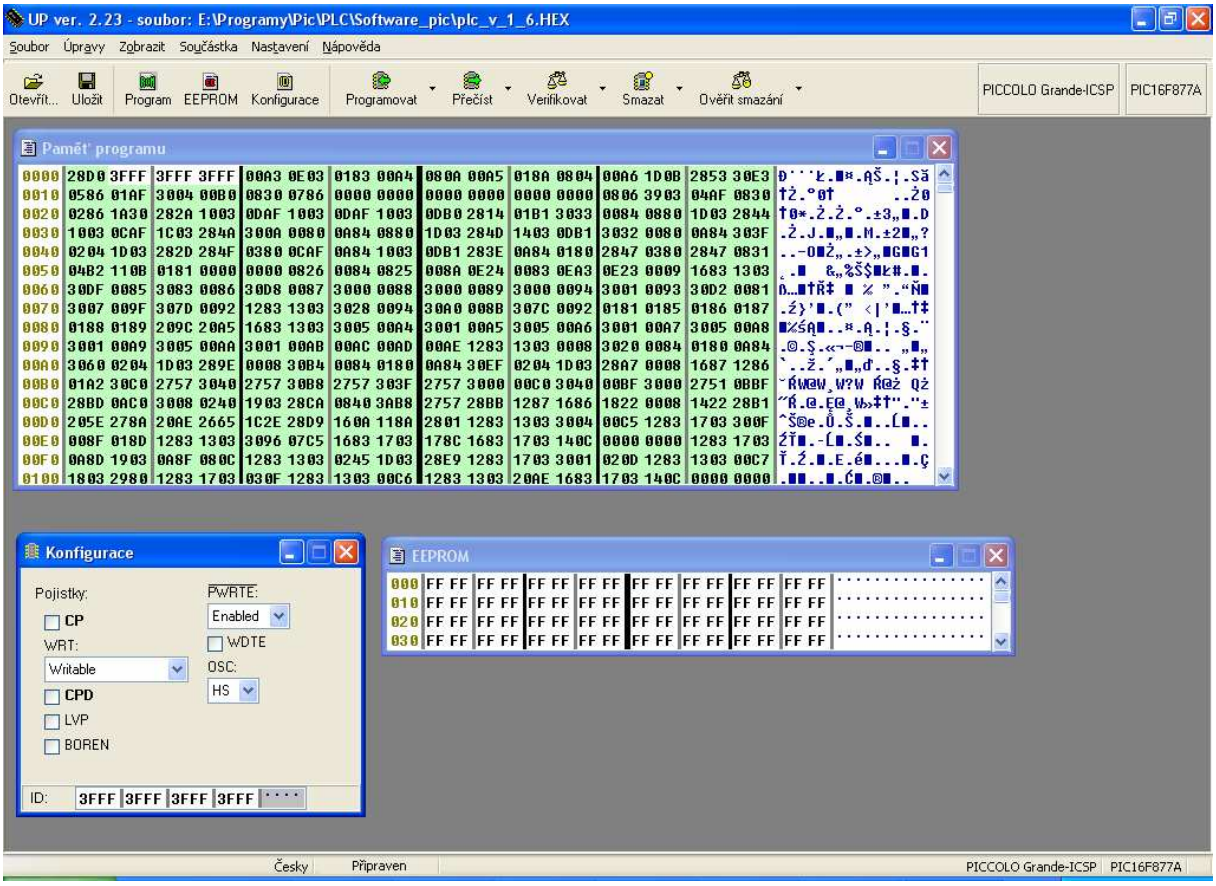
obr.4.16 – Způsob propojení aplikace a programátoru pomocí ICSP

4.2.1 Obslužný program UP

Součástí programátorů ASIX je příjemný ovládací software UP (obr.4.17) pro Windows 95/98/ME/NT/2K/XP. Uživatelské prostředí poskytuje snadné, intuitivní a rychlé ovládání všech funkcí programátoru. Obsahuje všechny standardní funkce (načtení souborů, programování, verifikace, vyčtení součástky, ...) i některé vylepšené.

- Pro všechny základní funkce lze využívat tzv. "horké klávesy" bez přeřazovačů (Ctrl, Alt...).
- Editace (včetně podpůrných funkcí) a zpětné uložení programovaných dat
- Barevné rozlišení dat podle původu (součástka, soubor, ruční editace...) a podle výsledku verifikace (nesouhlasící hodnoty jsou zvýrazněny)
- Nastavitelné parametry, umožňující rychlejší ovládání znalou obsluhou (úroveň hlášení, dotazy před provedením akcí, automatické zavírání informačních oken...)
- Uživatelské nastavení klávesových zkratek.
- Prostředí v maximální míře respektuje specifické vlastnosti PIC:
    - Formát zobrazovaných dat je optimalizován podle šířky instrukčního slova daného PIC
    - Možnost zobrazení a nastavení jednotlivých pojistek i přímá editace ve formátu konfiguračního slova
    - Je možné programovat i verifikovat zvlášť paměť programu, datovou paměť EEPROM a konfigurační slovo a ID.
    - Při použití vhodného programátoru (např. CAPR–PI) lze po ICSP naprogramování přímo aplikaci spustit (softwarové ovládání Stop/Run/Reset)
    - Inteligentní automatické generování sériových čísel s mnoha nastavitelnými možnostmi

- Režim hromadného programování podporuje obsluhu pomocí jediné klávesy a je doplněn o počítadla úspěšných i neúspěšných pokusů.
- Ukončení programování a jeho výsledek je volitelně doprovázeno nastavitelnou zvukovou signalizací.
- Software automaticky ukládá poslední nastavení - typ programované součástky, jméno souboru *.HEX (včetně celé cesty) a všechny potřebné parametry. Po znovuspuštění programu tedy lze pokračovat v práci tam, kde byla ukončena - bez nutnosti opětovného nastavování parametrů
- Optimální nastavení rychlosti komunikace pro daný typ počítače
- Nastavitelná doba náběhu napájecího napětí pro ICSP
- Adresa paralelního portu je libovolně nastavitelná. Kromě klasických portů LPT1 až 3 je tedy možno použít i nestandardní adresy v rozsahu 100h až 3FCh.



obr.4.17 - Rozhranní UP

4.3 Easily Applicable Graphical Layout Editor – EAGLE 4.16

Editor plošných spojů EAGLE je multiplatformní, snadno použitelný, uživatelsky přívětivý a výkonný nástroj pro návrh schémat a desek plošných spojů (DPS, PCB).

4.3.1 Vlastnosti programu EAGLE

Program se skládá ze tří hlavních modulů

- Editor spojů (obr.4.20)
- Editor schémat (obr.4.19)
- Autorouter

Tyto moduly jsou ovládány z jednoho uživatelského prostředí (obr.4.18). Proto není třeba konvertovat netlisty mezi schématy a deskami.

Vlastnosti programu (verze Professional)

- dopředná a zpětná anotace v reálném čase
- nápověda orientovaná podle obsahu
- žádná hardwarová ochrana programu!
- vícenásobná okna pro desku, schéma a knihovnu
- výkonný uživatelský jazyk
- integrovaný textový editor
- dostupný pro Windows® Linux® a MAC OS®

Editor spojů

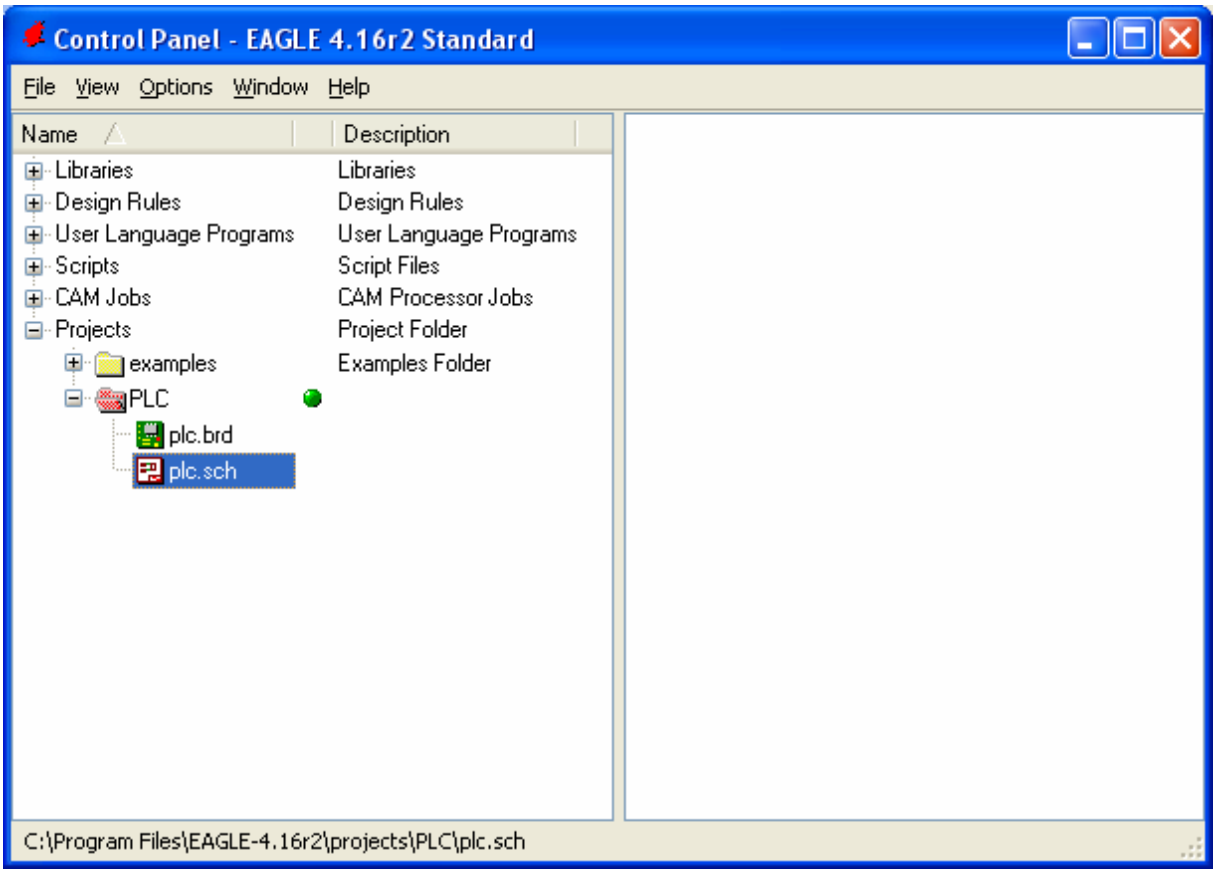
- největší rozměr výkresu 1.6 x 1.6m (64 x 64 inch)
- rozlišení 1/10.000 mm (0,1 mikronu)
- až 16 signálových vrstev
- klasické i SMD součástky
- dodává se s plnou sadou knihoven součástek
- snadné vytváření vlastních součástek v plně integrovaném editoru knihoven
- funkce vpřed/vzad pro LIBOVOLNÝ editační příkaz, do libovolné hloubky
- skriptové soubory pro dávkové zpracování příkazů
- pomědění ploch
- funkce kopírovat a vložit pro kopírování kompletních částí výkresu
- kontrola pravidel návrhu

Editor schémat

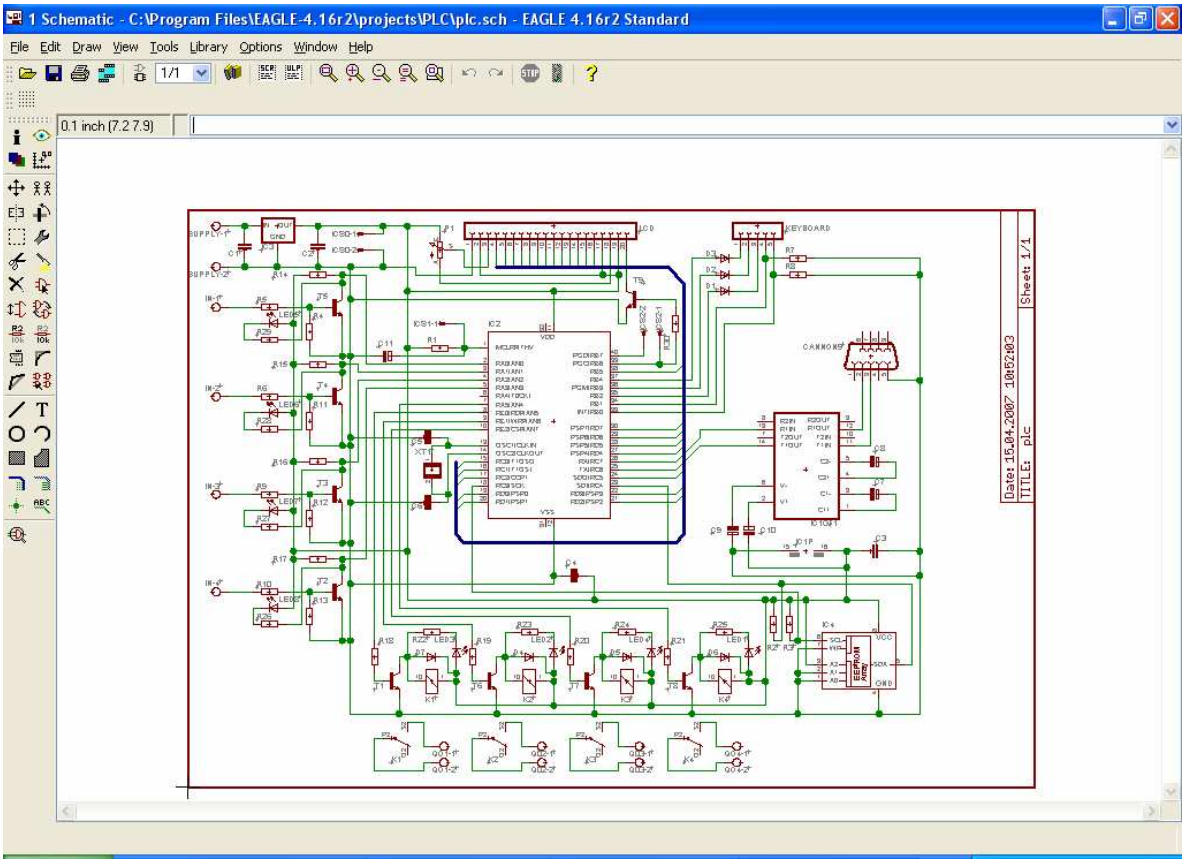
- až 99 listů jednoho schématu
- kontrola elektrických pravidel zapojení
- prohazování hradel a pinů
- vytvoření desky ze schématu jediným příkazem

Autorouter

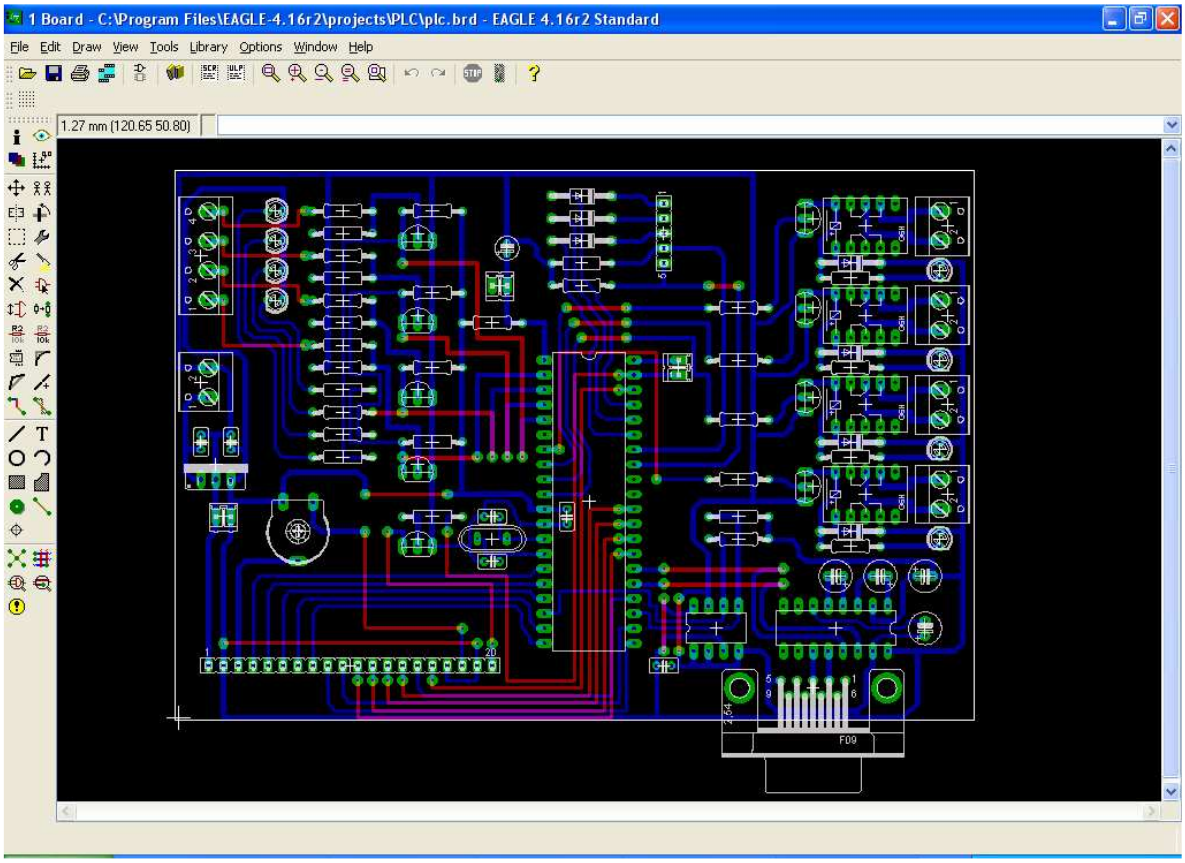
- ripup&retry router
- až 16 signálových vrstev
- strategie propojování nastavitelná uživatelem pomocí váhových faktorů
- CAM Processor
- Postscript
- perové plotry
- plotry Gerber
- soubory pro vrtačky Excellon a Sieb&Meyer



obr.4.18 – Uživatelské rozhranní EAGLE



obr.4.19 – Editor schémat



obr.4.20 – Editor desky plošných spojů

5. Firmware a jeho popis

Zde bych rád popsal, jakým způsobem (mně osobně) vyhovuje způsob tvorby firmwaru při vývoji aplikace s jednočipovým mikropočítačem.

Při návrhu aplikace s jednočipovým mikropočítačem se musí dobře zkombinovat jak hardwarové, tak softwarové řešení problému. Nejdůležitější literaturou při tvorbě firmwaru (i hardwaru) aplikace je Data Sheet (manuál) k mikroprocesoru a ke všem použitým součástem navrhovaného obvodu. Data Sheet obvykle přesně popisuje chování obvodu, způsob práce s ním, typické zapojení, jeho napěťové a proudové vlastnosti apod.

Na počátku je dobré vytvořit seznam (viz následující tabulka), jakým způsobem by mohli být v aplikaci zapojené vývody mikroprocesoru (zapojení se během práce může mírně pozměnit).

Zapojení mikroprocesoru PIC16F877A v aplikaci
Jednoduchý programovatelný logický automat

2	**RA0**	HW vstup I01
3	**RA1**	HW vstup I02
4	**RA2**	HW vstup I03
5	**RA3**	HW vstup I04
6	RA4	nc
7	**RA5**	HW výstup Q4

15	**RC0**	LCD - sig. R/S
16	**RC1**	LCD - sig. R/W
17	**RC2**	LCD - sig. E
18	**RC3/SCL**	I2C - serial clock
23	**RC4/SDA**	I2C - serial data
24	**RC5**	LCD - sig. CS2
25	**RC6/TX**	USART - TX
26	**RC7/RX**	USART - RX

8	**RE0**	HW výstup Q1
9	**RE1**	HW výstup Q2
10	**RE2**	HW výstup Q3

33	**RB0**	key. 2. sloupce
34	**RB1**	key. 1. sloupce
35	**RB2**	key. 1. řádek
36	**RB3**	key. 2. řádek
37	**RB4**	key. 3. řádek
38	**RB5**	LCD - sig. CS2
39	**RB6**	podsvícení LCD
40	RB7	nc

19	**RD0**	LCD - DB0
20	**RD1**	LCD - DB1
21	**RD2**	LCD - DB2
22	**RD3**	LCD - DB3
27	**RD4**	LCD - DB4
28	**RD5**	LCD - DB5
29	**RD6**	LCD - DB6
30	**RD7**	LCD - DB7

Před začátkem samotného programování je nesmírně důležité jednotlivé části programu popsat pomocí vývojových diagramů. Složitější části pak ještě pomocí osnovy vývojového diagramu. Takto vytvořené celky programů je dobré, pokud je to možné, naprogramovat a odladit.

Samotný program se skládá z několika částí.

- hlavní programová smyčka
- podprogramy volané hl. programovou smyčkou, nebo jiným podprogramem
- obslužný program přerušení

Všechny části programu obvykle potřebují rezervovat větší či menší část paměti pro ukládání svých výsledků, nebo sdílení dat s jinými podprogramy. Tato paměť se zpravidla nachází v paměti dat (RAM) mikroprocesoru. Velikost této paměti je u mikroprocesoru PIC16F877A 368B. Jakou část paměti pro jaký program rezervovat osobně provádím během psaní programu, kdy vycházejí najevo data potřebná k uložení. Organizace celé paměti je pak kompletní až po napsání celého programu.

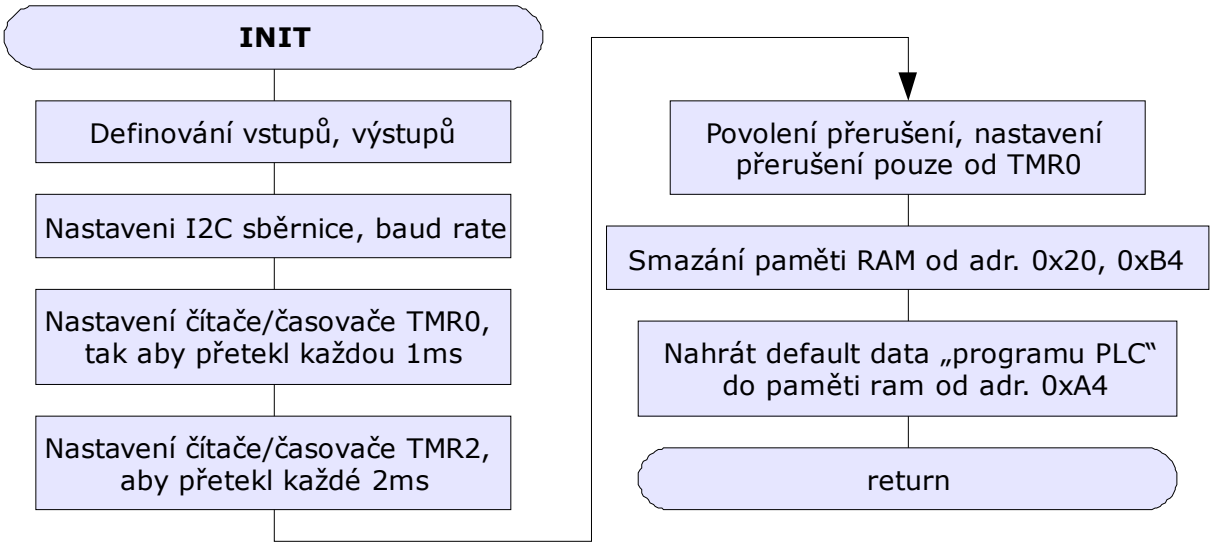
K ukládání většího množství dat pro práci programu lze využít v tomto mikroprocesoru ještě další uložiště - integrovaná paměť EEPROM (256B), a nevyužitá část programové paměti FLASH (8K x 14b slov). Práce s těmito paměťmi je ovšem trochu komplikovanější než s pamětí RAM. V aplikaci „Jednoduchý programovatelný logický automat" byla použita další externí paměť EEPROM o velikosti (128KB), ve které je v prvním bloku uložen vytvořený „program PLC", ve druhém pak nekompletní ASCII tabulka (125 znaků) pro LCD displej a několik speciálních znaků.

5.1 Základní funkční bloky programu

Při psaní programu jsem nezačal jako první hlavní programovou smyčkou, nýbrž podprogramy inicializace mikroprocesoru, čekací smyčky, obsluha I2C sběrnice, přerušení na přečtení klávesnice, práce s LCD displejem, výpis znaku ascii na LCD. Důvod byl ten, že je třeba nejprve napsat základní funkční bloky programu, než se začne pokračovat nástavbami pracujícími s těmito základními bloky.

5.1.1 Inicializace mikroprocesoru

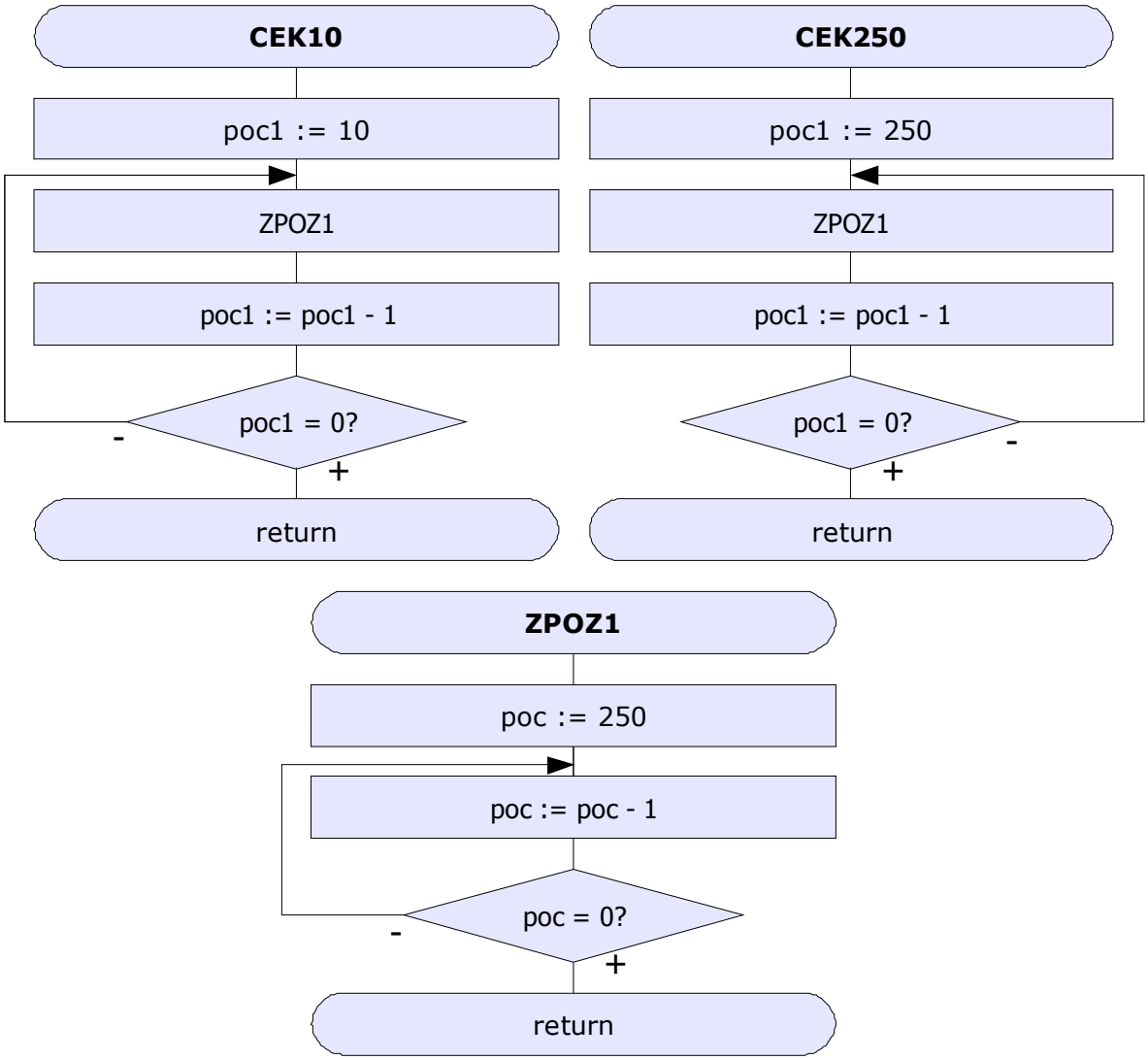
Následující osnova popisuje podprogram INIT. Jedná se o prvotní nastavení mikroprocesoru, definování vstupů, výstupů, zapnutí příslušných periferií, vymazání paměti RAM, nahrání některých dat do paměti RAM, a podobně.



5.1.2 Čekací smyčky

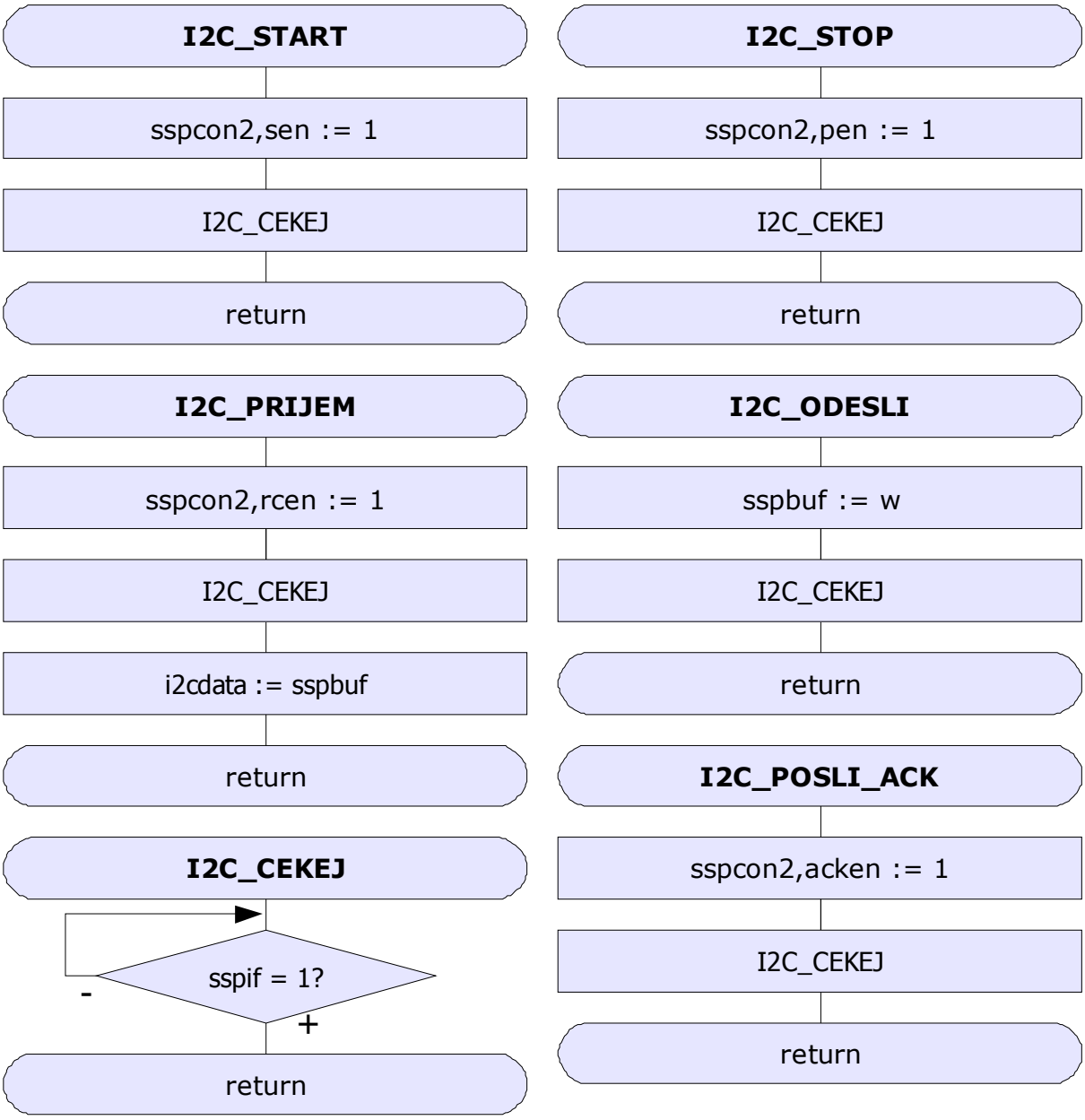
Jeden ze způsobů tvorby čekacích smyček je využití frekvence vykonávaného programu, kdy každá instrukce trvá jeden nebo víc instrukčních cyklů. Smyčky se mohou tvořit několika způsoby. V této aplikaci není nutné vytvářet přesné smyčky, proto jsem zvolil způsob jedné, trvající zhruba 500nS (ZPOZ1) a dalších, které tuto smyčku volají – CEK10 (výsledná doba 5ms), CEK250 (výsledná doba 125ms). Pracovní registry jsou:

poc (0x20) – pomocný čítací registr
poc1 (0x21) – pomocný čítací registr



5.1.3 Obsluha I^2C sběrnice

Podprogramy starající se o komunikaci po I^2C sběrnici. Pracovním registrem v podprogramu I2C_PRIJEM je i2c_data (0x2C). Před voláním podprogramu I2C_ODESLI je nutné do pracovního registru W vložit odesílané dato.

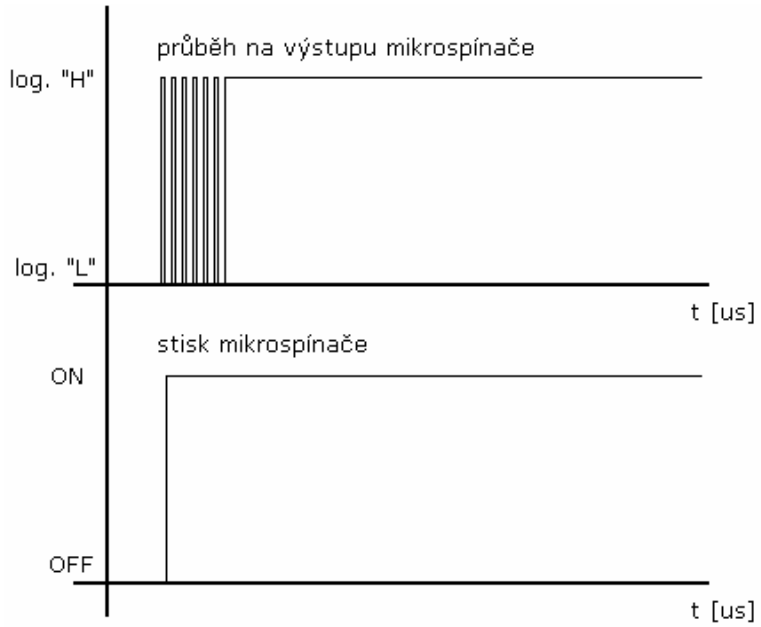


5.1.4 Obsluha maticové klávesnice 2x3 pomocí přerušení od TMR0

V INIT byl nastaven čítač/časovač TMR0 tak, aby přetekl každou 1ms, dále bylo povoleno, aby se při každém přetečení TMR0 vyvolalo přerušení mikroprocesoru. V tom případě je dokončena rozpracovaná instrukce a změněn programový čítač na adresu 0x0004, kde se nachází rutina obsluhy přerušení. Jak již zde bylo psáno při vstupu do přerušení, je velice důležité provést zálohu registrů W, STATUS, PCL a PCLATH. Jakým způsobem nejlépe zálohovat tyto registry, je popsáno v manuálu k mikroprocesoru. Po záloze následuje rozlišení, která periferie přerušení vyvolala a pokračuje se její obsluhou. Přerušení od TMR0 značí příznak TMR0IF, který směruje přerušení na obsluhu klávesnice.

5.1.5 Problém mikrospínačů v maticové klávesnici a jeho řešení

Kontakty v mikrospínačích po stisku několikrát odskočí „zakmitají" (viz ilustrativní obr.5.1), to je u mikroprocesoru o výkonu 2Mips relativní problém. Výsledkem takto „kmitajícího" tlačítka je vyhodnocení jako mnohonásobný stisk. To je ale nežádoucí, proto se dá problém řešit RS klopným obvodem, to znamená dalších několik zbytečných součástek navíc, nebo elegantněji ošetřit tento stav softwarově.



obr.5.1 – Ilustrativní znázornění zákmitů tlačítek

Nejdůležitějším výstupním registrem obsluhy klávesnice je KeyEvent (0x32). Tento registr je naplněn podle stisku příslušných spínačů podle obr.5.2.

KeyEvent (0x32)

bit7							bit0
x	x	TL6	TL5	TL4	TL3	TL2	TL1

obr.5.2 – Registr stavu klávesnice KeyEvent

Registr je nulován v hlavní programové smyčce po vykonání funkce příslušné klávesy. Obsluha klávesnice používá řadu pracovních registrů (16B) právě z důvodů ošetření „zákmitu" tlačítek.

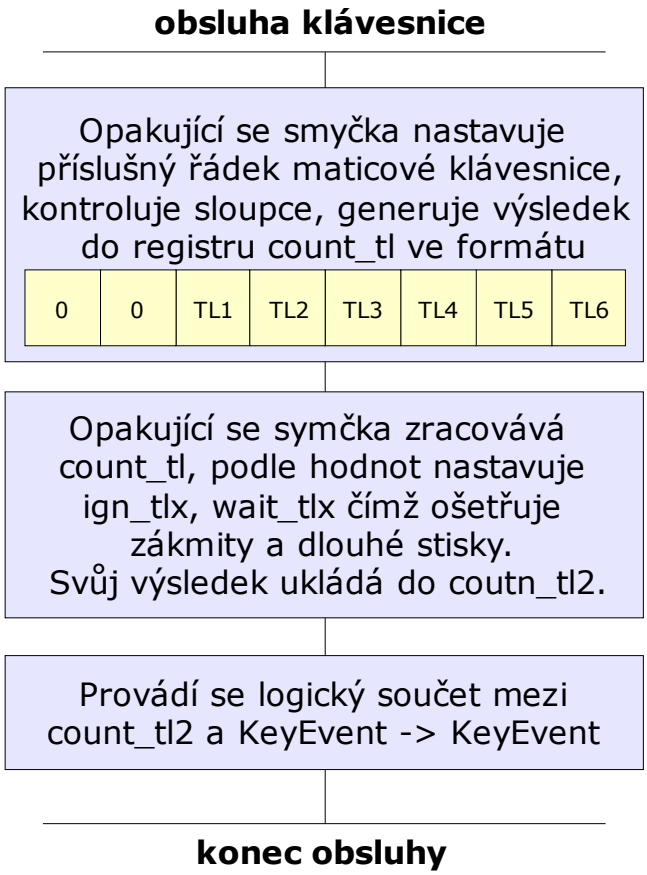
count_tl (0x2F) – stisknuté klávesy během smyčky
count_tl1 (0x30) – opakování smyčky
count_tl2 (0x31) – pomocná registru KeyEvent
KeyEvent (0x32) – údaj o stisknutých klávesách

12B od adresy 0x33
ign0 – ign5 - registry s údaji o délce trvání ignorace tlačítka po stisku v [ms]
wait_tl0 – wait tl5 - registry s údajem o délce čekání mezi opakováním vyhodnocení spínače jako stisknutého při nepřetržitém stisku.

5.1.6 Princip ošetření zákmitů tlačítek

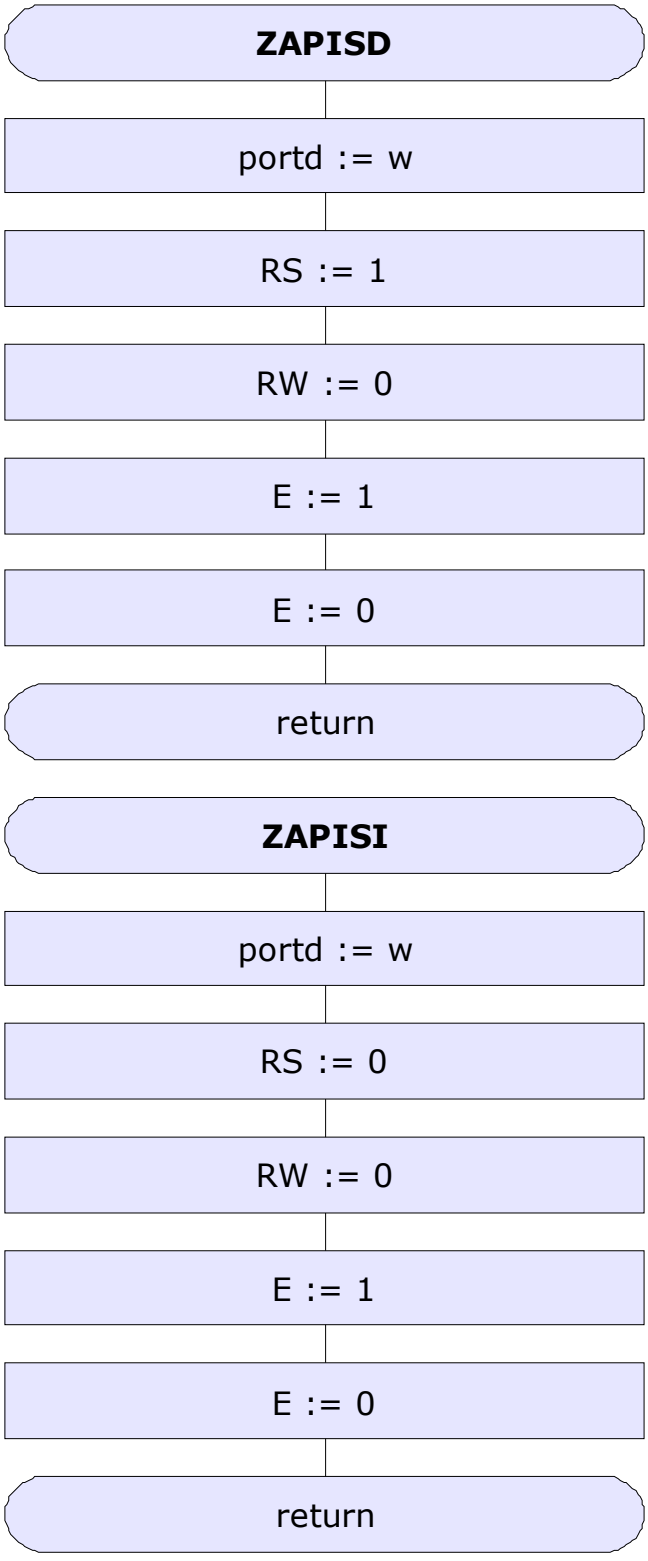
Princip ošetření „zákmitů tlačítek" spočívá v tom, že po vyhodnocení stisku se tlačítko 10ms netestuje, po uplynutí této doby je tlačítko znovu testované a pokud je stále stisknuté, zapracují registry wait_tl a klávesa se hned nevyhodnotí jako nový stisk, ale jako stále trvající starý stisk. Funkce tlačítek je pak stejná jako známe z běžného života – rychlé stisky znamenají rychlý pohyb a při dlouhých, nebo nepřetržitých stisknutých klávesách se kurzor přesouvá pomalu.

5.1.7 Osnova obsluhy maticové klávesnice



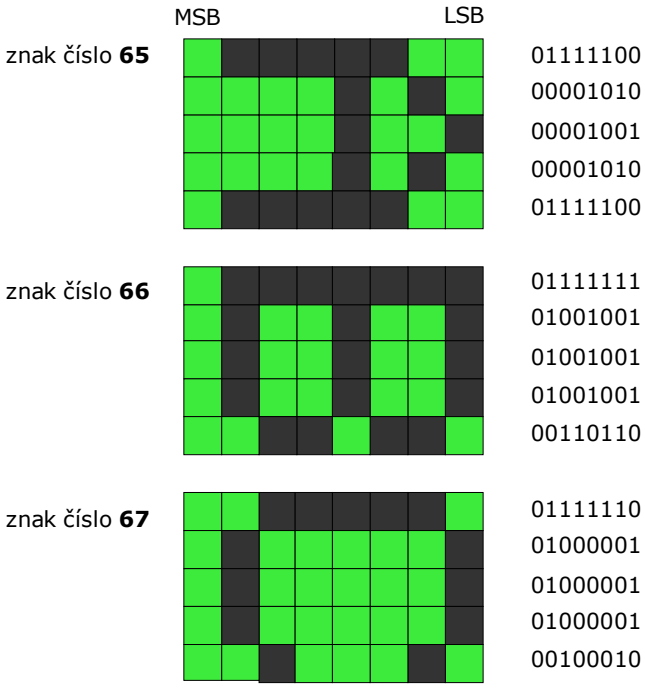
5.1.8 Práce s LCD displejem

Práce s grafickým LCD displejem Elatec je poměrně jednoduchá. Na obsluhu jsou napsané dva podprogramy ZAPISD, před jehož voláním je nutné požadované dato (které se bude zobrazovat) zapsat do W reg. Dalším je ZAPISI, který funguje obdobně s rozdílem zápisu nikoliv data, ale instrukce do displeje. Před zapisováním je důležité vybrat požadovaný řadič LCD řídícími signály CS1 a CS2.



5.1.9 Zápis znaku ASCII tabulky z EEPROM do LCD displeje

V externí paměti EEPROM, jak již bylo psáno je uložena nekompletní ASCII tabulka. Prvních 125 znaků je definováno v rozlišení 5x7 pixelů (obr.5.3).



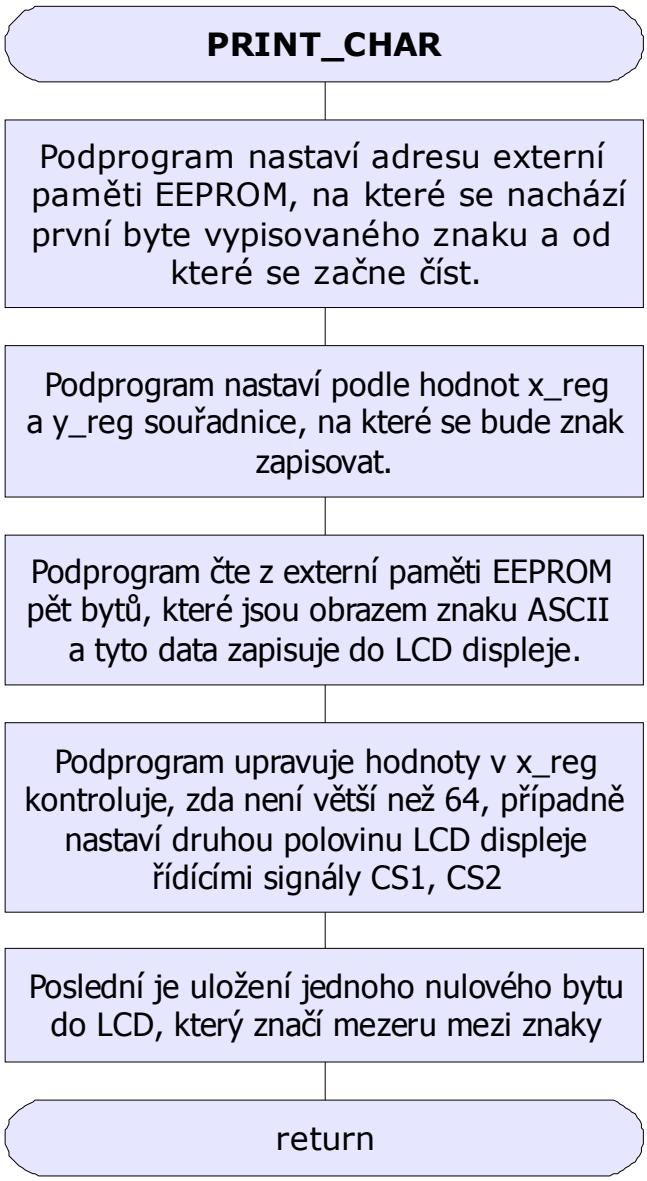
obr.5.3 – Způsob organizace dat znaků

O výpis znaku na LCD displej se stará podprogram PRINT_CHAR. Jeho vstupními proměnnými jsou:

chr_h (0x29) - horních 8 bitů adresy znaku v EEPROM
chr_l (0x2A) - spodních 8b adresy znaku
x_reg (0x27) - souřadnice x LCD displeje, na kterou se bude vypisovat znak
y_reg (0x28) - souřadnice y

Podle hodnot v registru x_reg podprogram vybere příslušný řadič (polovinu LCD) pomocí řídících signálů CS1, CS2, na který se budou posílat data. Během posílání podprogram kontroluje, zda x_reg není větší než 63 (konec první poloviny LCD), případně přepne na druhý řadič a zbytek znaku zapisuje na druhou polovinu LCD.

5.1.10 Osnova podprogramu PRINT_CHAR



5.1.11 Výpis souvislých textů na LCD displej

Řešení, jak vypisovat různé souvislé texty na LCD, nebylo jednoduché. Jako zajímavé řešení jsem považoval umístit veškeré texty pro přehlednost do jiného souboru texty.inc. Texty jsou pak definovány jako data v nevyužité paměti programu (FLASH). Data mají definovaný formát a přesnou posloupnost (viz obr.5.4).

- 0. byte pořadové číslo textu + 150 (počet možných textů 105)
- 1. byte souřadnice x LCD (x_reg) (127 sloupců)
- 2. byte souřadnice y LCD (y_reg) (8 řádek)
- 3. byte první znak vypisovaný na LCD
- může následovat několik znaků maximálně však 21 na řádek

- další byte může být odřádkování d'148', nebo konec textu d`149`
- v případě odřádkování je další byte údajem o x souřadnici LCD (x_reg)
- souřadnice y LCD (y_reg)
- minimálně 1 maximálně 21 znaků na řádek
- odřádkování nebo konec textu

Podprogram takto pracuje s daty příslušného textu, dokud nenajde konec a může se ukončit.

```
;text číslo 1
data        d'151'                  ;udava programu informaci o pozici textu cislo 1
data        d'12',d'1'
data        'B','a','k','a','l','a','r','s','k','a',' ','p','r','a','c','e',d'148'
data        d'12',d'3'
data        'P','R','O','G','R','A','M','O','V','A','T','E','L','N','Y',d'148'
data        d'12',d'4'
data        'L','O','G','I','C','K','Y',' ','A','U','T','O','M','A','T',d'148'
data        d'12',d'7'
data        'M','.','V','l','n','a',' ','M','V','T','p',' ','2','0','0','7',d'149'
```

obr.5.4 – Organizace dat v souboru texty.inc

5.1.12 Výpis textů podprogram PRINT_STR

Jak jsem již zmínil, podprogram PRINT_STR pracuje s přesně definovanými daty textů. Jeho povinným vstupem je pouze údaj čísla vypisovaného textu v rozmezí 0 až 105 v proměnné:

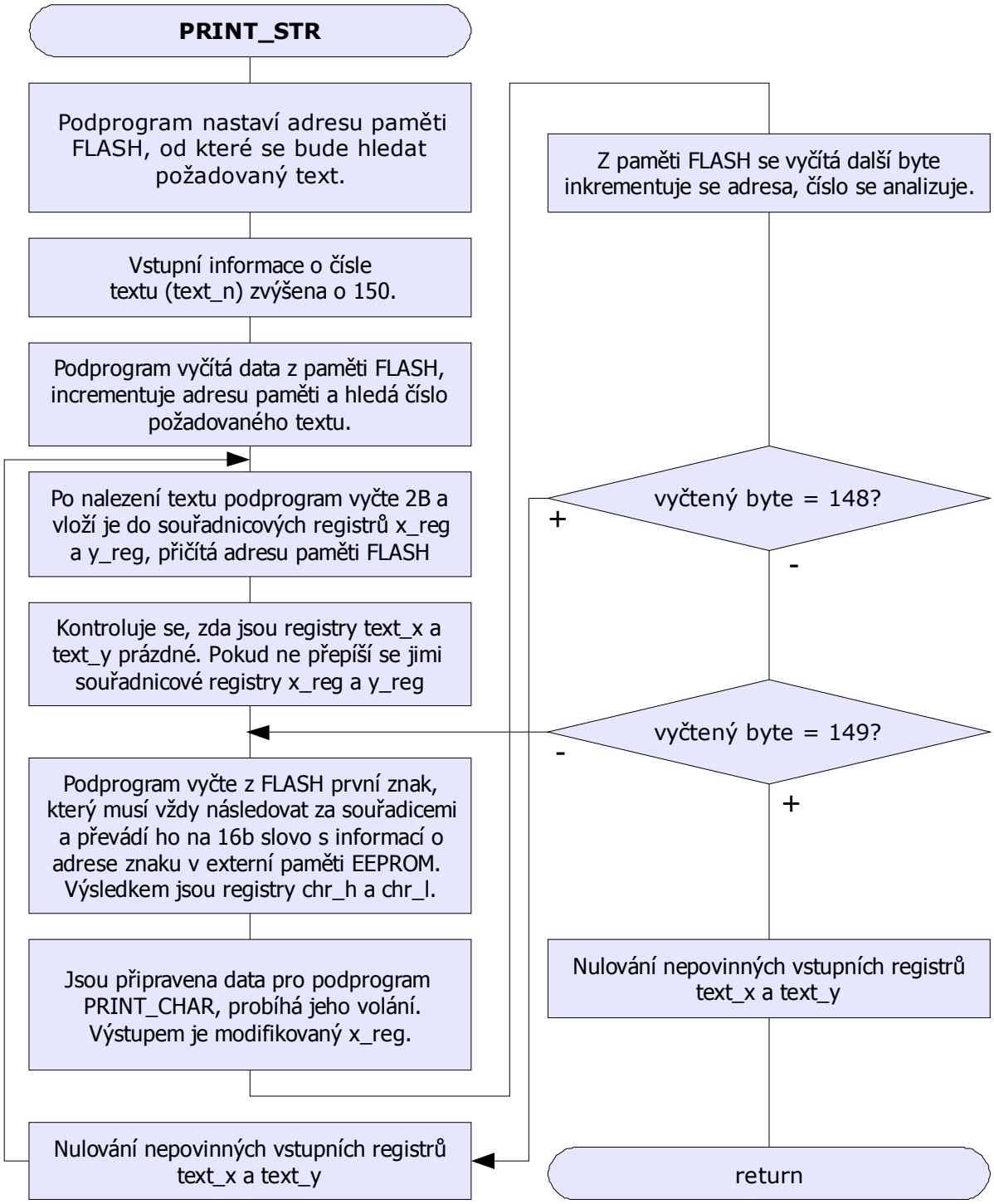
text_n (0x2D) – údaj o čísle vypisovaného textu

Nepovinnými proměnnými jsou:

text_x (0x4D) – souřadnice x prvního vypisovaného řádku
text_y (0x4E) – souřadnice y prvního vypisovaného řádku

Pokud nejsou proměnné text_x a text_y nulové, podprogram je chápe jako programově změněná souřadnice prvého řádku textu, než je uvedeno v souboru texty.inc. U víceřádkových textů by mohli způsobit problém. Toto řešení se osvědčilo jako poměrně užitečné a vyšlo najevo v průběhu psaní dalších částí programu.

5.1.13 Osnova podprogramu PRINT_STR

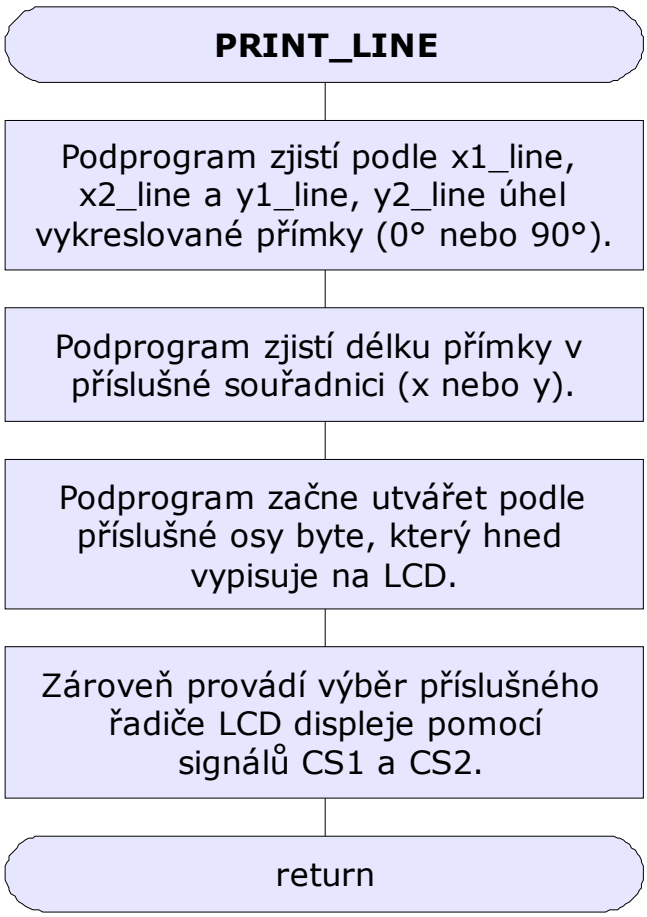


5.1.14 Zobrazování přímek na LCD

Při návrhu programu byla nutnost vytvořit podprogram tisknoucí jednoduchou grafiku (přímky 0° a 90°) na LCD displej. Grafika bude použita v editačním rozhranní „programu PLC" a různé grafické doplnění textů. O zobrazování přímek se stará podprogram PRINT_LINE. Vstupní informací jsou dva body A(x,y) a B(x,y), mezi kterými se má přímka vykreslit. Body musí být vkládány z leva doprava a odshora dolu.

Nebyli psány speciální podprogramy pro práci s LCD, jsou využity stávající ZAPISD a ZAPISI, tudíž při vodorovné lince jsou zapisovány do LCD celé byty. Výsledkem je to, že linka zaplní celý jeden řádek, nedají se tedy vytvářet linky přes již vypsaný text a podobně. Vstupními proměnnými podprogramu PRINT_LINE jsou:

x1_line (0x56) – souřadnice x bodu A
y1_line (0x57) – souřadnice y bodu A
x2_line (0x58) – souřadnice x bodu B
y2_line (0x59) – souřadnice y bodu B



5.1.15 Zobrazování více přímek najednou (zobrazení grafiky)

O zobrazování několika přímek najednou se stará Podprogram PRINT_GRAF. Grafika je, podobně jako souvislé texty, uložena v dalším souboru (v paměti FLASH) graf.inc s podobnou filosofií, formátováním a posloupností dat (obr.5.5). Při volání podprogramu PRINT_GRAF musí být v proměnné graf_n uloženo číslo grafiky, která se bude vykreslovat.

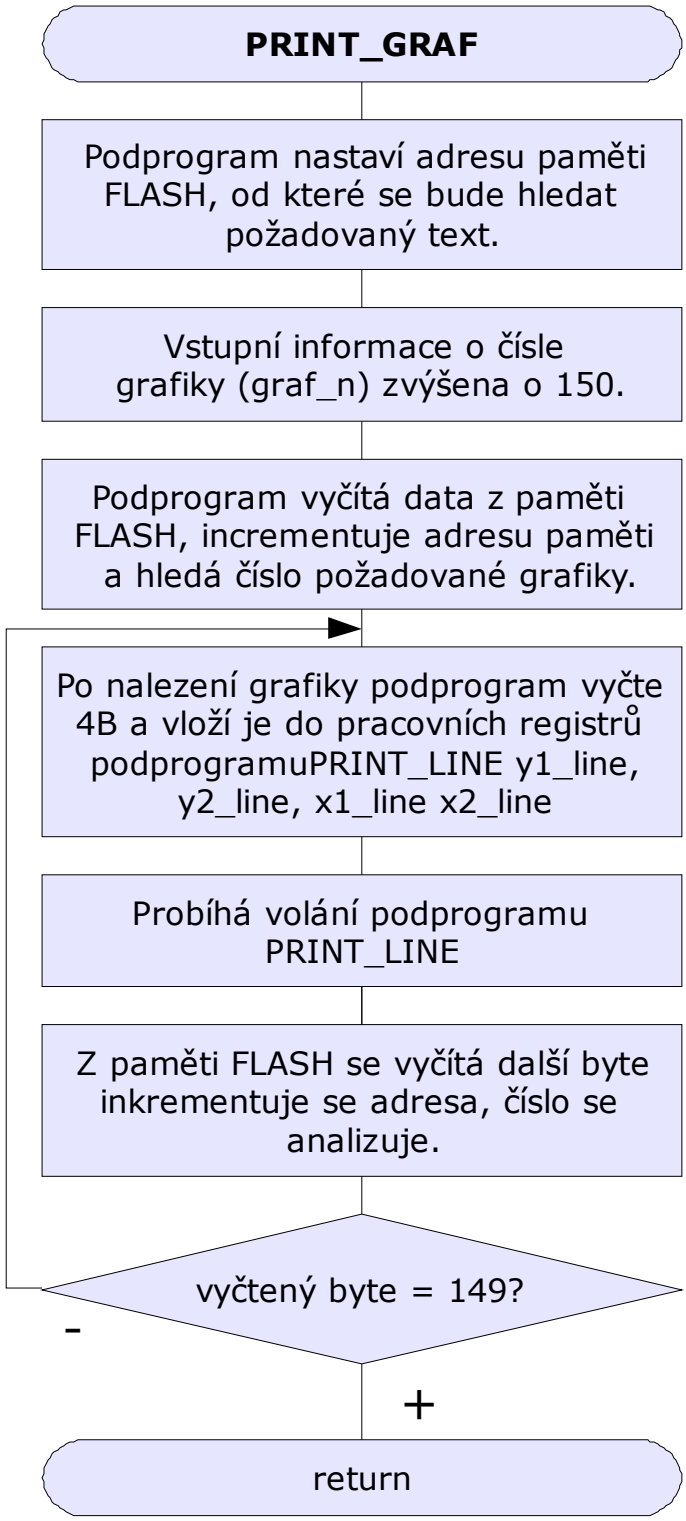
graf_n (0x4A) – číslo zobrazované grafiky

- 0.byte číslo grafiky + 150
- 1.byte souřadnice bodu A – x
- 2.byte souřadnice bodu A – y
- 3.byte souřadnice budu B – x
- 4.byte souřadnice bodu B – y
- 5.byte ukončovací znak d'149' nebo pokračování d'148'

```
;-----------------------------
            ;grafika číslo 3
            ;stavovy ramecek
            data        d'153'
            ;1.line
            data        d'9'                ;x1 souřadnice bodu A
            data        d'18'               ;y1 souřadnice bodu A
            data        d'115'              ;x2 souřadnice bodu B
            data        d'18'               ;y2 souřadnice bodu B
            data        d'148'              ;ukončovací nebo pokračovací znak
            ;2.line
            data        d'9'                ;x1 souřadnice bodu A
            data        d'35'               ;y1 souřadnice bodu A
            data        d'115'              ;x2 souřadnice bodu B
            data        d'35'               ;y2 souřadnice bodu B
            data        d'148'              ;ukončovací nebo pokračovací znak
            ;3.line
            data        d'9'                ;x1 souřadnice bodu A
            data        d'18'               ;y1 souřadnice bodu A
            data        d'9'                ;x2 souřadnice bodu B
            data        d'35'               ;y2 souřadnice bodu B
            data        d'148'              ;ukončovací nebo pokračovací znak
            ;4.line
            data        d'115'              ;x1 souřadnice bodu A
            data        d'18'               ;y1 souřadnice bodu A
            data        d'115'              ;x2 souřadnice bodu B
            data        d'35'               ;y2 souřadnice bodu B
            data        d'149'              ;ukončovací nebo pokračovací znak
```

obr.5.5 – Organizace dat souboru graf.inc

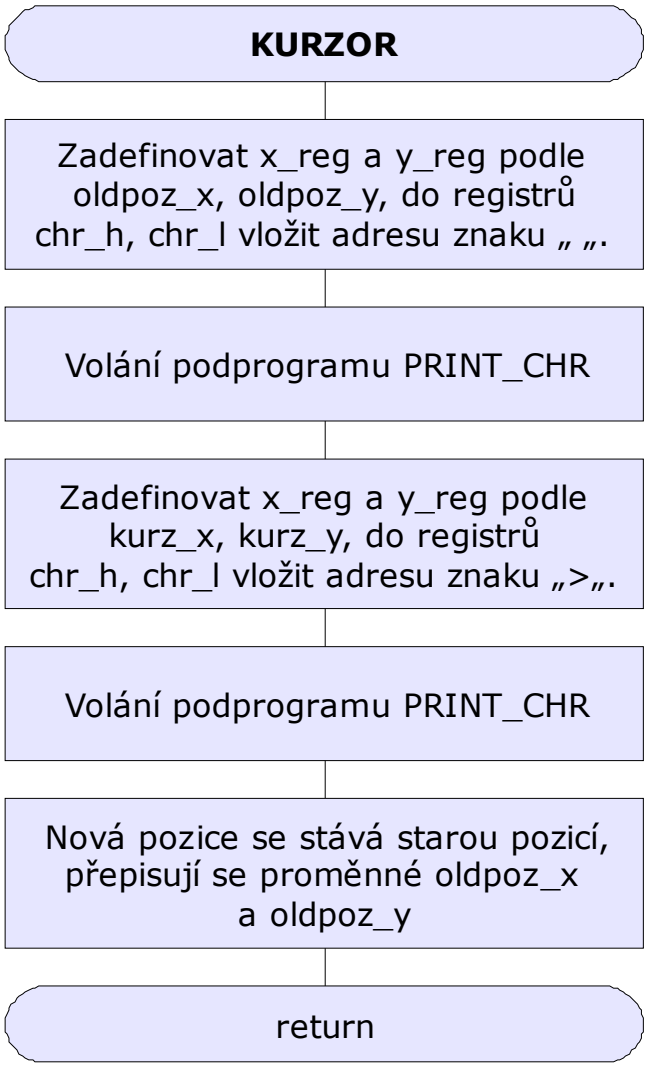
5.1.16 Osnova podprogramu PRINT_GRAF



5.1.17 Kurzor

Pro další části programu je nezbytný kurzor, který opticky ukazuje na pozici při pohybu v menu a v editačním rozhranní „programu PLC". Pro grafickou zřetelnost byl zvolen kurzor typu znaku „>" před vybraným řádkem. Povinné vstupní proměnné jsou pozice kurzoru x,y (kurz_x, kurz_y), nepovinné (za některých okolností povinné) x,y souřadnice předešlé pozice kurzoru.

kurz_x (0x43) – pozice x kurzoru
kurz_y (0x44) – pozice y kurzoru
oldpoz_x (0x41) – předešlá souřadnice x kurzoru
oldpoz_y (0x42) – předešlá souřadnice y kurzoru



5.2 Hlavní programová smyčka

Obecně se jedná o nejdůležitější část programu, na kterou obvykle po resetu mikroprocesoru ukazuje programový čítač. V aplikaci „Jednoduchý programovatelný logický automat" plní hlavní programová smyčka funkci jakéhosi výběrového menu využívající zde všech dosud popsaných podprogramů. Menu je přesně definované v posledním souboru menu.inc (obr.5.6). Soubor je psaný stejnou formou jako texty.inc a graf.inc, má přesně definovanou strukturu, práce s jeho daty je ale odlišná.

- 0.byte udává číslo menu + 150
- 1.byte informace o zobrazovaném textu ANO/NE, pokud ano číslo textu
- 2.byte informace o zobrazované grafice ANO/NE, pokud ano číslo grafiky
- 3.byte udává programu informaci o chování tlačítka ESC v tomto konkrétním menu
- 4.byte je počet pozic tohoto menu když je 0 jedná se o informační menu

```
;menu cislo 0          --          HLAVNI MENU
org         menu_ram
data        d'150'                              ;udava cislo menu + 150
data        b'01000010'                         ;udava text ANO (bit7), cislo textu 2
data        b'01000000'                         ;udava grafiku NE (bit7), cislo grafiky
data        d'0'                                ;registr udava chovani tlacitka ESC
data        d'3'                                ;pocet pozic v menu

            ;pozice 1   / PROGRAM
data        b'00101001'                         ;maska tlacitek reaguji Tl1, TL4, OK
data        d'24'                               ;souradnice kurzoru x
data        d'2'                                ;souradnice kurzoru y
data        d'0'                                ;pocet pozic doprava doleva
data        d'0'                                ;cislo offsetu vypisovaneho textu doprava/dol
data        d'0'                                ;cislo aktualni pozice
data        d'1'                                ;chovani tlacitka OK v aktualni pozici

            ;pozice 2   / NASTAVENI
data        b'00101001'                         ;maska tlacitek reaguji Tl1, TL4, OK
data        d'24'                               ;souradnice kurzoru x
data        d'3'                                ;souradnice kurzoru y
data        d'0'                                ;pocet pozic doprava doleva
data        d'0'                                ;cislo offsetu vypisovaneho textu doprava/dol
data        d'0'                                ;cislo aktualni pozice
data        d'2'                                ;chovani tlacitka OK v aktualni pozici

            ;pozice 3   / About
data        b'00101001'                         ;maska tlacitek reaguji Tl1, TL4, OK
data        d'24'                               ;souradnice kurzoru x
data        d'4'                                ;souradnice kurzoru y
data        d'0'                                ;pocet pozic doprava doleva
data        d'0'                                ;cislo offsetu vypisovaneho textu doprava/dol
data        d'0'                                ;cislo aktualni pozice
data        d'3'                                ;chovani tlacitka OK v aktualni pozici
```

obr.5.6 – Organizace dat souboru menu.inc

- pokud je informace o počtu pozic menu vyšší než 0, následuje tolik bloků (1 blok = 7B), kolik je pozic tohoto menu
    - 1.byte udává informaci, která tlačítka reagují v této pozici (1 = klávesa

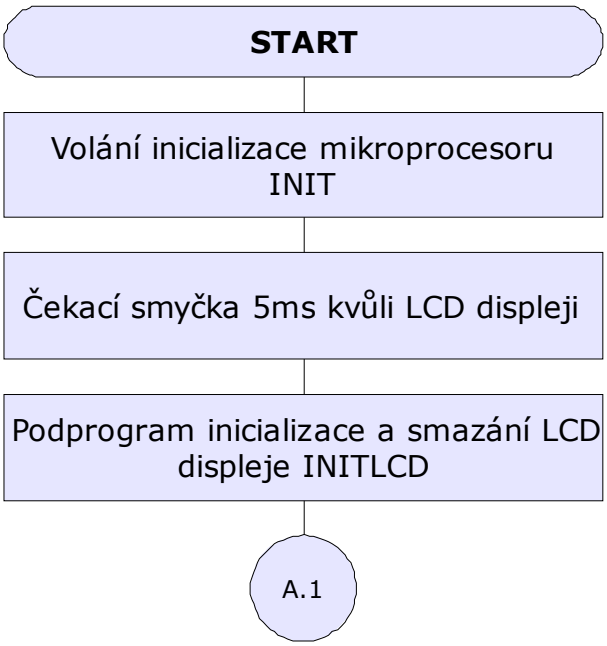
reaguje, 0 = klávesa nereaguje)
- 2.byte souřadnice x kurzoru
- 3.byte souřadnice y kurzoru
- 4.byte údaj o počtu pozic doprava/doleva v konkrétní pozici menu
- 5.byte začátek paměti s blokem vypisovaných textů v konkrétní pozici při výběru doprava/doleva
- 6.byte číslo aktuální pozice doprava/doleva
- 7.byte udává programu informaci o co má dělat, když byla v aktuální pozici stisknuta klávesa OK.

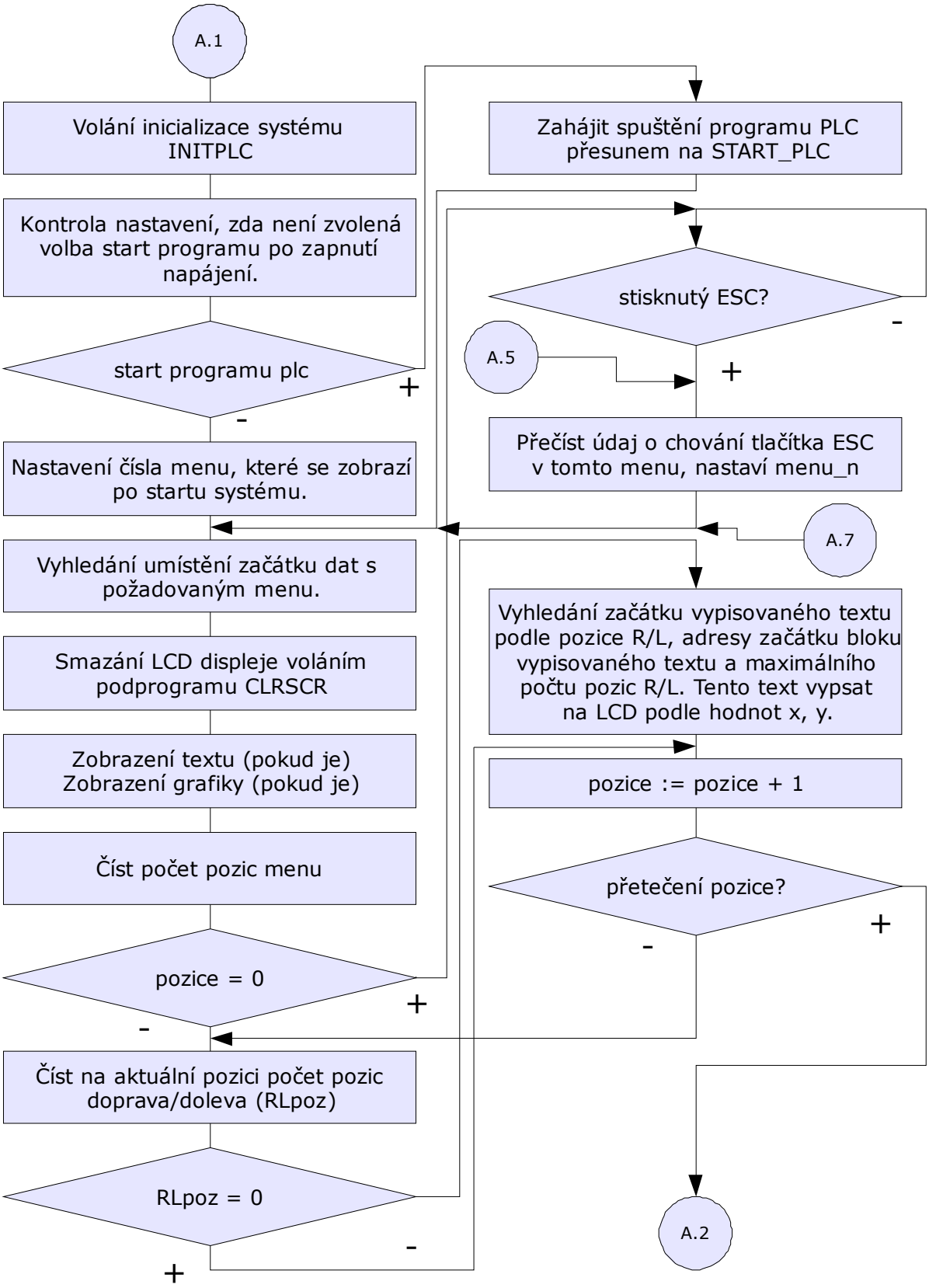
S takto definovanými daty pak pracuje hlavní programová smyčka a na základě informací z klávesnice směruje a větví program do požadovaných funkcí, jako je Nastavení, Editace (programu PLC), Start (programu PLC) apod.
Hlavní vstupní proměnnou je údaj o zobrazovaném menu:

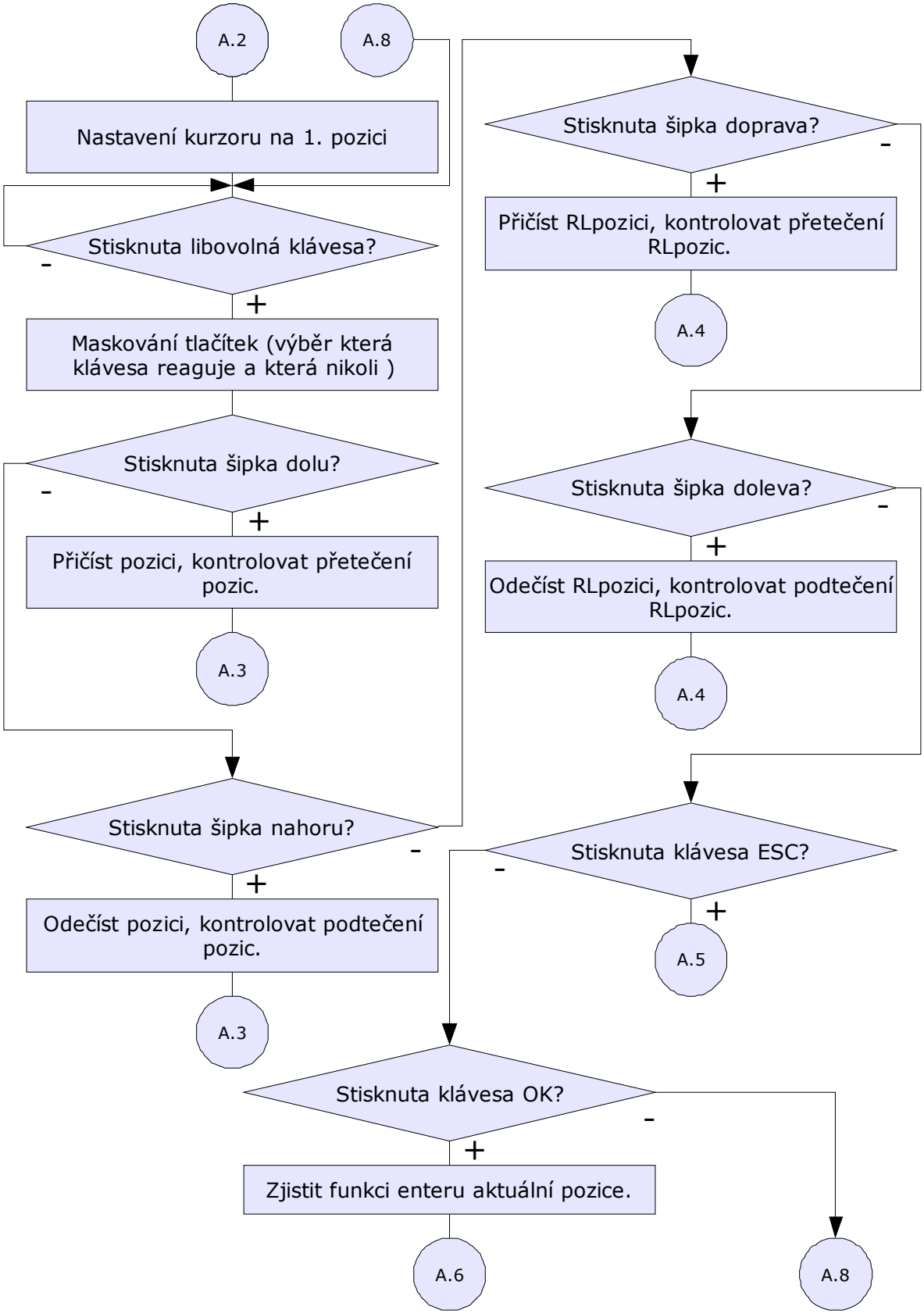
menu_n (0x45) – údaj o zobrazovaném menu

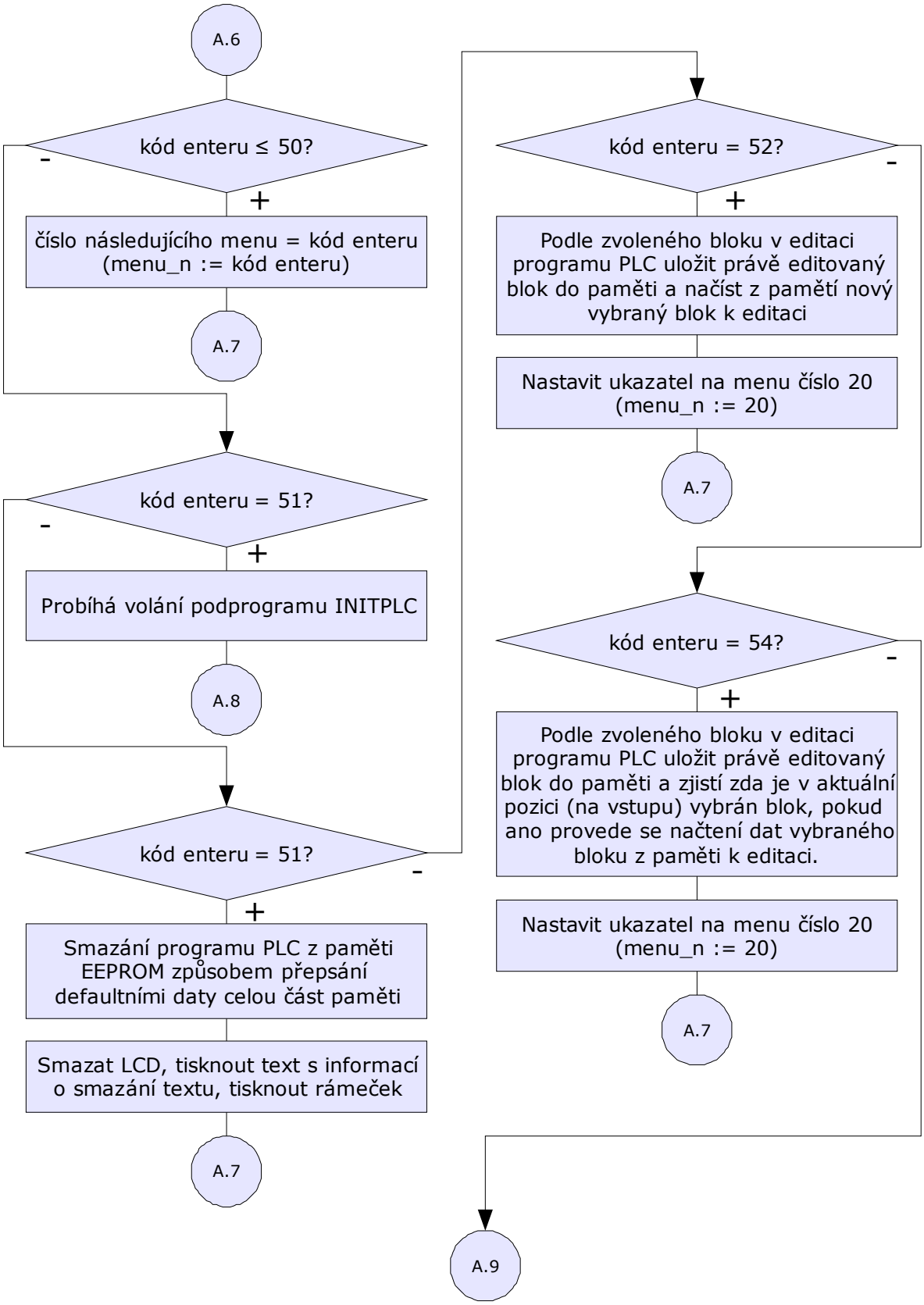
Novými pracovními proměnnými jsou:
menuadr (0x46,0x47) – 16b údaj o začátku paměti dat příslušného menu
madr (0x48,0x49) – 16b pracovní registr použitý při práci s pohybem paměti příslušného menu
akt_poz (0x4B) – registr udávající aktuální pozici kurzoru v příslušném menu
count_m (0x4C) – univerzální pomocný registr
menu_poz (0x4F) – počet pozic v aktuálním menu
maska_tl (0x50) – maska tlačítek s údajem která tlačítka v aktuální pozici reagují
KeyEvent2 (0x51) – pomocná proměnná k registru KeyEvent
RLpoz (0x52) – počet pozic porava/doleva v aktuální pozici příslušného menu
RLakt_poz (0x53) – aktuální pozice doprava/doleva

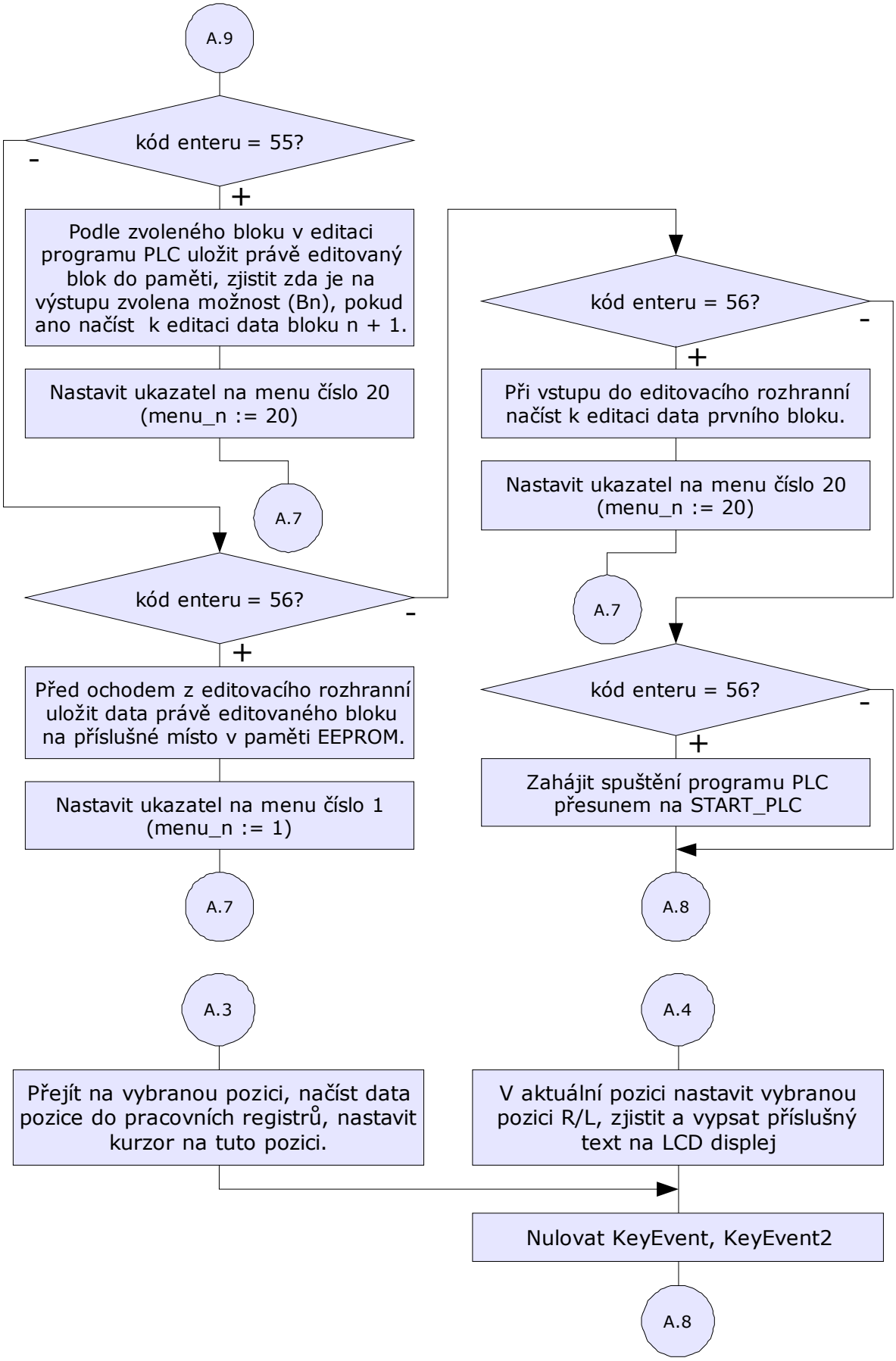
5.2.1 Osnova hlavní programové smyčky




A.1
Volání inicializace systému INITPLC
Kontrola nastavení, zda není zvolená volba start programu po zapnutí napájení.
start programu plc
+
-
Nastavení čísla menu, které se zobrazí po startu systému.
Vyhledání umístění začátku dat s požadovaným menu.
Smazání LCD displeje voláním podprogramu CLRSCR
Zobrazení textu (pokud je) Zobrazení grafiky (pokud je)
Číst počet pozic menu
pozice = 0
+
-
Číst na aktuální pozici počet pozic doprava/doleva (RLpoz)
RLpoz = 0
-
+
Zahájit spuštění programu PLC přesunem na START_PLC
stisknutý ESC?
-
A.5
+
Přečíst údaj o chování tlačítka ESC v tomto menu, nastaví menu_n
A.7
Vyhledání začátku vypisovaného textu podle pozice R/L, adresy začátku bloku vypisovaného textu a maximálního počtu pozic R/L. Tento text vypsat na LCD podle hodnot x, y.
pozice := pozice + 1
přetečení pozice?
+
-
A.2


A.2
A.8
Nastavení kurzoru na 1. pozici
Stisknuta libovolná klávesa?
-
+
Maskování tlačítek (výběr která klávesa reaguje a která nikoli)
Stisknuta šipka dolu?
-
+
Přičíst pozici, kontrolovat přetečení pozic.
A.3
Stisknuta šipka nahoru?
-
+
Odečíst pozici, kontrolovat podtečení pozic.
A.3
Stisknuta šipka doprava?
-
+
Přičíst RLpozici, kontrolovat přetečení RLpozic.
A.4
Stisknuta šipka doleva?
-
+
Odečíst RLpozici, kontrolovat podtečení RLpozic.
A.4
Stisknuta klávesa ESC?
-
+
A.5
Stisknuta klávesa OK?
-
+
Zjistit funkci enteru aktuální pozice.
A.6
A.8


A.6
kód enteru ≤ 50?
-
+
číslo následujícího menu = kód enteru (menu_n := kód enteru)
A.7
kód enteru = 51?
-
+
Probíhá volání podprogramu INITPLC
A.8
kód enteru = 51?
-
+
Smazání programu PLC z paměti EEPROM způsobem přepsání defaultními daty celou část paměti
Smazat LCD, tisknout text s informací o smazání textu, tisknout rámeček
A.7
kód enteru = 52?
-
+
Podle zvoleného bloku v editaci programu PLC uložit právě editovaný blok do paměti a načíst z pamětí nový vybraný blok k editaci
Nastavit ukazatel na menu číslo 20 (menu_n := 20)
A.7
kód enteru = 54?
-
+
Podle zvoleného bloku v editaci programu PLC uložit právě editovaný blok do paměti a zjistí zda je v aktuální pozici (na vstupu) vybrán blok, pokud ano provede se načtení dat vybraného bloku z paměti k editaci.
Nastavit ukazatel na menu číslo 20 (menu_n := 20)
A.7
A.9


A.9
kód enteru = 55?
-
+
Podle zvoleného bloku v editaci programu PLC uložit právě editovaný blok do paměti, zjistit zda je na výstupu zvolena možnost (Bn), pokud ano načíst k editaci data bloku n + 1.
Nastavit ukazatel na menu číslo 20 (menu_n := 20)
A.7
kód enteru = 56?
-
+
Před ochodem z editovacího rozhranní uložit data právě editovaného bloku na příslušné místo v paměti EEPROM.
Nastavit ukazatel na menu číslo 1 (menu_n := 1)
A.7
A.3
Přejít na vybranou pozici, načíst data pozice do pracovních registrů, nastavit kurzor na tuto pozici.
kód enteru = 56?
-
+
Při vstupu do editovacího rozhranní načíst k editaci data prvního bloku.
Nastavit ukazatel na menu číslo 20 (menu_n := 20)
A.7
kód enteru = 56?
-
+
Zahájit spuštění programu PLC přesunem na START_PLC
A.8
A.4
V aktuální pozici nastavit vybranou pozici R/L, zjistit a vypsat příslušný text na LCD displej
Nulovat KeyEvent, KeyEvent2
A.8

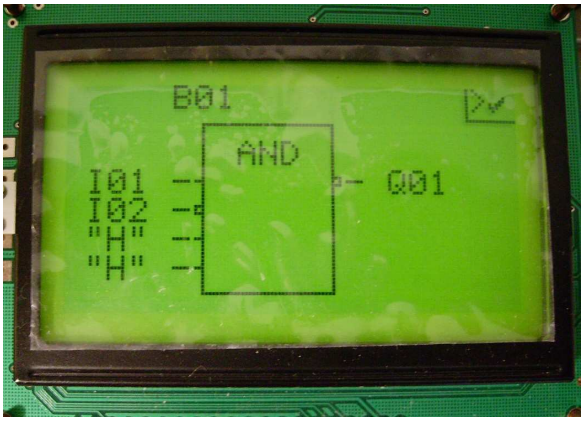
5.2.2 Menu

Takto vytvořený program umožňuje pohyb po definovaných pozicích, změna hodnoty v pozici, přesun do jiné části s novými definicemi pozic a podobně. Pohyb není samozřejmě jen grafický, během některých pohybů se mění příslušná data dané pozice a ty jsou ukládána do některé z pamětí. Tato filosofie umožňuje vytvoření menu aplikace „Jednoduchý programovatelný logický automat" a zároveň editovací rozhranní programu PLC.

obr.5.7 Hlavní menu aplikace

obr.5.7 Přesun z hlavního menu na menu PROGRAM

obr.5.8 Potvrzovací menu



obr.5.9 Editovací rozhranní aplikace

5.2.3 Vytvoření programu PLC pomocí editovacího rozhranní aplikace

Vytváření programu PLC se provádí v menu *PROGRAM -> Editace* grafickou formou pomocí bloků, kdy každý blok představuje nastavenou logickou funkci. Tato funkce může má 4 vstupy, které mohou být ze tří různých zdrojů:

- pevná logická úroveň „L", „H" (log.0, log.1)
- softwarové vstupy B01 – B32 (výsledek logické funkce jiného bloku)
- hardwarové vstupy I01 – I04 (log. „0" – 0V, log. „1" – 1V až 12V)

U každého vstupu lze vybrat negaci. Funkce jsou na výběr dvě základní:

- AND (logický součin)
- OR (logický součet)

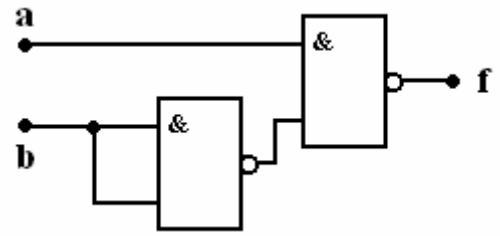
Výstup může být opět:

- pouze softwarový výstup Bn (dostupný pod názvem bloku B01 – B32)
- softwarový + hardwarový výstup Q01 – Q04 (sepnuté kontakty relé 1A)

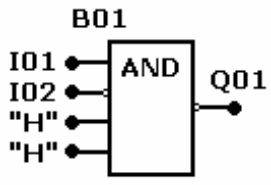
U výstupu lze také volit negace, což nám přidá dvě funkce NAND, NOR, při vhodně zapojených vstupech i funkci NOT.

Vytváření programu se pak tvoří podle libovolné zadané funkce, kdy si můžeme nejprve vytvořit pravdivostní tabulku, tu přepsat do rovnice (Karnaughova mapa, booleova algebra) a překreslit si rovnici na zapojení hradel (obr.5.10).

a	b	f
0	0	1
0	1	1
1	0	0
1	1	1



obr.5.10 – Zapojeni funkce pomocí hradel NAND



obr.5.11 – Zapojení obvodu v PLC

Takto překreslenou funkci můžeme překreslit podle možností editovacího rozhranní aplikace (např. podle obr.5.11) a pomocí editovacího rozhranní ji do automatu zadat (předchozí obr.5.9).

5.2.4 Organizace dat programu PLC v paměti EEPROM

Během editování programu dochází k ukládání nastavených dat do paměti EEPROM. Každý blok má v paměti velikost 11B. Program se v paměti nachází od adresy 0000h po adresu 0160h (11B x 32bloků = 352B).

- 1.byte – informace o vstupu 1
- 2.byte – informace o negaci vstupu 1
- 3.byte – vstup 2
- 4.byte – negace vstupu 2
- 5.byte – vstup 3
- 6.byte – negace vstupu 3
- 7.byte – vstup 4
- 8.byte – negace vstupu 4
- 9.byte – výběr logické funkce AND, OR
- 10.byte – negace výstupu
- 11.byte – informace o výstup (SW, HW+SW+číslo výstupu)

5.2.5 Zpracování a běh programu PLC

Spuštění vytvořeného programu může být dvěma způsoby. Pro testování se provádí v menu *PROGRAM -> Start* a při dlouhodobém provozu je dobré změnit v hlavním menu v *NASTAVENI* položku *Start po zapnuti* na *ANO*. Tato změna zajistí po přerušení dodávky elektrické energie a opětovném obnovení start programu PLC bez úvodních obrazovek a výběru *Start*.

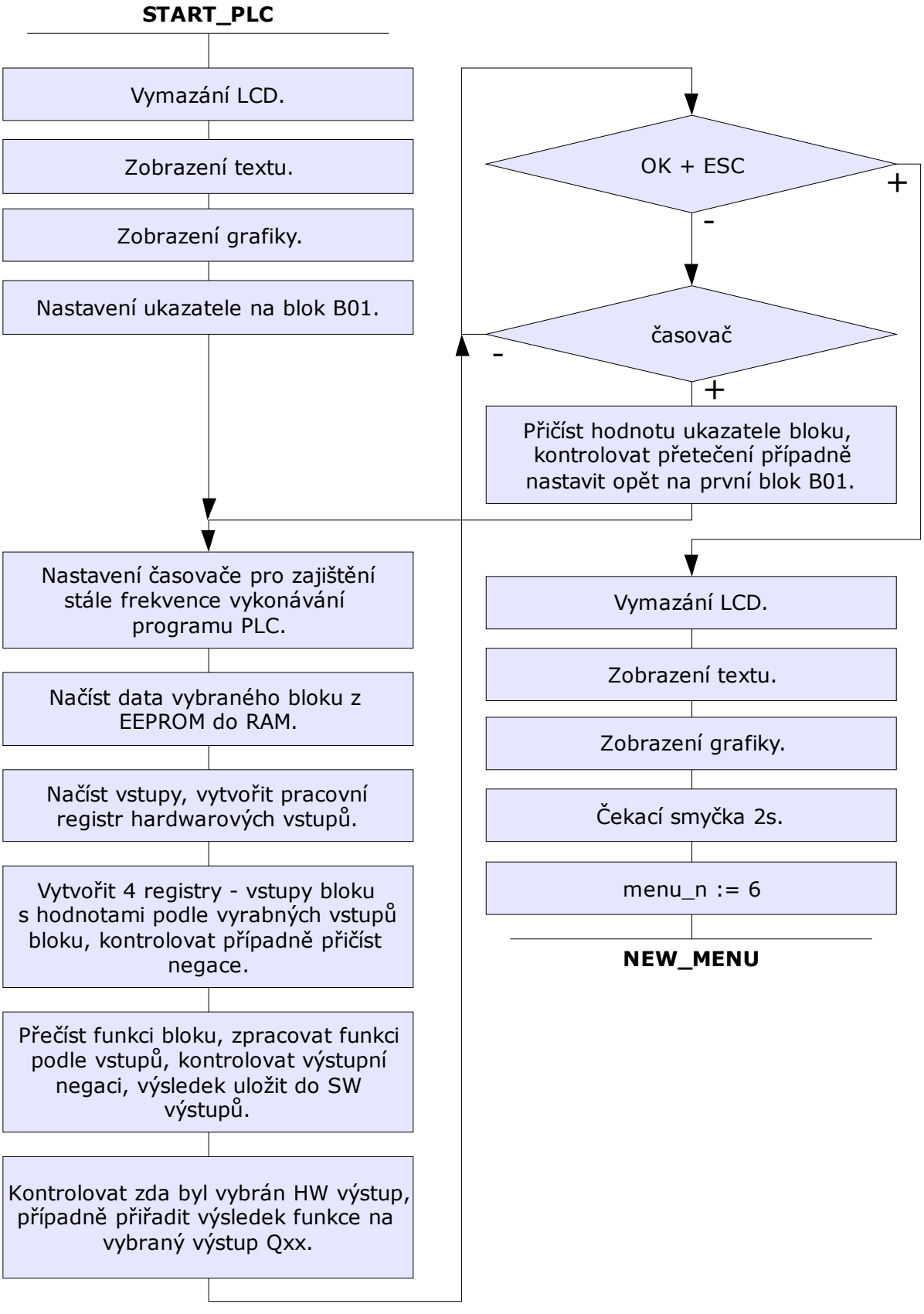
Po zahájení startu se program v mikroprocesoru přesune na smyčku běhu programu PLC. Smyčka se skládá z několik důležitých celků. Načtení dat, zjištění a definování vstupních informací společně s kontrolováním negací, provedení funkce příslušného bloku, výsledek zpracovat podle vybrané negace výstupu, uložit do softwarového výstupu, zkontrolovat hardwarový výstup případně sepnout výstupní relé.

blok_n	(0x62) – údaj o aktuálně zpracovávaném bloku
input	(0x63) – reprezentuje stav na vstupu na začátku smyčky
vstup_0	(0x64) – univerzální pracovní registr
neg_vst	(0x65) – údaj s negací vstupu
image_input	(0x66) – obraz vstupu (pracovní reg) na rozdíl od input se dá změnit
vystup_0	(0x67) – pracovní výstupní registr
output	(0x68) – výstupní registr

5.2.6 Softwarové výstupy programu PLC

Softwarové výstupy programu PLC je oblast dat o velikosti 32B, do které se během běhu programu ukládají výsledky každého bloku. Softwarové výstupy jsou uložené v paměti RAM mikroprocesoru a jsou během vykonávání programu libovolně přístupné nadřazeným programem nebo přerušením. Tato data se nachází ve druhé bance paměti dat mikroprocesoru od adresy 0xB4 a začínají výstupem bloku B01.

5.2.7 Osnova smyčky START_PLC



6. Obvodové řešení

Obvodové řešení je zřejmé z následujícího schématu. Na obr.6.1 je patrné rozložení součástek desky plošných spojů (DSP). DSP je dvouvrstvá - obr.6.2 a obr.6.3 jsou pouze ilustrativními obrázky. DSP v rozměrech 1:1 je k dispozici na přiloženém CD.

6.1 Rozpis součástek exportovaný z EAGLE 4.16r2

Partlist

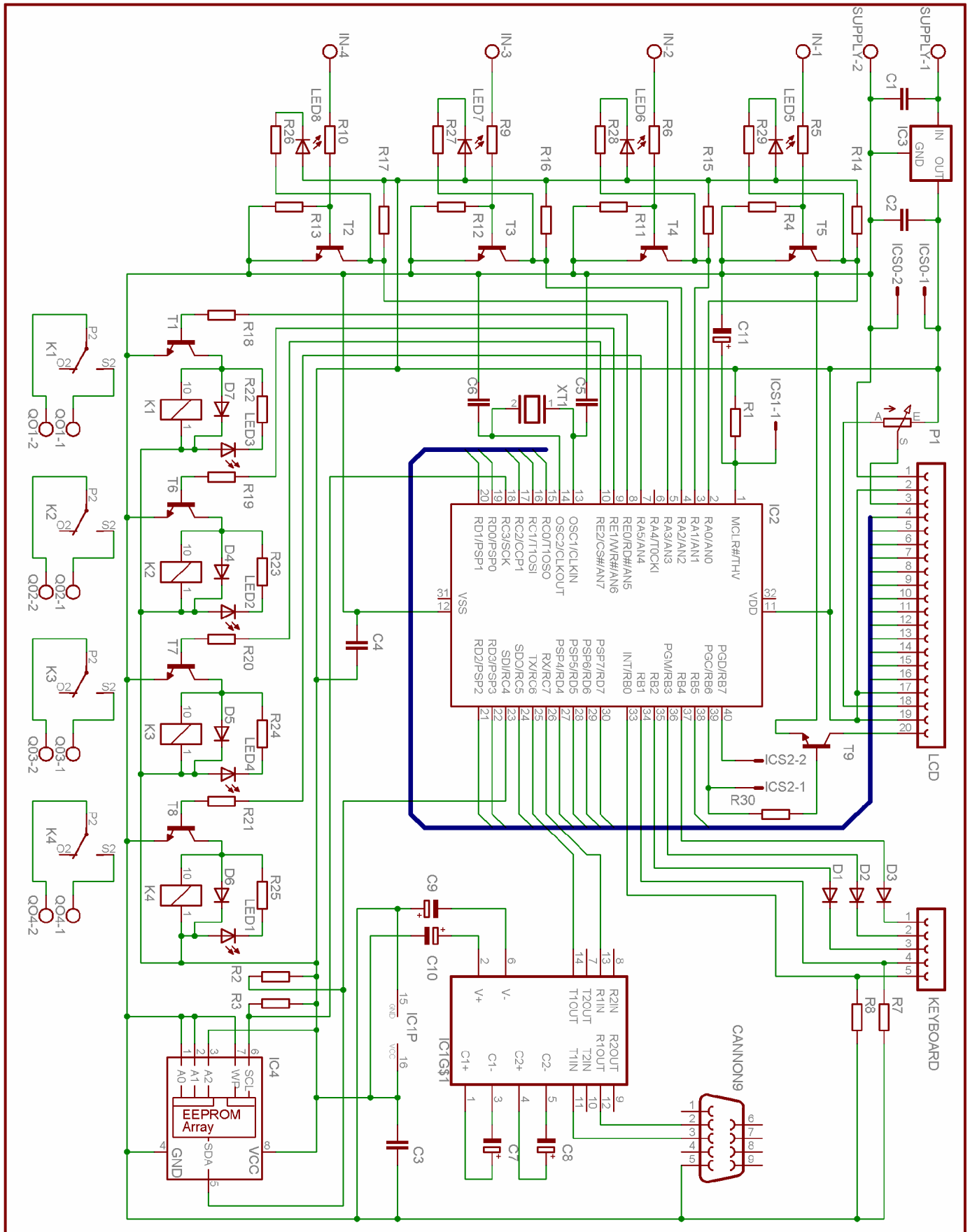
Exported from plc_double_dsp.sch at 23.04.2007 15:30:39

EAGLE Version 4.16r2 Copyright (c) 1988-2006 CadSoft

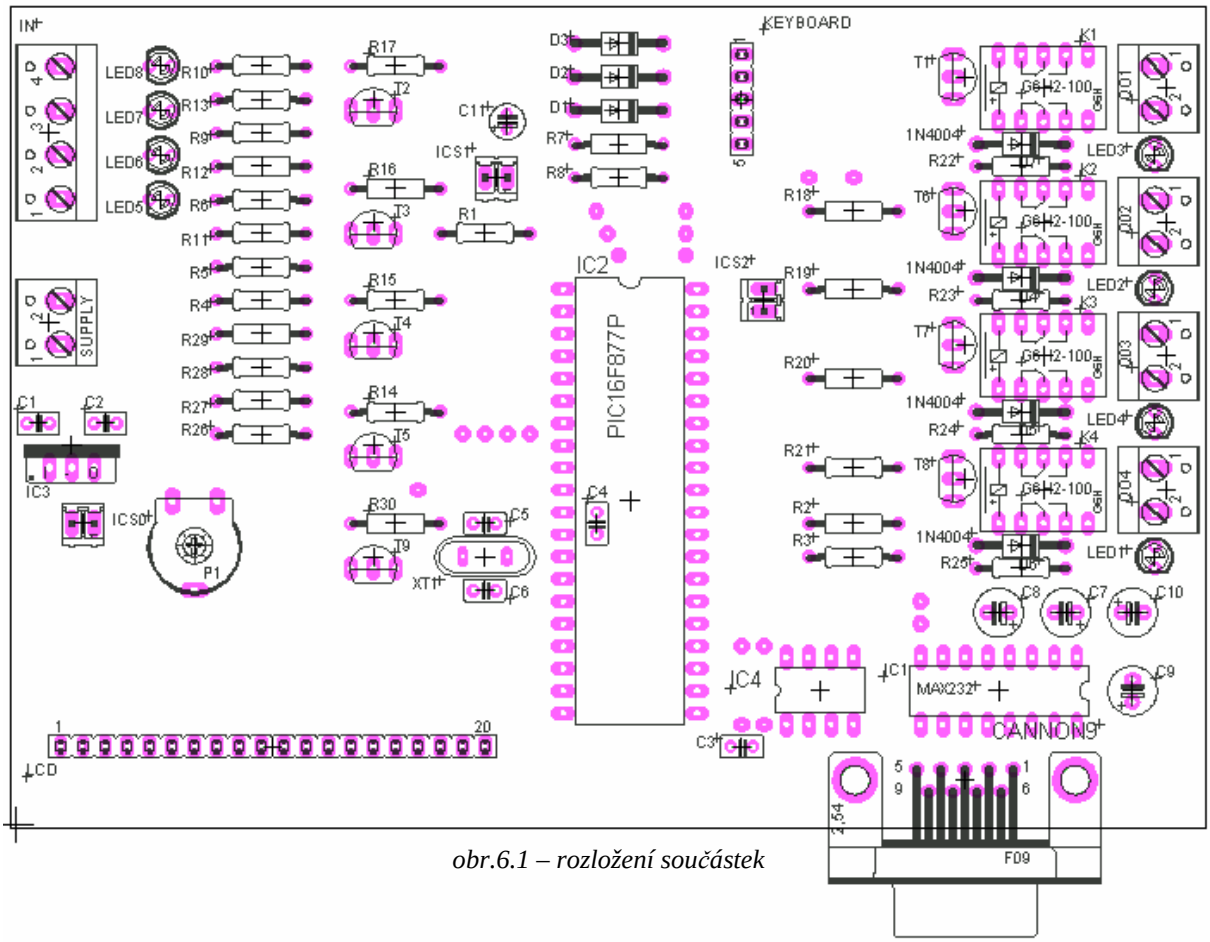
Part	Value	Device	Package	Library	Sheet
C1	100n	C-EU025-025X050	C025-025X050	rcl	1
C2	100n	C-EU025-025X050	C025-025X050	rcl	1
C3	100n	C-EU025-025X050	C025-025X050	rcl	1
C4	100n	C-EU025-025X050	C025-025X050	rcl	1
C5	33p	C-EU025-025X050	C025-025X050	rcl	1
C6	33p	C-EU025-025X050	C025-025X050	rcl	1
C7	1u/16V	CPOL-EUE2.5-6	E2,5-6	rcl	1
C8	1u/16V	CPOL-EUE2.5-6	E2,5-6	rcl	1
C9	1u/16V	CPOL-EUE2.5-6	E2,5-6	rcl	1
C10	1u/16V	CPOL-EUE2.5-6	E2,5-6	rcl	1
C11	1u/16V	CPOL-EUE1.8-4	E1,8-4	rcl	1
CANNON9		F09HP	F09HP	con-subd	1
D1	1N4004	1N4004	DO41-10	diode	1
D2	1N4004	1N4004	DO41-10	diode	1
D3	1N4004	1N4004	DO41-10	diode	1
D4	1N4004	1N4004	DO41-10	diode	1
D5	1N4004	1N4004	DO41-10	diode	1
D6	1N4004	1N4004	DO41-10	diode	1
D7	1N4004	1N4004	DO41-10	diode	1
IC1	MAX232	MAX232	DIL16	maxim	1
IC2	PIC16F877P	PIC16F877P	DIL40	microchip	1
IC3	78L05	78XXS	78XXS	v-reg	1
IC4	24FC1025	24AA256P	DIL8	microchip	1
ICS0		M02S	02P	con-amp-quick	1
ICS1		M02S	02P	con-amp-quick	1
ICS2		M02S	02P	con-amp-quick	1
IN		W237-4	W237-4	con-wago-500	1
K1	5V/1A	G6H2-100	G6H2-100	relay	1
K2	5V/1A	G6H2-100	G6H2-100	relay	1
K3	5V/1A	G6H2-100	G6H2-100	relay	1
K4	5V/1A	G6H2-100	G6H2-100	relay	1
KEYBOARD		FE05-1	FE05-1	con-lsta	1
LCD		FE20-1	FE20	con-lsta	1
LED1	RED	LED3MM	LED3MM	led	1
LED2	RED	LED3MM	LED3MM	led	1
LED3	RED	LED3MM	LED3MM	led	1
LED4	RED	LED3MM	LED3MM	led	1

LED5	RED	LED3MM	LED3MM	led	1
LED6	RED	LED3MM	LED3MM	led	1
LED7	RED	LED3MM	LED3MM	led	1
LED8	RED	LED3MM	LED3MM	led	1
P1	2k5	TRIMRPT10V	PT10V	#Trimry	1
Q02		W237-102	W237-102	con-wago-500	1
Q03		W237-102	W237-102	con-wago-500	1
QO1		W237-102	W237-102	con-wago-500	1
QO4		W237-102	W237-102	con-wago-500	1
R1	1k	R-EU_0207/10	0207/10	rcl	1
R2	4k7	R-EU_0207/10	0207/10	rcl	1
R3	4k7	R-EU_0207/10	0207/10	rcl	1
R4	1k	R-EU_0207/10	0207/10	rcl	1
R5	10k	R-EU_0207/10	0207/10	rcl	1
R6	10k	R-EU_0207/10	0207/10	rcl	1
R7	10k	R-EU_0207/10	0207/10	rcl	1
R8	10k	R-EU_0207/10	0207/10	rcl	1
R9	10k	R-EU_0207/10	0207/10	rcl	1
R10	10k	R-EU_0207/10	0207/10	rcl	1
R11	1k	R-EU_0207/10	0207/10	rcl	1
R12	1k	R-EU_0207/10	0207/10	rcl	1
R13	1k	R-EU_0207/10	0207/10	rcl	1
R14	10k	R-EU_0207/10	0207/10	rcl	1
R15	10k	R-EU_0207/10	0207/10	rcl	1
R16	10k	R-EU_0207/10	0207/10	rcl	1
R17	10k	R-EU_0207/10	0207/10	rcl	1
R18	10k	R-EU_0207/10	0207/10	rcl	1
R19	10k	R-EU_0207/10	0207/10	rcl	1
R20	10k	R-EU_0207/10	0207/10	rcl	1
R21	10k	R-EU_0207/10	0207/10	rcl	1
R22	180R	R-EU_0207/10	0207/10	rcl	1
R23	180R	R-EU_0207/10	0207/10	rcl	1
R24	180R	R-EU_0207/10	0207/10	rcl	1
R25	180R	R-EU_0207/10	0207/10	rcl	1
R26	180R	R-EU_0207/10	0207/10	rcl	1
R27	180R	R-EU_0207/10	0207/10	rcl	1
R28	180R	R-EU_0207/10	0207/10	rcl	1
R29	180R	R-EU_0207/10	0207/10	rcl	1
R30	10k	R-EU_0207/10	0207/10	rcl	1
SUPPLY		W237-102	W237-102	con-wago-500	1
T1	BC337	BC337-16-NPN-TO92-EBC	TO92-EBC	transistor	1
T2	BC337	BC337-16-NPN-TO92-EBC	TO92-EBC	transistor	1
T3	BC337	BC337-16-NPN-TO92-EBC	TO92-EBC	transistor	1
T4	BC337	BC337-16-NPN-TO92-EBC	TO92-EBC	transistor	1
T5	BC337	BC337-16-NPN-TO92-EBC	TO92-EBC	transistor	1
T6	BC337	BC337-16-NPN-TO92-EBC	TO92-EBC	transistor	1
T7	BC337	BC337-16-NPN-TO92-EBC	TO92-EBC	transistor	1
T8	BC337	BC337-16-NPN-TO92-EBC	TO92-EBC	transistor	1
T9	BC337	BC337-16-NPN-TO92-EBC	TO92-EBC	transistor	1
XT1	8MHz	XTAL/S	QS	special	1

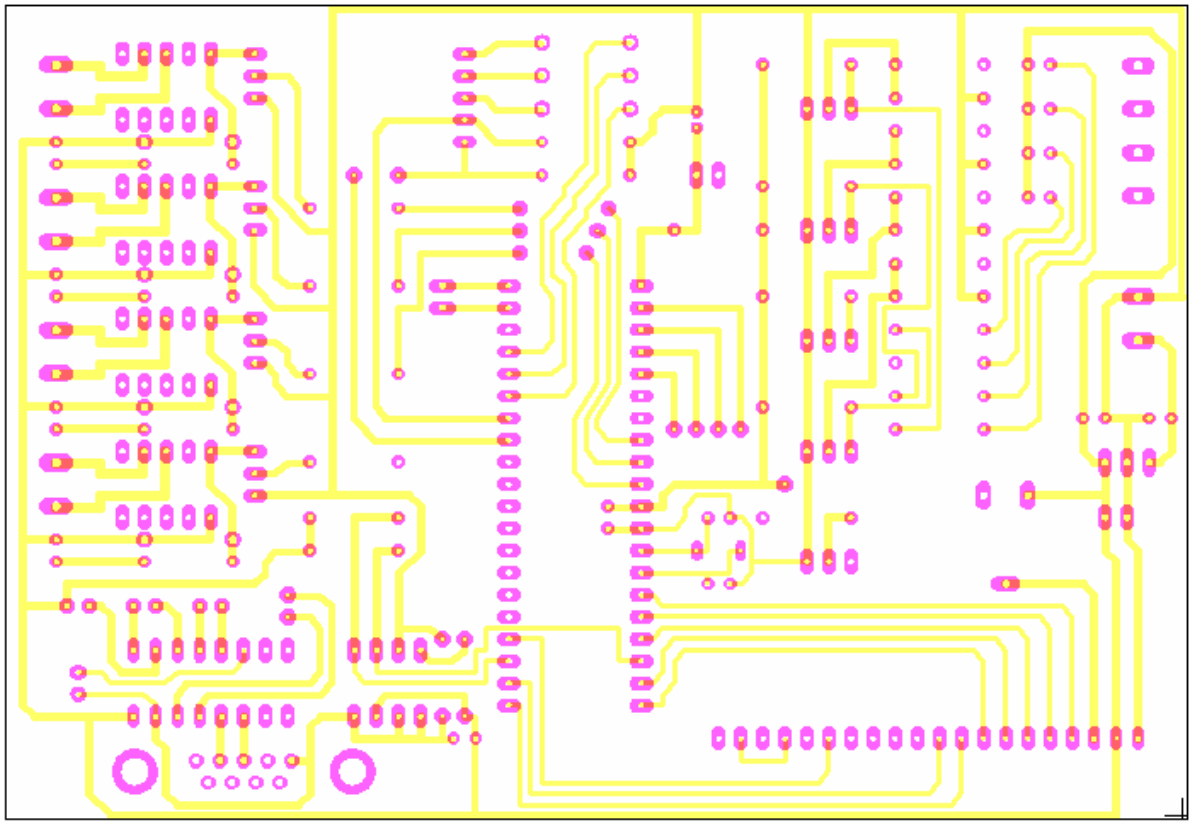
6.2 Schéma a deska plošných spojů



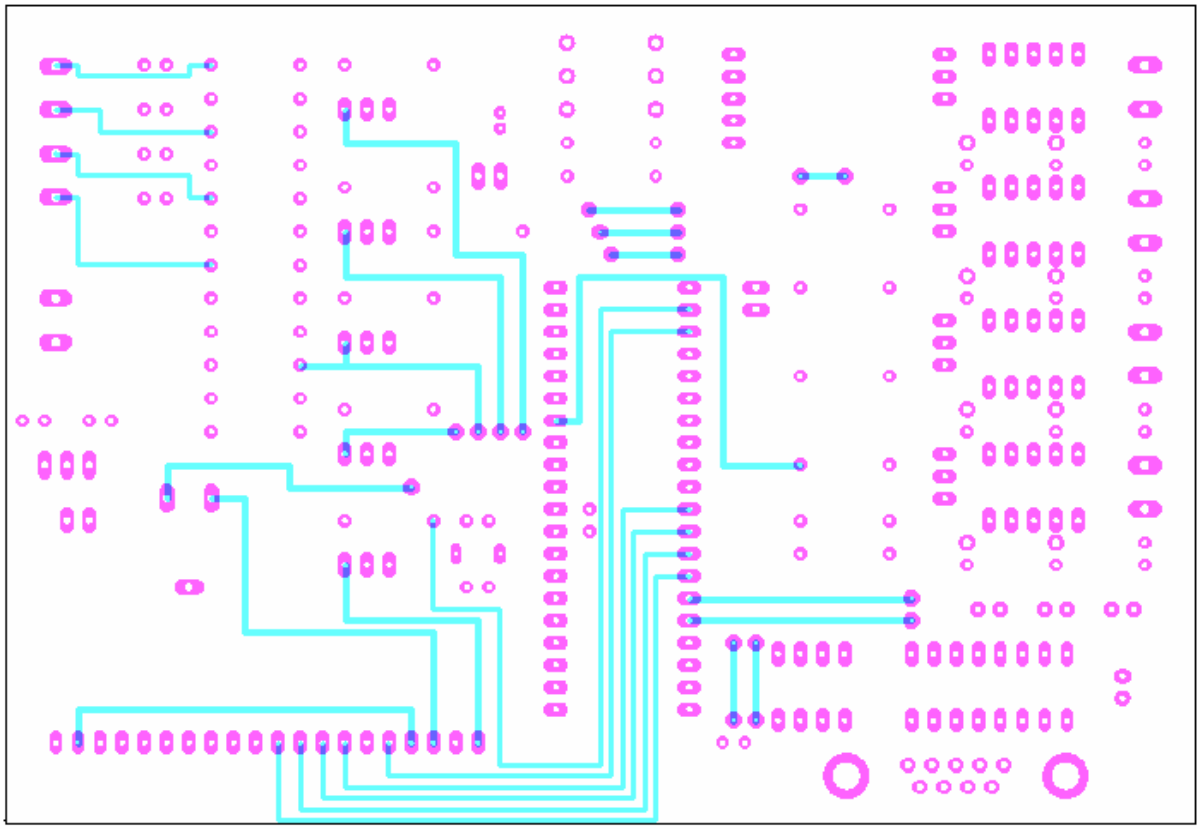
Date: 15.04.2007 10:52:03	
TITLE: plc	Sheet: 1/1



obr.6.1 – rozložení součástek



obr.6.2 – spodní (2.) vrstva DPS



obr.6.3 – horní (1.) vrstva DPS

7. Technické parametry zařízení

Rozměry zařízení:	120x145x33	mm
Rozsah vstupního napětí:	7 – 12	V
Maximální odebíraný proud:	450	mA
Přibližný rozsah provozní vlhkosti:	10 - 90	%
Počet programovatelných bloků	32	
Počet logických vstupů	4	
Počet logických výstupů	4	
Frekvence vykonávaného programu (SCAN):	32	Hz
Logické úrovně vstupů:	„0" – 0 „1" >1 - 12	V V
Proudové zatížení výstupů:	1	A

8. Závěr

Hodiny věnované aplikaci „Jednoduchý programovatelný logický automat" byly velice zajímavé, mně osobně daly mnoho zkušeností zejména v práci s použitým hardwarem. Ověření funkce a principu sériové sběrnice I^2C, práce s grafickými LCD displejem a v neposlední řadě detekování a zpracovávání dat z maticové klávesnice. Hlavním předmětem projektu bylo zvládnutí práce s jednočipovými mikropočítačem, což v mém případě nebyl z důvodů zkušeností z předchozích let téměř žádný problém.

Vývoji zařízení jsem věnoval kolem 400 hodin práce, je spousta možností, jak zařízení vylepšit a dovést do dokonalosti. Jak po HW stránce tak po SW. Jako zajímavé pokračování se nabízí připravená sériová linka na propojení s osobním počítačem. Zde by mohlo být vyvinuto programovací rozhranní, ve kterém uživatel sestaví program a ten se pak nahraje přes sériovou linku do aplikace. Zařízení by mohlo s programovacím rozhranním různě komunikovat, do běhu programu zasahovat, číst veškeré výstupy a podobně.

„Jednoduchý programovatelný logický automat" by se dal využít při realizování jednoduchých problémů, jako je kupříkladu chodbové osvětlení, řízení bazénů atd. Zkrátka všude tam, kde k realizaci stačí základní logické funkce. Lze ho využít při výuce číslicové techniky, při ověřování logických funkcí, sekvenčních klopných obvodů a podobně.

9. Literatura

[1] - http://dhservis.cz

[2] - http://ben.cz

[3] - http://elektronika.kvalitne.cz

[4] - http://hw.cz

[5] - http://mcu.cz

[6] - http://microchip.com

[7] - http://asix.cz

[8] - http://elcad.cz/eagle/

[9] - Microchip PIC16F877A Data Sheet

[10] - Microchip 24AA1025/24LC1025/24FC1025

[11] - User's Guide EL-12864 LCM (Licquid Crystal Diplay Module)

10. Přílohy