Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

2

REGISTRY

Registr je speciální paměť s kapacitou 8 bitů, kterou si můžeme představit takto:

7	6	5	4	3	2	1	0

Vidíme, že očíslování těchto bitů je takové, že nejméně významný bit začíná nulou ($2^0 = 1$).

Registr může obsahovat čísla od 0 do 255 (kladná čísla, neobsahující záporné hodnoty), nebo od –128 do +127 (celá čísla se znaménkovým bitem v bitu 7), nebo hodnoty reprezentované znaky v ASCII kódu (např. „A“), nebo jen osm osamocených bitů nemajících dohromady nějaký význam (např. pro osm signálů sloužících k zapínání/vypínání nějakých vnějších zařízení jako např. signalizačních LED, spínání pomocných relé, nebo naopak ke snímání stavu vnějších kontaktů apod.).

Speciální charakter registrů při srovnání s jinou pamětí se projeví tím, že:

- Mohou být použity přímo registrovými instrukcemi.
- Operace s jejich obsahem potřebují instrukce s pouze jedním slovem.
- Jsou spojeny přímo s centrální procesorovou jednotkou, s akumulátorem.
- jsou zdrojem i cílem při výpočtech, provádění instrukcí.

AVR MCU mají 32 registrů, označené R0 až R31. Můžeme jim však přiřadit další jména pomocí assemblerovské direktivy. Například:

```
.DEF MujRegistr = R16
```

Přitom direktiva assembleru, jako je tato, není assemblerem přeložena do nějakého kódu spustitelném na cílovém čipu AVR. Místo používání jména registru R16 můžeme nyní používat vlastní, námi definované, jméno MujRegistr, které můžeme použít místo R16 v příkazech. Takto napíšeme v zdrojovém textu o něco více znaků při používání tohoto registru, ale máme možnost mít název registru spojen s významem jeho obsahu.

Použití příkazové řádky

```
LDI MujRegistr, 150
```

znamená: *load the number 150 immediately to the register R16*, **L**oa**D** **I**mmediate (ulož číslo 150 přímo do registru R16). Tato instrukce uloží pevnou hodnotu nebo konstantu do tohoto registru. Následné přeložení tohoto kódu do programové paměti čipu AVR vypadá nějak takto:

```
000000 0xE906
```

Kód instrukce LDI, stejně jako cílový registr (R16) i hodnota konstanty (150) jsou částí hodnoty E906, i když to není snadno vidět. Není však nutné, abychom (ručně) vytvářeli sami tento strojový kód, protože to je úkolem překladače – assembleru, který sám vygeneruje E906.

V instrukci mohou být použity dva různé registry. Nejjednodušší instrukcí tohoto typu je instrukce MOV. Ta kopíruje obsah jednoho registru do druhého registru, jako toto:

```
.DEF MujRegistr = R16
.DEF DalsiRegistr = R15
  LDI MujRegistr, 150
  MOV DalsiRegistr, MujRegistr
```

První dva řádky tohoto programu jsou direktivy které definují nová jména registrů R16 a R15 pro assembler. Tyto řádky nevytvářejí žádný kód pro AVR. Řádky zdrojového kódu s LDI a MOV vytvářejí kód:

```
000000 0xE906
000001 0x2F01
```

Tyto příkazy zapisují 150 do registru R16 a kopírují jeho obsah do cílového registru R15.

Důležitá poznámka: *První registr je vždy cílový, zapisuje se do něho výsledek!*

(Pro začátečníka je to trochu nezvyklé, protože jsme, např. při psaní, zvyklí spíše na pohyb zleva doprava. Je to však obvyklá konvence, nicméně je to jeden z důvodů, proč se začátečníkům zdá assembler být složitý.)

Rozdílné registry

Když začínáme programovat v AVR assembleru, můžeme zkusit napsat místo výše uvedených řádků programu kód, který vypadá např. takto:

```
.DEF DalsiRegistr = R15
  LDI DalsiRegistr, 150
```

To je ale špatně! Protože: pouze **registry od R16 do R31 se smějí používat v instrukci LDI – uložení konstanty přímo do registru, R0 až R15 přitom nelze použít k tomuto účelu**. Toto omezení není příliš pěkné, ale musíme na něj pamatovat při programování pro AVR.

Existuje jedna výjimka z tohoto pravidla: nastavení registru na nulu. Příkaz

```
CLR MujRegistr
```

je platný pro všechny registry.

Kromě omezení u instrukce LDI je stejné omezení na používání stejné třídy registrů i u následujících dalších instrukcí:

- **ANDI Rx,K** ; bitové And registru Rx s konstantní hodnotou K,
- **CBR Rx,M** ; vymazání všech bitů v registru Rx, které jsou nastaveny na jedničku v konstantní masce hodnoty M,
- **CPI Rx,K** ; srovnání obsahu registru Rx s konstantní hodnotou K,
- **SBCI Rx,K** ; odečtení konstanty K a aktuální hodnoty příznaku přenosu od aktuální hodnoty registru Rx a uložení výsledku do registru Rx,
- **SBR Rx,M** (= ORI Rx,M) ; nastavení všech bitů v registru Rx na jedničku, které jsou jedničkové ke konstantní masce M,
- **SER Rx** ; nastavení všech bitů v registru Rx na jedničku (rovná se LDI Rx,255),
- **SUBI Rx,K** ; odečtení konstanty K od obsahu registru Rx a uložení výsledku do registru Rx.

Ve všech těchto instrukcích musí být registry mezi R16 a R31! Chcete-li používat tyto instrukce, musíte si vybrat některý z těchto registrů pro použití těmito instrukcemi. Tím se zjednoduší programování. To je dalším důvodem proč používat direktivu definující jméno registru, protože se snadněji dá měnit používání registrů.

Registry – ukazatele

Velmi zvláštní úlohu mají dvojice registrů R26:R27, R28:R29 a R30:R31. Důležitost těchto dvojic registru je podtržena tím, že mají zvláštní jména v assembleru: X, Y a Z. Tyto dvojice jsou 16bitové registry – ukazatele, schopné ukazovat na adresy s max. 16 bity v paměti SRAM (X, Y nebo Z) nebo v programové paměti (Z).

Dolní byte 16bitové adresy je umístěn v dolním registru, horní byte v horním registru. Obě části mají svá vlastní jména tj. vyšší byte ze Z je pojmenován ZH (=R31), dolní byte je ZL (=R30). Tato jména jsou definována ve standardních hlavičkových souborech pro MCU. Rozdělení těchto 16bitových ukazatelů do dvou různých bytů se dá udělat třeba takto:

```
.EQU Adress = RAMEND ; RAMEND je nejvyšší 16bitová adresa v SRAM
  LDI YH,HIGH(Adress) ; nastaví MSB
  LDI YL,LOW(Adress) ; nastaví LSB
```

Přístup prostřednictvím ukazatelů se programuje pomocí speciálně navržených instrukcí. Ke čtení se využívá LD (LoaD), pro zápis ST (STore), např. s ukazatelem X (X-pointer), *tab. 2.1.*

Tab. 2.1 Instrukce pro přístup prostřednictvím ukazatelů

Ukazatel	Popis	Příklad
X	Zápis/čtení z adresy X, ukazatel se nemění	LD R1, X ST X, R1
X+	Zápis/čtení z adresy X a poté inkrementace ukazatele o 1	LD R1, X+ ST X+, R1
–X	Dekrementace ukazatele o 1 a pak zápis/čtení z nové adresy	LD R1,–X ST –X, R1

Obdobně můžeme používat i Y a Z.

Existuje pouze jedna instrukce pro čtení z programové paměti. Je definovaná pro ukazatelovou dvojici Z a jmenuje se LPM (Load from Program Memory). Tato instrukce kopíruje byte na adrese Z v programové paměti do registru R0. Protože programová paměť je organizována po slovech (jedna instrukce na jedné adrese je složená z 16 bitů nebo dvou bytů nebo jednoho slova) nejméně významný bit vybírá dolní nebo horní byte (0 = dolní byte, 1 = horní byte). Z tohoto důvodu původní adresa musí být násobena 2 a přístup je omezen na 15 bitů neboli 64kB programovou paměť. Např.:

```
LDI ZH,HIGH(2*Adress)
LDI ZL,LOW(2*Adress)
LPM
```

Po těchto příkazech musí následovat inkrementace adresy, aby bylo možné přistupovat k následujícímu bytu v programové paměti. Protože tento případ nastává dosti často, je kvůli tomu definována speciální instrukce inkrementující ukazatel, aby mohla být takto použita:

```
ADIW ZL,1
LPM
```

8

PRVNÍ PROGRAM

Dosud jsme si uváděli jen fragmenty programů. Naše znalosti jsou nyní už takové, že můžeme začít v AVR assembleru programovat úplné, jednoduché aplikace. Zdrojový kód našeho programu přeložíme AVR assemblerem, čímž dostaneme výsledný strojový kód, kterým naprogramujeme mikrokontrolér. Ten bude připojen k vnějším periferiím, s nimiž bude spolupracovat při provádění námi napsaného programu. Bude to, stejně jako další programy, které si v této knížce uvedeme, jen školní, cvičný program. Pro odzkoušení takových programů se obvykle používá nějaký **Start Kit**. Firma ATMEL, kromě vlastních čipů, vyrábí i startkity. Jeden z posledních je STK500. Pro STK500 (stejně jako pro STK100, STK200 …) najdeme řadu ukázkových programů jak na CD či www stránkách firmy ATMEL, tak na webovských stranách dalších firem, škol i soukromých osob, hlavně studentů. Startkit STK500 lze dokonce zdarma získat při účasti na některé vývojářské konferenci firmy ATMEL. Účast na těchto konferencích není vázána na konferenční poplatek. Bohužel se ale tyto konference konají jen ve velkých městech v USA či Kanadě. Můžeme však použít i nějakou vlastní konstrukci startkitu. Budeme předpokládat použití startkitu s AT90S8515, uvedeného v příloze. Je navržen tak, že obsahuje pouze MCU AT90S8515 s krystalem, obvody resetu, 5V stabilizátor a potom už jen konektor pro připojení ISP programátoru a čtyři konektory pro připojení vnějších periferií k PORTA, PORTB, PORTC a PORTD. Piny těchto portů jsou ještě přes pull-up odpory připojené k +5 V. Kromě toho je možnost zapojit obvod MAX232 pro převod TTL signálů z vnitřního UARTu na úrovně RS232 signálů, přivedené na 9pinový konektor CANON. K portu B připojíme 8 LED. LED diody budou mít anody připojené přes vhodný odpor na +5 V, katody na piny PORTB0, …, PORTB7. Port D připojíme k osmi tlačítkům tak, že jeden jejich kontakt bude připojen na PORTD0, … PORTD7, druhé kontakty budou připojeny k 0 V (*obr. P1.1*). Stejně jsou připojena tlačítka a LED diody v STK500, takže si můžeme bez úpravy odzkoušet i aplikace původně napsané pro STK500.

Jako první aplikaci si napíšeme program, který bude střídavě rozsvěcet/zhasínat liché a sudé LED diody připojené k portu B. Nejdřív si napíšeme program tak, aby vykonával tuto činnost. Protože tato první verze programu není naprogramována příliš elegantně, naprogramujeme příklad 2 a 3 jako elegantnější variantu příkladu 1. V úvodu této publikace jsme se zmínili, že i pro stejný typ procesoru může

existovat několik assemblerů. Popíšeme si práci pomocí WAVRASM verze 1.3 spuštěného přímo jako WAVRASM.exe a dále pomocí AVR Studia verze 3.2, který AVR makroassembler verze 1.3. obsahuje jako součást. Přeložený soubor pak pomocí AVR PROG verze 1.31 a programátoru s AT90S1200 (popsaného v příloze) pak naprogramujeme do AT90S8515.

Nejprve si uvedeme zdrojový kód řešící střídavé rozsvěcování lichých a sudých LED:

```
;************ priklad1 **************

.include "8515def.inc"

.def Temp = R16
.def Delay1 = R17
.def Delay2 = R18
.def Delay3 = R19

;******** inicializace
RESET:
   ser Temp              ; nastavuje všechny bity v Temp na 1
   out DDRB,Temp         ; nastavuje PORTB jako vystup
   ser Delay1
   ser Delay2
   ser Delay3

LOOP:
   ldi Temp,0b01010101
   out PORTB,Temp

DLY1:
   dec Delay1
   brne DLY1
   dec Delay2
   brne DLY1
   dec Delay3
   brne DLY1

   ldi Temp,0b10101010
   out PORTB,Temp

DLY2:
   dec Delay1
   brne DLY2
   dec Delay2
   brne DLY2
```

```
dec Delay3
brne DLY2

rjmp LOOP
```

Program začíná direktivou assembleru **.include „8515def.inc**“, která ukládá assembleru připojit hlavičkový soubor 8515def.inc. Tento soubor umístíme do stejného adresáře, kde bude umístěn i výše uvedený zdrojový kód. Zdrojový kód budeme mít např. v souboru *priklad1.asm*. Potom následují direktivy **.def** přidělující registrům R16, R17, R18 a R19 jména Temp, Delay1, Delay2 a Delay3. Instrukcí **ser Temp** nastavíme v registru R16 všechny bity na 1.

Jako další se provede **out DDRB,Temp**, čímž se nastaví všechny piny portu B jako výstupní. Definice DDRB je uvedena v souboru *8515dec.inc*, hodnoty v jednotlivých bitech DDRB určují, zda odpovídající pin portu B je vstupní nebo výstupní. Jednička nastavuje příslušný pin jako výstupní, nula jako vstupní. Protože v předchozí instrukci jsme v Temp, tj. v R16, nastavili všechny bity na 1, budou po zkopírování do DDRB pomocí *out* nastaveny všechny piny jako výstupní.

Poznámka: *Možná, že se vám zdá divné, proč k poslání samých jedniček do DDRB potřebujeme dvě instrukce. Nebylo by jednodušší použít jedinou instrukci* ***out DDRB, 0b1111111****? Instrukce out, stejně jako některé další, totiž nemůže mít jako parametr konstantu, parametrem může být jen registr.*

Hlavní myšlenkou řešení našeho příkladu je střídavé posílání čísel 0b01010101 a 0b10101010 na výstup portu B. Jak jsme si právě řekli, parametrem instrukce *out* nemůže být konstanta, takže jinak hezké *out PORTB, 0b01010101* a *out PORTB, 0b10101010* je chybou a proto napíšeme např.

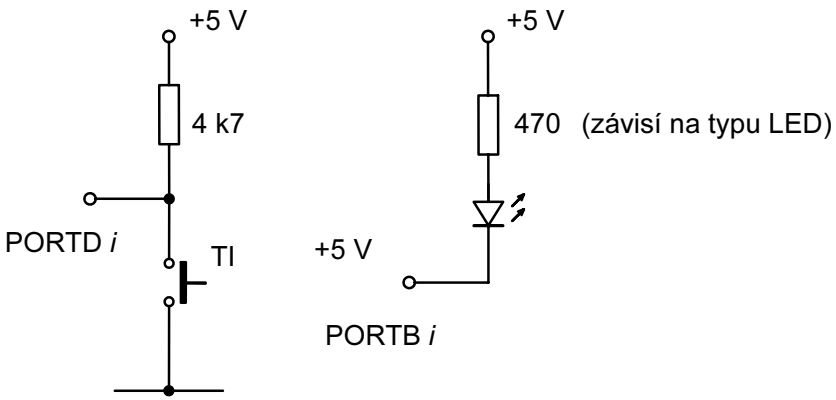
```
ldi Temp,0b01010101
out PORTB,Temp
```

a

```
ldi Temp,0b10101010
out PORTB,Temp
```

Instrukce **ldi** uloží konstantu do registru a tento registr je již použitelný jako parametr instrukce **out.** Protože chceme, aby docházelo k střídavému rozsvěcování lichých a sudých LEDek, potřebuje střídavě posílat 0b01010101 a 0b10101010 do portu B. Umístíme proto do zdrojového kódu návěští **LOOP** a skok **rjmp LOOP**, a mezi tato dvě místa instrukce pro ovládání LED na portu B. Máme tak naprogramovanou nekonečnou smyčku:

PŘÍLOHA 1



Obr. P1.1 *Zapojení osmi tlačítek a osmi LED připojovaných k PORTU D a PORTU B AT90S8515 tak, jak jsou používány programy v Příkladu 1 až Příklad 11 – kompatibilita s kitem STK500*

PŘÍLOHA 2

PŘÍKLADY ZAPOJENÍ S MCU AVR

Pro usnadnění získávání prvních praktických zkušeností s MCU AVR, účely vývoje nových zařízení na jedné straně a seznamování se s činností AVR studenty na druhé straně dodává ATMEL i několik Start kitů, např. již zmiňovaný STK500.

Můžeme použít i nějakou vlastní konstrukci. Pro účely výuky studentů SPŠE jsem vyvinul a realizoval několik konstrukcí umožňující seznámit se s činností některých mikroelektronických prvků a s jejich programováním (FPGA firmy ALTERA, PIC16F84 Microchipu, AT89S8252 firmy ATMEL a konečně AVR MCU základní řady). Navrhoval jsem je tak, aby na jednostranné desce plošných spojů s nepříliš složitým obrazcem byl v objímce např. MCU AVR. Je přitom použito základní katalogové zapojení. Kromě napájení obsahuje již jen obvody RESETu, krystal s kondenzátory a 10pinové konektory pro všechny porty MCU. Potom ještě jeden konektor pro spojení s ISP programátorem a 9pinový konektor CANON pro případnou komunikaci pomocí RS-232. Pro tyto účely obsahuje deska ještě patici pro MAX232 a příslušné kondenzátory. Tento obvod nemusíme mít osazený, nepředpokládáme-li sériovou komunikaci RS-232 (a dále u typu AT90S1200, který neobsahuje UART).

Protože zapojení 10pinových konektorů připojených k PORTům je na všech deskách (obsahujících různé typy MCU či jednočipových mikropočítačů) stejné, lze mít jednotnou sadu periferií – spínače DIP, LED výstupy, LCD displeje, A/D převodník, klávesnice atd. Jde o obdobný systém, který je používán v [14], [15] a [16].

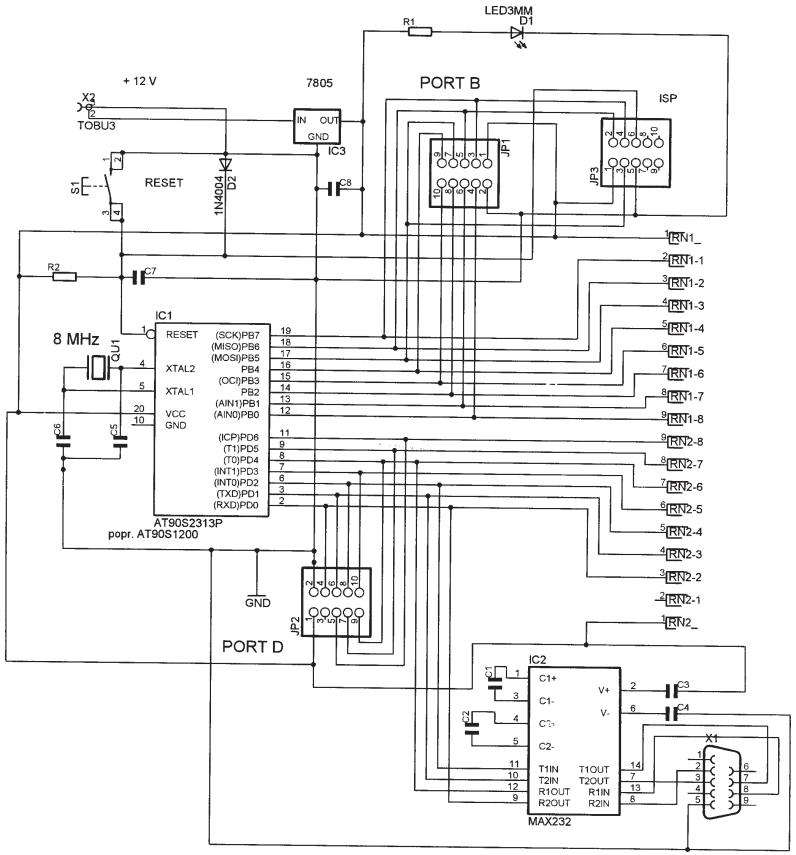
Zapojení základní desky pro AT90S1200 či AT90S2313, rozložení součástí a desky spojů jsou na *obr. P2.1*, *obr. P2.2* a *obr. P2.3* (pozn. AT90S1200 neosahuje UART, neosazovat MAX232).

Další tři obrázky, *obr. P2.4*, *obr. P2.5* a *obr. P2.6* představují základní desku s AT90S8515, popř. AT90S8535, rozložení součástí a spoje.

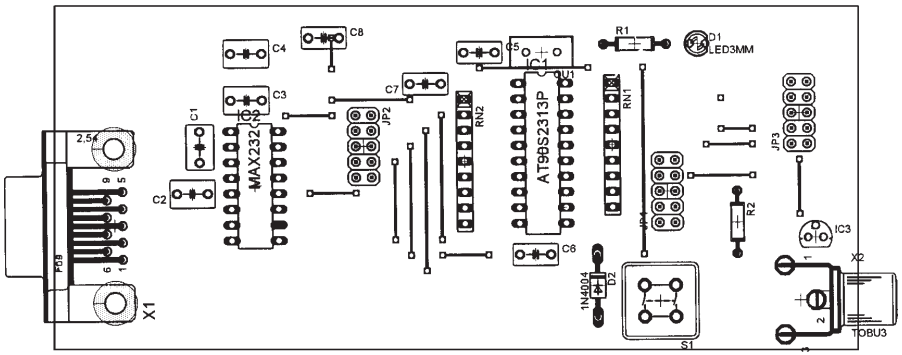
Protože typ AT90S8535 je v naší maloobchodní síti dodáván většinou v provedení PLCC, můžeme pro tento typ použít startkit uvedený na obr. *P2.7*, *P2.8*, *P2.9.*

Programovací jazyky pro AVR

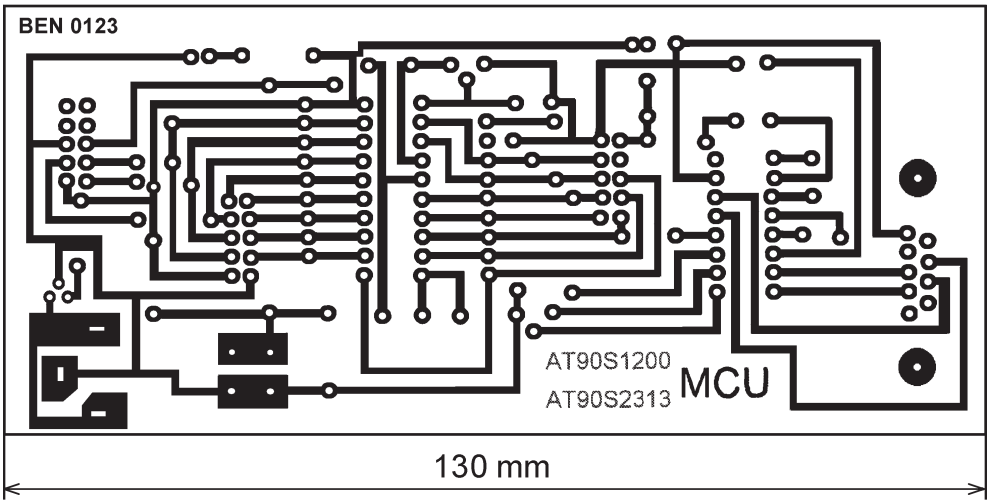
Protože tento díl není učebnicí programování AVR, seznámíme se s programovacími jazyky pro AVR jen velice krátce pro získání představy, co je k dispozici, zejména FREE verze či verze pro studenty.



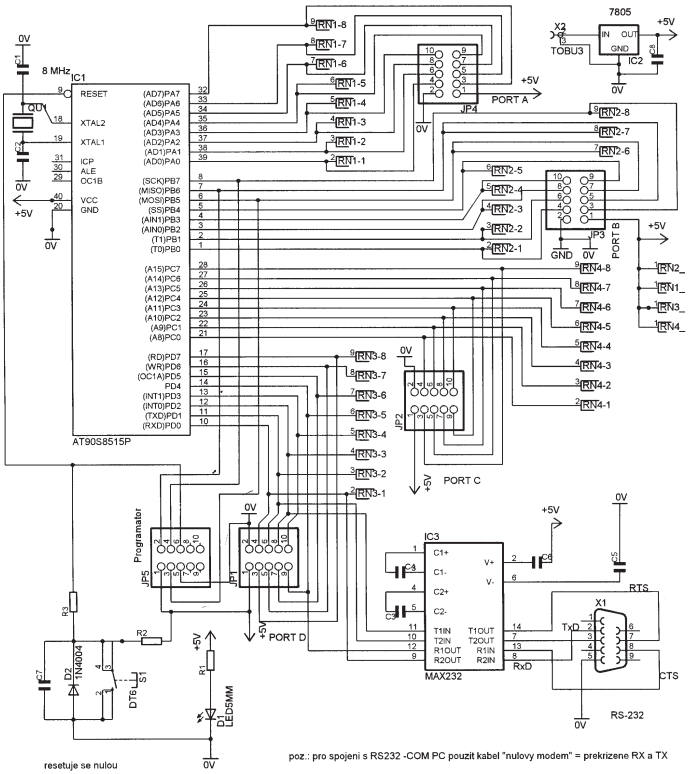
Obr. P2.1 *Základní deska – startkit pro AT90S1200 či AT90S2313*



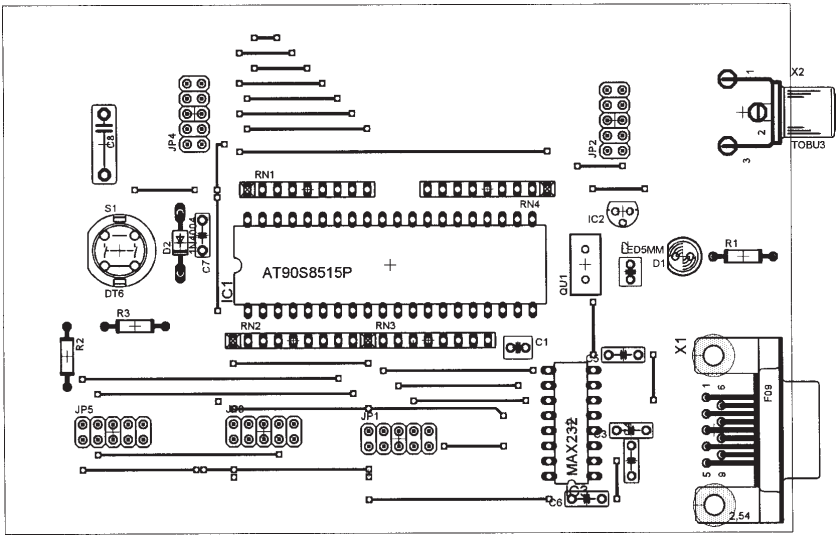
Obr. P2.2 *Rozložení součástí startkitu s AT90D2313*



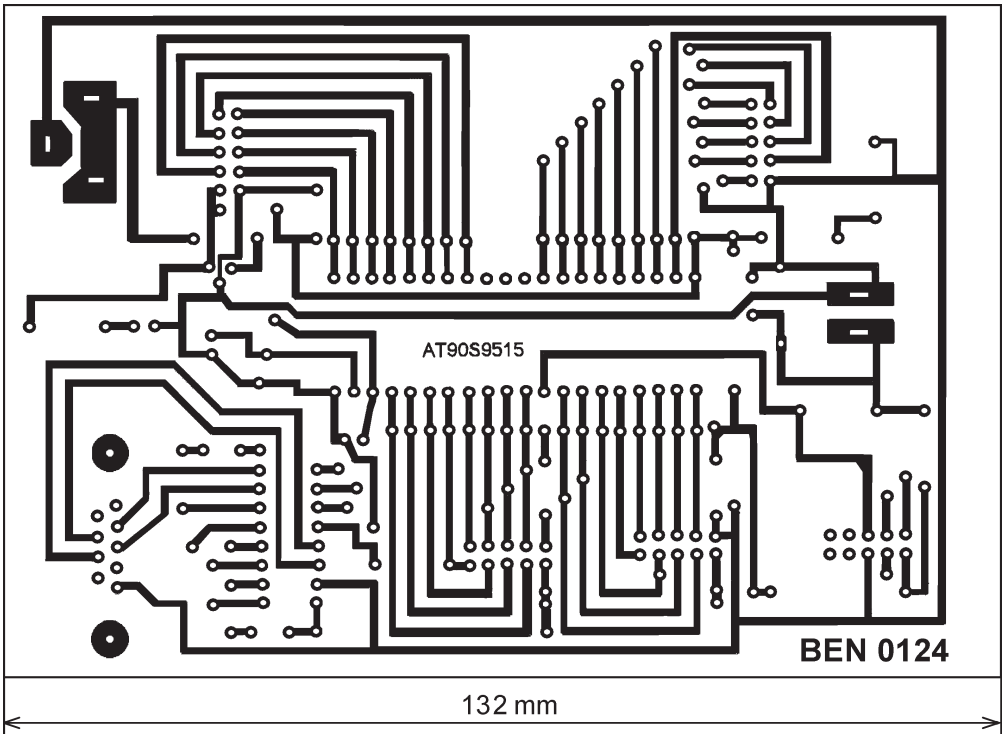
Obr. P2.3 *Schéma plošných spojů startkitu s AT90S2313, skutečná velikost (130 mm × 56 mm), BEN 0123*



Obr. P2.4 *Zapojení základní desky s AT90S8515*



Obr. P2.5 *Rozložení součástí kitu s AT90S8515*



Obr. P2.6 *Spoje kitu s AT90S8515, skutečná velikost (132 mm × 89 mm) – BEN 0124*