Využití platformy Arduino pro výuku programování mikropočítačů v jazyce symbolických adres

Bc. Štěpán Pavelka

Diplomová práce
2019

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Štěpán Pavelka**
Osobní číslo:	**A17368**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Učitelství informatiky pro střední školy**
Forma studia:	**prezenční**
Téma práce:	**Využití platformy Arduino pro výuku programování mikropočítačů v jazyce symbolických adres**
Téma anglicky:	**Using the Arduino Platform for the Tuition of Microcontroller Programming in the Assembly Language**

Zásady pro vypracování:

1. **Zpracujte literární rešerši na téma výuky programování mikropočítačů v jazyku symbolických adres.**
2. **Navrhněte a sestavte vhodné softwarové i hardwarové prostředky umožňující využití platformy Arduino k výuce programování v jazyku symbolických adres na střední škole.**
3. **Implementujte vhodné ukázkové programy v jazyku symbolických adres.**
4. **Vytvořte prezentace k teoretické části výuky.**
5. **Vypracujte podklady pro studenty a učitele pro praktickou část výuky.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **BARR, Michael a Anthony J MASSA. Programming embedded systems: with C and GNU development tools. 2nd ed. Sebastopol: O'Reilly, 2006, xxi, 301 s. ISBN 978-0-596-00983-0.**
2. **CATSOULIS, John. Designing embedded hardware. 2nd ed. Sebastopol, CA: O'Reilly, 2005, xvi, 377 p. ISBN 0596007558.**
3. **MARGOLIS, Michael. Arduino cookbook. 2nd ed. Sebastopol, Calif.: O'Reilly, 2012, xx, 699 p. ISBN 1449313876.**
4. **MATOUŠEK, David. Práce s mikrokontroléry ATMEL AVR ATmega16. 1. vyd. Praha: BEN - technická literatura, 2006, 319 s. uC. ISBN 80-730-0174-8.**
5. **MAZIDI, Muhammad Ali, Sarmad NAIMI and Sepehr NAIMI. The AVR microcontroller and embedded systems: using Assembly and C. Upper Saddle River, N.J.: Prentice Hall, 2011, xiv, 776 p. ISBN 01-380-0331-9.**
6. **PINKER, Jiří. Mikroprocesory a mikropočítače. 1. vyd. Praha: BEN - technická literatura, 2004, 159 s. ISBN 80-7300-110-1.**

Vedoucí diplomové práce: **Ing. Jan Dolinay, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání diplomové práce: **3. prosince 2018**

Termín odevzdání diplomové práce: **15. května 2019**

Ve Zlíně dne 7. prosince 2018

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Jméno, příjmení: Štěpán Pavelka

Název diplomové práce: Využití platformy Arduino pro výuku programování mikropočítačů v jazyce symbolických adres

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 15. 5. 2019

.....
podpis diplomanta

ABSTRAKT

Diplomová práce se věnuje přechodu výuky jazyku symbolických adres z již téměř 40 let starého mikropočítače 8051 na němž byla výuka assembleru na střední škole prováděna. Práce navrhuje novější mikropočítač (a novější mikrokontrolér) jenž je cenově dostupný pro každého studenta mj. z důvodu, aby si sám kontrolér mohl nosit domů a na práci pokračovat doma. Práce popisuje aktuální řešení školy, doplňuje jej praktickými příklady a navrhuje řešení, které by výuku assembleru mohlo ještě více zjednodušit oproti stávajícímu stavu.

Dále implementaci ukázkových programů s podrobným vysvětlením u některých z nich, popisu prezentací k teoretické části výuky a závěrem také podkladům pro studenty a učitele, jenž jsou právě kromě prezentací a ukázkových programů také další podklady a materiály.

Klíčová slova: Assembler, Jazyk symbolických adres, Arduino, AVR, CISC, RISC

ABSTRACT

The diploma thesis deals with transition of teaching Assembly language from at least 40 years old microcontroller 8051 where assembly language teaching was carried out. The thesis proposes a newer microcomputer (and newer microcontroller) that is affordable for students, among other things students can carry controller to their home and work with him. Thesis describes the current solution of the school, complements it with practical examples and proposes solution that could simplify teaching assembly language against current state.

Implementation of sample programs with detailed explanation in some of them, a description of theoretical presentation part of the curse and in the end of thesis is basis for studens and teachers which are besides of presentation and sample programs also other study materials.

Keywords: Assembler, Assembly language, Arduino, AVR, CISC, RISC

Poděkování, motto a čestné prohlášení, že odevzdaná verze bakalářské/diplomové práce a verze elektronická, nahraná do IS/STAG jsou totožné ve znění:

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Stále více se v dnešní době rozšiřuje programování ve vyšším programovacím jazyce. Je to logické, jelikož není až na výjimky důvod, programovat v nižších programovacích jazycích. Člověk(žák) se daleko lépe naučí programovat v „jazyku mu bližším“ než jazyku bližšímu pro samotný stroj. Nicméně i přesto se ještě dnes můžeme výjimečně setkat s firmami které potřebují své přístroje naprogramovat právě nižším programovacím jazykem (ať už důvody, které vedou firmu k tomuto kroku jsou jakékoli) nemůžeme říci, že nižší programovací jazyky zanikli a nepoužívají se, jelikož právě opak je pravdou.

Cílem této práce je vytvořit výukové zařízení pro výuku nižšího programovacího jazyku: jazyku symbolických adres – v angličtině Assembly Language (ale slangový název je Assembler). Jeho výuka na střední škole je zde zařazena hned ze dvou důvodů. První zde byl zmíněn, a to ten že se absolvent střední školy bude moct uplatnit znalosti programování nižším jazykem a druhý, který mu tedy předchází, a to, aby sama škola studentům ukázala, že existují i nižší programovací jazyky než právě i ty vyšší (se kterými se nejspíše budou setkávat častěji) a jenž jsou vyučovány v podobných předmětech.

První část práce popisuje mikrokontroléry, jenž jsou alfou a omegou programování ve strojovém kódu, dále mikropočítače (jenž nám programování a celkovou práci s mikrokontroléry usnadňují) a porovná zastaralé řešení střední školy (hardware který škola používala) s novým hardwarovým a dále i softwarovým řešením k výuce assembleru a jak jej nastavit.

Druhá část je zaměřena prakticky. Dle bodů zadaní byly vytvořeny ukázkové programy, jež ty klíčové jsou popsány více detailněji. Hlavní náplní práce pro výuku již v aktuálním roce byly výukové prezentace a závěrečná část praktické části se zabývá podkladovými materiály pro studenty a vyučující, jenž jsou nad rámec právě programů a prezentací.

I. TEORETICKÁ ČÁST

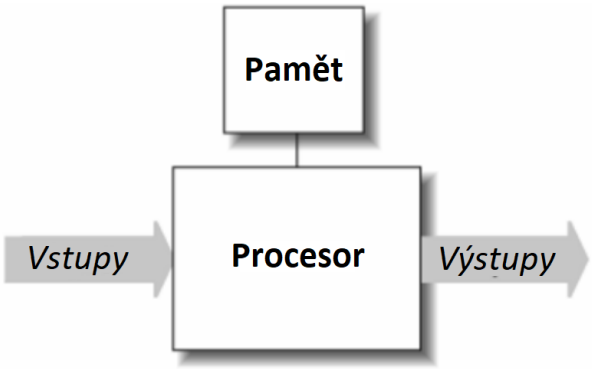
1 VÝUKA PROGRAMOVÁNÍ V JAZYKU SYMBOLICKÝCH ADRES

Letos je to přesně 70 let, co vnikla druhá generace nižšího programovacího jazyka. Tou první, která vznikal ještě dříve, byl prostý zdrojový kód, tedy programátor si musel pamatovat kódy jednotlivých instrukcí. Počítače vyráběné od druhé poloviny 50 let již umožňovali využít druhou generaci tohoto programování. Zde si již programátor nemusel pamatovat kód jednotlivé instrukce a postačovalo pouze znát instrukční sadu pro daný procesor.

Assembly language (anglicky jazyk symbolických instrukcí) má tedy různé instrukční sady pro různé procesory. Je to logické, že např. 32bitový procesor může mít instrukce pro práci s desetinnými místy, což např. 8bitový mít nemusí. Program napsaný v assembleru musí být někde uložen a vykonávat se krok po kroku. To může být realizováno dvěma způsoby.

1.1 Embed systémy, architektury a instrukční sady

Embed systém je sytém obsluhující několik vstupů a výstupů. Například mikrovlnná trouba má jako vstup tlačítka na předním panelu nebo teplotní sondu, výstupem může být displej mikrovlnné trouby nebo samotný mikrovlnný zářič. Výstupy embed systému jsou téměř vždy funkcí vstupních faktorů (čas, teplota, stisk tlačítka) [Chyba**! Nenalezen zdroj odkazů.**, str. 12]



Obrázek 1: Embed systém

Dnešní mikropočítače zvládají ovládat mikrovlnou troubu, jak přirovnal Michael Barr. Ten také již před 15 lety, rozdělil embed systémy do tří hlavních skupin. (Tabulka 1) Dle mého názoru je tato tabulka platná i dnes, kromě položek vývoje, kdy dříve byl vývoj dražší a

životnosti, která by dnes měla být ideálně dekády pro všechny tři třídy. [Chyba**! Nenalezen zdroj odkazů.**, str. 16]

Požadavky pro embeded systém			
Kritérium	**Nízká (třída)**	**Střední (třída)**	**Vysoká (třída)**
Procesor	4 nebo 8bitové	16bitové	32bitové
Paměť	méně než 64 KB	méně než 1 MB	více než 1 MB
Cena vývoje	< 100 000 $	< 1 000 000 $	> 1 000 000 $
Cena produktu	Méně než 10 $	Méně než 100 $	Více než 100 $
Životnost	Dny, týdny, měsíce	Roky	Desetiletí
Spolehlivost	Občas může selhat	Musí být spolehlivé	Ochrana proti selhání

Tabulka 1: rozdělení Embed systémů

(Mikro)počítače dělíme dle architektury na Hardvardskou a Von Neumannovu. Jejich hlavní rozdíl je ten, že v Hardvardské je oddělena paměť programu a dat, kdežto ve Von Neumannově architektuře nikoli.

1.1.1 Hardwardská architektura

Mezi nespornou výhodu této architektury právě pro využití mikrokontrolérů AVR spadá mj. to, že architektura umožnuje zároveň přistupovat do paměti programu i paměti dat.

1.1.2 Von Neumannova architektura

Tato architektura má společnou paměť programu a dat, je pomalejší ovšem má připojeny vstupy a výstupy přímo na ALU, tedy ve výjimečných případech může být rychlejší než Hardwardská, to však pouze u těch nejvíce primitivních programů. Žádný z mikrokontrolérů AVR ji nevyužívá.

1.1.3 CISC (Complex Instruction Set Computing)

Tato instrukční sada má téměř veškerý obsah instrukcí, má velmi široký okruh funkcí, které by jinak šli naprogramovat i pomocí jednodušších strojových instrukcí. Například, pokud chceme vynásobit obsah dvou paměťových buněk, bude nám k tomu stačit jedna instrukce. Že ve skutečnosti tato sada instrukci musí ještě požít preprocesor a výslednou operaci provede za více hodinových cyklů, je již právě „daň za onu komplexnost“.

1.1.4 RISC (Reduced Instruction Set Computing)

Jak z názvu vyplývá, jedná se o redukovanou sadu. Proto nemůžeme například násobit jednotlivé buňky paměti, ale základními instrukcemi si postupně obsah pamětí přesunout do registrů, v registrech provést násobení a poté výsledek zapsat na požadovanou adresu. Na první pohled by se zdálo že pro osobu, jenž bude využívat redukovanou sadu, „přibyde“ více instrukcí, které musí napsat, aby program provedl požadovanou operaci, nicméně, zde platí že téměř všechny instrukce se provedou během strojového cyklu.

Vlastnosti RISC jsou:

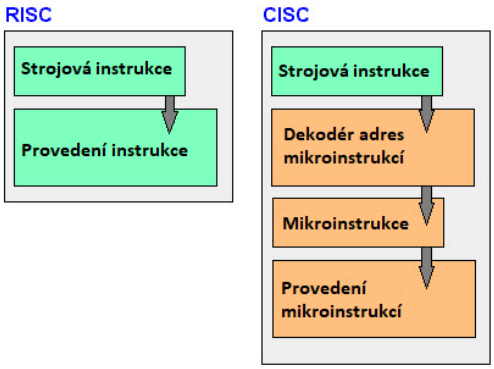
- V každém hodinovém cyklu by měla být dokončena jedna instrukce
- Instrukce mají pevnou délku a jednotný formát který vymezuje význam jednotlivých bitů.
- Je použit vyšší počet registrů
- Celkový počet instrukcí a způsob adresování, je malý

1.1.5 Ukázka rozdílu mezi CISC (8051) a RISC (ATmega328).

Níže je program v jazyku symbolických adres, jenž má vynásobit dvě paměťová místa (konkrétně obsah paměti na adresách 12 a 14):

CISC:	RISC:
MUL $12,$14	MOV R1,$12
	MOV R2,$14
	MUL R1,R2
	MOV $12,R1

Při prvním pohledu se může zdát, že CISC bude s jedním příkazem rychlejší, ale nenechme se zmást. Zatímco u RISC trvá obvykle jedna instrukce jeden hodinový cyklus („tik“) u CISC je tomu samozřejmě jinak. Podívejme se na následující (Obrázek 2). U CISC převezme (načte) strojovou instrukci, poté dekóduje, jaké další instrukce ještě musí provést, ty si uloží do paměti (RAM) a v posledním bodu je provede. Pokud se tedy podíváme do manuálu výrobce, zjistíme, že násobení trvá na 8051 čtyři strojové cykly (a jeden strojový cyklus trvá na 8051 dvanáct hodinových cyklů). Celkově tedy násobení na 8051 provede 48 hodinových cyklů.



Obrázek 2: Zpracování RISC a CISC instrukcí

Nyní program psaný v RISC. Jak již víme, většina instrukcí trvá jeden hodinový cyklus, tedy první dva řádky (načtení do registrů) trvají 2 hodinové cykly. Samotné násobení dvou registrů (jedna instrukce) trvá také dva cykly, a následně přesunujeme výsledek zpět což je jeden hodinový cyklus. V RISC variantě tedy násobení dvou osmibitových čísel trvalo pět hodinových cyklů, kdežto v CISC těchto cyklů proběhlo 48.

Přehledné shrnutí tedy je:

	CISC	RISC
Časová složitost instrukcí:	může probíhat mnoho hodinových cyklů	většina trvá jeden hodinový cyklus
Práce s pamětí:	jednoduchá	složitější
Instrukce:	komplexní (například více operandů než dva)	primitivní standardizované instrukce
Počet instrukcí	průměrně 100-200 i více	většinou méně než 100
Instrukce, které mohou přistupovat do paměti:	Load a Store	téměř všechny.

Tabulka 2: Rozdíl mezi úplnou a redukovanou instrukční sadou

1.2 Zastaralé řešení

V 90 letech (a i v následujícím desetiletí) byl jazyk symbolických adres vyučován nejčastěji na mikroprocesoru 8051. Nejspíše největším důvodem, proč tomu tak bylo je, že jej mohl z výrobců vyrábět kdokoli, kdo zaplatil malou část Intelu, který měl na procesor patent. Díky

tomu jej bylo snadnější získat (na českém trhu) než jiné typy mikrokontrolérů. Jeho programování je také jednodušší než u jiných mikrokontrolérů. Teď když známe rozdíl mezi RICS a CISC, jak však dostat program do samotného mikrokontroléru? Mikrokontrolér byl posazen na desce s dalšími obvody které zajišťovali komunikaci (nejen) s počítačem a jeden z nich je právě přes sériový port (standard RS-232) Pro představu si lze tedy celý mikropočítač představit tak jako můžeme vidět výukový KIT pro 8051 na (Obrázek 3) Tato vývojová deska lze však zakoupit dnes již pouze v zahraničí (Indie) a i tak nemusí být záruka, že je skutečně funkční. Jeho cena je sice pod 300,- Kč za kus, nejspíše však není možné jej objednat EU natož České republiky, a jinde než v Indii, jsem ji již ke koupi nenašel.

Obrázek 3: Mikroprocesor 8051 (uprostřed) a jeho vývojová deska.

Střední škola využívala jiné desky (neměli například právě displej), avšak programování bylo stejné bez ohledu na desku. Žáci mohli především ocenit právě jednodušší CISC.

Postupem času však počítače na školách nejen že přestali mít sériový port, ale i samotné desky postupně přestali fungovat. Škola jako vhodnou náhradu zvolila mikropočítač Arduino, jelikož právě jednou z jeho výhod je i mj. to, že má velmi příznivou cenu ze zahraničí (cena kopie i s poštovným je maximálně 200,- Kč), žáci si jej tedy mohou sami pořídit a díky tomu si mohou mikropočítač nosit i domů, kde na úkolech mohou pokračovat, což u starších modelů možné nebylo.

Programování assembleru na platformě Arduino s mikrokontroléry AVR má však úplně jinou instrukční sadu (RISC namísto CISC), vnitřní topologii, počet a číslování registrů a celkově je to znatelná změna. Než však budou popsány rozdíly mezi 8051 a ATmega328, uvedeme pár slov o firmě, jenž nové mikrokontroléry vyrábí.

1.3 Atmel AVR

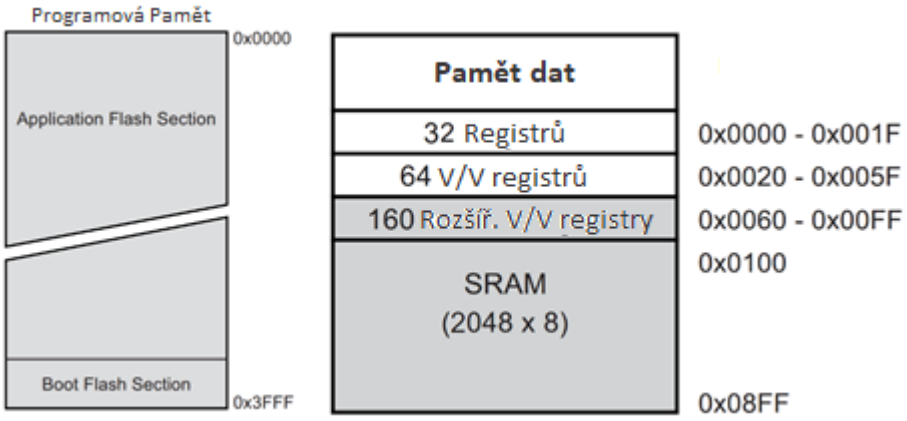
Firma Atmel byla založena roku 1984 a v dnešní době je stále světovou jedničkou v oblasti výroby mikroprocesorů. K dnešnímu dni se však již s názvem firmy Atmel nesetkáme, jelikož jej k 19. lednu 2016 koupila firma Microchip. Mikrokontroléry od firmy Atmel mají rodinu procesorů AVR, u které se běžně pracuje s informací, že zkratka AVR znamená: „Alf and Vegard's RISC procesor" což však samotná firma nikdy nepotvrdila. Mikrokontroléry Atmel AVR dříve byli řadou 8bitových nízko příkonových mikrokontrolérů na jádře AVR využívajícího koncepci Hardwardské a RISC architektury. To znamená, že je zde oddělena paměť pro data a paměť pro program, s redukovanou sadou instrukcí. Výhodou této koncepce je propojení registrů s ALU, jenž provede za jeden hodinový cyklus jednu operaci.

Mikrokontroléry AVR se vyrábějí ve třech řadách lišících se především ve využití. Prvním je ATtiny, které je vhodné pro primitivní a malé elektrické obvody. Druhá řada je ATmega, která má výkonné mikročipy, větší RAM a flash, či více rozhraní oproti předchozímu. Třetí skupina stojí nepatrně bokem od předchozích dvou a to proto, že vznikla později a namísto 8bitového procesoru má procesory 32bitové. Instrukční sada je zde podobná jako RISC, ovšem není kompatibilní.

1.3.1 Architektura AVR

AVR jádro je podobné většině RISC procesorů. Obsahuje 32 obecných 8bitových registrů, které jsou rychle přístupné a mohou obsahovat jak adresy, tak samozřejmě i data. Tato část je přímo propojena s aritmeticko logickou jednotkou (ALU). Na obrázku v tištěné příloze (Příloha P I) je možné vidět zjednodušené schéma AVR mikroprocesoru. Všechny části jsou připojeny na 8bitovou sběrnici. U mnoha jednoduchých mikroprocesorů je obvyklé, že jsou registry zobrazovány přímo v adresovém prostoru dat. Prvních 32bytů paměti (adresy 0x00 – 0x1F) odpovídá registrům R0 až R31. Díky tomu je možno zacházet s každým registrem použitím standartních odkazů(instrukcí) bez toho, aby programátor potřeboval znát řídicí

instrukce registrů. Dále můžeme na schématu vidět ke sběrnici připojené další prvky, jako jsou například čítače/časovače, jednotka přerušení, watchdog či vstupně výstupní linky. Také je zde paměť EEPROM[1], která však umožňuje pouze přibližně 100 000 zápisů, proto nebude ve výukách nijak využívána, protože při nesprávném použití ji může student během několika minut (nevhodný cyklus se zápisem dat) celou zničit.



Obrázek 4: Rozložení Vnitřní paměti (vlevo programová část, vpravo část dat)

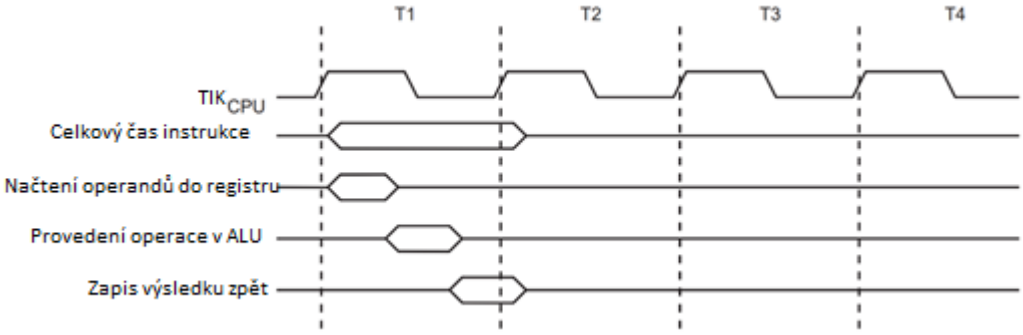
Datová paměť obsahuje také registry Vstupů/Výstupů, a rozšířené registry V/V. Lze tedy konkrétní V/V adresovat přímo adresou, nicméně dnešní vývojová prostředí již zvládají zkratky jednotlivých V/V registrů a není tedy třeba pamatovat jejich adresy v paměti.

AVR má také paměť programu přístupnou pomocí dvoustupňové pipeline. Mikrokontroléry AVR tedy využívají jednoduché, avšak účinný předvýběr instrukce tzv. pipelining. Ten v hodinovém cyklu načte (před vybírá instrukci) a v dalších hodinových cyklech se provádí daná instrukce a zároveň se před vybírá instrukce následující.

Strojový cyklus mikrokontrolérů ATmega přímo odpovídá hodinovému cyklu. Časování jedné instrukce je tak rychlé, že je schopno při jednom strojovém cyklu načíst oba operandy, aritmeticko-logické instrukce, vypočítat výsledek a zapsat jej zpět. (Obrázek 5) Toto umož-

[1] Electrically Erasable Programmable Read-Only Memory

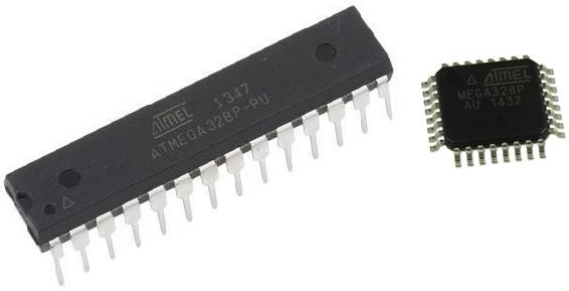
ňuje dosáhnout velkého výpočetního výkonu. Například procesory řady 8051, disponují instrukcemi o délce od 12 do 48hodinových cyklů, navíc musejí využít ke své práci akumulátor, který je v procesoru pouze jeden. [4]



Obrázek 5: Časování jednotlivé instrukce (zdroj: datasheet ATmega32)

1.4 ATmega328 (čip)

Základem mikroprocesoru ATmega32P je pouzdro DIP28[2]. ATmega také pokračuje na své předchůdce ze své rodiny, konkrétně ATmega8 a ATmega16. Nejmenší z nich, ATmega8 ovšem postrádá rozhraní JTAG, to je však jediný rozdíl kromě také samozřejmě toho, že číslo v názvu nám udává velikost flash(programové) paměti tedy u ATmega32 je to 32KB. Co se týče instrukční sady shodné a vyšší verze se liší ještě vyšším počtem SRAM paměti, jinak jsou piny na daných procesorech shodné. Jak již bylo zmíněno, používá RISC.



Obrázek 6: Mikroprocesor ATmega328P v pouzdře DIP28 a MLF (vpravo)

[2] Dual in-line package

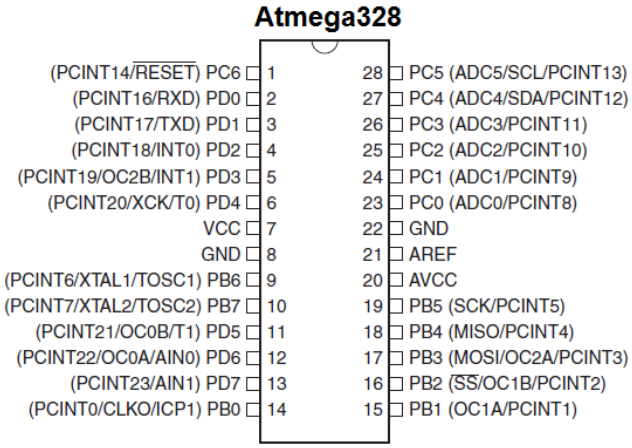
Poznámka: Procesor se může vyskytovat také v jiných pouzdrech, např. v Arduinu Uno r.3 se používají již MLF pouzdra (Obrázek 6) procesor, což má nepatrnou výhodu v úspoře místa, jelikož je procesor menší, nevýhodou že oproti starší variantě nemůže byt posazen do DIP patice ale v případě poruchy by musel byt přímo přepájen. Tedy pokud chceme mít více programů, a měnit je bez připojení k PC, ve verzi s paticí stačí jen prohazovat jednotlivé mikročipy s jejich programy do patice s obvodem, který mikrokontrolér ovládá.

Mikrokontrolér má 3 vstupně-výstupní porty: B, C a D, označované nejčastěji písmeny: P a písmenem portu (např. PD), Všechny tyto porty mohou pracovat při výstupním portu dle výrobce až 40mA z jednoho pinu. Porty jsou sdíleny s dalšími periferiemi (např. USART[3], I2C[4]). Je zde také RC oscilátor, jenž lze kalibrovat. Písmena portů jsou dle abecedy, jen na původním 40pinovém mikrokontroléru (ATmega32) byl právě i port A, který zde (ATmega328) chybí.

1.4.1 Parametry ATmega328P

- Programovatelná 32kB paměť (paměť programu), 2 kB SRAM (paměť dat)
- 1 kB EEPROM (teoreticky až 100 000 cyklů přepsání)
- Instrukční sada se 131 instrukcemi
- 32 8bitových registrů
- Čtyři 8bitové vstupně/výstupní porty
- Hodinový kmitočet 16 MHz
- Napájecí napětí 2,7 – 5,5 V

[3] Universal Synchronous Receiver and Transmitter
[4] I-squared-C



Obrázek 7: Popis pinů ATmega328

Z obrázku (a z tabulky – viz Příloha P II) je patrné, že port B má šest pinů, (B0–B5), C má piny C0 – C6, tedy celkem 7 a port D má piny D0–D7, tedy osm pinů, které mohou plnit jeden bite. Jaké to sebou přináší problémy pro výuku assembleru pro střední školy, je uvedeno v další části, nicméně je patrné, že není možné poslat celý bite na porty B a C.

1.5 Arduino

Arduino je otevřená (open source) platforma, založená na uživatelsky jednoduchém software a hardware. Tato platforma vznikla v Itálii jako jednoduchý nástroj pro studenty bez hlubší znalosti programování, či dokonce elektroniky. Pojmenována je po významné postavě města, ve kterém platforma vznikla. Pro svůj úspěch se začala dále vyvíjet od jednoduchých 8bitových desek, až pro produkty určené pro embedded systémy nebo dokonce 3D tisk.

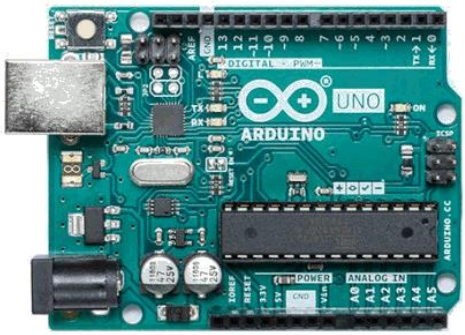
Obrázek 8: Logo společnosti Arduino

Jedná se o malý jednočipový mikropočítač, který je řízen mikropočítačem od (nyní už) společnosti Microtic. Mikropočítač má několik pinů, některé z nich jsou analogové, a většina z nich jsou digitální, přičemž všechny (kromě napájecích) mohou být jak vstupní, tak výstupní. Na digitálních pinech můžeme nastavovat napěťové úrovně (logická 1 nebo 0), nebo tyto stavy můžeme číst z vnějších periferií (tlačítka, senzory) a v reálném čase reagovat na tyto periferie například rozsvícením LED apod.

Pro výuku assembleru bude mikropočítač ovládat pouze základní externí periferie (převážně tlačítka a diody) ale jeho možnosti daleko převyšují tyto možnosti. S pomocí vhodně navrhnutého programu je mikropočítač chopen řídit téměř jakékoli elektronické zařízení.

1.5.1 Arduino Uno

Arduino Uno je nejrozšířenější vývojovou deskou. Jejím předchůdcem je Arduino Duemilanove ale novější Uno se liší jen mírným hardwarovým upgradem bootloaderu a části desky která je provázaná s Watchdogem. Tato nejrozšířenější verze je velmi dobře zdokumentovaná a pro začátečníky je nejspíše ideální volbou. Obsahuje právě již zmiňovaný ATmega328P. Dále je na desce LED dioda, která je připojena k pinu 13, resetovací tlačítko, 16MHz krystalový oscilátor, 14 vstupů/výstupů z nich až 6 lze použít pro PWM[5], nebo jako analogový vstup/výstup. Deska může být napájena zdrojem napětí (doporučeno 7-12V) ale pro napájení je možné využít i USB konektor typu B, který také slouží k nahrání programu.

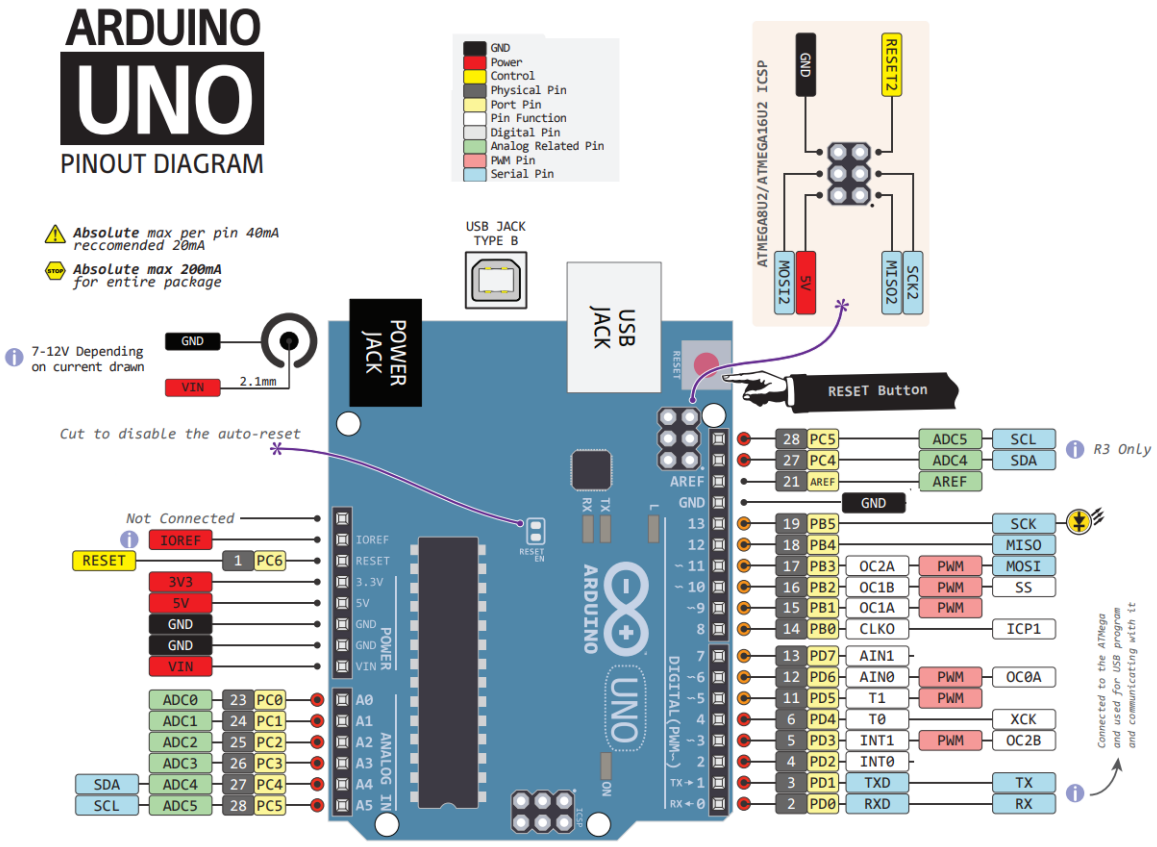


Obrázek 9: Arduino Uno r.3

[5] Pulse width Modulation

1.5.1.1 Piny Arduino Uno

Při pohledu na (Obrázek 10) zjistíme, že Arduino má piny k napájení (červené a černé), ovšem většina má i více funkcí. Typickým příkladem je sada pinů vlevo dole, kde můžeme vidět piny, které slouží jako analogový vstup (ADC0–ADC5) ale samy přímo se jedná o port C, respektive piny C0–C5 (popis pinů je žlutou barvou). Z tohoto hlediska není tedy složité, dohledat zbývající vstupně-výstupní porty B a D. Zde si také povšimněte že port B (stejně jako port C) má pouze 6 pinů (B0–B5). Tři z pinů portu B a D mohou být použity pro PWM, co je však podstatnější, jsou funkce pinů. Ty jsou popsány bílou barvou a v dále v práci bude na ně ještě odkazováno. Namátkou zmíním funkce INT0 (či INT1) které slouží pro extérní přerušení. Tato deska nám tedy nabízí (z „bajtového“ pohledu) kompletní port D (8 pinů) a nekompletní porty B a C. Pokud bychom tedy náš program měl do registru C uložit hodnotu 0xFF (binárně: 11111111), tak se logická jednička projeví pouze na pinech C0 až C5 (piny C6 a C7 neexistují), lze však s portem pracovat kompletně – například pokud by byla potřeba bitová rotace apod. Pokud tedy budeme chtít pracovat s externím zařízením, které pracuje na bázi celých byte, budeme si muset vystačit pouze s portem D, s porty B a C nikoli.



Obrázek 10: Rozložení pinů Arduino Uno

Samotné Arduino nabízí více desek, (např. Nano, či Mini) jenž se liší velikostí desky a počtem pinů. A právě tento problém by mohlo vyřešit Arduino Mega. Jeho cena je ze zahraničních obchodů cca jen o 100 – 150,- Kč vyšší ovšem počet pinů je více než dvojnásobný. Jeho cena tedy není v poměru ‚cena/výkon' vůbec špatná.

1.5.2 Arduino Mega

Tato deska je dle výrobců určena pro „náročnější uživatele" jenž nestačí počet pinů v základní verzi Uno. Vzhledem k výukovým periferiím je Arduino Mega vhodnější. Při bližším prozkoumání této desky (Příloha P III) zjistíme, že obsahuje daleko více portů než porty B, C, D jako tomu je právě u Uno. Šest z nich je kompletních (označeno zeleně), je zde 6 pinů pro externí přerušení – zvýrazněno žlutě (na Uno jsou dva také na portu D), jeden vstupní pin na čítaní signálů do čítače, a sériová linka je skrze port E (zvýrazněno modrou barvou), což je na Uno řešeno na pinech D0 a D1. To může na Uno způsobit problémy, pokud má uživatel/žák na desku připojeny periferie a periferie se snaží posílat logickou 1 na piny, kde je sériový přenos. Tím tento „přenos" ucpe a deska není schopna přijmout nový program.

1.6 Jazyk symbolických adres

V českém jazyce se pro jazyk symbolických adres používá i výraz Assembler. Tento překlad však není správný, protože výraz pro assembler je pouze pro překladač, který překládá z jazyku symbolických adres do strojového kódu.

1.6.1 Strojový kód

Každý procesor pracuje pouze s čísly (s nulami a jedničkami, ale ty mohou reprezentovat i vyšší čísla). Tyto čísla si bere z paměti, a jejich přečtením zjistí co má vlastně dělat. Pro lepší lidskou přehlednost se strojový kód zobrazuje v šestnáctkové soustavě (ale procesor jej samozřejmě vidí jen jako nuly a jedničky). Takovýto program při čtení procesorem může vypadat například následovně: „86 41 B7 01 00 BD 02 00". Zde se instrukce (co má dělat), nazývají operačními kódy a následují je operandy. Pokud chceme například načíst nějakou hodnotu do akumulátoru (hl. registru), musíme vědět její operační kód. Kolik vlastně chceme přičíst (vlastnost instrukce) nazýváme operandem. Zapamatování si však všech operačních

kódů bylo velmi nepraktické, proto byl vymyšlen jazyk symbolických adres, který tyto operační kódy nahradil symbolickými jmény. [2, str. 54]

1.6.2 Jazyk symbolických adres

Pokud je tedy definováno, že určitá slova (jenž si programátor snáze zapamatuje) mají význam konkrétních instrukcí, nejsnazší vysvětlení bude přímo na konkrétním programu. Následující program ukazuje totožný program, který je ovšem kromě strojového kódu popsán i v jazyku symbolických adres a je psán pro procesor 68HC11.

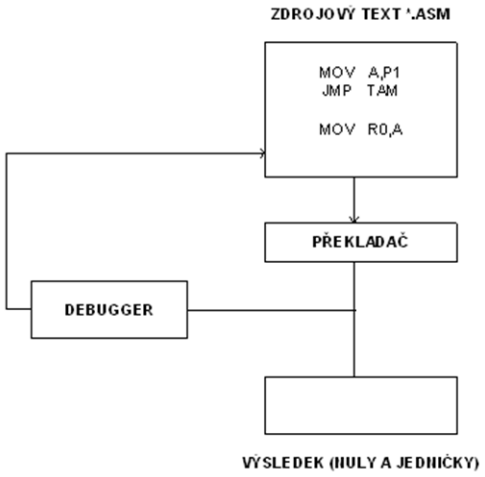
Strojový kód	Jazyk symbolických adres	Komentář
68 41	LDAA #$41	;načte hodnotu 41 do Akumul.
B7 01 00	STAA $0100	;načte obsah adresy 100H do A
BD 02 00	JSR $0200	;skočí na adresu 0200H (hex)

Zde tedy instrukce LDAA má strojový kód 68, a instrukce JSR strojový kód BD. Kolik za operandem následuje číslic je již definováno samotnou instrukcí, takže procesor sám rozpozná, zda číslo patří ještě k aktuální instrukci, nebo je již následující instrukcí.

Jazyk symbolických adres nám tedy zjednodušuje programování na nižší vrstvě procesoru a přímo umožňuje pochopit, jak procesor zpracovává instrukce a jak skutečně funguje. Ve skutečnosti však žádný procesor nerozumí těmto symbolickým adresám, jelikož sám nemá databázi instrukcí „v lidské podobě“ pouze v té strojové. Aby však byl život programátora jednodušší, vznikl v té době překladač nazývaný v angličtině Assembler. Vychází z anglického „Assembly“, kde právě ‚assembly language‘ je jazyk symbolických adres. V češtině se tedy sice nesprávně používá výraz pro překladač jakožto slangový název pro jazyk symbolických adres, ale v praxi to nezpůsobuje naštěstí takový problém a většina lidí používá raději slovo assembler než jazyk symbolických adres, protože je to kratší. Pokud tedy bude v dalších částech zmíněn assembler, bude se jednat o kód právě v jazyku symbolických adres – pokud nebude zmíněno jinak (například v popisu překladače). [2, str. 51]

1.6.3 Překladač

Programátor zapíše program ‚v assembleru‘ a než se program uloží do paměti, musí jej právě překladač přeložit do strojového kódu.



Obrázek 11: Diagram překladu zdrojového textu na strojový kód

Výsledný program si můžeme v počítači také prohlédnout, kde bude tedy v hexadecimálním tvaru, ale samotný procesor, jak již bylo řečeno si jej bude číst v tvaru binárním.

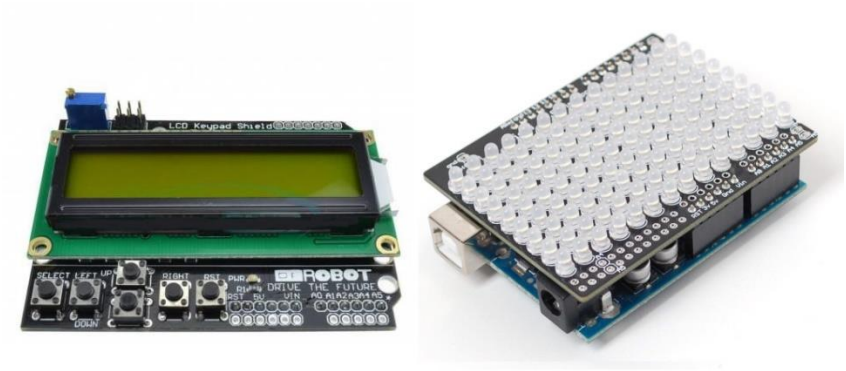
V programu Atmel studio tento výsledný soubor nalezneme dle nastavení překladače, nejčastěji však při defaultním nastavením v adresáři projektu v podadresáři Release.

2 NÁVRH HARDWAROVÉHO A SOFTWAROVÉHO ŘEŠENÍ

Začneme nejdříve hardwarovou částí. V první kapitole Arduino již popsáno je, není to však vše, co je potřeba z hardwarových součástek. Jak je popsáno v první kapitole, Arduino má své porty (B, C a D), a snahou je, aby žáci měli programování v Assembleru co nejjednodušší, co se týče hardwarového vybavení. Naprosté základy výuky programování na deskách Arduino (které tedy probíhají ve vyšším programovacím jazyku) ukazují základy pomocí desky a několika dalších periferií připojených nejčastěji přes nepájivé pole. To je možné i zde, ale má to dvě úskalí. Buď by si žáci museli kromě mikropočítače pořídit právě i nepájivé pole s dalšími součástkami (diody, odpory atd.) a druhé že by i na samotném nepájivém poli museli před každým příkladem vše zapojovat dle požadavků, což by opět zdržovalo žáky při samotném programování.

2.1 Hardwarové řešení

Pokud chceme připojit další periferie na Arduino, používají se tzv. Shieldy. Je to přímo označení pro jakoukoli periferii ve tvaru desky s piny tak aby odpovídali zapojení arduina připojenou na desku arduina z horní části. Některé tyto shieldy jsou možné zakoupit od výrobců jenž desku Arduino sami vyrábějí. Jedná se například o shield s dvouřádkovým displejem nebo maticí diod.



Obrázek 12: Příklady shieldů, vpravo matice LED shield přímo na desce Arduina

Tyto shieldy jsou však již natolik komplexní, že se spíše téměř nevyplatí s nimi pracovat na úrovni nižšího programovacího jazyka mj. proto, že kód je již natolik složitý, že tyto shieldy používají pro svou práci také již před programované knihovny.

2.2 SPŠ Shield a moduly

SPŠ Zlín dříve používalo některé externí moduly pro výuku Assembleru se starším mikropočítačem 8051 také i pro jiné mikropočítače či PLC. Jako komunikaci mezi jednotlivými periferiemi sloužili konektory PFL. Tyto konektory pro ploché kabely mohou mít až 64 pinů (tedy kabely až 64 nezávislých vodičů) o maximální délce 50 metrů[6]. V tomto případě však chceme na periferii vést obsah registru, a registry jsou zde jednobajtové (1 byte). Potřebujeme tedy 8 + 1 vodičů. Osm vodičů vede informaci (nula nebo jedna), devátý je zemnící vodič, který uzavírá elektricky obvod (např. pro 8 diod). Konektor pro ploché kabely pro 9 vodičů se nevyrábí, a proto se musí zvolit první vyšší – v tomto případě 10. Desátý vodič bychom mohli využit jako náhradní v případě poruše jednoho vodiče z celého kabelu, nebo například pro napájení externího modulu, což zde však využito není.

Obrázek 13: PFL konektor + zásuvka

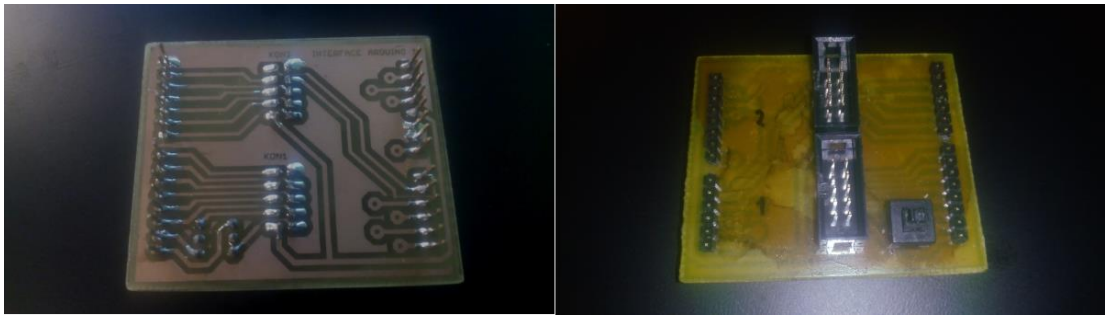
V tomto případě tedy chceme jeden konektor (pro port D) a jeden konektor pro další port. Zde však nastává problém popsaný v první kapitole a (Příloha P II). Arduino má na svých pinech vstup/výstup 6 pinů pro porty B a C. Externí moduly však očekávají 8 pinů/vodičů pro svou obsluhu. Proto byl vyroben shield, který dle mého názoru je poněkud nešťastným řešením této situace, a to ten, že sice druhý konektor obsahuje kompletně port D, ale první je rozdělen na prvních 6 pinů portu B, a zbývající dva jsou první dva portu C. nejsou tedy rozděleny ani „půl na půl“ a to přináší problém, který lze sice řešit, ale jelikož zde máme RISC, musíme 6x rotovat a žádný „jednodušší“ způsob jak piny ze dvou portů „srovnat“ do jednoho registru neexistuje. Pro vyšší přehlednost a pochopení následující tabulka:

[6] Byl by možný i delší kabel, prakticky jej ale žádná firma na světě nevyrábí delší než 50 m

PORT	PORT B						PORT C		PORT D							
PIN	0	1	2	3	4	5	0	1	0	1	2	3	4	5	6	7
Kon./Pin	1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	2/1	2/2	2/3	2/4	2/5	2/6	2/7	2/8

Tabulka 3: Zapojení Pinů pro dané konektory Shieldu.

První konektor má tedy na pinech 1 až 8 dva porty (B a C). To je patrné také zadní straně shieldu, kde můžeme vidět piny z PFL konektoru směřující k pinům v levé spodní části (Port D) a z horního konektoru vede 6 vodičů k pinům levé horní části (port B) a 2 cesty vedou na pravou stranu (port C). (obrázek X) Tyto shieldy však již byly vyrobeny v množství alespoň 20 kusů a pro aktuální školní rok, proto byly příklady, pokud možno vymyšleny pro tento typ zapojení, i když dle názoru autora může toto zapojení žáky v první moment mást.



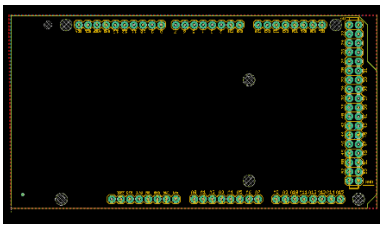
Obrázek 14: SPŠ Shield + jeho zapojení pinů(zadní strana)

Na obrázku je také možné si všimnout tlačítka. Je to právě tlačítko, které nám rozpojí (po dobu stisku tlačítka) cestu mezi modulem a shieldem (arduinem). Je tam z toho důvodu, jelikož při nahrávání programu je využíván právě sériový přenos, který ovšem hardwarově připojen na piny D0 a D1 (obrázek 10) a pokud modul „obsadí“ tyto piny, není možná komunikace, a tedy i nahrání samotného programu. (při nahrávání programu s připojeným shieldem je tedy nutné tlačítko držet)

2.2.1 Shield pro Aruduino Mega

Tato situace má řešení, které bohužel pracuje s myšlenkou, že původní shieldy budou „smeteny ze stolu“, žáci si namísto Arduina Una, jehož kopii mohou pořídit pod 100,- Kč vč. poštovného, zakoupí již zmíněné a popsané Arduino Mega. Jeho kopie stojí okolo 200,- Kč. A jak jsou popsány piny (viz příloha X), bylo by tedy možné vyrobit shiled, který by zvládl

obsluhu teoreticky až 6 možných portů, přičemž port D by zvládal obsluhu klávesnice, která potřebuje ke své práci 4 piny jenž dokáží detekovat změnu, o tom ale více v praktické části. Rozměry nového shieldu i rozmístěním pinů, lze nalézt na internetu, přidat na ně konektory a správně propojit piny je otázkou maximálně několika desítek minut.



Obrázek 15: přesné rozměry Shieldu pro Aruino Mega z platformy Github

2.2.2 Modul Tlačítek/Spínačů a diod

Tento modul je již byl vyroben jako pomůcka pro střední školu před více než dvaceti lety. Jeho výhodou je právě snadné propojení mezi mikropočítačem a modulem, Tento modul tedy má dvě na sobě nezávislé strany. První pracuje pouze jako výstupní. Máme na ní řadu 8 diod, a segmentový display. O tom, zda má být zapojen display nebo diody rozhoduje zapojení jumperu v horní části modulu.

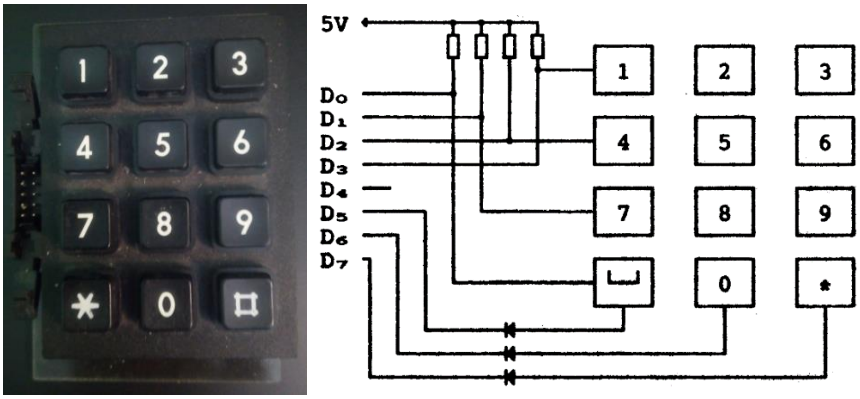
Pravá část má sadu osmi spínačů a tlačítek, které slouží jako vstupy. Spínače a tlačítka jsou zapojeny paralelně, aby mohli fungovat nezávisle na sobě, zda by měl uživatel stisknout jen krátce, či právě daný spínač sepnout na delší dobu. Tato část nám reprezentuje řadu osmi vstupů.

Obě strany modulu – vstupní (spínače) a výstupní(diody) jsou vyrobeny tak, že při logické nule dioda svítí a spínač se pro mikrokontroler tváří jako při log. 1 a naopak. Pokud tedy chceme rozsvítit všechny diody, musíme poslat na port 0000 0000 a naopak, pokud chceme porovnat stav, kdy jsou všechny spínače nesepnuté, kontrolujeme, zda je na portu hodnota 0xFF neboli 1111 1111. Tím, že jsou obě strany modulu „prohozeny“ nemusíme tedy obracet jejich stav. Tedy pokud program, který má za úkol rozsvítit ty diody, kterým odpovídají čísla portů seplým spínačům, budou sice tyto spínače dávat log 0, ale právě nula tyto diody rozsvítí (a obráceně).

2.2.3 Modul klávesnice

Klávesnice je zapojena tak, že jednotlivá tlačítka jsou zapojeny do matice. Pokud tedy chceme zjistit, že jsme stiskli např. č. 1, posíláme periodicky do 0, 1, 2 a 3 bitu portu, ve kterém je klávesnice zapojena log 1. Tato hodnota se pak po stisknutí tlačítka projeví na 5 pinu, jelikož se jedná o první řádek matice a v okamžik stisku byla poslána log. 1 právě do prvního sloupce. – Více o tom vypovídá schéma zapojení klávesnice.

Zde je však problém, že Arduino Uno má pouze dva piny, které umí detekovat přerušení,(konkrétně se jedná o piny D2 a D3) proto nebyly na tento modul vytvořeny žádné příklady a jsou tedy možné jen pro Arduino Mega. (kde přerušení umí detekovat piny D0 – D3) Také by bylo potřeba vytvořit primitivní adaptér, jenž by umožnil inverzní zapojení pinů klávesnice do portu mikropočítače.



Obrázek 16: modul klávesnice a jeho schéma zapojení

2.3 Softwarové řešení

Softwarové řešení se dá shrnout do 3 kategorií. První je kategorie zdarma, která nepodléhá licencím, druhou je kategorie, kdy licencím podléhá ale i přesto je zdarma, a třetí varianta je licence placená. První variantou lze tedy označit „software“ který umožnuje překlad do strojového kódu zdarma bez licence, zde se však nedá zcela úplně mluvit o SW, jelikož překlad do zdrojového kódu nám provede samotný procesor počítače. Jak přesně však ví, co dělat? V prvé řadě je potřeba mít soubor, který chceme převést do zdrojového kódu (program). K němu nasměrovat linker, překladač a do parametrů překladu i udat, jak chceme překládat. V ostatních variantách se nám o to stará přímo software, ale zde nikoli. V případě, že je zde vše správně nastaveno, překlad se podaří a může být odeslán do mikropočítače.

Výjimečně je přeskočena druhá varianta, která bude obsáhlejší a zmíním zde ještě třetí kategorii, a to je vývojové prostředí, které je za poplatek. Je sice výhodné, že licence (kromě podpory, která je na rok, a lze ji dokupovat na další roky), není dále nijak časově omezená, i tak se ale cena jedné licence pohybuje okolo ceny 5000,- Kč.

Poslední střední cestou mohou být hned dva programy, které podléhají licencím, ale jsou zdarma. Krátce o té nepříliš efektivní: Arduino jako takové má své vlastní IDE, které je zcela zdarma. Lze na něm také programovat jazyku symbolických adres, zde však musí být kód uveden ve funkci asm(); a obsah funkce je Assembler kód jenž musí být každý řádek ohraničen navíc nejen uvozovkami ale také ukončen sekvencí ‚\n' (je to řídící znak pro nový řádek), což je velmi nepraktické.

```
11 void setup() {
12   Serial.begin(9600);
13   asm (
14     "rcall  asmSetup    \n"
15   );
16 }
17 void loop() {
18   asm(
19     "rcall asmLoop    \n"
20   );
```

Obrázek 17: Příklad programování Assembleru v Arduino IDE

Není proto divu, že na většině webu, který odkazuje na prostředí dominuje Atmel studio 7. Se staršími návody se lze setkat s názvem AVR studio, od 6 verze však používá tento nový (Atmel Studio) název.

Nutno podotknout, že IDE prostředí, kterými lze programovat Arduino, existuje samozřejmě více, ne všechny však nativně umožnují programovat v Assembleru. Z těch, které to umožnují je možno zmínit: SimAVR, které však umožňuje pouze simulaci a není zde možné program nahrát do mikrokontroléru. Ostatní vývojová prostředí, která zde nejsou zmíněna většinou také obsahují simulátor, ovšem není zde možnost programovat v Assembleru.

Zdarma bez Licence 3. stran:		
	Výhody:	Nevýhody:
GCC inline assebler (kompilátor)	+ Je kompletně zdarma + Zabírá málo místa na disku	- Pouze Linux - Nemá vlastní IDE - Je spíše pro zkušenější uživatele
Zdarma s licencí třetích stran:		
Arduino IDE Atmel Studio*	+ Jednoduché + Atmel studio má i simulátor	- Zabírá dost místa* (alespoň 2 GB) - Arduino IDE zabírá o mnoho méně, ovšem nemá simulátor - Arduino IDE není příliš praktické na programování assembleru. (Obrázek 17)
Placené licence		
CodeVisionAVR	+ Nabízí optimalizaci (skoků, smyček a dalších)	- Oproti Atmel Studiu nenabízí kromě optimalizace kódu nic navíc - Vysoká cena
Proterus VSM	+ Kromě základního simulátoru umožnuje i simulaci s dalšími shieldy, či periferiemi (segmentový display, LED, potenciometr apod.)	- Vysoká cena

Tabulka 4: Porovnání dostupných software programů pro výuku Assembleru

2.3.1 Atmel studio

Toto vývojové prostředí pro mikroprocesory od firmy Atmel (Microchip) bylo vyvýjeno separátně pro 32bitové procesory a ostatní procesory, ovšem od šesté verze, kterou již má název Atmel studio, funguje prostředí pro všechny typy Atmel mikroprocesory.

2.3.1.1 AVR32 Studio (1.0 – 2.6)

První verze tohoto programu se jmenovala AVR32 Studio 1.0. Datum vydání bylo: 14. 6. 2007. Jak z názvu může vyplývat, tato verze se primárně snažila být vývojovým prostředím pro 32bitové procesory. Byl postaven ve vývojovém prostředí Eclipse, díky tomu byla tato

verze nejen pro Operační systém Windows, ale také pro Linux a minimální požadavky tehdy byly: Alespoň 512 MB RAM, 500 MB volného místa na disku, procesor Pentium 4 (a novější) a Windows 2000(XP) nebo Fedora 4, Ubuntu 6.06 či SUSE Linux 10.1. Samozřejmostí bylo také to, že tato verze již umožnovala programovat v Assembleru.

Druhá verze vyšla 4. 4. 2008, ovšem zásadní verzí byla až verze 2.5, která byla vydána 25. 2. 2010 a to verze 2.5. Tato verze byla stále napsána v Eclipe ovšem již nebyla kompatibilní se staršími operačními systémy – Windows 2000, Fedora 7(a nižší), naopak měla plnou podporu pro Windows 7 (64-bit).

V červenci toho roku také vyšla poslední verze obsahující v názvu ‚32' a tedy verze AVR32 Studio (v názvu již není číslo verze), od předchozí verze se však kromě novějšího frameworku nijak výrazně nelišilo.

2.3.1.2 AVR studio (4.13 – 5.1)

Souběžně s vývojem AVR32 Studio se také vyvíjelo IDE pro programování všech typu mikroprocesorů (tedy také 4, 8 a 16bitových) Tato verze byla vydána tedy již na konci roku 2008, (tedy dříve, než AVR32 2.5). Tato verze ovšem byla již k dispozici pouze pro operační systém Windows.

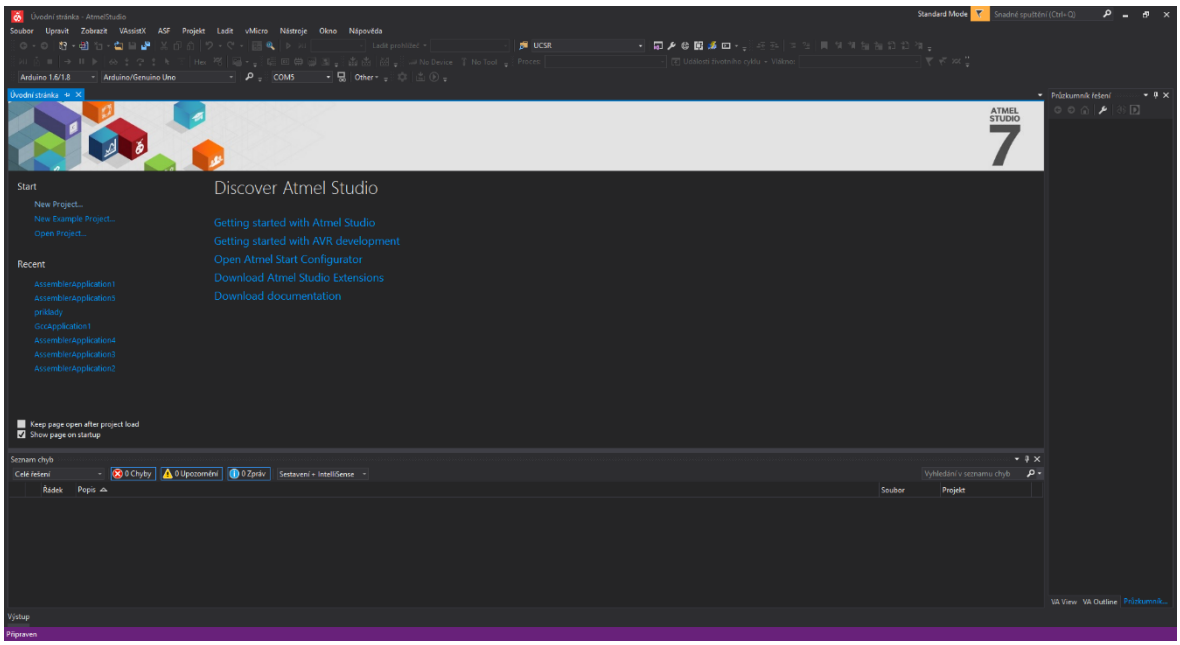
2.3.1.3 Atmel Studio (6.0 – 6.1)

Toto prostředí je vydáno v roce 2012 (nejdříve beta verze – únor) později hlavní verze. Je postavena na frameworku 4.0 a hlavní jádro programu je Visual Studio 2010 (dále VS) od firmy Microsoft. V případě že uživatel neměl nainstalované VS, instalátor nainstaloval izolovanou verzi VS. Nebylo to kompletní VS, uživatel tedy nemohl spustit VS (samostatně).

2.3.1.4 Atmel Studio 7.0

Obrázek 18: Logo Atmel studia

Tato (zatím) poslední verze, vydána v druhé polovině roku 2015 a běží na jádru VS 2015. Prostření je možné si prohlédnout níže. (Obrázek 19)



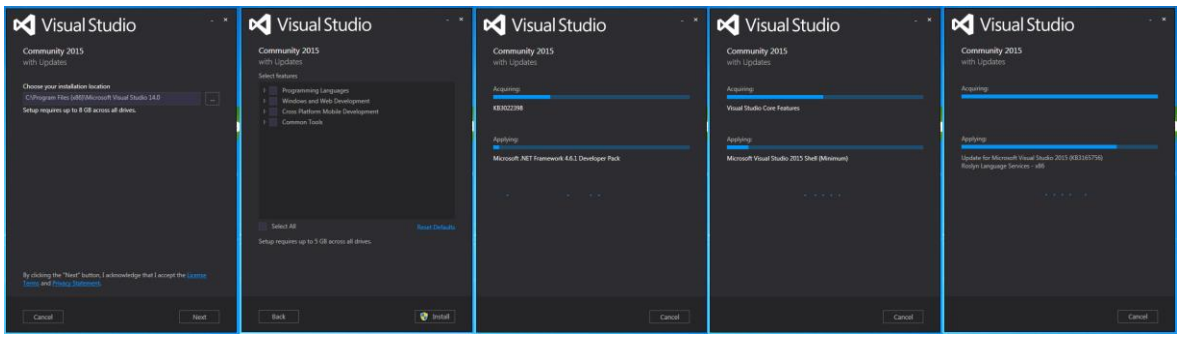
Obrázek 19: Atmel Studio 7 po spuštění

Tato verze z náhledu má kromě jiného, také českou lokalizaci programu (menu, popis kódu chyby, či nastavení). Atmel (Microchip) ji však nevytvořil a nepodporuje, jediná možnost, jak získat českou lokalizaci, je nainstalovat si Visual Studio Community 2015. Verzí Visual studií (kromě ústředních verzí – myšleno: verze např. 2013, 2015, 2017 či 2019) – je více (např. Express, Professional či Team), ovšem jedině Community verze, je zcela zdarma. Aby to bylo možné, je potřeba dodržet postup instalace. Poznámka: česká lokalizace bohužel není možná nainstalovat do Atmel Studia – instalátor češtiny nevyhodnotí program jako VS.

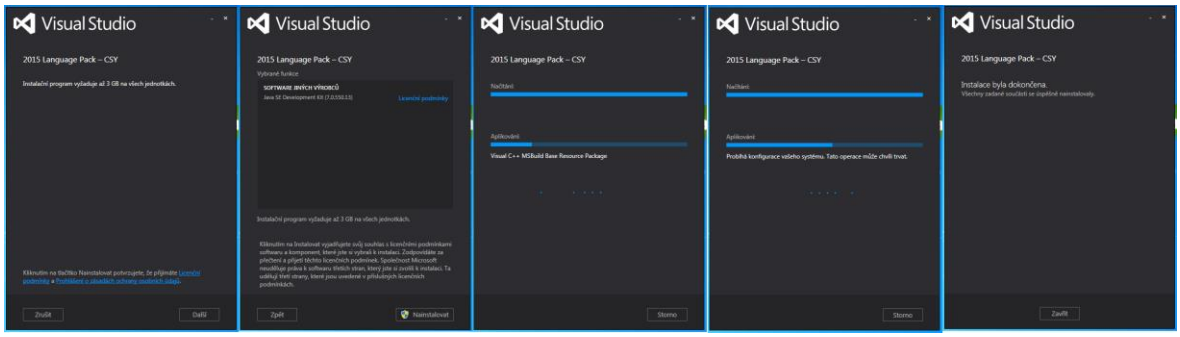
2.3.2 Postup instalace (vč. českého rozhraní)

V případě, že nepovažujete za nutné mít prostředí v českém jazyce, můžete tento bod přeskočit a pokračovat až bodem Instalace Atmel Studia.

V příloze naleznete 3 MB soubor jenž je instalátorem VS 2015, který Vás provede celou instalací. Během instalace je nutné být připojen k internetu. Poté co se instalátor zeptá, na umístění, kam VS nainstalovat, je možné zvolit funkce a doplňky (druhý obrázek z obrázku X). Pokud chcete využít i jiné možnosti VS, můžeme si zvolit funkce a doplňky, pokud však ne a instalujeme celé jádro jen kvůli češtině do Atmel Studia, nemusíme vybrat nic (viz obrázek) i tak se nainstaluje hlavní jádro VS, které má velikost přes 4 GB.



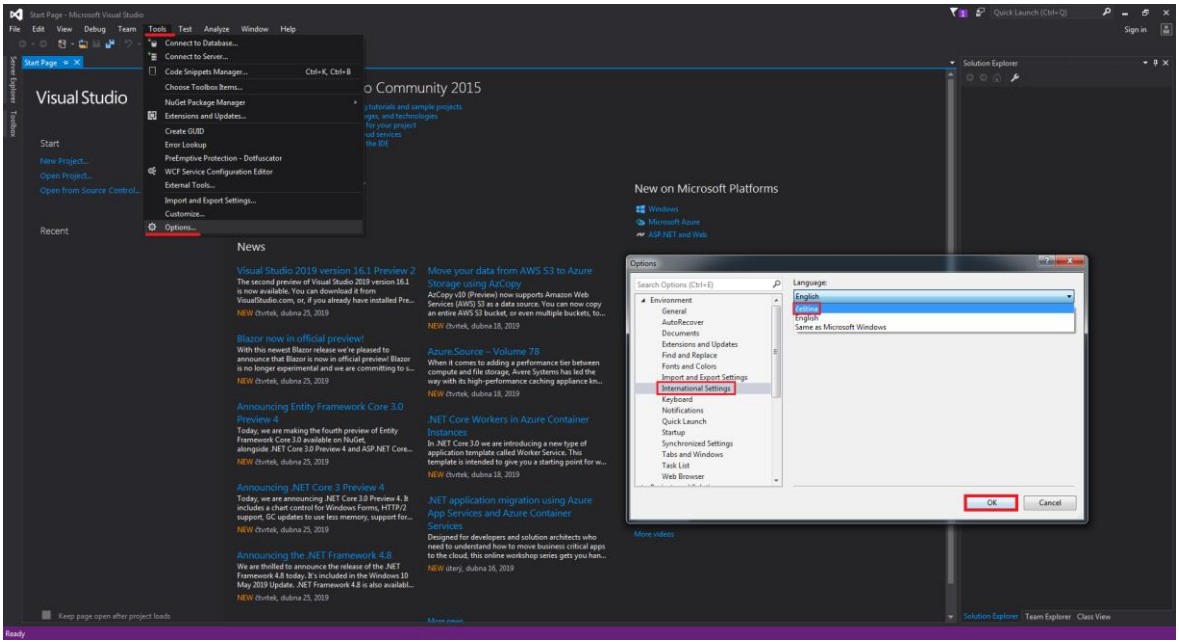
Obrázek 20: Instalace VS (průběžně)



Obrázek 21: Instalace českého jazyka

V příloze se také nachází nachází soubor vs_langpack.exe (221 MB) jenž právě obsahuje českou lokalizaci. Instalace je obdobná jako instalace VS. (obrázek X)

Poté, co je instalace dokončena, je potřeba ještě ve VS přepnout jazyk. Tuto možnost nalezneme v horním menu pod položkou menu „Tools“ a vybereme „Options“. Otevře se další okno s veškerým nastavením VS. V menu je první hlavní položkou Environment a jeho submenu nalezneme Internacional Settings. Zde již vybereme český jazyk. Po potvrzení bude prostředí vyžadovat restart celého prostředí, tedy v tomto případě jej stačí zavřít.



Obrázek 22: Nastavení českého jazyka ve VS

2.3.2.1 Instalace Atmel Studia

Následně již zbývá jen nainstalovat Atmel studio – soubor: as-installer-7.exe (2,4 MB). Jedná se opět o webový instalátor a během instalace je potřeba připojení k internetu. Ve druhém kroku je třeba také vybrat Architektury. Pro programování s jakoukoli verzí Arduina (Duemilanove, Uno, Mega) stačí pouze 8-bitová architektura – ostatní (32-bitová a SMART) architektury jsou nepotřebné a je zbytečné je instalovat.

Pokud jste správně nainstalovali českou lokalizaci, ihned po prvním spuštění Atmel studia bude již prostřední (program) v českém jazyce. V případě že se pro českou lokalizaci rozhodnete dodatečně, je možné VS i s češtinou doinstalovat později.

II. PRAKTICKÁ ČÁST

3 IMPLEMENTACE UKÁZKOVÁCH PROGRAMŮ

Ukázkové programy jsou realizovány do dvou sekcí a to tak, že do první skupiny spadají programy, jenž se provádějí v simulátoru a není na ně potřeba žádné zařízení. Jedná se o příklady, kde je pointou žákům vysvětlit základy, tedy jak instrukce pracují, co provádějí a jaké bity ve stavovém registru.

3.1 Programy prováděné v simulátoru

Prvních 8 příkladů je pro vyzkoušení a pochopení instrukcí, tyto příklady jsou uvedeny v přílohách a v rozumné míře jsou doplněny komentáři pro sekce, jež mohou vyučujícímu objasnit daný řádek (délku řádků)

Číslo příkladu	Popis	Využité instrukce
1	Práce s registry a pamětí, přesun obsahu registrů	LDI, LDS, LDI
2	Záměna obsahu registrů a obsahu adresy v paměti	MOV, LDS, STS
3	Přímé a nepřímé adresování	LDI, STS a ST
4	Součet po sobě jdoucích čísel a násobení registrů	ADD, MUL
5	Rozdíl 16bitových čísel	SUB a SBC
6	Součet více 16bitových čísel	ADD a ADC
7	Násobení a rotace a „dělení“ dvěma	LSL a ROL
8	Přesun obsahu na základě nepřímé adresy	LD, ST, DEC a instrukce skoku

Tabulka 5: Vzorové příklady pro simulátor

V této fázi nejsou žákům plně vysvětleny podmíněné a nepodmíněné skoky, a to hned ze dvou důvodů. Prvním je že do nepodmíněných skoků spadá i volání podprogramů a samotný skok na jiné návěstí. Podmíněný skok je na návěstí při splněné podmínce. Toto však lze pochopit již při základech vyššího programovacího jazyka. Zde praktický příklad:

Elementární podmíněný výraz „IF“. Zde podmíněný výraz vyhodnotí podmínku a na jejím základě se běh programu přesune na „větev kódu“ která se má vykonat při splnění podmínky a v případě nesplnění podmínky se běh přesune na zápornou větev programu. Tyto základy podmínek (a vyššího programovacího jazyka) mají studenti tohoto předmětu již v předchozím ročníku, samotný princip jak tedy na základně podmínky program „skočí“ na dané návěstí, je sice žákům zmíněno a zopakováno, ale dle názoru autora není potřeba samotnou problematikou se zabývat, jelikož by (již) problematice měli rozumět. Podrobněji jsou rozepsány instrukce, které provádí skoky podmíněné a objasní, jak fungují. Například pokud chceme porovnat hodnotu, zda je hodnota menší než hodnota porovnávající, obě hodnoty ALU porovná, a v případě, že je porovnávaná hodnota menší, projeví se to tak, že po porovnání bude nastaven carry bit. [5]

3.2 Programy prováděné mikropočítačem a moduly

Tyto příklady již využívají reálného mikropočítače a modulů. Který modul je potřeba (a jeho port jsou uvedeny v tabulce XX). Pokud však nastane případ, kdy si bude student nejistý, lze většina příkladů realizovat i v simulátoru, jelikož je zde možnost, nastavit během chodu simulace změnu registru jednotlivých vstupně-výstupních portů. Jak bylo ukázáno v první kapitole (obr XX) paměť dat (SRAM) obsahuje vstupně výstupní porty. Tyto registry jsou každý na konkrétní adrese (např. PORTD je v SRAM na adrese 2B) Tyto adresy si však není nutné pamatovat, jelikož samotné prostředí v Atmel studiu obsahuje databázi názvů všech registrů, jak jsou uváděné v datasheetu výrobce. V případě, že tedy chceme zjistit, zda je na portu B, který je nastaven jako vstup přítomno napětí (log. 1) není potřeba zadávat hexadecimální adresu 23, ale klíčové slovo PINB. Pro objasnění (a zopakování toho) jak funguje překlad z jazyku symbolických adres do strojového kódu si však žáci mohou vyzkoušet, že namísto klíčových slov použijí adresy. Tyto adresy jsou udávány výrobcem a není možné je měnit. Dále ještě existují oddíly paměti, jež jsou nevyužité, ale jsou tzv. rezervované. Do této části paměti není možné zapisovat, přestože se paměť jeví jako volná. Ve většině případů není zápis ani hardwarově povolen.

Č. příkladu:	Popis	Instrukce	Modul a port
9	Modul osmi LED (výstup) a příklady na efekty s modulem (rotace, blikání)		V/V portD
10	Modul osmi přepínačů (vstupů) a příklady		V/V portB a C
11	Realizace klopného obvodu		V/V
12	Realizace „vzduchovodu“ (výroková logika)		V/V
13	Blikání diody (na základě externího přerušení)		V/V
14	Práce na sedmi segmentovém displeji		V/V
15	BCD dekodér (využití tabulky proměnných)		V/V
16	Realizace čítače a časovače		V/V
17	Určeni počtu jednotlivých bitů		V/V
18	Multivětvení		-
19	Zobrazení čísla zadaného z klávesnice		klávesnice
20	Primitivní kalkulačka		klávesnice
21	Sériový přenos		V/V

Tabulka 6: Vzorové příklady pro Mikropočítač s moduly (Arduino Uno)

V případě že je uveden pouze modul, a nejsou uvedené porty, potom je vstupně-výstupní modul zapojen následovně: Port D je vyveden na výstupní část modulu (reprezentováno osmi LED) a na portech B a C je přivedena část spínačů, jenž reprezentuje právě vstupy.

Jelikož však (jak je zmíněno v druhé kapitole) tento miktrokontrolér (Arduino Uno) má jen dva piny schopné detekovat externí přerušení (respektive lze vyvolat externí přerušení na jakémkoli pinu, přerušení se však vyvolá při „jakékoli změně“ na pinu. To však znamená že při krátkém stisku tlačítka jsou vyvolány minimálně dvě změny (stisk a následné puštění tlačítka) a změn může být více díky překmitům tlačítka jemuž se právě věnuje také právě

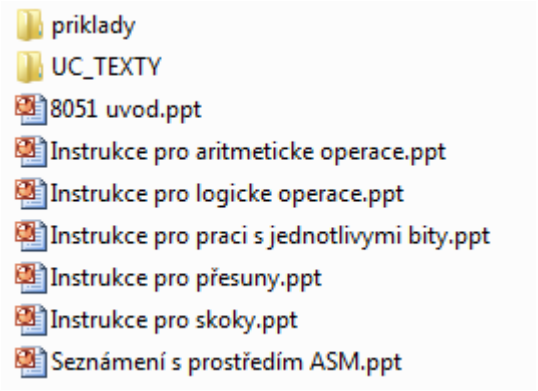
čtrnáctý příklad. Z tabulky je také patrné, že příklady s klávesnicí nakonec nebyly úspěšné realizovány, právě z důvodu, pro nedostatek pinů přerušení na straně mikrokontroleru.

3.2.1 Problémy při implementaci ukázkových programů

Při implementaci ukázkových programů se autor setkal s několika problémy, jenž některé z nich jsou vyřešeny úspěšněji, jiné bohužel ne. Nejčastějším problémem byla situace, kdy stisk tlačítka vyvolal tzv. zákmit tlačítka (tlačítko externího modulu). Tato situace se však dá žákům demonstrovat, proč vzniká a jak se jí softwarově „bránit“. Je na to i věnovaný příklad. Co však působí větší problém, je pokud je potřeba nastavit pin aby při změně vyvolal výjimku (v pořádku), ale ne několikrát díky překmitům. Softwarová ochrana proti překmitům je reprezentována alespoň 5 ms čekáním, nicméně obsluhy přerušení u mikrokontroléru fungují tak, že po vyvolání přerušení sice „nepřijímají“ další přerušení (aby nové přerušení nepřerušilo aktuální přerušení) – tedy proces přerušení je nedělitelný. Co se však stane, pokud díky zákmitům nastane více přerušení? Mikrokontrolér si při prvním přerušení zavolá obsluhu pro přerušení, ovšem po dobu přerušení si dál zaznamenává další impulsy přerušení. Jakmile je přerušení dokončeno, mikrokontrolér se vrátí do hlavního programu provede alespoň jednu instrukci (hlavního programu) a posléze může nastat další přerušení (a tak stále dokola). Z příkladu zde uvedeného je tedy patrné, že pokud by bylo zadáním, aby vyvolané přerušení zvýšili hodnotu registru o jedna, pak by stisk tlačítka generoval teoreticky více přerušení a softwarové ošetření pro přerušení možné není. Dalším problémem při zpracování komplexnějších úloh může být to, že piny pro vyvolání přerušení (externího či z čítačů) jsou na Arduino Uno pouze na portu D. Tedy není možné zapojit více spínačů a vyvolávat nezávislé přerušení např. z dvou vstupně-výstupních modulů. Poslední problém který nastal souvisí s maticovou klávesnicí, a to tak, že samotná klávesnice je zapojena jako matice, a pro detekování potřebuje 4 piny jako vstupní, ovšem na Arduinu Uno jsou pouze dva. Proto jsou příklady s klávesnicí v () označeny červeně, jelikož se jejich ukázkový příklad nepodařilo implementovat. Pro příklady s klávesnicí je tedy jediným možným řešením „přepracovat“ výuku na Arduino MEGA.

4 PREZENTACE K TEORETICKÉ ČÁSTI VÝUKY

Prezentace vycházejí částečně z původních prezentací předmětu Stavba a programování mikropočítačů, a to ve smyslu, že předmět již měl pro výuku vytvořeny dvě stěžejní prezentace (úvodní a hlavní) a dále pak prezentace pro operace s jednotlivými registry. Jak vypadalo původní rozložení prezentací je možné vidět na obrázku () níže:



Obrázek 23: Původní rozložení prezentací pro mikropočítač 8051

Autor práce se proto rozhodl dodržet toto schéma i pro budoucí výuku stylizovat prezentace do stejného rámce. Tedy prezentaci, jenž bude úvod k mikropočítači a úvod celkově k jazyku symbolických adres a prezentací pro prostředí v jejímž rámci budou již úvodní primitivní příklady, jenž právě vyučující realizuje se studenty právě ještě bez mikropočítačů a software pouze na tabuli, aby byl vysvětlen právě elementární princip některých operací.

Poslední částí jsou prezentace jednotlivých bloků instrukcí.

4.1 Úvodní prezentace

Úvodní prezentace jsou dvě, přičemž je zde doplněn dokumentový soubor (PDF) jenž shrnuje jednočipový mikropočítač z pohledu rozpisu učiva předmětu SPM.

4.1.1 První prezentace

První prezentace (v příloze soubor: 01_zakladní_pojmy.pptx) obsahuje popis jaký je rozdíl mezi mikropočítačem a mikroprocesorem aby žák byl obeznámen s tím, že neprogramuje „celou desku“ Arduina, ovšem jen její čip, který je v některých případech možné i měnit a

tím i měnit program. Tato prezentace také vysvětluje základní pravidla jazyku symbolických adres (konstanty či zápis čísel). Závěrem také nastíní registry.

Druhá prezentace

Druhá prezentace uvádí rozdělení počítačů, jak je popsáno v první kapitole teorie této práce, jak se dekódují instrukce a z jakých bloků je složen mikrokontroler ATmega328 a jak funguje překlad z jazyku symbolických adres přímo do strojového kódu.

Dále se prezentace věnuje organizaci paměti, popisu jednotlivých portů (jenž jsou popsány také v kapitole dvě a problémům jež můžou nastat při připojení periferií.

Dále se prezentace podrobně věnuje čítačům a časovačům a v poslední části je popsán jaké možnosti komunikace mikrokontroler nabízí (jejich porovnání, použitelnost) apod. [6=]

K této prezentaci je ještě vytvořen PDF soubor, který spadá do páte kapitoly podkladů pro učitele, kde bude uvedeno více.

4.1.2 Prezentace jednotlivých bloků instrukcí.

Prezentace byly rozděleny dle autora na bloky, které se liší v provedení samotné operace. První mohou provádět operace využívající se v ALU[7], dále na problematiku s práci s jednotlivými bity, práci s přesuny (z paměti do registrů či obráceně) a závěrem práci se skoky (a přeskoky)

4.1.2.1 Prezentace Instrukcí pro aritmetické operace

Tato prezentace se věnuje instrukcím pro sčítání, odčítání, násobení a jaké důsledky tyto operace mohou mít pro status registr. – Například pokud chceme násobit dvě 8bitové hodnoty a výsledek bude 16bitový, kladný, záporný, či nulový co vše dané operace mohou ovlivnit právě v registru jenž slouží k zaznamenávání stavů po provedení těchto instrukcí.

4.1.2.2 Prezentace instrukcí pro logické operace

Zde jsou pro studenty popsány syntaxe, a příklady instrukcí pro logické operace. V posledním snímku prezentace je dopodrobna popsán stavový registr.

---

[7] Arithmetic logic unit (Aritmeticko logická jednotka)

4.1.2.3 Prezentace instrukcí pro práci s jednotlivými bity

Prezentace se věnuje popisu instrukcí, jenž ovlivňují pouze konkrétní bit nikoli celý bajt. Tyto operace jsou velmi praktické pro práci s porty kde však pro vstupně/výstupní registry je potřeba využít jiné instrukce, než je tomu u standartních registrů.

4.1.2.4 Prezentace instrukcí pro přesuny

Tato prezentace se věnuje přesunům hodnot(dat) z paměti dat do registrů a obráceně. Samozřejmostí je také popsáno nepřímé adresování, jenž je možné také využít. Závěr prezentace vysvětluje teoreticky i prakticky jak lze využít zásobník (v paměti dat) na mikrokontroleru.

4.1.2.5 Prezentace instrukcí pro skoky (a přeskoky)

Poslední materiál uvádí kromě svého předchůdce (tedy skoků) také tzv přeskoky. To jsou instrukce, které při stavu konkrétního bitu mohou přeskočit následující instrukci. Vztahuje se to ovšem pouze na jednu následující instrukci a v případě, že je třeba přeskočit více instrukcí, nezbývá než využít standartní ať již podmíněný či nepodmíněný skok. Prezentace uvádí jak popis, tak praktický příklad instrukce podmíněného i nepodmíněného skoku.

Všechny prezentace jsou v příloze v kořenovém adresáři.

5 PODKLADY PRO STUDENTY A UČITELE

Podklady jsou rozděleny pro studenty a učitele, a jelikož poklady pro učitele nemusí obsahovat vše, co je obsahem této práce, je možné jako podklad pro učitele považovat mj. i tuto práci.

5.1 Podklady pro studenty

Pro studenty autor práce nachystal (kromě vzorových příkladů ve třetí kapitole) ještě dva další materiály. Jeden je v podstatě „ořezaný“ datasheet v českém jazyce a druhým je webový soubor, jenž počítá, jak velké smyčky je třeba pro zdržení programu.

5.1.1 „Tahák instrukcí“ a důležité registry

Tento tahák se nachází v příloze a je takovým silně okleštěným datasheetem. Celý dokument má něco lehce přes 20 stran, (originál jich má téměř 300). Jsou zde popsány (z pohledu příkladů) důležité registry (jejich bity a co nastavují pro vstupně výstupní porty) popsána celá paměť registrů (SRAM) a na posledních třech stranách jsou přehledně rozdělené a popsané instrukce.

5.1.2 Webový soubor počítající prodlevu pro zdržení

Pokud máme program, jehož cílem je aby dioda blikala, musíme také vědět, jak často chceme aby blikala (čas po který je zhasnutá a po který je rozsvícená). Mikrokontrolér však sám například sekundu „počkat“ neumí. Proto jej musí člověk, jenž jej programuje ‚zaměstnat‘. To provede tak, že procesor nechá provádět cyklus tolikrát, kolikrát je třeba. Víme, že frekvence mikrokontroleru je 16 MHz tedy za jednu sekundu stihne 16 milionů jedno cyklových instrukcí. Ve vyšším programovacím jazyku, který provádí program na mikrokontroléru je funkce Delay (nebo millis). Ta reprezentuje, kolik milisekund má procesor čekat. O obsluhu této funkce se stará čítač0 a ten díky realizaci zpoždění skrze čítač může ‚čekat‘ nejdéle přibližně 25 dní (cca 50 dní, pokud je použít bezznaménkový long integer. [3]

Zde však budou žáci potřebovat rozblikat diodu ještě dříve, než se vůbec dostanou k čítačům/časovačům. Jak tedy procesor zaměstnat například na dobu 1 sekundy? Obsah jednoho registru je číslo od 0 do 255, pokud tedy číslo budeme postupně snižovat dokud nebude

nulové, zaměstnáme procesor na 256 instrukcí. Pokud provede „cyklus v cyklu“ již zaměstnáme procesor právě na cca 65 536 instrukcí. Pokud však využijeme ještě 3 registr, do třetí vnější smyčky, vyjde nám 16 777 216 instrukcí, což je více než 1 sekunda. Takto by student musel počítat kolik vlastně musí být instrukcí, aby se „trefil do jedné sekundy“. Na to je naprogramovaný webový soubor, jenž může být nahrán kdekoli na web školy jenž algoritmicky spočítá kolik je instrukcí potřeba pro daný čas a navrhne pro dané zdržení kód v jazyku symbolických adres. Tento soubor se také nachází v příloze ve složce podklady pro studenty. Jedná se o prostý HTML soubor, jenž je možné si ve webovém prohlížeči či v externím programu prohlédnout také kód.

5.2 Podklady pro učitele

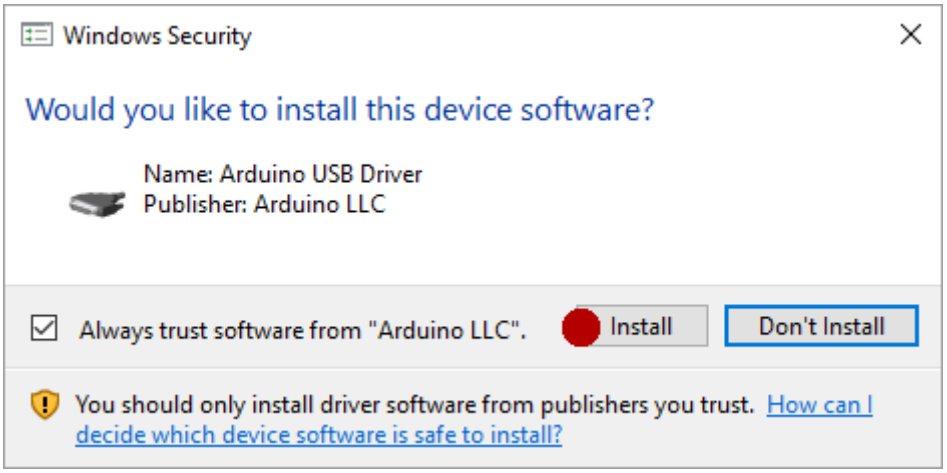
Jako podklady pro učitele jsou nachystány pomocné soubory, k prezentacím, které umožnují lépe porozumět látce tak, jak je potřebné pro výklad.

Dále je zde ještě podklad, jak nastavit Atmel studio, aby úspěšně bylo možné nahrát program do mikrokontroleru. Je zde totiž ta skutečnost, že Atmel studio samo o sobě nemá „ovladače“ pro zkontaktování arduina a domluvení se komunikace mezi arduinem a počítačem. Je to dáno z toho důvodu, že i když Atmel studio obsahuje sice obsahuje datasheety, instrukční sady aj. k nepřebernému množství miktrokontrolérů, samotný mikrokontrolér lze právě programovat přes např. sériový přenos a k tomu nejsou zapotřebí žádné periferie. Aby tedy program šel úspěšně nahrát do Arduina, je ještě nutné před první výukou nainstalovat program, jenž umožní, aby „počítač věděl“ že se jedná o arduino a podle toho s ním také komunikoval. V opačném případě počítač i přes USB rozhraní nepřidělí zařízení virtuální sériový port.

Nejnovější ovladače k arduinu lze najít na webových stránkách výrobce. Výrobce zde nabízí již v teoretické části zmíněné IDE, které sice nevyužijeme, jelikož je pro práci s nižším programovacím jazykem poměrně nepohodlné, jak je zmíněno v teoretické části, nicméně je nutné si jej nainstalovat, jelikož samostatné ovladače k desce k dispozici přímo nejsou, nebo jej přinejmenším autor práce nenašel. Jedná se tedy o jednu o jednu z největších nevýhod, jelikož v případě, že chceme programovat v jazyku symbolických adres pohodlně s českým prostředím, musí být v počítači nainstalováno Arduino IDE (po instalaci zabírá přibližně 500 MB), Visual Studio 2015 (přibližně 4 GB + něco málo přes 1 GB po instalaci české lokalizace) Proč po počeštění program tolik nabyl na velikosti se autorovi práce nepodařilo zjistit.

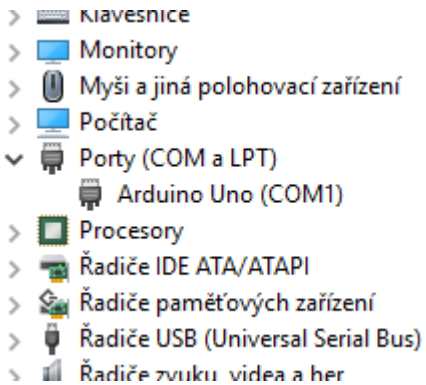
A na závěr samozřejmě nesmíme zapomenout na samotné Atmel Studio (po instalaci necelé 2 GB). Celkově tak softwarová složka obsadí minimálně 7,5 GB na pevném disku. Píši minimálně jelikož se autor obává, že s přibývajícím updaty může číslo ještě nepatrně narůst. Bohužel je to takto dané že i když z Visual studia není využito kromě jádra nic, stejně je potřeba celé, protože samotný Microsoft jej jinak nenabízí.

Zpět tedy k instalaci Arduino IDE, během instalace je tedy nainstalován i ovladač k Arduinu viz obrázek ()



Obrázek 24: Potvrzení instalace ovladače pro desku Arduina

Po dokončení instalace je zapojené Arduino možné nalézt ve správci zařízení

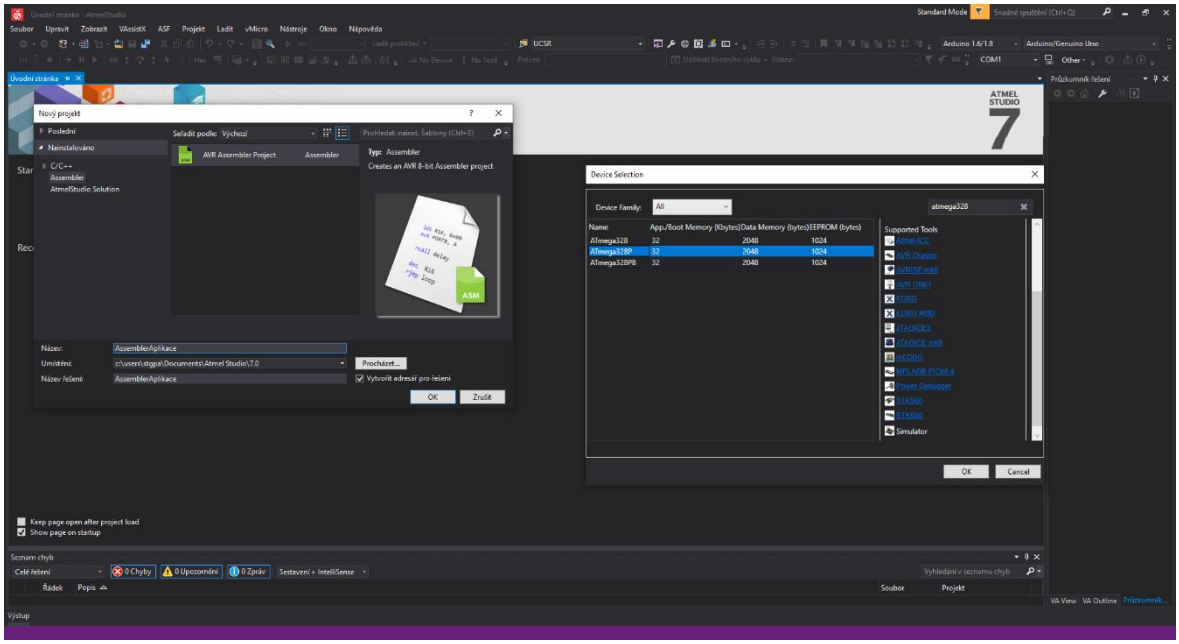


Obrázek 25: úspěšně detekovaná deska Arduino Uno

V tomto případě je zařízení na portu COM1, ovšem je možné tento port i změnit dle potřeby. Jakmile počítač dokáže komunikovat s Arduinem, zbývá (ještě bohužel) ještě poslední krok, který bude muset být nastaven při každém otevření nového projektu.

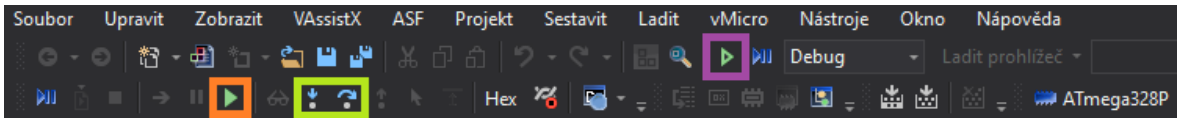
5.2.1 Seznámení se s prostředím Atmel studia

V případě, že jste si nastavili prostředí do češtiny, není těžké se v programu orientovat, proto budou popsány jen elementární věci, které neumí být zřejmé na první pohled. Po spuštění a vytvoření nového projektu viz obrázek (obr) vybereme nový assebler program a vybereme odpovídající mikrokontrolér. Tato část by měla být snadná a intuitivní i bez tohoto návodu.



Obrázek 26: Vytvoření nového projektu a vybrání odpovídajícího mikrokontroléru

Co již však plně intuitivní být nemusí, je to, jak vlastně říct prostředí, zda chceme využít simulátor, nebo přímo nahrát program do mikropočítače.

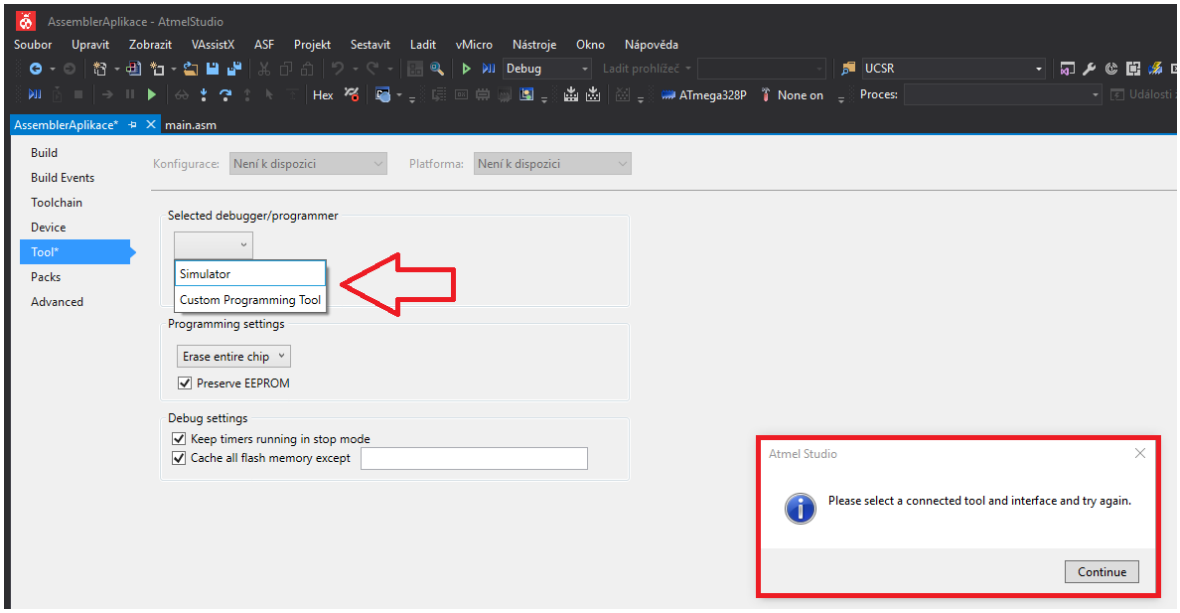


Obrázek 27: Ikony, některé i přes svou podobnost slouží k jiným účelům

Začneme tím, jako bychom vytvářeli skutečný program. Při prvním napsání programu bychom zcela jistě využili simulátor. V hlavní nabídce se kromě hlavního menu nahoře line minimálně dvouřada (závisí na rozlišení) ikon, z nichž hned dvě nápadně připomínají spuštění programu. Simulaci bychom spustili plnou ikonkou spuštění (zvýrazněno oranžově) –

tuto možnost však nedoporučuji. V případě, že již bude simulátor nastaven (bude popsáno ihned o pár řádků níže) tak se program spustí v simulaci a prakticky běží stejně rychle, jako miktrokontrolér (16 MHz) což je pro člověka příliš rychlé a dá se zde akorát využít při testování simulace programu které mají reagovat na dané vstupy. Proto pokud chceme sledovat, jak v simulaci pracují jednotlivé registry a podobně, doporučuji po dopsání programu zvolit jednu ze dvou ikonek zvýrazněných zeleně (při prvním stisknutí a pro programy, jenž nemají podprogramy, je jedno kterou vyberete) Takto budete moct klávesami F10 či F11 krokovat každou jednotlivou instrukci a prohlédnout si obsah registrů, ale také například stavového (SREG) registru, pokud si například nejste jistí, jaké příznaky (flags) vyvolá daná instrukce pro konkrétní hodnoty.

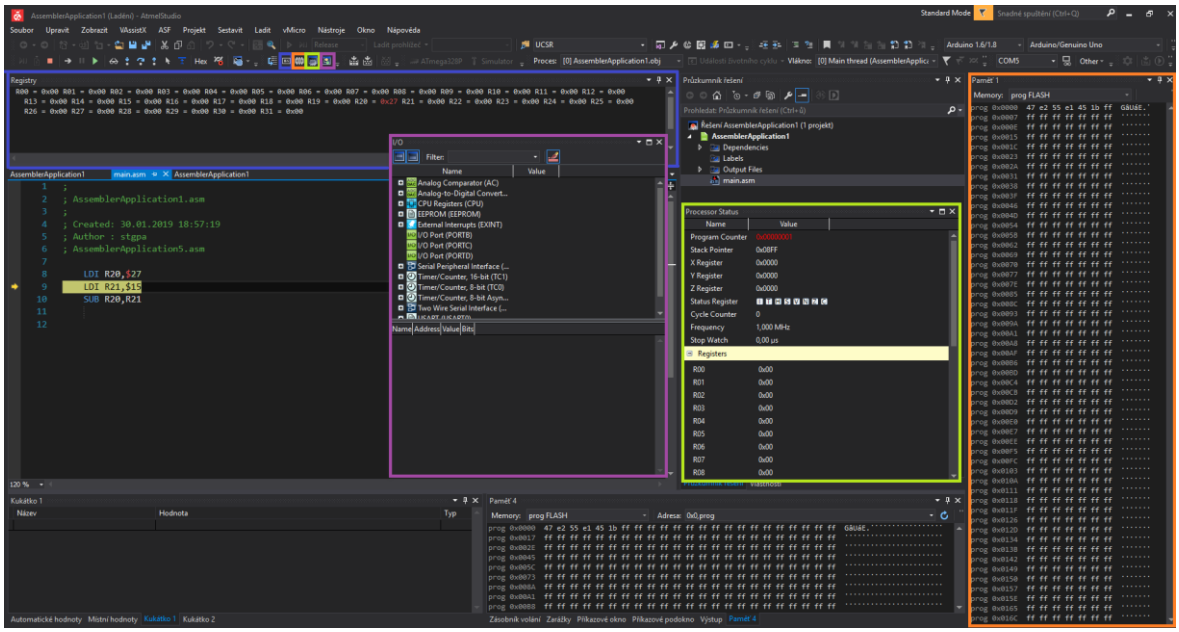
Jakmile tedy stisknete klávesu pro krokování (F10 či F11), otevře se další záložka projektu s nastavením. V případě, že byste si ji omylem zavřeli, najdete ji v hlavním menu kde se nachází Projekt jehož poslední možnost je: „vlastnosti (názevProjektu)“.



Obrázek 28: Výběr programátora či simulace

Zde se musí ještě vybrat, (stačí jednou) zda má být program simulován (simulátor) nebo zda se má využít „vlastní programovací nástroj“. K této možnosti se ještě vrátíme, nyní však dokončíme popis simulátoru. V případě, že tedy vyberete simulátor, pak již stačí klikat pro simulaci na ikonky zvýrazněné zeleně z obrázku (Obrázek 27) (nebo klávesovou zkratkou) a začne samotná simulace.

Na (Obrázek 29) jsou znázorněny jednotlivá okna pro samotnou simulaci. Samozřejmostí je si je v prostředí přesunou dle libosti a v případě zavření je možné jej znovu otevřít právě zvýrazněnými ikonkami nahoře v horní liště. Registry ukazují aktuální obsah registrů, Okno statusu procesoru zobrazuje obsah PC (Program Couter/u) což je počítadlo instrukcí a pro mikropočítač je důležité primárně z toho důvodu, aby v případě podprogramu či přerušení věděl, kam se vrátit (například probíhá 320 instrukce a mikročip si může uložit pouze číselné hodnoty do zásobníku) tedy uloží si hodnotu (ve skutečnosti adresu – je to instrukce na adrese 320 v paměti programu) do zásobníků a začne vykonávat podprogram/přerušení. Dále je zde vidět adresa na kterou ukazuje vrchol zásobníku, Status registr a registry a také frekvence krystalu. Níže jsou ještě znovu registry, v této části okna však lze registry přepnout z hexadecimálního do binárního zobrazení (což je výhodné při práci s jednotlivými bity či rotacemi) a u okna ‚klasických' registrů, to možné není. Dále máme podokno paměti, je na člověku, zda si nechá zobrazit paměť FLASH (paměť programu) či paměť dat (data REGISTERS). Poslední okno je I/O (z anglického Input/Output) jenž nám zobrazuje obsah vstupních a výstupních registrů. Pro potřeby simulace můžeme jednotlivé bity, které nastavíme v programu jako vstupní klikáním změnit, ovšem změna se projeví až při otočce cyklu celého programu. Je jen potřeba myslet na to, že např. právě moduly jež jsou součástí výuky mají ve vypnutém stavu log 1 a při sepnutém log 0. což ze začátku může zmást osobu, jenž si chce „odkrokovat" program pro práci s externím modulem.



Obrázek 29: Popis oken při simulaci programu v Atmel studiu

Simulace se vypíná červeným tlačítkem „Ukončit ladění“. Plný červený čtverec nelze přehlédnout. Ještě je třeba podotknout, že i když se rozdílnými ikonami spouští simulátor a nahrávání do mikropočítače, samotné prostředí i přesto vyžaduje přepnutí programátoru ve vlastnostech projektu.

5.2.2 Nastavení AVRDUDE

Název této utility je AVR Downloader/UploaDEr. Tato utilita se vyvíjí od roku 2002 avšak pro 8bitové mikročipy které se v podstatě dodnes vyrábějí prakticky beze změny, není problémem, že nejnovější verze je již více než 3 roky neaktualizovaná. Je to dáno dle autora i tím, že předchozí verze byly aktualizovány průměrně alespoň 2x do roka a je tak již dost možné, že vše již bylo odladěno tak jak má být. Tento program tedy umožnuje nahrávat program do AVR mikrokontrolerů. Je to tedy to, co vezme přeložený strojový kód („nuly a jedničky“) a nahraje jej přímo do mikrokontroléru.

Asi se může zdát, že bude potřeba ještě znova stahovat a instalovat další dodatečné programy. Naštěstí AVRDUDE je právě součástí Arduino IDE a pokud jste tedy nainstalovali Arduino IDE, v adresářích, kde je vývojové prostředí nainstalováno, AVRDUDE naleznete. Nyní je potřeba na něj odkázat. Zde by autor rád upozornil, že mu samotný program (v Atmel studiu) nefungoval, a nejspíše to bylo právě tím, že cesta k souborům obsahuje mezery.

Program AVRDUDE najdete tedy:

C:\Program Files (x86)\Arduino\hardware\tools\avr\bin\avrdude.exe

Dále je ještě potřeba konfigurační soubor, jenž se nachází na v adresáři:

C:\Program Files (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf

Jak již bylo řečeno, (alespoň autorovi práce) tyto cesty k souborům obsahující mezeru způsobily problém, proto se rozhodl oba soubory přesunout do adresáře: C:\avrdude\. A nyní již tedy k samotnému finále, aby po překladu programu byl program také nahrán do mikrokontroléru. Na (Obrázek 28) pokud je vybrána možnost kdy nechceme simulovat, objeví se ještě níže textové pole pro příkaz programátoru. Prakticky stačí jen zadat adresu k utilitě, jaký typ mikrokontroléru je použit, port a odkaz k samotnému strojovému kódu (soubor.hex). Je dobré vědět, že cesty k souborům se mohou lišit, zvláště, pokud si utilitu nepřesunete například do podadresáře, jenž se nachází v kořenu oddílu.

Příkaz tedy je:

C:\avrdude\avrdude.exe -C "C:\avrdude\avrdude.conf" -p m328p -c arduino -P COM1 -b 115200 -U flash:w:"$(OutputDirectory)\$(OutputFileName).hex":i"

Nejdůležitější parametry jsou následující: (v závorce je uvedeno, zda jsou povinné či nikoli)

-C - udává cestu ke konfiguračnímu souboru (zde povinný*)

-p - udává typ mikrokontroléru. (povinný)

-c - typ zařízení (programátor) (povinný)

-P - port (nepovinný) Pokud jej nezadáme, bude implicitně zkoušet port COM1

-b - přenosová rychlost (z anglického baudrate) (nepovinný)

-V - nastaví, aby se data před nahráním nekontrolovala (nepovinný)

-U -nastavuje, kam a zda se bude nahrávat či číst, viz příklad níže (povinný)

Nejdříve probereme parametr -U. Zde je několik možností, co za něj uvést. Prvním slovem je pamět, se kterou má pracovat, může to být jak *flash* (pamět programu) tak například *eeprom* nebo *boot* (sekce). Za pamětí následuje dvojtečka a písmeno, jaká operace je požadována.

r z paměti zařízení se bude číst

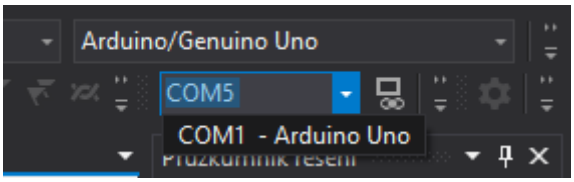
w do paměti zařízení se bude zapisovat

v stejné jako ‚r' jen s tím rozdílem, že slouží pro více zařízení.

Následuje další dvojtečka a za ní je cesta k souboru použitá zápisem, jak jen využívá Visual studio, tedy zápisem pro adresář Release aktuálního adresáře, a souboru, jehož obsah se má přenést.

Barevně je zvýrazněna ta část příkazu, která se právě může lišit. Úplně nejčastěji se bude lišit port. Jaký port je momentálně využíván arduinem se dozvíte ve správci zařízeni. V případě, že by toto řešení bylo nainstalováno na školních počítačích, kde by žáci neměli přístup ke správci zařízení, jsou ještě možné dvě možnosti, jak port zjistit, bez administrátorských práv. Prvním je spuštění IDE Arduina, kde v nabídce „Nástroje" je možné vidět port, na kterém je připojeno právě Arduino. Druhou pohodlnější možností je doinstalování pluginu do Atmel studia. Tento plugin je primárně určený k tomu, aby bylo možné programovat v Atmel studiu stejně jako v Arduino IDE (vyšší programovací jazyk). Odtud tedy název:

Arduino IDE for Atmel Studio 7. Doplňky lze získat přímo v programu v menu v položce Nástroje -> Rozšíření a aktualizace. Velikost tohoto doplňku je přibližně 30 MB.



Obrázek 30: Rozšíření, jenž ukazuje port, na kterém je připojeno Arduino

Zde však platí že tento doplněk, kromě toho, že nám v horní liště vývojového prostředí ukáže Port, na kterém je připojen mikropočítač, (jelikož port se může měnit v závislosti na tom, do kterého USB portu žák mikropočítač připojí)

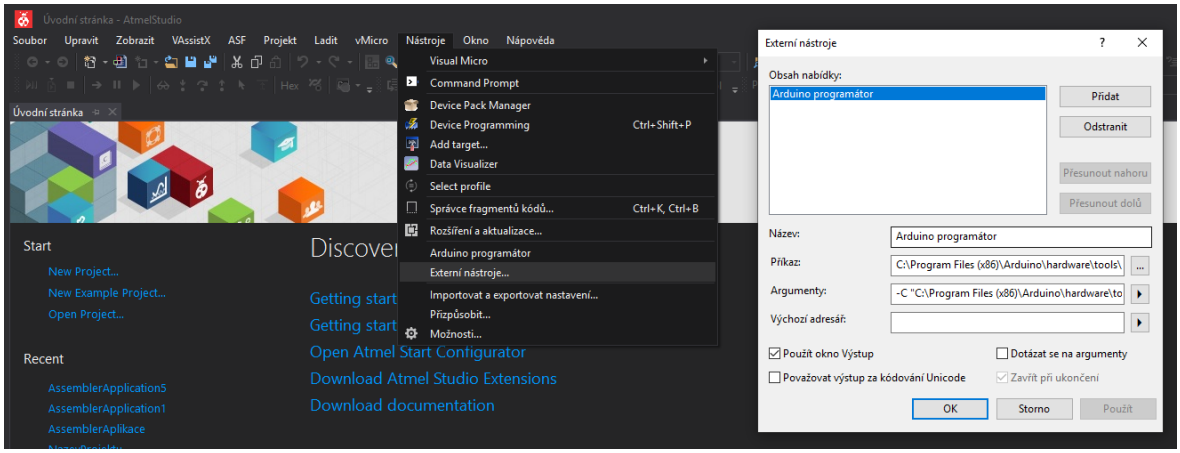
*Co se týče parametru -C, ten je sice dle dokumentace nepovinný, ovšem pokud není definována cesta ke konfiguračnímu souboru, na jehož základě se hledá mj. typ mikrokontroléru (parametr -p), tak nahrání programu do mikrokontroléru nebude úspěšné.

Pokud jste příkaz nastavili správně, pak vám nahrání programu musí fungovat a to poznáte hned dvěma způsoby, prvním je, že uvidíte že program dělá to co má (například že bliká LED) a tím ne nesporné, že program byl do mikrokontroléru nahrán. Pokud však program nemá jak „dát vědět“ zda je vše v pořádku, je to patrné také z výpisu (Output) kde se přepne výstup na „Custom programing tools“ Výstup by měl vypadat následovně a nejdůležitější částí je tedy poslední řádek jenž nás informuje, že proces byl dokončen. Na předposlední řádek je potřeba si dávat pozor, jelikož i když bude příkaz nastaven špatně či nebude fungovat komunikace, vždy na výstupu avrdude vytiskne kousek výše seznam chyb a ukončí se právě řádkem: „*avrdude.exe Done. Thank you*“ což může na první pohled mást, že vše proběhlo v pořádku, ovšem není tomu tak. Toto nahrání se provede ikonkou nevyplněného trojúhelníku (Obrázek 27). Nevýhodou je to, že příkaz k programátoru musíme vložit do každého nového projektu, jenž vytvoříme (pokud tedy není vytvořen projekt pro simulace). Níže je ukázka úspěšného nahrání:

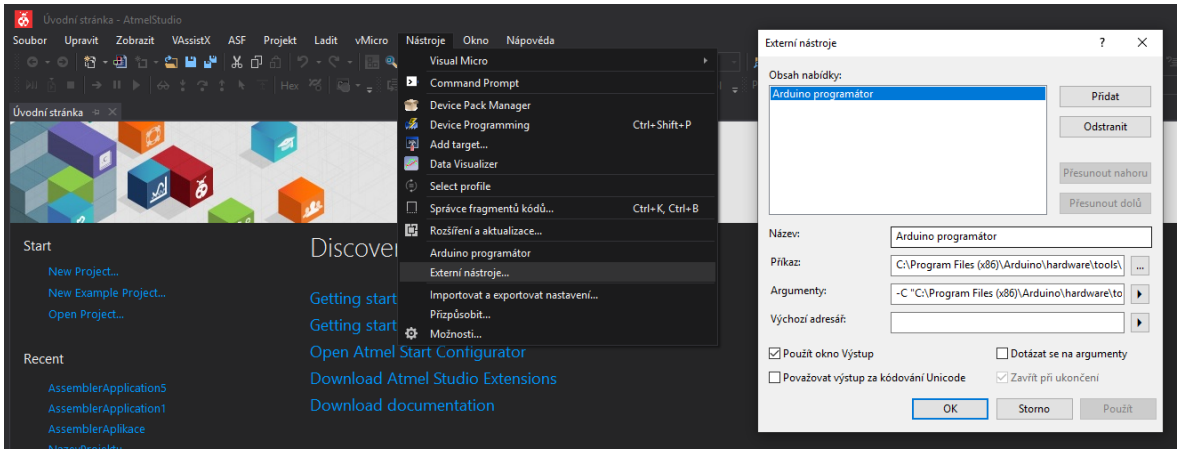
```
Executing Custom tool:
	Command: C:\avrdude\avrdude.exe -C "C:\avrdude\avrdude.conf" -p m328p -c ar-
duino -P COM1 -b 115200 -U flash:w:"C:\Users\stgpa\Documents\Atmel Studio\7.0\Assem-
blerApplication5\AssemblerApplication5\Release\AssemblerApplication5.hex":i"
	Parsed executable: C:\avrdude\avrdude.exe
	Parsed arguments: -C "C:\avrdude\avrdude.conf" -p m328p -c arduino -P COM1 -b
115200 -U flash:w:"C:\Users\stgpa\Documents\Atmel Studio\7.0\AssemblerApplica-
tion\AssemblerApplication\Release\AssemblerApplication.hex":i"
[stderr] avrdude.exe: AVR device initialized and ready to accept instructions
```

```
[stderr] Reading | ################################################## | 100% 0.00s
[stderr] avrdude.exe: Device signature = 0x1e950f (probably m328p)
[stderr] avrdude.exe: NOTE: "flash" memory has been specified, an erase cycle will
be performed
[stderr]              To disable this feature, specify the -D option.
[stderr] avrdude.exe: erasing chip
[stderr] avrdude.exe: reading input file "C:\Users\stgpa\Documents\Atmel Stu-
dio\7.0\AssemblerApplication\AssemblerApplication\Release\AssemblerApplication.hex"
[stderr] avrdude.exe: writing flash (206 bytes):
[stderr] Writing | ################################################## | 100% 0.04s
[stderr] avrdude.exe: 206 bytes of flash written
[stderr] avrdude.exe: verifying flash memory against C:\Users\stgpa\Documents\Atmel
Studio\7.0\AssemblerApplication\AssemblerApplication\Release\AssemblerApplica-
tion.hex:
[stderr] avrdude.exe: load data flash data from input file C:\Users\stgpa\Docu-
ments\Atmel Studio\7.0\AssemblerApplication\AssemblerApplication\Release\AssemblerA-
pplication.hex:
[stderr] avrdude.exe: input file C:\Users\stgpa\Documents\Atmel Studio\7.0\Assemble-
rApplication\AssemblerApplication\Release\AssemblerApplication.hex contains 206 by-
tes
[stderr] avrdude.exe: reading on-chip flash data:
[stderr] Reading | ################################################## | 100% 0.03s
[stderr] avrdude.exe: verifying ...
[stderr] avrdude.exe: 206 bytes of flash verified
[stderr] avrdude.exe: safemode: Fuses OK (E:00, H:00, L:00)
[stderr] avrdude.exe done.  Thank you.
Process exited with code 0
```

Na SPŠ se ještě kromě tohoto postupu používá ještě alternativní. V Nabídce Nástroje lze nalézt položku Externí nástroje. Zde lze představit programátor tak, aby již byl v nabídce menu. Níže v (



Obrázek 31) lze v pravé části vidět již nakonfigurovaný programátor a ve střední části již je v rámci menu nad položkou externí nástroje právě položka „Arduino programátor“ (tento název si lze však libovolně zvolit). Uživatel tedy má možnost nastavit ‚nalikování‘ k AVRDUDE zde (stejně jako v předchozím případě – nastavení a parametry jsou totožné), a poté stačí stisknout v menu právě toto a není potřeba v každém projektu vložit příkaz v nastavení projektu.



Obrázek 31: Použití externího nástroje v Atmel studiu

Bohužel toto má však jedno úskalí. V případě této varianty může zejména začínající uživatel (student) snadněji přehlédnout stav, kdy nahrávání nebylo úspěšné, (například při úspěšné komunikaci ale chybě v kódu) jelikož v předchozím případě při chybě (v jakémkoli případě) nás prostředí informuje stavovým oknem, že proces byl neúspěšný. V tomto případě se však informace o tom, že proces nebyl úspěšných „schová“ do výpisu který může vypadat následovně: (v konzoli text není zvýrazněn, to pouze zde)

```
avrdude.exe: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude.exe: Device signature = 0x1e950f (probably m328p)
avrdude.exe: NOTE: "flash" memory has been specified, an erase cycle will be perfor-
med
             To disable this feature, specify the -D option.
avrdude.exe: erasing chip
avrdude.exe: reading input file ""C:\Users\user\Documents\Atmel Studio\7.0\Assemble-
rApplication\AssemblerApplication\Debug\.hex""
avrdude.exe: can't open input file "C:\Users\user\Documents\Atmel Studio\7.0\Assem-
blerApplication\AssemblerApplication\Debug\.hex": Invalid argument
avrdude.exe: read from file '"C:\Users\user\Documents\Atmel Studio\7.0\AssemblerAp-
plication\AssemblerApplication\Debug\.hex"' failed

avrdude.exe: safemode: Fuses OK (E:00, H:00, L:00)

avrdude.exe done.  Thank you.
```

ZÁVĚR

Cílem této práce bylo nachystat popsat nové prostředí pro výuku jazyka symbolických adres pro Střední průmyslovou školu ve Zlíně. Alfou i omegou této práce bylo využití nápadu, že se přejde na finančně vhodné řešení, jenž umožní žákům si mikrokontrolér samotný pořídit a procovat na něj z domu, a využít původní moduly, které škola využívala.

Teoretická část se tedy okrajově věnuje zastaralému řešení, jeho výhodám a nevýhodám a proč se přešlo na řešení novější. Popisuje platformu AVR od firmy Microchip a i přestože prakticky není mikrokontrolér používán, je zde nastíněn i Arduino Mega, jelikož, jak je popsáno v teoretické části, Arduino Uno, na kterém má být práce primárně konstruována, má nedostatečné množství portů.

V praktické části jsou popsány moduly, jež měli být v ideálním případě využity pro ukázkové příklady, bohužel však právě z důvodu úspornější verze Una není možné příklady pro modul klávesnice v současném stavu implementovat. Dále jsou v příloze vyhotovené příklady, jenž obsahují praktické komentáře, a v neposlední řadě je zde také primitivní kalkulátor jenž je schopna vypočítat zpoždění které by se jinak v nižším programovacím jazyku muselo počítat vždy manuálně.

SEZNAM POUŽITÉ LITERATURY

[**Chyba! Nenalezen zdroj odkazů.**] BARR, Michael a Anthony J MASSA. Programming embedded systems: with C and GNU development tools. 2nd ed. Sebastopol: O?Reilly, 2006, xxi, 301 s. ISBN 978-0-596-00983-0

[2] CATSOULIS, John. Designing embedded hardware. 2nd ed. Sebastopol, CA: O'Reilly, 2005, xvi, 377 p. ISBN 0596007558.

[3] MARGOLIS, Michael. Arduino cookbook. 2nd ed. Sebastopol, Calif.: O'Reilly, 2012, xx, 699 p. ISBN 1449313876

[4] MATOUŠEK, David. Práce s mikrokontroléry ATMEL AVR ATmega16. 1. vyd. Praha: BEN – technická literatura, 2006, 319 s. μC. ISBN 80-730-0174-8.

[5] MAZIDI, Muhammad Ali, Sarmad NAIMI and Sepehr NAIMI. The AVR microcontroller and embedded systems: using Assembly and C. Upper Saddle River, N.J.: Prentice Hall, 2011, xiv, 776 p. ISBN 01-380-0331-9.

[6] PINKER Jiří. Mikroprocesory a mikropočítače. 1 vyd. Praha: BEN – technická literatura, 2004, 159 s. ISBN 80-7300-110-1.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ALU	Arithmetic logic unit (Aritmeticko-logická jednotka)
AVR	Alf and Vegard's RISC procesor
CISC	Complex Instruction Set Computing (Kompletní instrukční sada)
DIP	Dual in-line package
EEPROM	Electrically Erasable Programmable Read-Only Memory
GB	Gigabajt
GCC	GNU Compiler Collection
GNU	GNU's Not Unix
I/O	Input/Output (česky: vstup/výstup, je zde i pod českou zkratkou V/V)
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
JTAG	Join Test Action Group
KB	Kilobajt
LED	Light-Emitting Diode
MB	Megabajt
PC	Program Counter
PDF	Portable Document Format
PWM	Pulse Width Modulaction
RISC	Reduced Instruction Set Computer (Redukovaná instrukční sada)
SRAM	Static Random Access Memory
SREG	Status Register (statusový registr)
UART	Universal Asynchronous reciiver-transmitter
V/V	Vstupně/Výstupní
VS	Visual Studio

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Tištěné:

Příloha P1

Příloha P2

Příloha P3

Elektronické:

Instalační soubory k Atmel Studiu

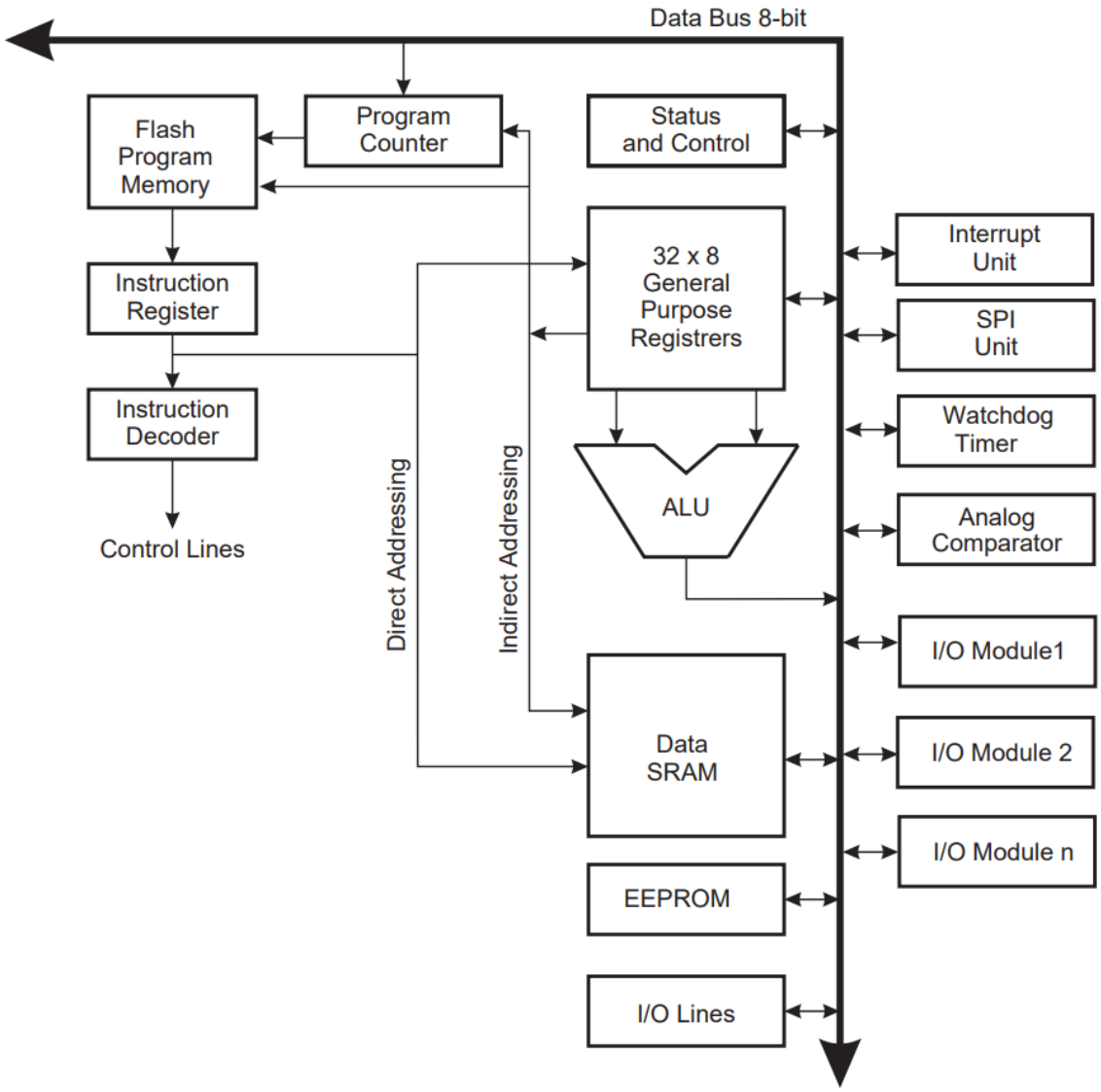
Prezentace k teoretické části předmětu

Ukázkové programy k samotné výuce

Kalkulátor časového zpoždění (webový soubor)

„Tahák instrukcí a registrů“

PŘÍLOHA P I: ZJEDNODUŠENÉ BLOKOVÉ SCHÉMA AVR ARCHITEKTURY, JENŽ KORESPONDUJE S ATMEGA32

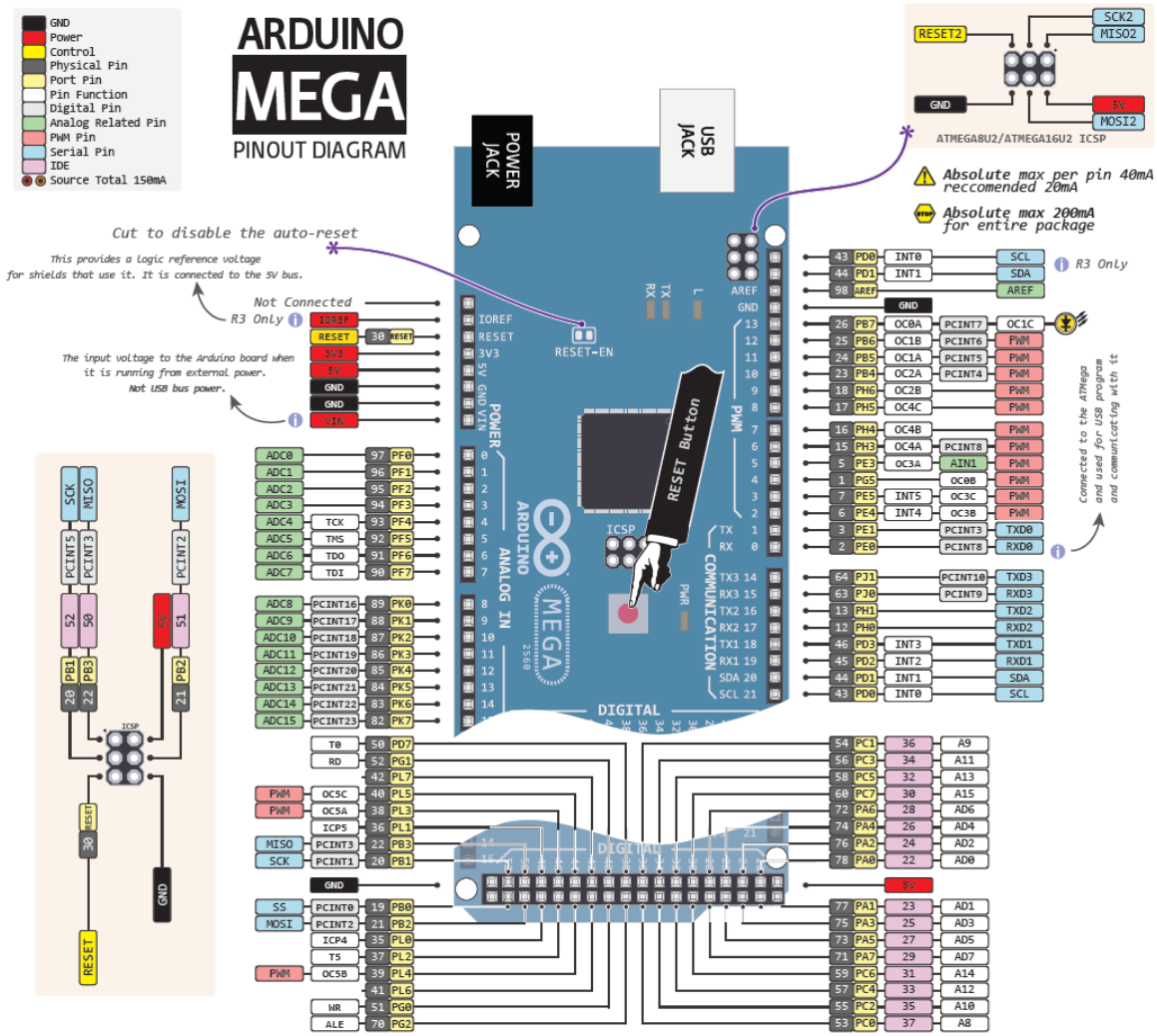


Obrázek 32: Zjednodušené blokové schéma ATmega32 (zdroj: datasheet)

PŘÍLOHA P II: PINY MIKROKONTROLÉRU ATMEGA328P

Číslo pinu	Popis (PORT)	Funkce
1	PC6	Reset
2	PD0	Digitální pin (RX)
3	PD1	Digitální pin (TX)
4	PD2	Digitální pin
5	PD3	Digitální pin (PWM)
6	PD4	Digitální pin
7	Vcc	Napětí (ze zdroje do mikrokontroleru)
8	GND	Zem (GND)
9	XTAL 1	Krystalový oscilátor
10	XTAL 2	Krystalový oscilátor
11	PD5	Digitální (PWM)
12	PD6	Digitální Pin (PWM)
13	PD7	Digitální Pin
14	PB0	Digitální Pin
15	PB1	Digitální Pin (PWM)
16	PB2	Digitální Pin (PWM)
17	PB3	Digitální Pin (PWM)
18	PB4	Digitální Pin
19	PB5	Digitální Pin
20	AVCC	Pozitivní (+) Stejnosměrné napětí
21	AREF	Referenční napětí
22	GND	Zem (GND)
23	PC0	Analogový vstup
24	PC1	Analogový vstup
25	PC2	Analogovy vstup
26	PC3	Analogový vstup
27	PC4	Analogový vstup
28	PC5	Analogový vstup

PŘÍLOHA P III: ROZLOŽENÍ PINŮ A PORTŮ ARDUINO MEGA:



Je port na desce? / číslo pinu	0	1	2	3	4	5	6	7
PORTA	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano
PORTB	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano
PORTC	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano
PORTD	Ano	Ano	Ano	Ano	-	-	-	Ano
PORTE	Ano	Ano	-	Ano	Ano	Ano	-	-
PORTF	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano
PORTG	Ano	Ano	Ano	-	-	Ano	-	-
PORTH	Ano	Ano	-	Ano	Ano	Ano	Ano	-
PORTK	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano
PORTL	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano

Tabulka: Piny jednotlivých portů – zda fyzicky jsou na desce Arduina Mega