Čo je to MCU? Čo je to program?

Už niekoľkokrát som ľuďom zo svojho okolia, pomáhal pri prvých krokoch v programovaní mikrokontrolérov.
Aby som sa znova neopakoval, svojim poznámkam som dal akú-takú štábnu kultúru, a ponúkam ich v tomto miniseriáli širokej verejnosti.
Miniseriál vyžaduje od čitateľa minimálne znalosti z elektrotechniky (ohmov zákon, RAM, ROM) a matematiky (BIN, HEX, DEC).

Mikrokontrolér (MCU, mikroprocesor, jednočipový procesor, jednočipák,...)

Je to logická elektronická súčiastka, ktorej správanie v el. obvode je predurčené sadou návestí, povelov, inštrukcií,... skrátka - programom.
Tento program je trvale uložený vo vnútri MCU (ROM pamäť) a ostáva zachovaný aj pri neprítomnosti napájacieho napätia.
Program je však možné zmeniť čím sa z MCU stáva univerzálny blok na doske s plošnými spojmi, ktorý eliminuje počet logických a periférnych integrovaných obvodov, a užívateľ dokáže flexibilne meniť správanie sa celého zapojenia.

Ako to funguje

V MCU sa po privedení napájacieho napätia, rozkmitá oscilátor (podľa druhu integrovaný v MCU alebo externe pripojený k MCU), ktorý je pre celý mikrokontrolér niečo ako nepokoj v naťahovacích hodinkách. Tento signál zabezpečuje celý chod MCU. Nazýva sa aj MainClock (hlavné hodiny), alebo CLK.
Pokiaľ nenastane zmena tohto signálu, MCU nič nevykoná. Všetka jeho činnosť je synchronizovaná práve týmto signálom.
Keď už beží hodinový signál MCU síce "žije", ale nič nerobí. Chýba mu povolenie aby robil.
Na to slúži signál RESET.
Tento je pripojený na jeden pin MCU a je ovládaný z okolitého zapojenia.
Zmena logickej úrovňe na tomto signále zabezpečí, nastavenie celého MCU do východiskového stavu. Jeho spätné prestavenie povolí MCU vykonávanie definovanej činnosti.
A tou je spracovávanie programu. Ten sa nachádza v pamäti ROM. Interná logika vykonávania programu zabezpečuje čítanie krok-po-kroku jendotlivých návestí (inštrukcií), predanie tejto inštrukcie do jadra MCU (aritmetická jednotka), ktorá následne túto inštrukciu vykoná. Na pomoc má sadu registrov (časť pamäte RAM), ktoré sa využívajú na ukladanie výsledkov logických výpočtov, pomocných premenných a pod.
Všetko samozrejme časované signálom CLK.

Príklad inštrukcií pre potrebných pre blikanie s LED diódou:

1. :100000000EC026C025C024C023C022C021C020C0ED
2. :100010001FC01EC01DC01CC01BC01AC019C01124A7
3. :100020001FBECFEDCDBF10E0A0E6B0E0E6E7F0E008
4. :1000300002C005900D92A036B107D9F710E0A0E6F6
5. :10004000B0E001C01D92A036B107E1F702D012C0A6
6. :10005000D7CFC398BB9AC39AC398C39AC398C39A7D
7. :10006000C398C39AC398C498BC9AC49AC49880E0B1
8. :0600700090E00895FFCFAF
9.

```
:00000001FF
```

Toto je obsahu pamäte ROM z MCU, zarovnaný formátom IntelHex.

Formát inštrukciií je síce čitateľný pre samotné MCU, ale pre užívateľa ktorý chce zmeniť chovanie MCU v zapojení (čítaj: programátora, ktorý chce modifikovať program) je to trocha neprehľadné.
Preto má každá inštrukcia aj svoj názov (Mnemonics). A aby to zasa nebolo tak jednoduché, každý druh MCU, má svoj definovaný formát inštrukcií, ich názvov, bitovej dĺžky, počtu CLK cyklov potrebných k jej vykonaniu, a tak ďalej.....

A ako by teda vypadal výpis programu v mnemokódoch?
Asi takto (toto nezodpovedá vyššie uvedenému kódu! Jedná sa iba o príklad...):

```
1.
    ldi     R16, $44        ;  Load low byte address of end of RAM into register  R16
2.
    out     SPL,R16                     ;  Initialize stack pointer to end of internal RAM
3.
    ldi     R16, $89      ;  Load high byte address of end of RAM into registe r R16
4.
    out     SPH, R16        ;  Initialize high byte of stack pointer to end of i nternal RAM
5.


6.
    ldi     AL,low(-$fe3)
7.
    ldi     AH,high(-$fe3)    ;A=-$fe3
8.
    ldi     BL,$40
9.
    ldi     BH,$20                  ;B=$2040
10.
    rcall   comp16s       ;Compare
11.
    mov     G1L,AL                  ;Move low byte
12.
    mov     G1H,AH                  ;Move high byte
13.


14.
    ldi     AL,$00
```

```
15.
        ldi     AH,$50                  ;A=$5000
16.
        ldi     BL,$bc
17.
        ldi     BH,$4f                  ;B=$4fbc
18.
        rcall   comp16s     ;Compare
19.
        mov     G2L,AL                  ;Move low byte
20.
        mov     G2H,AH                  ;Move high byte
21.
22.
        ldi     AL,low(-$7f34)
23.
        ldi     AH,high(-$7f34)    ;A=-$7f34
24.
        ldi     BL,low(-$3e12)
25.
        ldi     BH,high(-$3e12)    ;B=-$3e12
26.
        rcall   comp16s       ;Compare
27.
        mov     G3L,AL                  ;Move low byte
28.
        mov     G3H,AH                  ;Move high byte
```

Takýto formát sa nazýva zdrojový kód v assembleri.
Toto je už pre programátora prijateľný zápis programu. Ovšem pre MCU nepoužiteľný. Preto programátor musí použit prekladač (čítaj: kompilér), aby sa tento zápis očistil od komentárov (texty za bodkočiarkou), otestoval na syntax, a pod..., ale hlavne každý mnemokód bude nahradený kódom inštrukcie pre MCU, ktorý sa definuje pri preklade. Po ukončení kompilácie ktorej výstupom je objektový súbor, sa ešte spustí linker, ktorý definitívne usporiada tento kód podľa dlžky inštrukcií do formátu vhodného na prepis do ROM pamäte, a nahradí návestia pre volania programov konrétnymi adresami v pamäti ROM.
Potom už stačí použiť iba formátovaciu utilitku a jej výsledkom je pekný IntelHEX formát (napríklad), ktorý pomocou programátora nahráme (napálime, naloudujeme) do pamäte ROM.

Ako vidno, zdrojový kód v asembléri, je prvý možný čitateľný kód pre programátora.
Je to programovací jazyk najnižšej úrovne.
Výhodou tohoto programovacieho jazyka je, že programátor má plne pod kontrolou chod programu ako aj samotné zostavovanie a členenie programu, čím má možnosť zostaviť výkonnejší alebo neštandartný program.
Nevýhodou (??) je, že tento program je použiteľný výhradne pre MCU pre ktorý bol napísaný (nie je prenositeľný), pretože je

pevne naviazaný na inštručnú sadu MCU.

Odstránenie týchto nevýhod (a výhod) riešia programovacie jazyky vyššej úrovne (mno aj keď by som ich tak nemal nazvať).
Tie mjú za úlohu skompilovať program v definovanom formáte syntaxe.
Výhodou týchto jazykov je prenositeľnosť zdrojového kódu. Rovnaký zápis programu sa dá (samozrejme do určitej miery) použiť pre rôzne rodiny MCU.
Nevýhodou týchto jazykov, je strata výkonnosti (v určitých situáciach), strata absolútnej kontroly nad výsledným programom.
Výhodou je jednotná syntax pre rôzne platformy a lepšia prehľadnosť kódu.
Medzi tieto jazyky patria:

- C

- Pascal

- Basic

- ...

Ja v nasledujúcich praktických príkladoch použijem programovací jazyk "C".
Ako MCU použijem radu AVR od firmy Atmel.

AVR (Advanced Virtual RISC) je označenie celej série 8-bitových mikrokontrolérov, ktoré ako prvé začali používať integrovanú (On-Chip) pamäť kódu (ROM) typu FLASH (elektricky prepisovateľná, na rozdiel dovtedy používanej OTP - jedenkrát programovateľná).
MCU sa aj vďaka tejto vlastnosti plus nízkej cene a jednoduchým programovacím hardwérom, rýchlo rozšíril.

MCU majú okrem spomínanej FLASH pamäte integrovaný RAM pamät (pamäť údajov), pamäť EEPROM (pamäť dát uchovávajúca tieto dáta aj po odpojení napájania) plus široké spektrum periférnych funkcií ako USART, I2C (TWI), AD prevodníky, DA prevodníky, RTC (obvod reálneho času), PWM moduly, časovače, externé preušenia a pod.
Jednou z nezanedbateľných výhod je aj integrovaný oscilátor, vďaka čomu môže MCU pracovať s minimom podporných súčiastok.

Toto sú všetko vlastnosti, pre ktoré si tieto MCU vyberajú hlavne začínajúci programátory pre svoje pokusy (čím netvrdím že je určený na to).

Preto skúsim na tomto type MCU s použitím programovacieho jazyka C, predviesť prvé kroky pri programovaní MCU.

2. časť -->