07 - Vyšší programovací jazyky pro mcu

Omezení a rozdíly vůči programování pro PC

- absence operačního systému (u jednodušších mcu)
  - absence dynamické alokace
  - přímý přístup k HW
- norma je jazyk C
  - s omezenými možnostmi (např. paměť)
- navíc se zde oproti PC nachází instrukce, co manipulují s jednotlivými bity
  - `SET`, `CLEAR`, `TEST`
- někdy je nutno softwarově implementovat zásobník (přes ukazatel)
- **omezená nebo zakázaná rekurze**

Datové typy

- většina datových typů podporována
- navíc se zde oproti PC nachází typ **bit**
- lze určit proměnnou jako "volatile", což umožní ji asynchronně menit

Výhody a nevýhody použití assembleru

- kvalitní kód dosáhne vyšší úrovně optimalizace
- delší doba tvorby kódu
- menší přehlednost
- kód není přenosný
- u některých programů se v assembleru optimalzuje pouze jeho část

Požadavky vyšších programovacích jazyků na mcu

Použití vyšších programovacích jazyků z důvodu:

- snížení kódové nadbytečnosti
- efektivní implementace typických konstrukcí
  - pole, větvené, podprogramy…

Požadavky:

- **více pracovních registrů**
  - charakteristika RICS architektur
  - aby se data pokaždé nemusely číst z paměti
- **krátký instrukční cyklus**
  - charakteristika RICS architektur
  - ideálně jeden cyklus na instrukci
- **rozšířená podpora ukazatelů**
  - pro přístup k datům
  - pre-dekrement a post-inkrement
- **indexování polí**
  - relativní adresy pro přístup k prvkům pole
  - ukazatel je na začátek pole a obsahuje určitý offset, který slouží jako index
- **šíření příznaku nuly**
  - instrukce v sobě zahrnuje výsledek předchozí operace
  - např. když má číslo více bitů než máme k dispozici
  - pokud se nerovnalo v předchozím bitu, číslo nemůže být rovno
  - příkladem jsou instrukce "… with carry"
- **bitové proměnné**

    - používají se pro zjištění hodnoty na portech a reprezentaci hodnoty true/false
    - samostatná skupina instrukcí v instrukčním souboru
    - `SET`, `CLEAR`, `TEST`
- **ukazatel na zásobník**
    - stack pointer
    - díky němu je možno instrukci i operand přečíst v jednom cyklu

Optimalizace kompilátoru

- základ optimalizace je správný návrh kódu programátora
- dělí se na:
    - optimalizaci závislé na hardwaru
    - vyžadují podporu od zařízení (mcu)
    - optimalizaci nezávislé na hardwaru

Optimalizace závislé na hardwaru

- **registrované proměnné**
    - proměnné a parametry funkcí se umisťují do registrů místo do RAM
- **optimalizace jednoduchým přístupem**
    - kompilátor využije nejvíce se hodící instrukce pro danou činnost
    - např. bitové instrukce pro porovnávání bitů
- **reorganizace kódu**
    - změna typu smyčky (kompilátor použije FOR místo WHILE, je-li to efektivnější)
    - někdy je efektivnější čítat z druhé stranya testovat na 0, místo testování na vrchní hodnotu

Optimalizace nezávislé na hardwaru

- **zpracování konstant**
    - výpočty s více konstatami lze předpočítat v době kompilace
- **vyloučení opakujících se výpočtů**
    - kompilátor si může uložit hodnotu když zjistí, že se používá znovu
    - do registru
- **optimalizace skokových příkazů**
    - některé vnořené příkazy lze nahradit skokem přímo na cílovou adresu
    - kompilátor také volí, zda použije absolutní nebo relativní skok podle délky skoku
- **vyloučení "mrtvého" kódu**
    - pokud se v programu nachází kód, který nikdy nebude proveden, odstraní se z programu
    - "nedosažitelný kód"
- **nahrazení opakujících se úseků programu skoky**
    - pokud program zjistí, že se určité instrukce opakují, vytvoří pro to podprogram
    - program skočí na začátek instrukcí
- **negace skoků**
    - jednu větev podmínky lze odstranit negací podmínky
- **optimalizace plnění**
    - při inicializaci proměnných lze nastavit nějakou počáteční hodnotu
    - např. proměnnou "pro jistotu" vymazat
    - pokud program zjistí, že se proměnná nepoužívá, tuto inicializační funkci může vynechat
- **optimalizace jednoduchých cyklů**
    - místo použití cyklů lze kód nakopírovat za sebou
    - zvyšuje velikost paměti programu, ovšem zrychluje ho
- **rotace smyček**
    - lze zaměnit pořadí provádění instrukcí
    - pokud na sobě nejsou závislé
- **optimalizace řídícího toku**
    - náhrada za switch-case
    - switch-case se převádí na if

- pokud jednotlivé podmínky tvoří kaskádu, lze skočit přímo na správnou větev