19 Vyšší programovací jazyky pro mcu

Wednesday, 19 January 2022 09:11

Požadavky vpj na architekturu mcu, omezení a rozdíly vůči programování pro osobní počítače, optimalizace kompilátoru.

- programování pro mcu
  - absence operačního systému u jednodušších mcu
  a. C
    - norma
    - přímý přístup k HW
    - práce s jednotlivými bity (u PC řešeno softwarově)
    - omezená nebo zakázaná rekurze
    - někdy je nutno implementovat stack (zásobník) přes pointery
    - omezené HW prostředky
      - redukována velikost paměti
      - hloubka implementace zásobníku
      - podpora ALU
    - absence služeb OS
      - absence dynamické alokace
      - potřeba vlastní implementace multitaskingu
    - datové typy
      - podporována většina
      - velikost int závislá na platformě
      - float/double nejsou podporovány HW
        - potřeba SW emulace
        - výjimkou jsou novější a výkonnější ARM (Cortex M4)

  b. ASM
    - složitější
    - vysoká uroveň optimalizace
    - nepřenosnost mezi platformami

  - požadavky vyššího programovacího jazyka
    i. více pracovních registrů
      - charakteristika RISC (*Reduced Instruction Set Computer*)

    ii. SCI (*Single Cycle Instruction*)
      - RISC
      - pipelining

    iii. rozšířená podpora pointerů
      - lze jimi adresovat celou paměť

    iv. indexace polí
      - instrukce obsahuje adresu začátku pole a relativní displacement (poloha od začátku)

    v. minimálně 4 paměťové ukazatele
      - zdroj, cíl, stack pointer, datový segment

    vi. šíření příznaku nuly
      - instrukce zahrnující výsledek předchozí operace
      - carry příznakový registr

    vii. bitové proměnné
      - zjištění hodnoty na portech
      - set, clear, test (skip if bit)

    viii. instrukce i s operandy na jedné adrese
      - charakteristika RISC

    ix. HW stack pointer

    x. podpora AL instrukcí s vyšší šířkou než nativní

        - např. word - zdvojené registry
- optimalizace kompilátoru
    - cílem je minimální velikost nebo nejvyšší rychlost programu
        - tyto požadavky jdou většinou proti sobě

a. závislé na HW
    i. registrové proměnné
        - kompilátor se pokusí proměnnou uložit do registrů místo do operační paměti
    ii. optimalizace jednoduchým přístupem
        - je-li to možné, použijí se bitové operace
    iii. změna typu/směru smyčky
        - pro CPU bývá jednodušší číslování sestupně směrem k 0
        - náhrada for za while

b. nezávislé na HW
    i. zpracování konstant
        - výpočty lze předpočítat v době kompilace
    ii. vyloučení opakovaných výrazů
        - uložení hodnoty do registru
    iii. optimalizace skokových příkazů
        - vnořené příkazy lze nahradit relativním/absolutním skokem na cílovou adresu
    iv. vyloučení mrtvého kódu
        - nedosažitelný kód se odstraní z programu
    v. náhrada opakujícího se kódu za skoky
        - při opakování instrukcí se vytvoří podprogram
    vi. negace skoků
        - odstranění jedné větve podmínky její negací
    vii. překrývání dat
        - sdílení statických proměnných v několika funkcích
    viii. optimalizace plnění
        - inicializaci nepoužívaných proměnných vynechá
    ix. optimalizace smyček
        - místo cyklů se kód nakopíruje za sebe
        - zvýšení paměti, ale i rychlosti
    x. rotace smyček
        - záměna pořadí instrukcí, pokud jsou na sobě nezávislé
    xi. optimalizace řídícího toku
        - switch-case -> if