Assembler pro mainframe

Tomáš Oberhuber

Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

Zápis instrukcí

- umíme už zapisovat instrukce v binárním tvaru
- to je silně nešikovné
- pro snazší vývoj programů vznikl jazyk symbolických instrukcí - **assembler**
- každé instrukci je přidělena zkratka
- operandy lze zapisovat také v snáze čitelnější formě
- na rozdíl od vyšších programovacích jazyků platí, že jeden příkaz assembleru vytváří typicky jednu instrukci procesoru
  - assembler umožňuje definovat návěští, která negenerují žádnou instrukci
- assembler byl postupně vylepšen na macroassembler
  - umožňuje psaní maker, která se rozvinou do několika příkazů assembleru a tím pádem generují i více instrukcí
- rozlišujeme tedy:
  - strojové instrukce (machine instructions)
  - asemblerové instrukce (assembler instructions)
  - makro instrukce (macro instructions)

Překlad kódu v asembleru

- překlad probíhá obdobně jako u jiných jazyků
- samotný překlad provádí program hlasm
  - high-level assembler
- následuje linkování

Zápis kódu v asembleru

- návěští (labels), pokud je použito, musí začínat na první pozici
- kód operace (opcode) musí být od návěští oddělen alespoň jednou mezerou
- operandy musí být od kódu instrukce odděleny alespoň jednou mezerou
- komentáře musí být od operandů odděleny alespoň jednou mezerou
- používá se následující konvence:
  - návěští začíná na pozici 1
  - opcode začíná na pozici 10
  - operandy začínají na pozici 16
  - komentáře začínají na pozici 35 nebo 40
- pro navázání na další řádek:
  - napíše se jiný znak než mezera na pozici 72
  - na novém řádku se začíná na pozici 16
- pokud řádek začíná hvězdičkou *, bere se celý řádek za komentář

Ukázka kódu

```
1  PGMNM      SETUP
2             PROGRAM CODE
3             SNAPSHOT          PGMNM,ENDPGM
4             ENDIT
5             CONSTANTS AND DATA
6  ENDPGM     DS        D
7             END
```

První příklad

Napíšeme jednoduchý program, který sečte dvě celá čísla.

- dekadicky: `54321+12345=66666`
- hexadecimálně: `x'0000D431'+ x'00003039' = x'0001046A'`

První příklad

Napíšeme jednoduchý program, který sečte dvě celá čísla.

- dekadicky: `54321+12345=66666`
- hexadecimálně: `x'0000D431'+ x'00003039' = x'0001046A'`

Šablona s příklady a pomocná makra lze najít v:

- `MCOE.CLASS.ASM.DATA`
- `MCOE.CLASS.ASM.SOURCE`

První příklad

```
 1  PGM1        SETUP
 2              L          2,A
 3              A          2,B
 4              ST         2,SUM
 5              SNAPSHOT   PGM1,ENDPGM
 6  ENDIT
 7  A           DC         F'54321'
 8  B           DC         F'12345'
 9  SUM         DS         F
10  ENDPGM      DS         D
11              END
```

Rozbor kódu

- `pgm1` je název programu
- `setup` makro které provede tzv. housekeeping routine – uložení obsahu registrů
- `l 2,a`
  - load
  - načte 4 bajty (word) na adrese `a` do registru 2
- `a 2,b`
  - add
  - přičti word na adrese `b` do registru 2
- `st 2,sum`
  - store
  - ulož obsah registru 2 na adresu `sum`

Rozbor kódu

- pro sečtení dvou čísel jsme tedy využili registr 2
- lze používat bez obav registry 2 až 11
- registry 0, 1, 12, 13, 14, 15 mají speciální význam

- čísla určená ke sčítání jsou uložena na adresách `a` a `b`
- výsledek se zapisuje na adresu `sum`
- `a`, `b` a `sum` lze chápat jako proměnné ve vyšších jazycích
- definují se pomocí příkaz§ `ds` a `dc`

Rozbor kódu

- `a dc f'54321'`
  - define constant
  - vytvoří v paměti místo pro typ `f` tj. fullword (4 bajty)
  - instrukce definuje návěští `a`, které ukazuje na stejné místo v paměti
  - assembler zároveň na toto místo zapíše binární tvar pro číslo 54321
  - instrukce se jmenuje define constant, ale proměnná není konstanta, lze jí přepsat
- `sum ds f`
  - define storage
  - funguje stejně jako `dc`, ale na výsledné místo v paměti neukládá žádnou hodnotu

Rozbor kódu

- `snapshot pgm1,endpgm`
  - jde o makro, které uloží na výstup obsah paměti mezi adresami `pgm1` a `endpgm`
  - to nám slouží k tomu, abychom si mohli překontrolovat, zda náš kód pracuje správně
- `endit`
  - makro, které generuje ukončení programu a předání řízení operačnímu systému
- `end`
  - instrukce assembleru značící konec programu

Překlad kódu

- překlad kódu je mnohem jednodušší než u vyšších jazyků
- assembler používá ukazatel do výsledného "objekt souboru" tzv. `location counter`
- s každou instrukcí v kódu zjistí její velikost a dekóduje operandy
- na výstup vloží příslušnou posloupnost bajtů
- pokud vložíme instrukci `DS`, vynechá se příslušně velká mezera
  - proměnné jsou tedy uloženy přesně tam, kam je napíšeme
- vnitřní ukazatel asembleru lze ovládat speciálními příkazy
- makra se zpracovávají obdobně