Структура на програма в C

Част 2 - типове, функции

Иван Георгиев, Христо Иванов, Христо Стефанов

Технологично училище "Електронни системи",
Технически университет, София

10 март 2019 г.

Съдържание

Съдържание

Инструкции и формат на данни

- Компютърните процесори обработват данни, които се намират в RAM паметта на компютъра
- Обработването се извършва като процесорът изпълнява *последователност от инструкции* (*програма*), които манипулират данните
- Почти всеки съвременен процесор поддържа инструкции за работа с целочислени и реални числа
- За да може една инструкция да обработва данни те трябва предварително да са записани в определен вид, с който инструкцията може да работи

Инструкции и формат на данни

```
# store 1 byte integer number 3 to address 0x1000
line1: storeb   $0b00000011, [$0x1000] # 3 in decimal
# store 1 byte integer number 1 to address 0x1001
line2: storeb   $0b00000001, [$0x1001] # 1 in decimal
# load 1 byte at address 0x1000 to register r1
line3: loadb    [$0x1000], %r1
# load 1 byte at address 0x1001 to register r2
line4: loadb    [$0x1001], %r2
# add the two 1 byte integer numbers found in registers r1 and r2
# and save the result in register r3
line5: addb     %r1, %r2, %r3
# store 1 byte from register r3 to address 0x1002
line6: storeb   %r3, [$0x1002]
# result in memory at address 0x1002 is 00000100,
# which corresponds to number 4 in decimal
line7: outb     %r3
```

Фрагмент 1: Програма на асемблер за събиране на целочислени числа с големина 1 байт за измисления процесор PROG1

Инструкции и формат на данни

```
# store 4 byte real number 3.0 starting at address 0x1000
line1: storel   $0b01000000010000000000000000000000, [$0x1000]
# store 4 byte real number 1.0 starting at address 0x1004
line2: storel   $0b00111111100000000000000000000000, [$0x1004]
# load the 4 bytes starting at address 0x1000 to register fl1
line3: loadl    [$0x1000], %fl1
# load the 4 bytes starting at address 0x1004 to register fl2
line4: loadl    [$0x1004], %fl2
# add the two 4 byte real numbers found in registers fl1 and fl2
# and save the result in register fl3
line5: addf     %fl1, %fl2, %fl3
# store the 4 bytes in register fl3 to address 0x1008
line6: storel   %fl3, [$0x1008]
# result in memory at addresses 0x1008..0x100b
# is 01000000 10000000 00000000 00000000, which
# corresponds to number 4.0 in decimal
line7: outl     %fl3
```

Фрагмент 2: Програма на асемблер за събиране на реални числа с големина 4 байта (IEE 754) за измисления процесор PROG1

Съдържание

Типове данни

- Програмните езици от високо ниво (C, C++ и др.) целят да улеснят писането на машинни програми
- В езика C всички данни имат *тип*
- Типът определя какъв вид стойности и самите стойности, които могат да се съдържат в данните от типа. Също определя и какви операции могат да се извършват с тях.
- Знаейки какви данни, от какъв тип са и какви операции се извършват върху тях от програмата, компилаторът може да подбере правилните инструкции при генерирането на машинен код за определен процесор
  - След като са известни инструкциите, компилатора може да запише данните в правилния формат за тях

Типове данни

- Езикът C се опитва да поддържа възможно най-много видове процесори, което налага някои аспекти на езика да са специфицирани по-общо
- Например в повечето случаи броят байтове, които заема единица данни от определен тип, не е посочен
- От една страна това позволява една и съща програма да работи непроменена върху различни процесори
- От друга страна прави писането на коректни програми по-трудно

Основни типове в C

Тип данни	Описание
`signed char`	Цяло число със знак, което се събира в 1 байт памет. Може да съдържа поне числата [-127, +127]
`unsigned char`	Цяло число без знак, което се събира в 1 байт памет. Може да съдържа поне числата [0, 255]
`signed short int`	Цяло число със знак. Може да съдържа поне числата [-32767, +32767]
`unsigned short int`	Цяло число без знак. Може да съдържа поне числата [0, 65535]
`signed int`	Цяло число със знак. Може да съдържа поне числата [-32767, +32767]
`unsigned int`	Цяло число без знак. Може да съдържа поне числата [0, 65535]

Таблица 1: Основни целочислени типове данни в C (1/2)

Основни типове в C

Тип данни	Описание
`signed long int`	Цяло число със знак. Може да съдържа поне числата: [-2147483647, +2147483647]
`unsigned long int`	Цяло число без знак. Може да съдържа поне числата: [0, 4294967295]
`signed long long int`	Цяло число със знак. Може да съдържа поне числата: [-9223372036854775807, +9223372036854775807]
`unsigned long long int`	Цяло число без знак. Може да съдържа поне числата: [0, 18446744073709551615]

Таблица 2: Основни целочислени типове данни в C (2/2)

Основни типове в C

Тип данни	Описание
`float`	Реално число. Съдържа числа с малка точност.
`double`	Реално число. Съдържа числа с по-голяма точност.
`long double`	Реално число. Съдържа числа с възможно най-голяма точност.

Таблица 3: Реални типове данни в C

Съдържание

Подпрограми

- При писане на програми често се налага използването на един и същ алгоритъм повече от веднъж (напр. смятане на корен квадратен, повдигане на степен и т.н.)
- Това води до дублиране на кода всеки път, когато използваме алгоритъма
- *Подпрограмите* са програми, които могат да бъдат стартирани от програма за да решат някаква задача върху определени данни наречени *параметри*.
- Най-често подпрограмите изчисляват *резултат*, който може да бъде използван от програмата, която е стартирала подпрограмата
- Подпрограмите помагат да се избегне дублирането на код, което значително улеснява писането на програми

Подпрограми

```
line1:  jump :main

cube:
line2:  loadb  [$0x1000], %r1
line3:  loadb  [$0x1000], %r2
line4:  mulb   %r1, %r2, %r3
line5:  mulb   %r3, %r1, %r2
line6:  storeb %r2, [$0x2000]
line7:  loadl  [$0x3000], %rl1
line8:  jump   %rl1

main:
line9:  storeb $0b00000100, [$0x1000]
line10: storel :line12, [$0x3000]
line11: jump   :cube
line12: loadb  [$0x2000], %r1
line13: outb   %r1
line14: storeb $0b00000101, [$0x1000]
line15: storel :line17, [$0x3000]
line16: jump   :cube
line17: loadb  [$0x2000], %r1
line18: outb   %r1

line19: storeb $0b00000110, [$0x1000]
line20: storel :line22, [$0x3000]
line21: jump   :cube
line22: loadb  [$0x2000], %r1
line23: outb   %r1
```

Фрагмент 3: Програма на асемблер за смятане на $4^3, 5^3, 6^3$, използваща подпрограма, за измисления процесор PROG1

Съдържание

Поддръжка на подпрограми в C. Функции

- Езикът C позволява писането и стартирането на подпрограми в рамките на програма.
- Особеност на езика е, че не позволява промяната на *главната* програмата, която винаги е една и съща.
- Всяка програма написана на C винаги стартира една от подпрограмите, която е отбелязана като *главна подпрограма*, след което приключва своята работа
  - Това не е ограничение, тъй като програмистът има свободата да променя главната подпрограма, както реши
- Подпрограмите в C е прието да се наричат *функции*.
  - Името е взаимствано от математиката, тъй като математическите функции имат *аргументи* (*параметри*) и след пресмятане имат *резултат*

Функции

- При езиците от ниско ниво подпрограмите се различават по адреса на първата си инструкция, а данните по своя адрес в паметта
- Тъй като помненето на адреси е по-трудно от помненето на думи, в C се използват *идентификатори* (имена) за отделните функции и отделните данни
- Съответно всяка функция написана от програмиста има както свое име, така и имена за отделните данни върху които работи (наречени още *аргументи на функцията*)

Функции

- За да бъде написана функция трябва да се окажат:
  - нейното име
  - имената на отделните данни, които функцията обработва, заедно с техните типове
  - типа на данните, които съдържат резултата след изпълнение на функцията
- Самият код на функцията се описва в нейното *тяло*
- Една функция се отбелязва за *главна функция* (подпрограма), като се избере нейното име да е *main*

```
float sum(float a, float b) {
}
```

Фрагмент 4: Пример за функция в C с празно тяло