Guião 3

Programação em C e Assembly

João Paulo Sousa
jpsousa@fe.up.pt

3.1 Introdução e objectivos **1**
3.2 Programação em C *versus* programação em *assembly* **1**
3.2.1 Vantagens e desvantagens do *assembly* 2
3.2.2 Vantagens e desvantagens do C 2
3.2.3 Conclusão 2
3.3 Problemas propostos **3**

3.1 Introdução e objectivos

O desenvolvimento de programas para sistemas embutidos é uma area que apresenta problemas específicos normalmente relacionados com as limitadas capacidades do *hardware* de suporte. Para optimizar o aproveitamento das potencialidades do *hardware* e maximizar a facilidade e rapidez de desenvolvimento do *software* é necessário um conhecimento aprofundado do modelo de programação e uma escolha adequada da linguagem de programação a utilizar. Este trabalho tem como objectivos a consolidação de conhecimentos sobre o modelo de programação da família 51 quer do ponto de vista da programação em linguagem *assembly* quer do ponto de vista da programação em linguagem C.

A metodologia a usar será a análise de segmentos de código representativos seguida de resolução de problemas. Como ferramentas de desenvolvimento recomendam-se am aplicações *u-Vision* e *d-Scope* da KEIL Software, disponíveis em versão de demonstração na rede do DEEC.

Ainda que de nível introdutório, assume-se a existência de conhecimentos prévios sobre a linguagem assembly desta familia de microcontroladores. Como elemento de estudo recomenda-se [**?**, capítulos 4 e 8] onde estes assuntos estão apresentados em detalhe.

3.2 Programação em C *versus* programação em *assembly*

Embora se possam considerar outras linguagens[1], a maioria das aplicações para a família 51 são desenvolvidas em C e/ou *assembly* pelo que é essencial saber escolher a linguagem a usar.

[1] BASIC, Forth, PL-M.

3.2.1 Vantagens e desvantagens do *assembly*

A programação em *assembly* é o nível mais baixo em que se pode programar com alguma comodidade. As desvantagem de uma linguagem de baixo nível são:

- Grande desproporção entre o conjunto de instruções e as tarefas que o microprocessador normalmente é chamado a executar. Esta desproporção obriga a decompor manualmente cada tarefa em operações elementares que, além de ser um processo demorado e sujeito a erros, não ajuda a manter o código estruturado.

- Grande dependência relativamente aos detalhes internos do *hardware* que tem como consequência directa a reduzida portabilidade dos programas.

A principal (e actualmente única) vantagem de programar em *assembly* é poder aproveitar ao máximo as características particulares do *hardware* onde vai ser executado, conseguindo assim resultados optimizados quer em tempo de execução quer em tamanho de código.

Outra vantagem é a existência de assembladores gratuitos ou de baixo custo para a totalidade dos microprocessadores pelo que é sempre possível programar em *assembly*, qualquer que seja o microprocessador escolhido. O mesmo já não acontece com linguagens de alto nível, onde nem sempre é possível encontrar um compilador para um dado microprocessador[2].

3.2.2 Vantagens e desvantagens do C

As razões para a popularidade da linguagem C em aplicações embutidas são bem conhecidas: o acesso fácil ao *hardware*, os baixos requisitos de memória e o elevado desempenho dos actuais compiladores facilitam o *rápido* desenvolvimento de aplicações *eficientes* e de *fácil manutenção*.

Igualmente bem conhecidas são as desvantagens da linguagem: sintaxe demasiado livre e propensa a erros difíceis de detectar e demasiados pontos onde a norma ISO não define o comportamento da linguagem ou remete para a implementação de cada compilador.

3.2.3 Conclusão

A tendência actual é a favor de uma programação mista, usando principalmente linguagens de mais alto nível (C em particular) e recorrendo à linguagem *assembly* apenas em rotinas onde a eficiência do código seja o objectivo principal. Esta tendência explica-se por três motivos:

- existem actualmente bons compiladores para a maioria dos microprocessadores, alguns até de domínio público,

- os avanços na microelectrónica permitem que a rapidez de execução se consiga facilmente por aumento da frequência de funcionamento

- a pressão do mercado obriga a encurtar o tempo de desenvolvimento e a aumentar a facilidade de manutenção do código.

[2] Ou pelo menos um compilador de baixo custo...

3.3 Problemas propostos

A única maneira de aprender a programar é escrever programas! Nos problemas seguintes, a não ser que seja feita referência a endereços específicos, deverão ser utilizados segmentos de dados recolocáveis.

1. Escreva um programa em *assembly* que calcule $\boldsymbol{x = x + y}$ supondo que:
   (a) x e y são variáveis de 8 bits em memória de dados interna (MDI).
   (b) x e y são variáveis de 16 bits em MDI.
   (c) x e y são variáveis de 8 bits em memória de dados externa (MDE).
   (d) x e y são variáveis de 16 bits em MDE.
2. Repita o problema anterior mas agora em C. Configure o compilador para mostrar o código gerado e compare com o código que desenvolveu anteriormente. Comente.
3. Escreva um programa que copie os 4 bits mais significativos de $\boldsymbol{x}$ para os 4 bits menos significativos de $\boldsymbol{y}$ e coloque a zero os 4 bits mais significativos de $\boldsymbol{y}$. Considere:
   (a) variáveis de 8 bits em MDI.
   (b) variáveis de 8 bits em MDE.
4. Repita o problema anterior mas agora em C, comparando o código gerado pelo compilador com o código desenvolvido anteriormente.
5. Considere que ***temp[i]*** contém as medições de temperatura efectuadas em 16 pontos diferentes de uma estufa. Escreva um programa em *assembly* e outro em C que calculem a temperatura média no interior da estufa. Suponha que está a trabalhar com valores de temperatura entre 0 e $200°C$ e que o resultado é guardado em ***tempmed***.
6. Suponha um vector v com 50 elementos de um *byte* cada em MDE representados em complemento para 2. Escreva um programa em *assembly* e outro em C que contem quantos elementos negativos, nulos e positivos existem no vector. As contagens devem ser guardadas nas variáveis nn, nz e np respectivamente. Declare estas variáveis num segmento de dados em MDI.
7. Repita em C.
8. Suponha já existente a rotina *Upcase* que converte para maiúscula a letra minúscula cujo código ASCII está no acumulador, devolvendo o resultado da conversão também no acumulador.
   (a) Escreva um programa que, recorrendo a essa rotina, converta para maiúsculas um texto em MDE com início apontado por DPTR e cujo fim é indicado pelo código 0.
   (b) Escreva a rotina *Upcase*.
9. Repita em C.
10. Suponha já existente a rotina *CountBits* que conta o número de bits a um da posição de MDI apontada por R0, devolvendo o resultado no acumulador.
    (a) Escreva um programa que, recorrendo a essa rotina, conte o número de bits a um de uma zona de memória com início apontado pelo registo R0 e comprimento indicado no registo R7. O resultado deve ficar em R7.
    (b) Escreva a rotina *CountBits*.
11. Repita em C.