EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 4 – Representação e Armazenamento de Dados

Profa.Wu Shin-Ting

OBJETIVO: Armazenamento e manipulação de dados na memória

ASSUNTOS: Tipos de dados e operações entre dados, conversão entre diferentes tipos de dados, alocação estática e dinâmica de dados na memória, operações sobre endereços dos dados.

O que você deve ser capaz ao final deste experimento?

Usar *casting* em linguagem C.

Usar os tipos de dados em linguagem C.

Saber operar endereços em linguagem C.

Saber alocar estatica e dinamicamente os dados em linguagem C.

Gerar um arquivo de funções referentes à manipulação das botoeiras do *shield* EA871.

INTRODUÇÃO

A unidade de memória num sistema digital é *byte*, de forma que cada endereço de memória, ou ponteiro, aponte sempre para um determinado *byte*. Entretanto, podemos definir numa linguagem de programação as variáveis de tamanhos maiores que um *byte*. Diferentemente dos registradores dos módulos do nosso microcontrolador, que vimos nos experimentos anteriores, os locais, onde as variáveis que precisamos num programa devem ser armazenadas, não são previamente mapeados no espaço da memória. É necessário alocá-los. Em linguagem C esta alocação pode ser feita **estaticamente** com a declaração das variáveis, indicando o seu tipo de dados e os qualificadores do tipo de dados, ou **dinamicamente** durante a execução do programa.

Neste experimento vamos exercitar, através do programa em [2], formas de representação de dados e alocação de memória para os dados utilizados num programa em C do microcontrolador KL25Z e apresentar a programação de uma interface com dois periféricos de entrada: os três *push-buttons* PTA4, PTA5 e PTA12 disponíveis na placa auxiliar (*shield* EA871). Implementaremos neste experimento um **temporizador** com um sinalizador luminoso cuja cor é configurável pelo usuário através da botoeira PTA5 e cujo intervalo de [2]tempo em que o sinalizador ficará aceso é configurável pelo usuário através da botoeira PTA12. A botoeira PTA4 é utilizado para iniciar a contagem de um intervalo de tempo.

EXPERIMENTO

1. *Vamos entender como a linguagem C armazena os dados na memória?* **Leia** atentamente o documento [1] para se ter uma ideia como os dados são representados na linguagem C

2. *Vamos ver como os conceitos são traduzidos na prática através de um* programa [2]? Estendemos a funcionalidade do hello_world.zip permitindo que um usuário mude a cor do *led* RGB através da botoeira PTA5. Note que a função de atraso usada neste programa é

`delay10us` que gera um atraso em múltiplos *t* de *10us*. Ao pressionar a botoeira PTA5 a cor do *led* transitaria na sequência vermelha → verde → azul → branca em cada 0.25s. Estas cores correspondem à sequência de intervalos de tempo que o temporizador "contaria": 1.5s → 2.5s → 3.5s → 4.5s. Ao soltar a botoeira o estado atual do intervalo de tempo é armazenado. Para isso pré-definimos em arranjos/vetores 4 conjuntos de estados dos *leds:* 4 cores e 4 intervalos distintos. Dois vetores foram alocados dinamicamente e um estaticamente. **O vetor alocado estaticamente tem os seus elementos definidos na inicialização**, enquanto **os vetores alocados dinamicamente tem os seus elementos inicializados no tempo de execução**, antes de entrar no laço de controle do temporizador. Estes vetores são, *de* fato, *lookup tables* que mapeiam o índice entrado pelo usuário em cor/intervalo correspondente no vetor pré-definido. Como um usuário entraria diferentes índices (numéricos) de cor/intervalo só com as botoeiras disponíveis no *shield* EA871?

No programa, com `delay10us`, são utilizados como índices dos vetores os múltiplos de 0.25s (*t=0.25s/0.000010s*) em que um usuário pressione uma botoeira. As três botoeiras do *shield* EA871 estão conectadas nos pinos 4, 5 e 12 da porta A [3] e o sinal adquirido por elas é digital, 0V (botoeira pressionada) e 3V3 (botoeira aberta). No laço principal da rotina `main` são verificados os estados das duas botoeiras, PTA5 e PTA12, com a rotina `le_pta`. Quando o sinal de entrada da botoeira PTA5 for 0, inicia-se a contagem de tempo com uso de `delay10us` até que o estado da botoeira volte ao nível 1 na rotina `tempo_espera`. O tempo transcorrido é contabilizado ciclicamente em múltiplos de 0.25s. A realimentação visual do tempo transcorrido se dá pela mudança da cor do *led* RGB. A cada cor corresponde um intervalo de tempo programado do temporizador. Quando o sinal de entrada da botoeira PTA12 for 0, é iniciada a função de temporização: o *led* RGB acende na cor correspondente ao intervalo de tempo "programado" e inicia-se a contagem regressiva até que transcorra o tempo programado.

Em relação à codificação do fluxo de controle no programa [2], chamo atenção ao uso de diferentes tipos de dados, `uint8_t` (1 *byte*), `short` (2 *bytes*) e `int` (4 *bytes*), e ao uso de renomeação de algumas estruturas de dados e variáveis via `typedef`. O primeiro tem como objetivo otimizar o uso do espaço de memória, alocando somente quantidade de *bytes* estritamente necessários (Dica: Consulte a aba *Variable* na perspectiva *Debug* do *IDE CodeWarrior*), e o segundo, como já vimos no roteiro 1 [5], permite tornar o nosso código mais inteligível. A mistura de tipos num programa requer conversão de tipos ao movermos os dados de um tipo a uma posição de memória reservada para outro tipo. Conversões explícitas são preferíveis para evitar conversões implícitas inadequadas.

Vale também chamar atenção nas operações em ponto flutuante sobre a variável `multiplo` no seguinte trecho de código do programa [2]

```
float multiplo = *tmp/1.69;

multiplo *= 1.3;

multiplo *= 1.3;

multiplo /= 0.5;
```

A sequência não seria equivalente à última linha de comando "`multiplo /= 0.5;`"? Compare o comportamento do *led* azul nas duas versões do programa, uma com as 4 instruções e outra somente com a última linha de instrução, para você ver que, na prática, os resultados são diferentes. Isso decorre do fato de que nos sistemas digitais muitos valores reais não podem ser representados de forma exata [1][6]. As diferenças entre os valores representados e os valores reais são conhecidas como erros de arredondamento e estes erros de arredondamento podem se acumular com uma sequência de operações sucessivas, como na sequência acima, e distorcer o resultado final. Portanto, recomenda-se tomar decisões em cima de valores inteiros e reduzir a quantidade de operações em ponto flutuante, quando possível.

Observe ainda que, no lugar das instruções das redefinições (`define`) dos endereços dos registradores, foi incluso o arquivo-cabeçalho *derivative.h* no qual é definida uma série de macros úteis para programar os registradores do nosso microcontrolador. Vale também notar diferentes formas para acessar um elemento de um vetor, pode ser (a) por índice ou (b) pelo endereço do elemento que corresponde à soma do endereço inicial do vetor com o índice do elemento (Dica: Consulte a aba *Variable* na perspectiva *Debug* do *IDE CodeWarrior*). Noutro detalhe que precisamos atentar é a liberação do espaço de memória alocado dinamicamente após seu uso com a função `free`. Finalmente, chamo atenção a dois diferentes formatos de disposição de dados com tamanho maior que um *byte* na memória: *little endian* (os *bytes* são armazenados na ordem do *byte* menos para o mais significativo em endereços crescentes de memória) e *big endian* (os *bytes* são guardados na ordem do *byte* mais para o menos significativo em endereços crescentes de memória). O formato adotado no nosso micro-controlador é *little endian* (Dica: Veja na aba *Memory* o conteúdo do vetor de frequências).

3. *Vamos praticar o que aprendemos?* Crie um projeto novo e insira os arquivos do programa [2] nas pastas apropriadas. Substitui a função `delay10us` pela função `delay20us` que você implementou no experimento 3 [4]. Esta nova função gera um atraso em múltiplos de *20us*. Para que o projeto execute corretamente, é necessário
   a) configurar os registradores de controle dos pinos que servem as três botoeiras, PTA4, PTA5 e PTA12 no arquivo `pushbutton.c` com uso das macros pré-definidos em `derivative.h`: habilitar o *clock* do módulo PTA, setar a função dos pinos para GPIO, configurar tipo de sinal digital que o pino servirá através da substituição de xxxx. Veja em [1] que o pino que serve a PTA4 tem um tratamento diferenciado, pois o seu valor de *reset* é NMI (0b111).
   b) substituir as macros do arquivo `ledRGB.c` pelas macros pré-definidas em `derivative.h`, pois foram removidas as definições das macros SIM_* e PORTx_* do arquivo `ledRGB.h`.

4. *Vamos ver se você entendeu?* Vamos aprimorar o projeto do temporizador [2] de forma que a variedade de cores sinalizadoras, e em conseqüência dos intervalos de tempo, seja aumentada para vermelha (1.0s) → verde (1.5s) → azul (2.0s) → magenta (2.5s) → amarela (3.0s) → ciana (3.5s) → branca (4.0s). Vamos ainda alterar a forma de programação do temporizador. A cada *click* da botoeira PTA5 é alterada a cor dos *leds* ciclicamente e o valor de tempo correspondente é programado no temporizador. Você tem ideia como mudar a configuração pelo

`tempo_espera` para contagem cíclica de quantidade de vezes que a botoeira PTA5 foi pressionada? Caso você não conseguiu uma solução, dê uma olhada no procedimento abaixo em que estabeleci xxx iterações como tempo de espera máximo pelo subsequente *click*:

1. quantidade_clicks ← 0;
2. Se PTA5 estiver pressionada então
    1. incrementa quantidade_clicks;
    2. tempo_espera ← 0;
    3. Se o estado do PTA5 alternou então volta para 2.1;
    4. senão incrementa tempo_espera;
    5. Se tempo_espera < xxx então volta para 2.3;
    6. senão retorna quantidade_clicks;

Finalmente, muitos pontos podem ser melhorados no código. Melhore o código da rotina *main* de forma que

a) os espaços de memória alocados para as variáveis sejam os mínimos necessários, evitando redundância de dados;
b) seja reduzida a quantidade de instruções necessárias para o cálculo dos endereços dos dados nos vetores;
c) se faça uso das macros disponíveis na *derivative.h* para configurar os registradores do microcontrolador; e
d) comparações entre valores em ponto flutuante sejam substituídas pelas comparações entre valores inteiros.

RELATÓRIO

O relatório deste experimento é o seu projeto com os códigos-fonte devidamente documentados com a sintaxe *Doxygen*. Limpe as pastas do seu projeto (*Project* > *Clean* …), exporte-o para um arquivo de extensão **zip** e suba o arquivo no sistema Moodle.

REFERÊNCIAS

Todas as referências podem ser encontradas nos *links* abaixo ou na página do curso.

[1] D. S. Oliveira e Wu Shin-Ting. Linguagem C: Dados
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/RepresentacaoDados.pdf
[2] ledRGBPB.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/ledRGPB.zip
[3] Wu Shin-Ting e A.A.F. Quevedo. Ambiente de Desenvolvimento – *Hardware*
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/AmbienteDesenvolvimentoHardware.pdf
[4] Wu, Shin-Ting. EA871 - Roteiro 3 – 1s2018
http://www.dca.fee.unicamp.br/cursos/EA871/1s2018/T/roteiros/exp3.pdf
[5] Wu, Shin-Ting. EA871 - Roteiro 1 – 1s2018
http://www.dca.fee.unicamp.br/cursos/EA871/1s2018/T/roteiros/exp1.pdf
[6] Tools & Thoughts: IEEE-754 Floating Point Converter

https://www.h-schmidt.net/FloatConverter/IEEE754.html
[7] *KL25 Sub-Family Reference Manual – Freescale Semiconductors* (doc. Number KL25P80M48SF0RM), Setembro 2012).
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf
[8] Wu, Shin-Ting e D. S. Oliveira. Linguagem C: Operações sobre os Dados.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/Operadores.pdf

Agosto de 2016
Revisado em Fevereiro de 2017
Revisado em Julho de 2017
Revisado em Fevereiro de 2018