# EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

## EXPERIMENTO 5 – 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 variáveis de tamanhos maiores que um *byte*. Diferentemente dos registradores dos módulos do nosso MCU, que vimos nos experimentos anteriores, os locais, onde as variáveis que precisamos num programa devem ser armazenados, 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 o tempo de execução.

Neste experimento vamos exercitar formas de representação de dados e alocação de memória para os dados utilizados num programa em C do microcontrolador KL25Z através do programa em [2], e apresentar a programação da interface com dois periféricos. O nosso programa permite que dois *push-buttons* PTA5 e PTA12 disponíveis na placa auxiliar (*shield* EA871) controlem, respectivamente, a cor e a frequência das piscada dos *leds* do *kit* FRDM-KL25Z.

## EXPERIMENTO

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

2. *Vamos ver como os conceitos são traduzidos na prática através de um* programa [2]? Como já comentado anteriormente, estendemos a funcionalidade do hello_world.zip permitindo que um usuário mude a cor e a frequência das piscadas do *led* RGB. Para isso pré-definimos em arranjos/vetores 6 conjuntos de estados dos *leds:* 6 cores e 6 frequências distintas. 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 das piscadas. Estes vetores são, de fato, *lookup tables* que mapeiam o índice entrado pelo usuário em

cor/frequência correspondente. Como um usuário entraria diferentes índices (numéricos) de cor/frequência só com as botoeiras disponíveis no *shield* EA871?

No programa são utilizados como índices dos vetores os múltiplos de 0.25s em que um usuário mantiver uma botoeira pressionada. 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 numa botoeira for 0, inicia-se a contagem de tempo com uso de `delay10us` implementado no experimento 4 até que o estado da botoeira volte ao nível 1 na rotina `tempo_espera`. O tempo transcorrido é mapeado em múltiplos de 0.25s. Outro detalhe é a conversão do valor de frequência em intervalo de tempo (meio período) necessário para que os *leds* mantenham um mesmo estado (aceso ou apagado) com uso da rotina `delayfreq`.

Em relação à codificação do fluxo de controle elaborado no programa, 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, 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 utilizadas no programa para assegurar conversões implícitas inadequadas.

Observe que, no lugar das instruções das redefinições dos endereços dos registradores, incluimos 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 de memória) e *big endian* (os *bytes* são guardados na ordem do *byte* mais para o menos significativo em endereços 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. Complete os pontos marcados com xxxx nos arquivos, insira as instruções nas rotinas vazias, altere 0.25us para 0.4us como base de tempo para computar índices dos elementos nos vetories de cores e de frequências, e substitua o tipo da variável `tmp` na rotina `main` de (`float *`) para `float` fazendo devidas modificações no código.

4. *Vamos ver se você entendeu?* Há muitos pontos que 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) ao invés de alternar os estados dos *leds*, programe os *leds* para que fiquem piscando (liga e desliga em cores diferentes); e
   d) Adicione mais uma alternativa de cor (verde) e uma de frequência (15.5Hz) .

## RELATÓRIO

O relatório deste exeperimento é o código em C do item 4 devidamente documentado. 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/EA871/1s2017/ST/codes/ledRGBPB.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

Agosto de 2016
Revisado em Fevereiro de 2017