EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 3 – Linguagem de Montagem (*Assembly*) - I

Profa. Wu Shin-Ting

OBJETIVO: Introdução à linguagem de montagem (*assembly*) do MKL25Z128 **(ARM)**

ASSUNTO: Linguagem de montagem

O que você deve ser capaz ao final deste experimento?

Ter uma noção do repertório de instruções ARM e as diretivas do seu *assembly.*

Saber como as instruções e os dados de um programa são organizados na memória.

Saber os passos de inicialização do MCU ao ser rebootado.

INTRODUÇÃO

Normalmente, várias etapas do projeto são automatizadas pelo IDE. Como vimos no experimento anterior, é bastante simples criar um projeto. O ambiente CodeWarrior gera para nós um esqueleto de programa main.c no qual podemos adicionar instruções para controlar os módulos agregados ao núcleo Cortex M0+ do nosso MCU da família Kinetis L. Esses módulos são também conhecidos como módulos de periférico da família Kinetis L. Entre os diversos módulos disponíveis no MCU, destacamos o módulo de integração do sistema SIM, o módulo de interface paralela de uso genérico GPIO, o módulo de interface de comunicação serial assíncrona UART, o módulo de interface de comunicação síncrona IIC, o módulo de interrupções periódicas PIT, o módulo de relógio em tempo real RTC, e o módulo de temporizador/modulação por pulso TPM.

Para que o MCU controle todos os periféricos de forma apropriada, várias coisas acontecem “nos bastidores”. Todo programa carregado nele precisa executar uma série de inicializações para, por exemplo, que o *program counter* (PC) tenha o endereço da primeira instrução do nosso programa. Mostramos no experimento 2 que este programa pode ser escrito em linguagem de alto nível C. Embora seja uma linguagem mais próxima da nossa linguagem de comunicação, ela está bem distante da forma como os processadores executam as operaçõs. Precisamos compilá-la e fazer *linking*. Pois, em última instância, os processadores operam com padrões binários, “0” e “1”, implementados usualmente como a ausência (p.ex., tensão nula) e a presença (p. ex. tensão 3.3V ou 5V) de sinais elétricos.

Neste experimento vamos introduzir o modelo de programação do núcleo Cortex-M0+ utilizando os mnemônicos dos seus códigos de máquina, ou seja a sua linguagem de montagem (*assembly*) [1]. Os programas em mnemônicos são traduzidos para os códigos binários da máquina pelo montador (*assembler*) GNU no ambiente IDE CodeWarrion [2]. E o processo de conversão de mnemônicos para códigos binários é muito mais simples do que uma compilação e *linking*. Esse processo de conversão é conhecido como montagem (*assembling*).

Sendo um processador RISC, o núcleo Cortex-M0+ apresenta um repertório de instruções bem menor e bem mais eficiente que o de um processador CISC. Por outro lado, ele requer uma quantidade maior de instruções para executar um mesmo programa, demandando um espaço maior de memória. Como memória é um quesito crítico para os MCUs, foi proposto o repertório de instruções *Thumb* de 16 *bits* como uma alternativa para as instruções de 32 *bits* da arquitetura ARM.

Porém, algumas instruções da arquitetura ARM não podem ser codificadas com 16 *bits*. Foram então adicionadas ao repertório *Thumb* algumas instruções de 32 *bits*. O núcleo Cortex M0+ suporta o repertório de instruções Thumb e algumas instruções codificadas em 32 *bits* [1].

EXPERIMENTO

1. *Vamos entender o que se trata de uma linguagem de montagem e como é esta linguagem para o processador integrado no microcontrolador MKL25Z?* **Leia** atentamente o documento [3] para ter uma noção da linguagem de montagem.

2. *Vamos ver como os conceitos são traduzidos na prática através de um programa em assembly em [4]*? Este programa é equivalente ao programa em C, apostila.c, que vocês trabalharam nos experimentos 1 e 2.

    a) Crie um novo projeto configurado para a linguagem de programação ASM e substitua o código do arquivo *main.s* pelo código do arquivo asm.s. Rode o programa no FRDM-KL25Z.
    b) Adicione em cada linha de instrução a função da instrução como comentário.
    c) Veja que cada instrução em C é associado a uma sequência de instruções de montagem no arquivo. Faça a associação entre os dois conjuntos.
    d) Observe que foi utilizada a diretiva *.word* para alocar um espaço na memória para armazenar os endereços dos registradores. Podemos substituir a diretiva *.word* por *.equ*? Jusrtifique.
    e) Observe que a constante 500000 no programa apostila.c é armazenada na memória no programa asm.s através da diretiva *.word*. Por quê não utilizamos o modo de endereçamento imediato para carregar este valor no registrador r0, como "movs r1,#500000" no lugar de uma instrução mais lenta "ldr r0,COUNT" com COUNT representando o endereço onde está armazenado o valor 500000?
    f) Qual é a função da diretiva "*.align 4*" antes da alocação de memória para diferentes rótulos/símbolos?
    g) Veja no Capítulo 6 em [5] número de ciclos de relógio utilizados para cada instrução e esitme o tempo gasto para executar a rotina *delay*. É possível reduzir a quantidade de instruções nesta subrotina. Como?
    h) Compare o código de montagem gerado pelo compilador a partir do programa apostila.c com o código asm.s em temos de ocupação de memória e de tempo de execução.

3. *Vamos praticar um pouco o que você aprendeu?* Adicione instruções em *assembly* no programa asm.s que faça o *led* verde piscar juntamente com o *led* vermelho.

4. *Vamos ver se você entendeu?* Reescreva o seu programa apostila.c de piscadas em cor ciano na linguagem de montagem. Cada linha de instrução deve ser comentada com a função da instrução.

RELATÓRIO

O relatório deve conter as respostas das questões (c-g) do item 2 e o programa *main.s* de pisca-pisca em cor ciano (*led* verde + *led* azul) na linguagem de montagem (*assembly*). Suba o arquivo no sistema Moodle.

REFERÊNCIAS

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

[1] ARM. ARMv6-M Architecture Reference Manual
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/ARMv6-M.pdf
[2] The GNU Assembler
http://tigcc.ticalc.org/doc/gnuasm.html
[3] Wu Shin-Ting. Linguagem de Montagem
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/LinguagemMontagem.pdf
[4] asm.s
http://www.dca.fee.unicamp.br/cursos/EA871/1s2017/ST/codes/asm.s
[5] ARM7TDMI – Technical Reference Manual
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/ARM7TDMITechnicalManual.pdf

Agosto de 2016
Revisado em Fevereiro de 2017