EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 3 – Linguagem de Montagem (*Assembly*)

Profa. Wu Shin-Ting

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

ASSUNTO: Combinação de linguagem de montagem e linguagem C.

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.

Saber integrar os códigos em linguagem de montagem nos códigos em linguagem C.

INTRODUÇÃO

Normalmente, várias etapas do projeto são automatizadas pelo *IDE*. Como vimos nos experimentos anteriores, 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 microcontrolador 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”. Qualquer 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 operações. Precisamos compilá-la, ou traduzi-la para linguagem de montagem, e ligá-la com as funções que estejam definidas em outros arquivos. 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*) (Seção A6.7 em [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 linkagem (ligação). O processo de conversão dos mnemônicos 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 microcontroladores, 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 asm.s [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) Veja em [5] que cada linha de comando em C é associada a uma sequência de instruções de montagem no arquivo. Faça a associação entre os dois conjuntos de códigos, apostila.c e asm.s. Dica: Compile apostila.c e veja as instruções de montagem na aba *Disassembly* da perspectiva *Debug* do *IDE CodeWarrior*.
   c) 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.
   d) 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? Dica: A maioria das instruções *Thumb* ocupam 16 *bits*.
   e) Qual é a função da diretiva “*.align 4*” antes da alocação de memória para diferentes rótulos/símbolos?
   f) Veja no Capítulo 6 em [6] o número de ciclos de relógio gastos para cada instrução. Estime o tempo gasto para executar a rotina *delay* no programa apostila.c/asm.s, sabendo que a frequência de operação do núcleo do nosso microcontrolador está configurado para 20.971520 MHz. Compare o tempo estimado com o tempo medido com uso de um osciloscópio. Dica: O acesso ao sinal do pino PTB18, que alimenta o *led* vermelho, não é fácil. No Anexo A, a ponta de prova do osciloscópio foi colocado no pino PTE23 da placa auxiliar [8]. Para que o sinal PTB18 seja “replicado” no pino PTB23, basta programar (em C) o pino PTE23 de forma similar ao pino PTB18. Vamos adicionar a configuração do pino PTE23 no programa apostila.c.
   g) Determine o espaço de memória em *bytes* que a rotina *delay* no programa apostila.c/asm.s ocupa. É possível reduzir este espaço de ocupação mantendo a mesma funcionalidade? Justifique.
   h) Para você determinar com maior precisão o tempo em que são executadas as instruções (de espera) dentro a rotina *delay* no programa apostila.c, podemos substituir a instrução de laço

em C pelas instruções em *assembly*. Como seria a interface entre os códigos em C e os códigos em *assembly*?

i) Qual é a quantidade máxima de argumentos de uma rotina que são passados pelos registradores numa chamada desta rotina no nosso ambiente de desenvolvimento *CodeWarrior*? Qual é a vantagem de codificarmos uma rotina com uma lista pequena (até 4) de argumentos?

.

3. ***Vamos colocar em prática o nosso aprendizado?*** Substitua o código da rotina *delay* no programa apostila.c pela rotina void *delay10us(unsigned int t)*, de forma que a execução da rotina *delay10us* seja em t*0.00001s.

a) Elabore um programa em *assembly* equivalente ao laço de espera da função *delay* em C, com o número máximo de iterações configurável [5]. Dica: A instrução NOP não faz nada, só gasta tempo correspondente a um ciclo de relógio do núcleo. (veja Seção A6.7.47 em [1]).
b) Como se passa o valor do argumento *t* de uma chamada em C para as instruções em *assembly* dentro da função *delay10us(unsigned intt t)*?
c) Verifique a acurácia do seu código para os seguintes valores de t com uso de osciloscópio conforme a imagem no Anexo A: 1, 10, 100, 1000, 10000 e 50000.
d) Substitua a rotina *delay* do seu programa de pisca-pisca de cor branca do segundo experimento [7] pela nova função *delay10us* e modifique o seu projeto para que os *leds* pisquem na frequência de 2Hz. Documente o seu código com a sintaxe de Doxygen,

RELATÓRIO

O relatório deve ser devidamente identificado e conter as respostas das questões (c-i) do item 2 e o resultado de testes de acurácia da questão c do item 3. Suba o relatório em pdf e o seu projeto com a nova função *delay10us(unsigned int t)* da questão d do item 3 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] 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
[6] *ARM7TDMI – Technical Reference Manual*
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/ARM7TDMITechnicalManual.pdf
[7] Wu, Shin-Ting. EA871 – Roteiro 2 – 2s2017
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/roteiros/exp2.pdf
[8] 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

ANEXO A: PONTOS DE ACESSO A PTE23 e TERRA

Fonte: Roteiro do Experimento 4 do Segundo Semestre de 2015 (http://faraj7.github.io/).

Baseado nos Roteiros dos Experimentos 3 e 4 do Segundo Semestre de 2015 (Agosto de 2016)
Revisado em Fevereiro de 2017
Revisado em Julho de 2017