8.2.1 Compilar código assembly e código C e linkar ambos.

Foi adaptado o seguinte código (add.s e main.c):

add.s

```
.text

.globl myadd

myadd: @ Function "myadd" entry point.

add r0, r0, r1 @ Function arguments are in R0 and R1. Add together and put the result in R0.

mov pc, lr @ Return by branching to the address in the link register.
```

main.c

```
#include <stdio.h>

extern int myadd(int a, int b);

int main()

{

int a = 4;

int b = 5;

printf("Adding %d and %d results in %d\n", a, b, myadd(a, b));

return (0);

}
```

Observe que o código em C chama myadd que está em assembly. No assembly, a função myadd recebe os parâmetros em R0 e R1 e retorna o resultado em R0. Existe uma convenção de como o C passa os parâmetros para uma função em assembly, e de como recebe o retorno dessa função. Veja "Chapter 2. Consulte o capítulo 2 do doc DUI0056.pdf.

Para compilar:

```
student:~/src$ gcc add.s main.c
```

que gera o codigo a.out

Fazendo gdb a.out :

Loading section .jcr, size 0x4 vma 0x1097c

Start address 0x8110

Transfer rate: 139776 bits/sec.

(gdb) b main

Breakpoint 1 at 0x8230: file main.c, line 7.

(gdb) r

Starting program: /home/student/src/a.out

Breakpoint 1, main () at main.c:7

(gdb) n

Adding 4 and 5 results in 9

(gdb)

Repita o que foi feito, criando o seu primeiro código que mistura C e assembly.

int2str

Crie a função em assembly int2str(inteiro, pontstr) que transforma um inteiro em um string. Por exemplo: o inteiro 1 deve ser transformado na sequencia de bytes 0x31 , 0x0 ; pois o zero representa o final do string. O string deve ser colocado no ponteiro apontado por pontstr.

Use a rotina de divisão da E3 para dividir o número por 10 e ir montando o string. Exemplo, supondo que inteiro =123 esse valor é dividido por 10 com resto 3 ; soma 3 com 0x30 (para formar o caractere ASCII '3'). Coloca o 0x33 em pontstr.

Depois dividido por 10 com resto 2 gera 0x32 ; move todos os bytes para direita dentro de pontstr e insere 0 0x33 em pontstr.

Seguindo, dividido por 10 com resto 1. Repete formando a sequência de bytes 0x31, 0x32, 0x33, que são colocados a partir de pontstr.

Crie uma função em C (main) *char impnum(int num): entrada: num - número, que imprime o número que recebeu, em decimal. Para converter o número deve usar a rotina assembly int2str. Dica: use "puts".

Teste a sua função assim:

char straluno[100]; /_ preferencialmente declarado como variavel global -> fora de qualquer funcao._/

(...)

int2str(numero, straluno);

PRINTSCREEN: tire um printscreen de sua tela logo depois que o número é impresso pela função em C. Teste com seu número USP.

Adapte em imprime.c

Altere a função imprime em imprime.c usado no planejamento, trocando o printf por chamadas de int2str e puts

8.2.2 Alterar o código C inserindo assembly no meio do código C.

Estude o código imprime.s vindo de imprime.C

No seu relatório responda:

Existe algum formato para os labels gerados automaticamente em imprime.s?

Observe que os labels gerados automaticamente seguem um padrao como .L(\d+) ou seja ".L" seguido de dígitos. É importante observar isso porque ao inserir qualquer código em assembly, você não poderá inserir labels desse tipo pois você não tem o controle de como o compilador funciona.

Como o número é passado como parâmetro? Como o fp é usado para isso?

Em uma função, os registradores poderiam ser salvos e recuperados da forma que segue:

BL myfunction

myfunction

.....

STMFD sp!, {r4-r10, lr}; guarda os registradores

.....

.....

LDMFD sp!, {r4-r10, pc}; recupera os registradores; o retorno da funcao eh feito colocando lr em pc.

Para entender melhor como funciona a chamada de rotinas, veja: https://www.eecs.umich.edu/courses/eecs373/readings/ARM-AAPCS-EABI-v2.08.pdf e Responda no relatório:

Quais são os registradores atribuídos a: fp, ip, sp, lr?

Para que serve o fp?

A seguinte pergunta não precisa ir para o seu relatório. Se você tiver uma boa ideia sobre a resposta (porque trabalha com compiladores) discuta a resposta com o Professor.

Para que serve o ip?

A resposta na internet é confusa:

Register r12 (IP) may be used by a linker as a scratch register between a routine and any subroutine it calls (for details, see §5.3.1.1, Use of IP by the linker). It can also be used within a routine to hold intermediate values between subroutine calls. Essa resposta é extremamente confusa, mas é o que está escrito em aapcs.pdf. Em: https://stackoverflow.com/questions/16120123/arm-why-do-i-need-to-push-pop-two-registers-at-function-calls tem-se que a ARM quer o alinhamento de 8 bytes na memória (para o ARM 64 bits). Isso explica o motivo de se ter um registrador scratch (r12, ip) colocado na pilha - porque do ponto de vista da recursão em si, não faz sentido… Dentre os 2 registradores fp e ip, o mais confuso é o ip, sem dúvida, e é chamado de Intra-Procedure-call scratch register. https://community.arm.com/developer/tools-software/oss-platforms/f/dev-platforms-forum/5436/i-want-to-know-meaning-of-r12-register.

Inserindo assembly no meio do código C

Para inserir código assembly no meio do código C (imprime.c), pode-se usar "inline assembly code" como no exemplo a seguir (ver: https://www.ic.unicamp.br/~celio/mc404-s2-2015/docs/ARM-GCC-Inline-Assembler-Cookbook.pdf) :

```
asm (

"ldr r3, [fp, #-16]\n\t"

"mov r0,r1\n\t"

);
```

onde cada linha corresponde a uma instrução assembly. Obviamente é MUITO ARRISCADO mexer nos registradores dessa forma, pois o C também está usando esses registradores, e qualquer alteração pode causar erros no seu código. Apesar dessa ressalva, vamos alterar um ou outro registrador nesse item.

Em C temos variáveis locais e variáveis globais. As variáveis locais são definidas na pilha e para o C qualquer função pode ser recursiva. As variáveis globais são definidas em posições fixas na memória.

Vamos definir a variável global "mostra", no código em C, como:

```
int mostra;
```

Deve ser atribuído o valor 1 logo no começo de seu código. Veja o assembly gerado pelo c, através da opção -S, e procure entender como o valor 1 foi atribuído à variável mostra. No relatório coloque o código correspondente a essa atribuição e explique (dica: veja str)

Observe que o código em assembly gera um monte de labels que começam com ponto (ex: .L2). Muito provavelmente, o assembly gerado usa um destes labels para atribuir 1 à variavel mostra. Quando você usar o "inline assembly", você NÃO DEVE usar nenhum label como .L2 pois isso iria se misturar com os labels gerados pelo C. O desafio proposto é:

Escreva em assembly, um código referente a "mostra = 1" usando inline assembly (ou seja, usando asm). Verifique no gdb que o valor foi devidamente escrito. Dica: Use as instruções em assembly LDR e STR e lembre-se de que às vezes fazemos LDR …, = (na dúvida veja a aula sobre LDR e STR), para isso

você precisará no máximo de 3 instruções: 2 para acertar registradores e 1 para o STR. Você também não deve fazer algo como .word…. declarando um ponteiro, copiando o que foi feito pelo compilador C. Dentro do gdb, para saber o valor de mostra basta fazer:

p mostra

Se tiver dúvida de que as coisas estão funcionando, experimente variar o valor a ser colocado em "mostra" e verifique se o "p mostra" varia de acordo.

Para saber a posição de memória para onde foi parar a variável "mostra" é necessário gerarmos a tabela de símbolos. Supondo que o código gerado seja a.out, então podemos fazer:

student:~/src$ arm-elf-objdump -t a.out|grep mostra

00010ab8 g O .bss 00000004 mostra

OU

student:~/src$ arm-elf-nm a.out | grep mostra

00010ab8 B mostra

A posição 0x10ab8 corresponde ao endereço da variável "mostra". Verifique se essa posição foi alterada ao rodar o código no gdb (variável mostra). No relatório coloque o código correspondente à essa atribuição e explique (dica: veja str).

8.2.3 observar como funciona a recursão.

Observe como o "imprime" do pré-lab imprime um caractere no código assembly (gerado ao compilar .c com -S).

Utilizando o gdb observe como o fp, ip e sp são utilizados em:

stmfd sp!, {fp, ip, lr, pc}

e como são desempilhados em:

ldmfd sp, {r3, fp, sp, pc}

O arm-elf-gcc primeiro passa os parâmetros por registradores e depois empilhando-os dentro da rotina, por isso os parâmetros são acessados via [fp - Número] pois foram empilhados depois da entrada da rotina que empilhou ip, sp, fp. Supondo que uma função tenha 5 parâmetros, a função terá que, obrigatoriamente, empilhar alguns antes da chamada da rotina. Esses parâmetros vão ser acessados via [fp + Número]. Verifique colocando 5 parâmetros na função recursiva. Observe como eles são empilhados e acessados.

Observe que entre o stmfd e o ldmfd, o sp é alterado; por isso lr é repassado para o pc.

Um grupo de alunos (Lucas, Ricardo, Gabriel e Jonatas, de anos anteriores) enviou uma foto de como a pilha se comporta nas chamadas recursivas: pilha.PNG

Perguntas: Como o parâmetro é passado para imprime? Na resposta explique o caso da rotina ter 5 parâmetros (via registrador e pilha) e da rotina ter poucos parâmetros (via registrador).

Como esse parâmetro é empilhado (isso é necessário em caso de chamadas recursivas)? Como é aberto um espaço na pilha para o parâmetro de imprime? Onde isso é feito no código?

Por que se faz fp-16 para acessar o parâmetro?

Como esse parâmetro é desempilhado? Observe que o ip é repassado para sp e com isso, a alteração na pilha para abrir o espaço para o parâmetro de "imprime" é automaticamente refeito.

Para o seu relatório, gere imagens semelhantes às apresentadas pelo grupo do Lucas, Ricardo, Gabriel, Jonatas.

Declare a variável local "int lixo" na função imprime.

Dentro da função recursiva faça, lixo++. Observe o código assembly gerado pelo compilador. Responda no relatório: Como lixo foi referenciado? Como foi aberto espaço na pilha para o lixo? Retire printscreens comparando a pilha sem o uso do int lixo e com o uso de int lixo. Variáveis locais devem ser empilhadas pois caso, a função seja recursiva elas fazem parte da recursão.

Crie imprime.s para que imprima 7 números de 1 a 7

Gere o código imprime.s. Estude como o fp (frame pointer) é usado para marcar uma posição na pilha empilhando parâmetros e variáveis locais. Apresente no relatório como a pilha e o fp estão sendo usados.

Rode essa versão de imprime no gdb. Retire printscreen antes e depois de cada instrução que faça alteração na pilha, em particular:

stmfd sp!, {fp, ip, lr, pc}

ldmfd sp, {r3, fp, sp, pc}

Pergunta: Quando a função imprime chama recursivamente imprime é necessário que haja um ponteiro para o fp anterior. Como isso é feito? Quando imprime retorna para uma instância anterior é necessário que o fp retorne para servir de base para os parâmetros e variáveis locais anteriores. Explique como isso é feito.

Para ver a pilha no gdb faca x/16 $sp