Organização Básica de Computadores e Linguagem de Máquina

Aula de exercícios

Aula de Exercícios

1 - Descreva registradores *callee-save* e *caller-save*

2 - Descreva pilha cheia e pilha vazia.

Aula de Exercícios

1 - Descreva registradores *callee-save* e *caller-save*.

Callee-save são registradores cujo valor deve ser salvo pela rotina que está sendo chamada. Caller-save são registradores cujo valor deve ser salvo pelo trecho de código que chamou a rotina.

2 - Descreva pilha cheia e pilha vazia.

Pilha cheia é uma pilha cujo SP aponta para o dado que está no topo da pilha. Pilha vazia é uma pilha cujo SP aponta para a posição (vazia) subsequente à posição do dado que está no topo da pilha.

Aula de Exercícios

3 - Descreva pilha crescente e pilha decrescente?

4 - Descreva a convenção do ARM quanto à política do uso da pilha (cheia/vazia, crescente/decrescente) e dos registradores (passagem de parâmetros, retorno, *callee-save* e *caller-save*).

Aula de Exercícios

3 - Descreva pilha crescente e pilha decrescente?

A pilha crescente cresce do menor endereço para o maior. A pilha decrescente cresce do maior endereço para o menor.

4 - Descreva a convenção do ARM quanto à política do uso da pilha (cheia/vazia, crescente/decrescente) e dos registradores (passagem de parâmetros, retorno, *callee-save* e *caller-save*).

Os registradores R0 a R3 são *caller-save*. R0 e R1 são usados para retorno de valores em funções. R0 a R3 são usados para passagem de parâmetros. Os registradores R4 a R11 são *callee-save*. A pilha é decrescente cheia.

Aula de Exercícios

5 - Descreva os tamanhos e tipos de dados que podemos ler e escrever na memória com instruções da arquitetura ARM.

6 - Descreva qual o significado das *flags* V, Z, N e C.

Aula de Exercícios

5 - Descreva os tamanhos e tipos de dados que podemos ler e escrever na memória com instruções da arquitetura ARM.

Podemos ler e escrever palavras inteiras (32 *bits*), meia palavra (16 *bits*) e *bytes* (8 *bits*). Estes podem ainda ser com ou sem sinal.

6 - Descreva qual o significado das *flags* V, Z, N e C.

V indica se houve *overflow* na representação com sinal. Z indica se o resultado da operação é zero. N indica se o resultado da operação é negativo. C indica se houve *carry-in* ou *carry-out*.

Aula de Exercícios

7 - Qual o valor nos registradores r2, r3, r4 e r5 ao fim da execução dos trechos de código abaixo? Condições de execução das intruções com sufixos são cs: C = 1, eq: Z = 1, vs: V = 1, pl: N = 0.

```
ldr r0, =-1

ldr r1, =1
ldr r2, =5
adds r0, r0, r1

mov r3, r2
mov r4, r2
mov r5, r2
```

```
addcs r2, r2, r1

addeq r3, r3, r1
addvs r4, r4, r1
addpl r5, r5, r1
```

Aula de Exercícios

```
ldr r0, =-1
ldr r1, =1
ldr r2, =5
adds r0, r0, r1
@ gera as flags:
@ z = 1, v = 0, c = 1, n = 0

mov r3, r2
mov r4, r2
mov r5, r2
@ r2 = r3 = r4 = r5 = 5
```

```
@ condições:
@ cs: c = 1
@ vs: v = 1

addcs r2, r2, r1
soma
addeq r3, r3, r1
soma
addvs r4, r4, r1
efetua
addpl r5, r5, r1
soma
```

```
eq: z = 1
pl: n = 0

@ efetua

@ efetua

@ nao

@ efetua
```

Aula de Exercícios

8 - Qual o valor nos registradores r1 e r2 ao fim da execução do trecho de código abaixo?

```
ldr r0, =8000
    ldr r1, =ACDE0F10
    str r1, [r0]
    ldrsb r2, [r0, #1]
    ldrsb r1, [r0, #2]
```

Aula de Exercícios

8 - Qual o valor nos registradores r1 e r2 ao fim da execução do trecho de código abaixo?

```
        ldr r0, =8000
        ldr r1, =ACDE0F10
    str r1, [r0]                    @ 0x8000: ACDE0F10
    ldrsb r2, [r0, #1]              @ r2 <= 0000000F
    ldrsb r1, [r0, #2]              @ r1 <= FFFFFFDE
```

Aula de Exercícios

9 - Transcreva o código em C para ARM, considerando a struct descrita.

```
struct id {
	struct id* next;
		short val;
}

struct id* find_element (struct id* p, short v) {
	for (; p != 0; p = p->next)
			if (p->val == v)
				return p;
		return 0;
}
```

Aula de Exercícios

9 - Transcreva o código em C para ARM, considerando a struct descrita.

```
find_element:
        cmp r0, #0              @ if (p != 0)
        beq fim                         @ sai do loop
        ldrsh r2, [r0, #4]      @ r2 <= p->val
        cmp r2, r1                      @ if (p->val == v)
        beq fim                         @ return p
        ldr r0, [r0]                    @ r0 <= p->next
        bne find_element        @ realiza loop
fim:
    mov pc, lr                  @ retorno da funcao
```

Aula de Exercícios

10 - Transcreva o código em C para ARM, considerando a struct descrita e a assinatura da função *bar*.

```
struct reg {
      short a[10];
            int b, c, d;
}
void bar (int u, int v, int x, int* y, int* z);
struct reg x;

void soma (int a, int b) {
      bar (a, b, x.b, &x.c, &x.d);
}
```

Aula de Exercícios

10 - Transcreva o código em C para ARM, considerando a struct descrita e a assinatura da função *bar*.

soma:

```
        stmfd sp!, {lr}          @ empilha lr
        ldr r3, =x                       @ r3 <= &x
        add r2, r3, #28          @ r2 <= &x.d
        stmfd sp!, {r2}          @ empilha r2
        ldr r2, [r3, #20]        @ r2 <= x.b
        add r3, r3, #24          @ r3 <= &x.c
        bl bar                           @ chama funcao
        add sp, sp, #4           @ desempilha
        ldmfd sp!, {pc}          @ retorno da funcao
```

Aula de Exercícios

11 - Transcreva o código em C para ARM, considerando a assinatura da função *bar*. Declare *y* como uma variável global e *x* como variável local.

```
void bar (int* a, int b);

int y = 2;

int foo (int a, int b, int c, int d, int e, int f) {
	int x = 0;
		bar (&x, y);
		return x+b+f;
}
```

Aula de Exercícios

11 - Transcreva o código em C para ARM, considerando a assinatura da função *bar*. Tome *y* como uma variável global e *x* como variável local.

```
y: .word 0x0000000002
foo:
        mov r0, #0                      @ x := 0
        stmfd sp!, {r0-r1, r11, lr}     @ empilha valores
        add fp, sp, #12                 @ atulaliza fp
        add r0, sp, #4                  @ r0 := &x
        mov r1, =y                      @ r1 := &y
        mov r1, [r1]                    @ r1 := y
        bl bar
        ldmfd sp!, {r0, r1}             @ desemplilha x e b
        add r0, r0, r1                  @ r0 := x + b
        ldr r1, [fp, #8]                @ carrega param. f em r1
        add r0, r0, r1                  @ r0 += f
        ldmfd sp!, {r11, pc}
```

Aula de Exercícios

12 - Transcreva o código em C para ARM, considerando a struct descrita.

```
struct aluno {
	char nome[128];
		unsigned short ra,
		unsigned short ingresso;
}

unsigned short atualiza_ingresso (struct aluno* t, unsigned short ano) {
	unsigned short aux = aluno->ingresso;
		aluno->ingresso = ano;
		return aux;
}
```

Aula de Exercícios

12 - Transcreva o código em C para ARM, considerando a struct descrita.

```
atualiza_ingresso:
        ldrh r2, [r0, #130]     @ r2 <= aluno->ingresso
        strh r1, [r0, #130]     @ aluno->ingresso <= ano
        mov r0, r2              @ r0 <= r2
        mov pc, lr              @ retorno
```