Trabalho de Arquitetura de Sistema e Computadores I

Pedro Anjos (45558), Rafael Silva (45813)

31/05/2020

Conteúdo

1 Introdução

Pretende-se desenvolver uma calculadora que opere na notação polaca inversa (RPN - Reverse Polish No-tation). A calculadora deverá ler strings da consola, realizar as operações indicadas e mostrar o estado da memória da calculadora na forma de uma pilha.

2 Desenvolvimento do trabalho

2.1 Imprimir-pilha

```
1  imprimir_pilha:
2
3  lw $a0,0($sp)
4  beq $a0,0x00,saida
5  nop
6  li $v0,1
7  syscall
8  li $v0,4
9  la $a0,$espaco
10 syscall
```

Esta função tem como objetivo imprimir os valores da pilha. O branch equal verifica se o numero colocado é zero. Caso se verifique, salta para a função saida (Não imprime mais nenhum numero). De seguida encontramos dois load imidiates, sendo que o primeiro é responsável por imprimir os números inteiros(la v0,1) e outro imprime uma word (la vo,4). Por fim temos um load adress que passa o endereço espaço para a memória, fazendo uma quebra de linha. Sucessivamente, de 4 em 4 bits, avançamos na pilha.

2.2 Multii

```
1  multii:
2
3  li $t6,10
4  mult $t7,$t6
5  mflo $t7
6  add $t7,$t7,$t0
7  j ciclo
8  nop
```

Esta função é responsável por imprimir números maiores que 9. Para que possa colocar esse número, é necessário fazer um load immidiate, que atribui o valor de 10 ao registo t6. De seguida, multiplicamos esse registo com o registo t7, que tem como valor 0. O resultado fica guardado no registo t7, e logo após é feita uma soma com o registo t0 (numero introduzido pelo utilizador). Por fim,

faz jump para a função ciclo.

2.3 Ciclo

```
ciclo:

lb $t0,0($a0)
addi $a0,$a0,1
beq $t0,0x0a, continua
nop
beq $t0, 0x20 ,continua
nop
blt $t0, 0x30, operacao
nop
bgt $t0, 0x3a, operacao
nop
```

A função ciclo é responsável pela leitura caracter a caracter. Os branchs equals são responsáveis para verificar caracteres nulos e espaços. Caso o mesmo se verifique, salta para a função continua. De seguida, as pseudoinstruções verifica se são sinais operacionais ou um operador unário.

2.4 Encontra-num

```
encontra_num:

j multii
addi $t0,$t0,-48
```

Esta função tem como funcionalidade converter os números ASCII para decimal..

2.5 Continua

```
continua:

addi $sp,$sp,-4
sw $t7,0($sp)
li $t7,0
j ciclo
nop
```

Esta função simplesmente guarda o valor no topo da pilha. Quando o número é colocado no topo da pilha, salta para a função para que possa continuar a analisar o resto da string

2.6 P-swap

```
1  p_swap:
2
3  lb $t0,0($a0)
4  bne $t0,0x77,operacao2
5  nop
6  addi $a0,$a0,1
7  lb $t0,0($a0)
8  bne $t0,0x61,operacao2
9  addi $a0,$a0,1
10 lb $t0,0($a0)
11 beq $t0,0x70,swap
12 addi $a0,$a0,1
```

Esta função verifica se a string introduzida no input é um swap. Como a letra s foi atribuida na função operação, o mesmo faz várias comparações para verificar o 'w', 'a' e 'p'. Caso se verifique, salta para a função swap.

O mesmo acontece nas funções p.neg, p.off, pd, p.drop, p.clear, e p.help, que difere apenas os caracteres que compara.

2.7 Operacao

```
1  operacao:
2
3  addi $t3,$zero,104
4  beq $t0,$t3,p_help
5  nop
6  addi $t3,$zero,43
7  beq $t0 ,$t3,soma
8  nop
9  addi $t3,$zero,45
10 beq $t0,$t3,subtracao
11 nop
12 addi $t3,$zero,42
13 beq $t0,$t3,multi
14 nop
15 addi $t3,$zero,47
16 beq $t0,$t3,divi
17 nop
18 addi $t3,$zero,115
19 beq $t0,$t3,p_swap
20 nop
21 addi $t3,$zero,110
22 beq $t0,$t3,p_neg
23 nop
24 addi $t3,$zero,100
25 beq $t0,$t3, pd
26 nop
27 addi $t3,$zero,99
28 beq $t0,$t3,p_clear
29 nop
30 addi $t3, $zero,111
31 beq $t0,$t3,p_off
32 nop
```

A função operacao verifica se o que foi introduzido no input corresponde a uma operação. A mesma comparação é feita com os inúmeros branch equals, que atraves do registo t3 (guarda o decimal de cada operação), verificando o mesmo. Quando é encontrado, salta para a função correspondente. Caso não não haja correspondência, continua o programa.

2.8 Operacao2

```
operacao2:

lb $t0,0($a0)
addi $a0,$a0,1
beq $t0,0x20,operacao2
nop
addi $t3,$zero,43
beq $t0 ,$t3,soma
nop
addi $t3,$zero,45
beq $t0,$t3,subtracao
nop
addi $t3,$zero,42
beq $t0,$t3,multi
nop
addi $t3,$zero,47
beq $t0,$t3,divi
nop
addi $t3,$zero,115
beq $t0,$t3,p_swap
nop
addi $t3,$zero,110
beq $t0,$t3,p_neg
nop
addi $t3,$zero,100
beq $t0,$t3,pd
nop
addi $t3,$zero,99
beq $t0,$t3,p_clear
nop
bgt $t0, 0x2f, encontra_num
nop
li $v0,4
la $a0, $pilha
syscall
jal imprimir_pilha
nop
li $v0,4
la $a0,$espacos
syscall
j Input
nop
```

Esta função verifica se existe algum tipo de operação ou numeros depois dos já introduzidos. Caso se encontre um número, através da pseudoinstrução salta para a LABEL encontra-num, que se encontra no ciclo. Caso seja uma operação, ela realiza-a, retornando à LABEL operacao2, até que a mesma não encontre nenhuma operação ou número. Se isso se verificar, chama a função imprimir-pilha, que por sua vez imprime no output.

2.9 Soma

```
1  soma:
2
3  lw $a2, 0 ($sp)
4  lw $a3, 4 ($sp)
5  addi $sp,$sp,8
6  add $v0, $a2, $a3
7  addi $sp,$sp,-4
8  sw $v0, 0 ($sp)
9  j operacao2
10 nop
```

O pedaço de código apresentado retira os dois números que se encontram no topo da pilha. Após a incrementação realiza a soma, guardando o resultado em v0. Por sua vez é guardado no topo da pilha. Por fim, é feito um jump que verifica se existem mais operações.

As seguintes funções (subtracao, multi, divi) realizam o mesmo, diferindo apenas a operação a realizar (- , *, /)

2.10 Help

```
1 help:
2
3 li $v0,4
4 la $a0,$help
5 syscall
6 j operacao2
7 nop
```

Este pedaço de código imprime o help quando o mesmo é invocado, que se encontra no .data.

2.11 Negacao

```
1 negacao:
2
3 lw $a2, 0 ($sp)
4 addi $sp,$sp,4
5 sub $v0, $zero, $a2
6 addi $sp,$sp,-4
7 sw $v0, 0($sp)
8 j operacao2
9 nop
```

O troço apresentado retira o número que se encontra no topo da pilha. Após a incrementação, é feita uma subtração que vai guardar em v0 o seu simétrico, guardando o mesmo no topo da pilha.

2.12 Swap

```
swap:

lw $a2, 0 ($sp)
lw $a3, 4 ($sp)
addi $sp,$sp,8
addi $sp,$sp,-8
sw $a2, 4 ($sp)
sw $a3, 0 ($sp)
j operacao2
nop.
```

Esta função retira os dois números que se encontram no topo da pilha. Logo após, troca os mesmos de posição, guardando na pilha.

2.13 Dup

```
dup:

lw $a2, 0 ( $sp)
addi $sp,$sp,4
addi $sp,$sp,-8
sw $a2, 0 ($sp)
sw $a2, 4 ($sp)
j  operacao2
nop
```

O troço apresentado retira o número que se encontra no topo da pilha, duplicando o seu valor. Após esse comando, o mesmo é guardado da pilha.

2.14 Drop

```
drop:

lw $a2, 0 ($sp)
addi $sp, $sp, 4
j  operacao2
nop
```

Esta função elimina o número que se encontram na pilha.

2.15 Clear

```
clear:

li $sp,0x7fffeffc
j  operacao2
nop
j Input
nop
```

A função coloca o stack pointer no seu estado inicial, não contendo qualquer número no seu interior.

2.16 Off

```
off:

li $v0,4
la $a0,
syscall
li $v0,10
syscall
```

Por fim, encontramos a função off que imprime o off quando o mesmo é invocado, que se encontra no .data.

3 Conclusão

Conseguimos cumprir as metas do trabalho, a pilha encontra-se operacional, mas cometemos erros no qual achamos inportante referir.

Quando colocamos o valor '0' na pilha, ou quando o resultado de uma operação é '0', o mesmo não é mostrado no output, sendo o que se encontra posteriormente a esse número não é apresentado no output .

Ao imprimir a pilha, só são apresentados os dez primeiros valores, não pelo tamanho da pilha, mas por imprimirmos os valores por uma iteração manual.