ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
DISCIPLINA: LABORATÓRIO DE PROCESSADORES- PCS3732
1° QUADRIMESTRE/2021

Aula 03

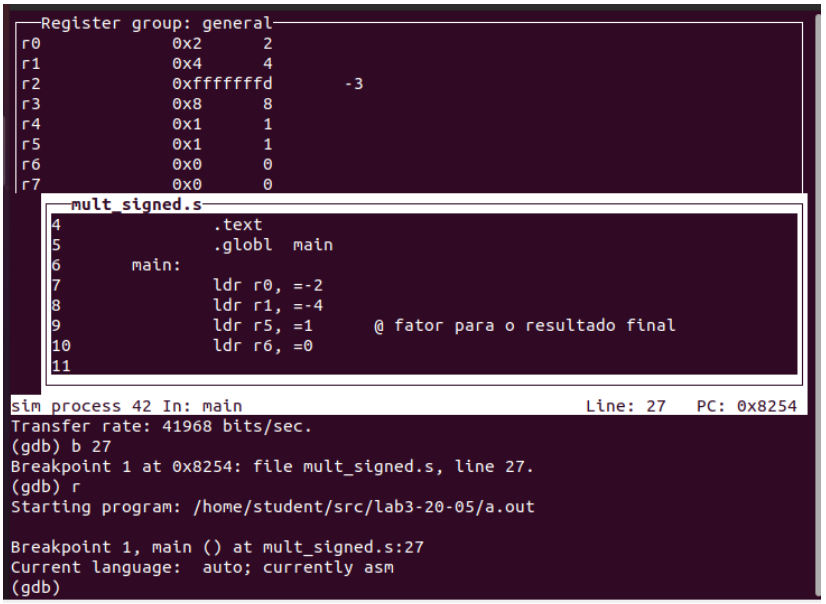
20 de Maio de 2021

GRUPO 10

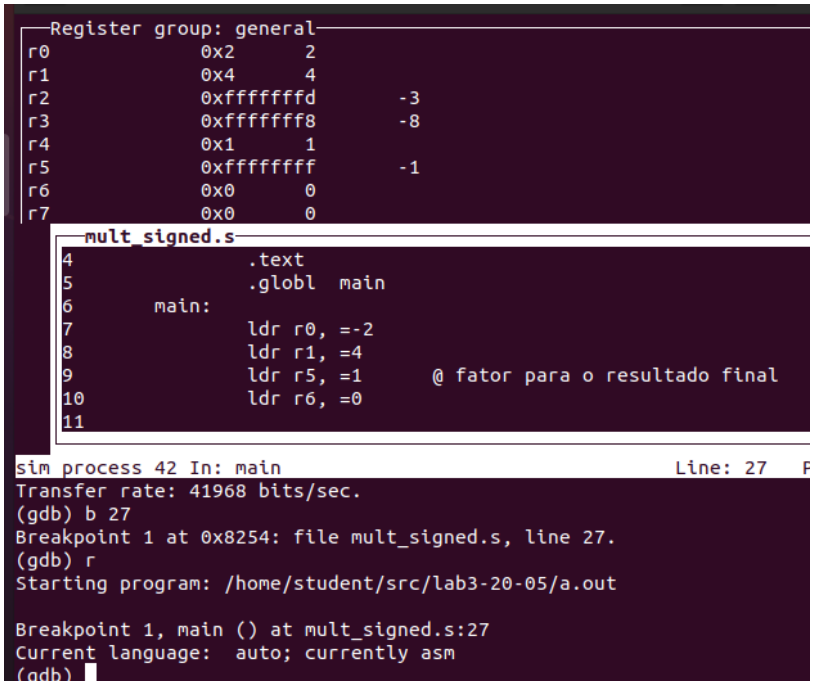
Arthur Pires da Fonseca	NUSP: 10773096
Bruno José Móvio	NUSP: 10336852
Iago Soriano Roque Monteiro	NUSP: 8572921

Sumário

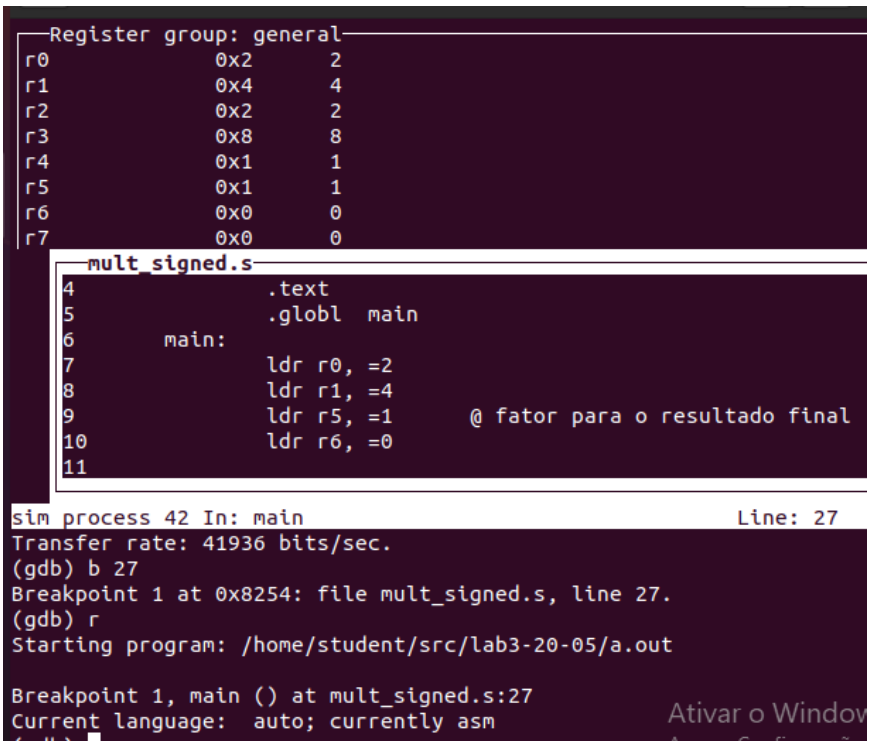
Exercício 3.10.5 - Signed multiplication



Multiplicando dois números negativos.

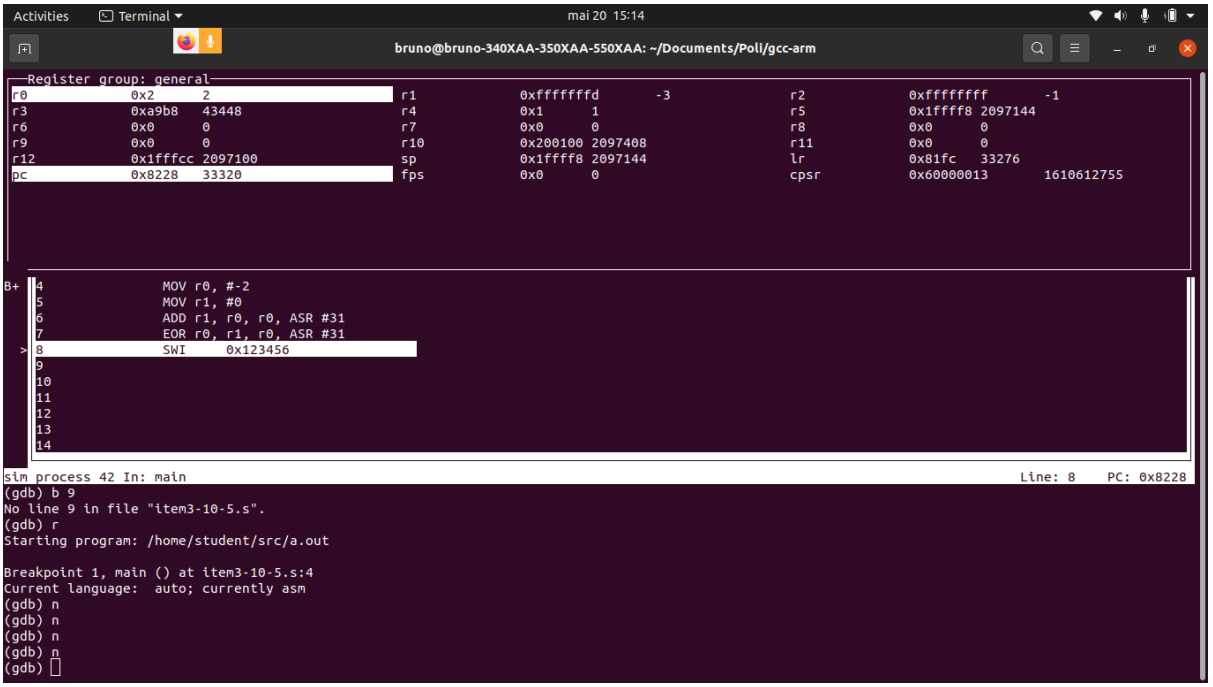


Multiplicando um número negativo e um positivo

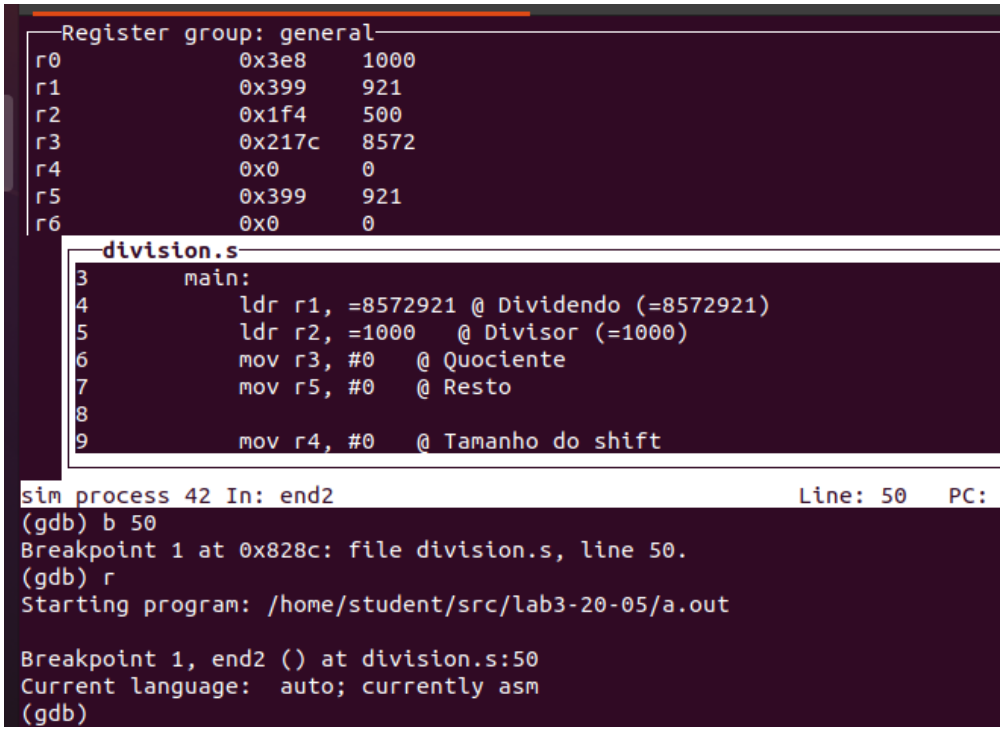


Multiplicando dois números positivos

Exercício 3.10.6 - Absolute Value

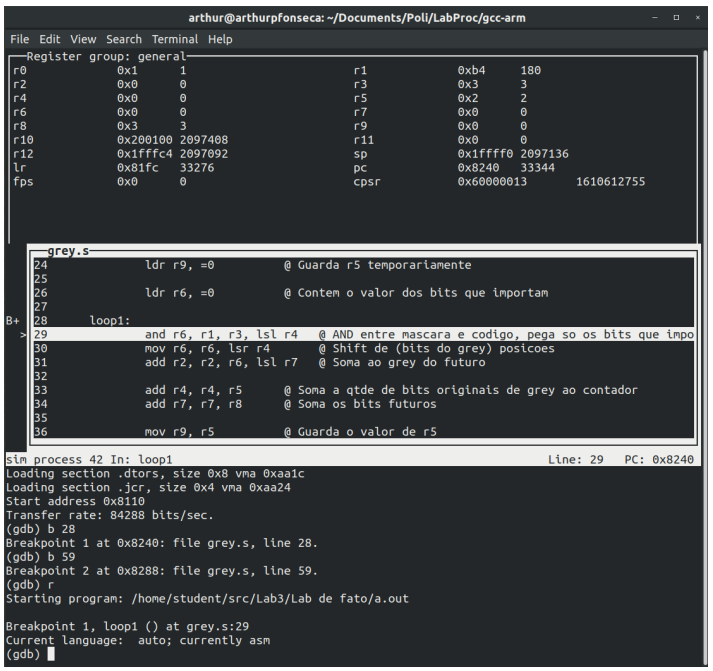


Exercício 3.10.7 - Division

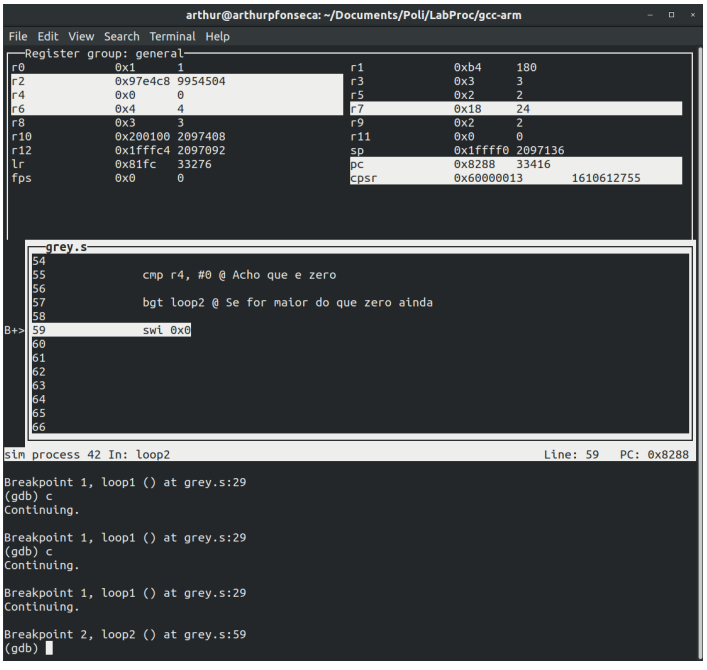


Exercício 3.10.8 - Gray codes

Antes de rodar o código:



Depois de rodar o código:



Apêndice

1. Exercício 3.10.5 - Signed Multiplication

```
@ Exercicio 3.10.5 da apostila
@ Para debugar este codigo:
@ gcc mult_signed.s && gdb a.out

   .text
   .globl    main
main:
   ldr r0, =2
   ldr r1, =4
   ldr r5, =1    @ fator para o resultado final
   ldr r6, =0

   CMP r0, #0
   SUBLT r5, r6, r5, lsl #0
   CMP r1, #0
   SUBLT r5, r6, r5, lsl #0

   MOV r2, #0
   ADD r2, r1, r1, ASR #31
   EOR r1, r2, r1, ASR #31

   ADD r2, r0, r0, ASR #31
   EOR r0, r2, r0, ASR #31

   MUL r3, r0, r1
   MUL r3, r5, r3

   swi 0x0
```

2. Exercício 3.10.6 - Absolute Value

```
@ Exercicio 3.10.6 da apostila
@ Para debugar este codigo:
@ gcc absolute_val.s && gdb a.out

.text
        .globl  main
main:
   MOV r0, #-2
   MOV r1, #0
   ADD r1, r0, r0, ASR #31
   EOR r0, r1, r0, ASR #31
   SWI    0x123456
```

3. Exercício 3.10.7 - Division

```
@ Exercicio 3.10.7 da apostila
@ Para debugar este codigo:
@ gcc division.s && gdb a.out

.text
        .globl  main
main:
        ldr r1, =0x000003FD @ Dividendo (=1021)
        mov r2, #25   @ Divisor (=25)
        mov r3, #0   @ Quociente
        mov r5, #0   @ Resto

        mov r4, #0   @ Tamanho do shift
        mov r0, r2   @ Guarda o valor inicial do divisor
        b align

        @ Divisao
        @ Alinhar bits mais à esquerda

align:
        mov r2, r2, lsl #1
        cmp r2, r1
        ble align    @ r2 (divisor) < r1 (dividendo) => continua

div_loop:
        cmp r1, r2
        bge quociente_1 @ r1 (dividendo) >= r2 (divisor) ?
        b quociente_0
loop_end:
        mov r2, r2, lsr #1 @ Desloca para a direita o divisor
        cmp r1, r0
        blt end @ r1 (dividendo) < r0 (divisor original) => acabou divisão
        b div_loop @ Senao, continua o loop

quociente_1:
        sub r1, r1, r2 @ Tira o divisor do dividendo
        mov r3, r3, lsl #1 @ Desloca o quociente
        add r3, r3, #1 @ Adiciona um 1 ao fim do quociente
        b loop_end

quociente_0:
        mov r3, r3, lsl #1
        b loop_end

end:
        cmp r2, r0
        blt end2          @ Se o divisor voltou ao valor original, nao tem mais o
que fazer
```

```
        mov r2, r2, lsr #1 @ Divide divisor por 2
```

4. Exercício 3.10.8 - Gray codes

```
@ Exercicio 3.10.8 da apostila
@ Para debugar este codigo:
@ gcc grey.s.s && gdb a.out
@ Codigo grey exemplo: 000 001 011 010 110 111 101 100
@ Grey 2: 10 11 01 00
@ Algoritmo: 0 10, 0 11, 0 01, 0 00, 1 00, 1 01, 1 11, 1 10
@ 10 11 01 00 = 0xb4
@ VIRA
@ 100 101 111 110 010 011 001 000 = 1001 0111 1110 0100 1100 1000 =
0x97e4c8

        .text
        .globl  main
main:
        ldr r0, =1          @ Constante 1

        ldr r1, =0x000000b4 @ Codigo grey de 2 bits
        ldr r2, =0          @ Futuro codigo grey de 3 bits

        ldr r3, =0x00000003 @ Mascara para pegar 2 bits
        ldr r4, =0        @ Contador de bits para shift no codigo original
        ldr r7, =0          @ Contador de bits para shift no codigo futuro
        ldr r5, =2        @ Qtde de bits do codigo grey original
        ldr r8, =3          @ Qtde de bits do codigo grey futuro

        ldr r9, =0          @ Guarda r5 temporariamente

        ldr r6, =0          @ Contem o valor dos bits que importam

loop1:
        and r6, r1, r3, lsl r4   @ AND entre mascara e codigo, pega so os bits
que importam
        mov r6, r6, lsr r4      @ Shift de (bits do grey) posicoes
        add r2, r2, r6, lsl r7   @ Soma ao grey do futuro

        add r4, r4, r5      @ Soma a qtde de bits originais de grey ao contador
        add r7, r7, r8      @ Soma os bits futuros

        mov r9, r5          @ Guarda o valor de r5
        mov r5, r0, lsl r5  @ Sobrescreve com o valor do shift (1 * 2^(bits do
grey))
        cmp r4, #8 @r0, lsl r5  @ Ve se r4 e maior que 2^(bits do grey)
        mov r5, r9          @ Recupera valor de r5

        blt loop1          @ Continua o loop se contador < 2^(bits do grey)
loop2:
```

```
        sub r4, r4, r5
        @sub r7, r7, r8

        and r6, r1, r3, lsl r4 @ Mesmo codigo de cima
        mov r6, r6, lsr r4
        add r6, r6, r0, lsl r5 @ Soma 100 (qtde de zeros necessarias) pra fazer
um complemento do loop1
        add r2, r2, r6, lsl r7

        @sub r4, r4, r5
        add r7, r7, r8

        cmp r4, #0 @ Acho que e zero

        bgt loop2 @ Se for maior do que zero ainda

        swi 0x0
```