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

Aula 6

17 de Junho de 2021

GRUPO 10

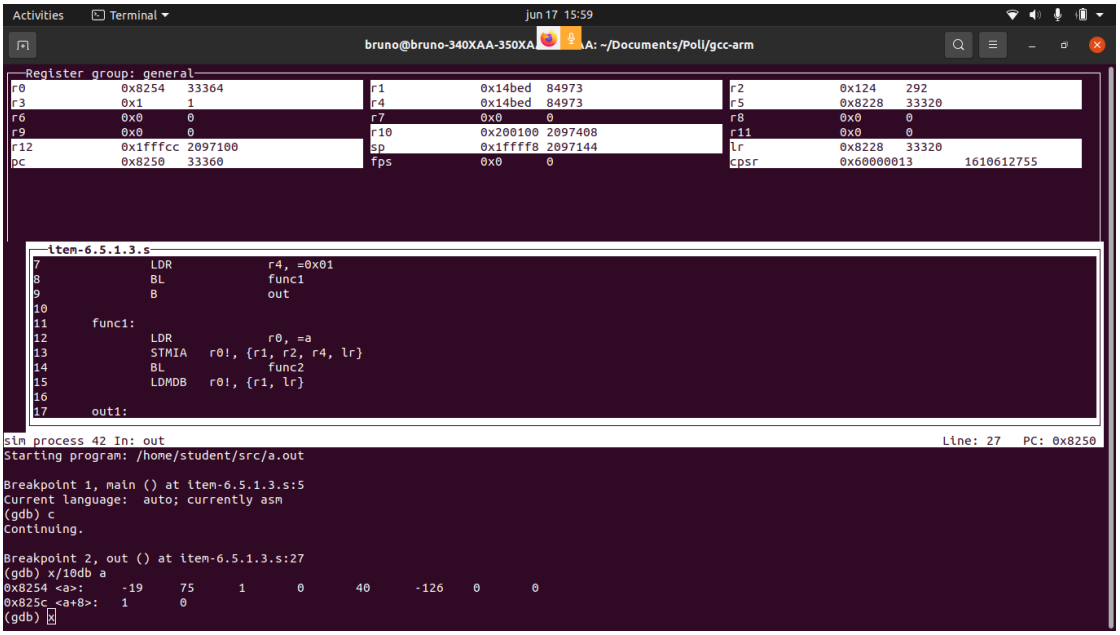
Arthur Pires da Fonseca	NUSP: 10773096
Bruno José Móvio	NUSP: 10336852
Iago Soriano Roque Monteiro	NUSP: 8572921

Sumário

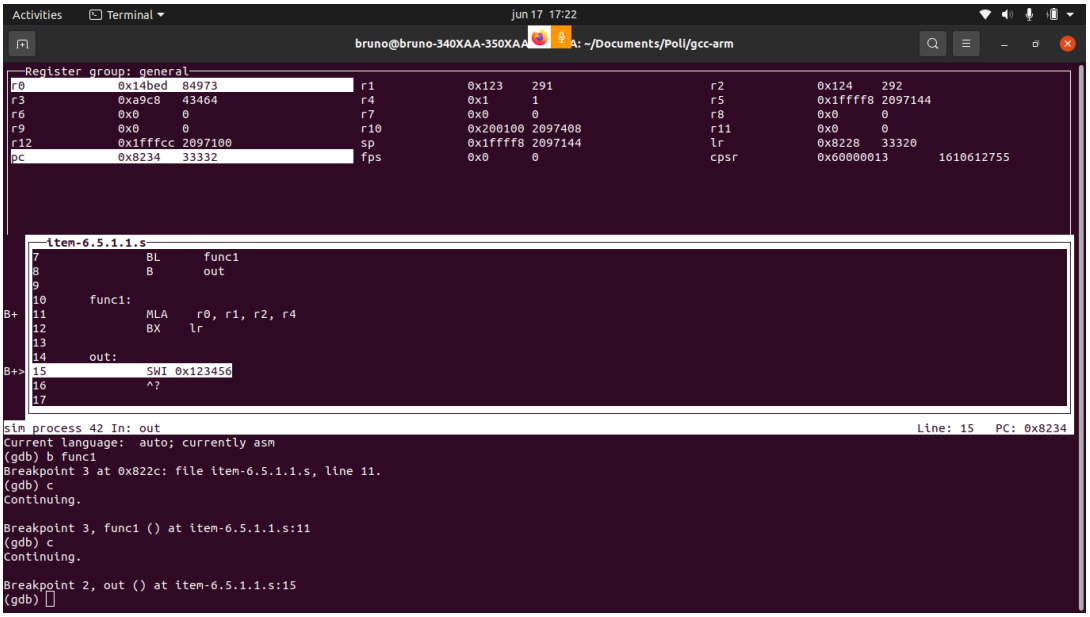
Exercício 6.5.1 Transmission of arguments

transmit the arguments by way of registers with one subroutine, func1

Dados armazenados nos registradores r1, r2 e r4 (b, c e d, respectivamente)

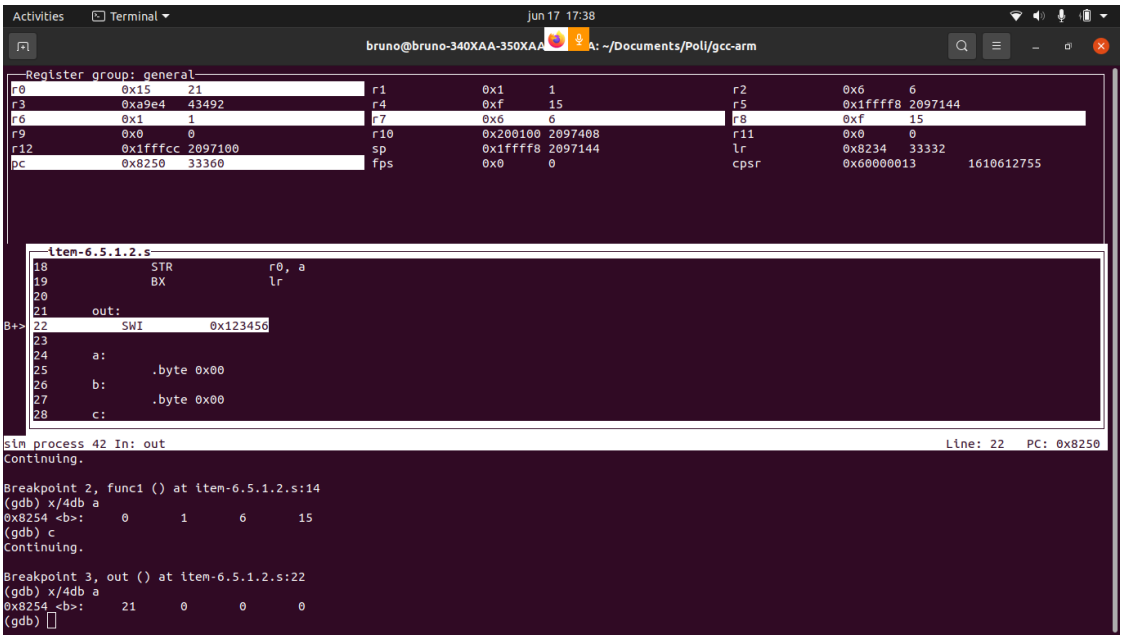


Resultado armazenado em r0 (variável a)



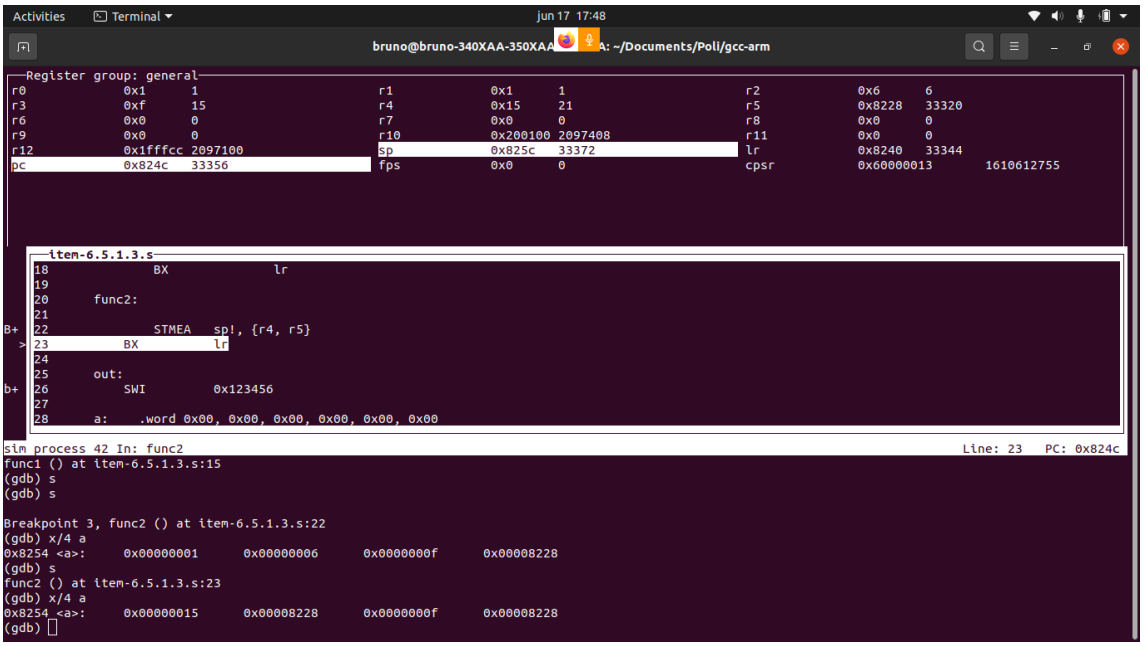
transmit the arguments by way of the addresses with one subroutine, func1

Antes, exibindo os valores de a (0), b (1), c (6) e d (15)
Após o Continue exibe os valores finais, com a = 21 = 1 x 6 + 15

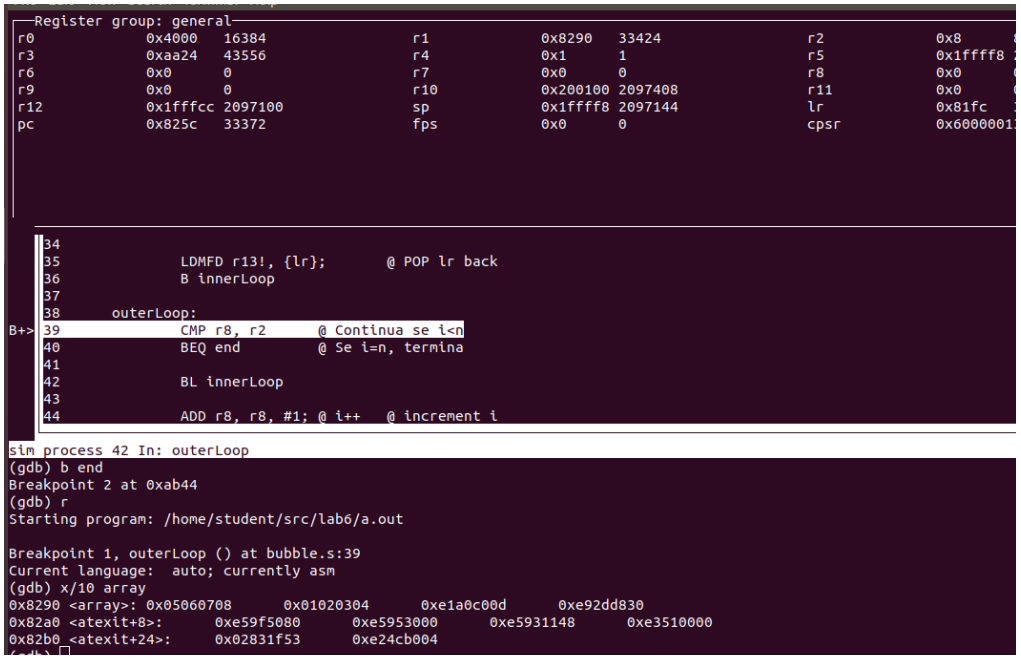


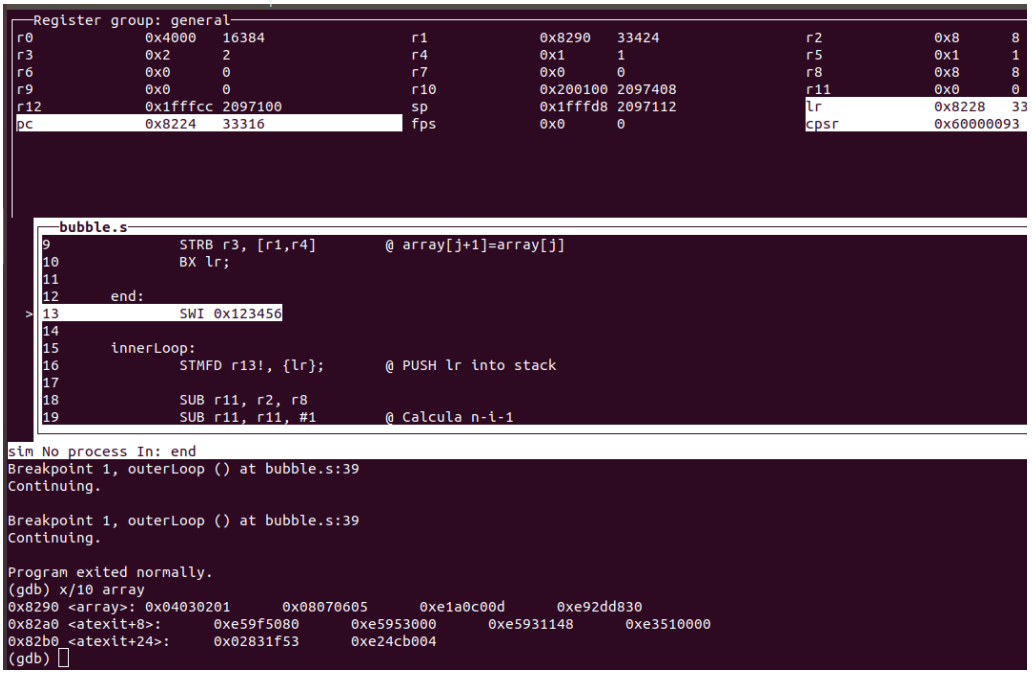
transmit the arguments by way of the addresses with one subroutine, func1

Antes exibe os valores de b (0x01), c(0x06) e d (0x0f).
As operações foram realizadas em base 16.
O resultado encontrado é a = 0x15 = 21



Exercício 6.5.2 Bubble sorting





Exercício 6.5.3 Magic squares

gdb.txt enviado junto com este relatório.

Exercício 6.5.4 More stacks

O código escreve no stack com base na variável r2 settada.

Apêndice

1. Exercício 6.5.1.1

```
        .text
        .globl  main
main:
        LDR    r1, =0x123
        LDR    r2, =0x124
        LDR    r4, =0x01
        BL    func1
        B    out

func1:
        MLA    r0, r1, r2, r4
        BX      lr

out:
        SWI 0x123456
```

2. Exercício 6.5.1.2

```
        .text
        .globl  main
main:
   LDR          r1, =0x01
   LDR          r2, =0x06
   LDR          r4, =0x0f
   STRB     r1, b
   STRB    r2, c
   STRB    r4, d
   BL           func1
   B      out

func1:
   LDRB    r6, b
   LDRB    r7, c
   LDRB    r8, d
   MLA          r0, r6, r7, r8
   STR          r0, a
   BX           lr
```

```
out:
        SWI    0x123456

a:
   .byte 0x00
b:
   .byte 0x00
c:
   .byte 0x00
d:
   .byte 0x00
```

3. Exercício 6.5.1.3

```
        .text
        .globl  main

main:
   LDR          r1, =0x01
   LDR          r2, =0x06
   LDR          r4, =0x0f
   BL           func1
   B            out

func1:
   LDR          sp, =a
   STMEA        sp!, {r1, r2, r4, lr}
   LDMEA        sp!, {r1-r3, r5}
   MLA          r4, r1, r2, r3
   BL           func2
   LDMEA        sp!, {r1, lr}
   BX           lr

func2:

   STMEA        sp!, {r4, r5}
   BX           lr

out:
   SWI    0x123456
```

```
a:      .word 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

4. Exercício 6.5.2

```
@ Exercicio 6.5.2 do livro
@ Para debugar este código:
@ gcc bubble.s && gdb a.out
   .text
   .globl   main

swap:
   STRB r5, [r1,r9]   @ array[j]=array[j+1]
   STRB r3, [r1,r4]   @ array[j+1]=array[j]
   BX lr;

end:
   SWI 0x123456

innerLoop:
   STMFD r13!, {lr};   @ PUSH lr into stack

   SUB r11, r2, r8
   SUB r11, r11, #1    @ Calcula n-i-1
   CMP r9, r11         @ Continua se j<n-i-1
   BXEQ lr             @ Se j=n-i-1, volta ao loop externo.

   LDRB r3, [r1, r9]   @ r3=array[j]
   ADD r4, r9, #1      @ r4=j+1
   LDRB r5, [r1, r4]   @ r5=array[j+1]

   CMP r3, r5;


   BLGT swap           @ if(array[j] > array[j+1]) swap()

   @ increment
   ADD r9, r9, #1;   @ j++

   LDMFD r13!, {lr};   @ POP lr back
   B innerLoop

outerLoop:
   CMP r8, r2   @ Continua se i<n
   BEQ end    @ Se i=n, termina

   BL innerLoop

   ADD r8, r8, #1;   @ i++   @ increment i
   MOV r9, #0x0;   @ j=0   @ restart j
```

```
   B outerLoop

main:
   LDR r0, =0x4000    @ endereço do tamanho da array
   ADR r1, array;

   MOV r2, #0x8     @ array de 8 endereços
   STRB r2, [r0]    @ mem[0x4000]=8

   MOV r8, #0x0;    @ i=0
   MOV r9, #0x0;    @ j=0

   B outerLoop

array:    .byte 0x08, 0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01;
```

5. Exercício 6.5.3

```
@ Exercicio 6.5.3 do livro
@ Para debugar este codigo:
@ gcc magic_squares.s && gdb a.out
   .text
   .globl    main

main:
   MOV r0, #0x03;    @ N
   ADD r1, r0, #1; @ N+1
   MOV r2, #0x0;    @ constante = N(N*N+1)/2
   MOV r3, #0;     @ contador i
   MOV r9, #0;    @ ehmagico
   MOV r6, #0;    @ S. soma da fila. Começa em 0

   ADR r10, quadrado;
   ADR r11, ehmagico;

   BL calcularConstante;
   BL checkPrimeDiagonal;
   BL checkMagic;

   MOV r7, #1;    @ contador j
   MOV r6, #0;    @ zera S antes de continuar.
   BL checkSecondaryDiagonal;
   BL checkMagic;

   MOV r7, #0;    # contador de filas
   MOV r6, #0;    @ zera S antes de continuar.
   MUL r1, r0, r0;       @ r1=N²
   BL checkColumns;

   MOV r7, #0;    # contador de filas
```

```
   MOV r6, #0;    @ zera S antes de continuar.
   BL checkRows;

   MOV r9, #1;    @ ehmagico=1
   STR r9, [r11];
   B fim;

checkColumns:
   STMFD r13!, {lr};    @ PUSH lr into stack
   CMP r7, r0;

   BXEQ lr;
   MOV r3, r7;     @ setta contador da coluna
   BL addColumn;
   BL checkMagic;
   MOV r6, #0;    @ zera S antes de continuar.
   ADD r7, r7, #1;

   LDMFD r13!, {lr};    @ POP lr back
   B checkColumns;

addColumn:
   CMP r3, r1;
   BXGT lr;
   BXEQ lr;
   MOV r4, r3, LSL #2;     @ Multiplica i por 4 para indexar por palavra
   LDR r5, [r10, r4];     @ r5=array[i];
   ADD R6, R6, R5;     @ S+=r5;
   ADD r3, r3, r0;     @ i+=N

   B addColumn;

checkRows:
   STMFD r13!, {lr};    @ PUSH lr into stack
   CMP r7, r0;    @ j(r7) indica a row atual. Uma vez que j=N, terminaram as
rows.
   BXEQ lr;
   MUL r3, r7, r0;     @ i(r3) conta o indice na array.
   BL addRow;
   BL checkMagic;
   MOV r6, #0;    @ zera S antes de continuar.
   ADD r7, r7, #1;
   LDMFD r13!, {lr};    @ POP lr back
   B checkRows;

addRow:
   ADD r4, r7, #1;       @ r4=j+1
   MUL r5, r0, r4;       @ r5=(j+1)N
   SUB r5, r5, #1;       @ r5=(j+1)N-1
   CMP r3, r5            @ se i>(j+1)N-1, chegou ao final da row
   BXGT lr;

   MOV r4, r3, LSL #2;     @ Multiplica i por 4 para indexar por palavra
   LDR r5, [r10, r4];     @ r5=array[i];
```

```
   ADD R6, R6, R5;    @ S+=r5;
   ADD r3, r3, #1;    @ i++

   B addRow;

checkMagic:
   CMP r6, r2;
   STRNE r9, [r11];   @ ehmagico=0
   BNE fim;   @ Se primeira diagonal já não for, pára.
   BX lr;

checkSecondaryDiagonal:
   CMP r7, r0;          @ ver se j > N
   BXGT lr;     @ se j > N, terminou o loop

   SUB r1, r0, #1;      @ r1=N-1
   MUL r4, r1, r7;    @ r4=j(N-1)
   MOV r4, r4, LSL #2;    @ Multiplica por 4 para indexar por palavra
   LDR r5, [r10, r4]   @ r5=array[i(N+1)]

   ADD r6, r5, r6;      @ S+=array[i(N+1)]

   ADD r7, r7, #1;      @ j++
   B checkSecondaryDiagonal

checkPrimeDiagonal:
   CMP r3, r0;          @ ver se i > N
   BXEQ lr;     @ se i = N, terminou o loop

   MUL r4, r1, r3;    @ r4=i(N+1)
   MOV r4, r4, LSL #2;    @ Multiplica por 4 para indexar por palavra
   LDR r5, [r10, r4]   @ r5=array[i(N+1)]

   ADD r6, r5, r6;      @ S+=array[i(N+1)]

   ADD r3, r3, #1;      @ i++
   B checkPrimeDiagonal


calcularConstante:
   MUL r2, r0, r0;    @ r2=N²
   ADD r7, r2, #1;    @ r7=N²+1
   MUL r8, r7, r0;    @ r8=N(N²+1)
   MOV r2, r8, LSR #1;    @ r2=N(N²+1)/2
   BX lr;
fim:
   SWI 0x123456;

quadrado:
   .word 0x02, 0x07, 0x06, 0x09, 0x05, 0x01, 0x04, 0x03, 0x08;
   .align 3
   @.word 16, 3, 2, 13, 5, 10, 11, 8, 9, 6, 7, 12, 4, 15, 13, 1;
ehmagico:
   .word 0x0;
```

6. Exercício 6.5.4

```
@ Exercicio 6.5.4 do livro
@ Para debugar este codigo:
@ gcc more_stacks.s && gdb a.out
   .text
   .globl    main
main:
   mov r2, #1              @ 1 = byte, 2 = half-word, 4 = word
   ldr r1, =0x12345678    @ valor a ser armazenado
   mov r4, #1

   cmp r2, #2
   bllt byte       @ adiciona byte
   bleq hword             @ adiciona half-word
   blgt word     @ adiciona word

   swi 0x0

byte:
   strb r1, [sp, r4, lsl #2]!    @ escreve byte no stack
   mov pc, lr

hword:
   strh r1, [sp, r4, lsl #2]!    @ escreve hword no stack
   mov pc, lr

word:
   str r1, [sp, r4, lsl #2]!    @ escreve word no stack
   mov pc, lr
```