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

Aula 5

10 de junho de 2021

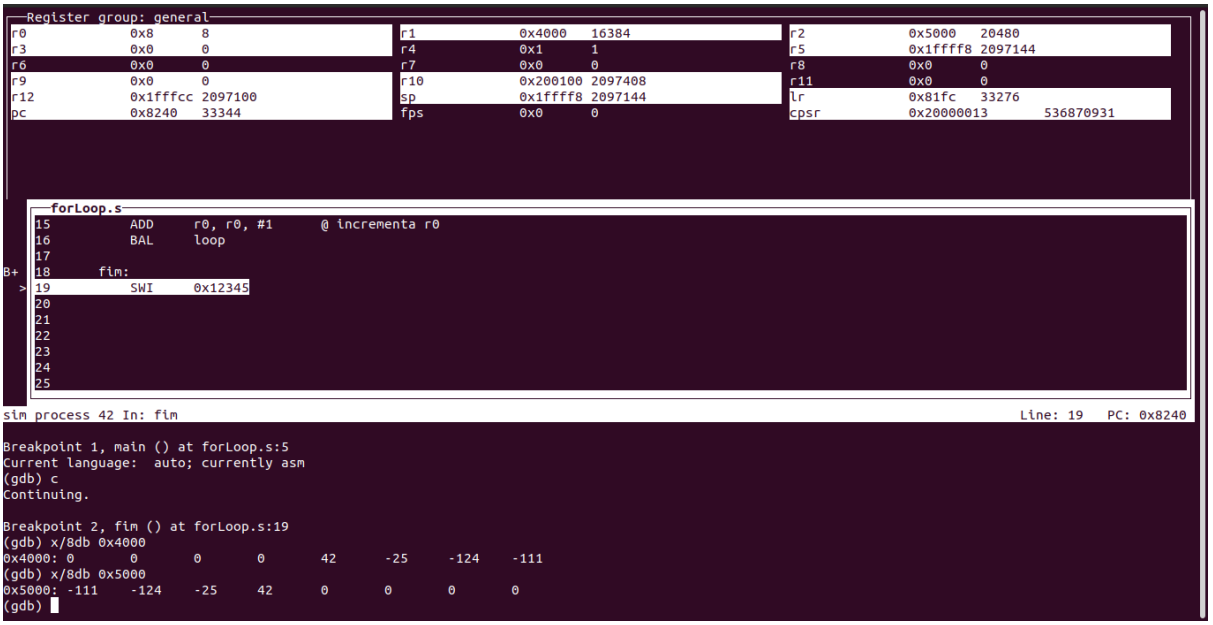
GRUPO 10

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

Sumário

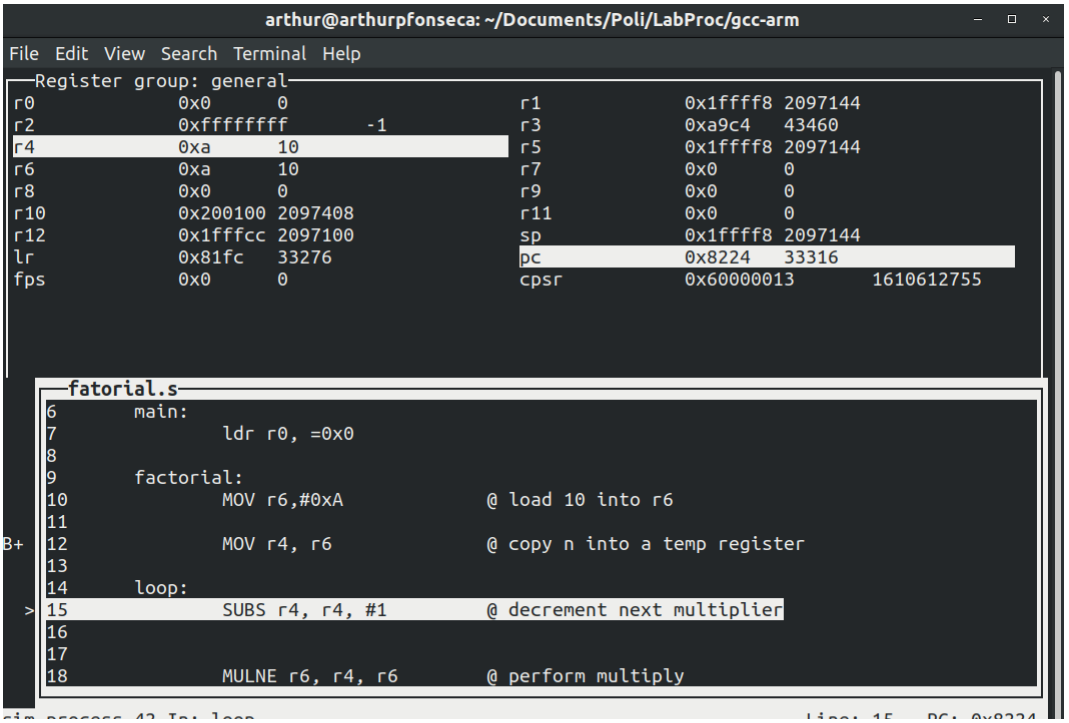
Exercício 5.5.1 - For Loop

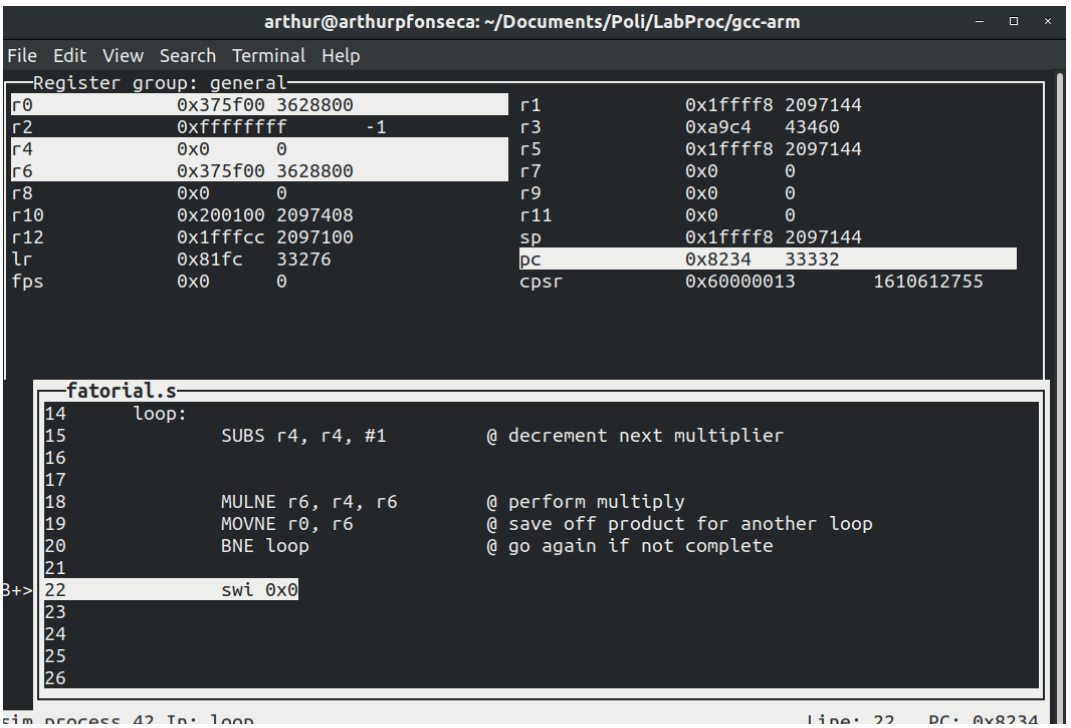
No print, é possível observar que o vetor b em 0x5000 recebeu os elementos do vetor a em 0x4000 em ordem inversa.



Exercício 5.5.2 - Factorial Calculation

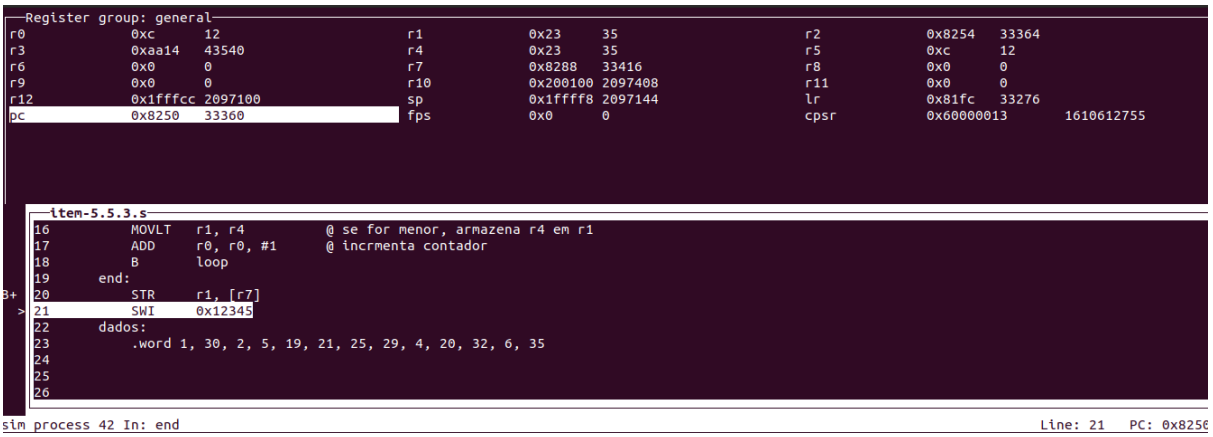
Observamos o resultado de (r6)! em r0. Na 2ª imagem, r0 = 10! = 3628800.





Exercício 5.5.3 - Find Maximum Value

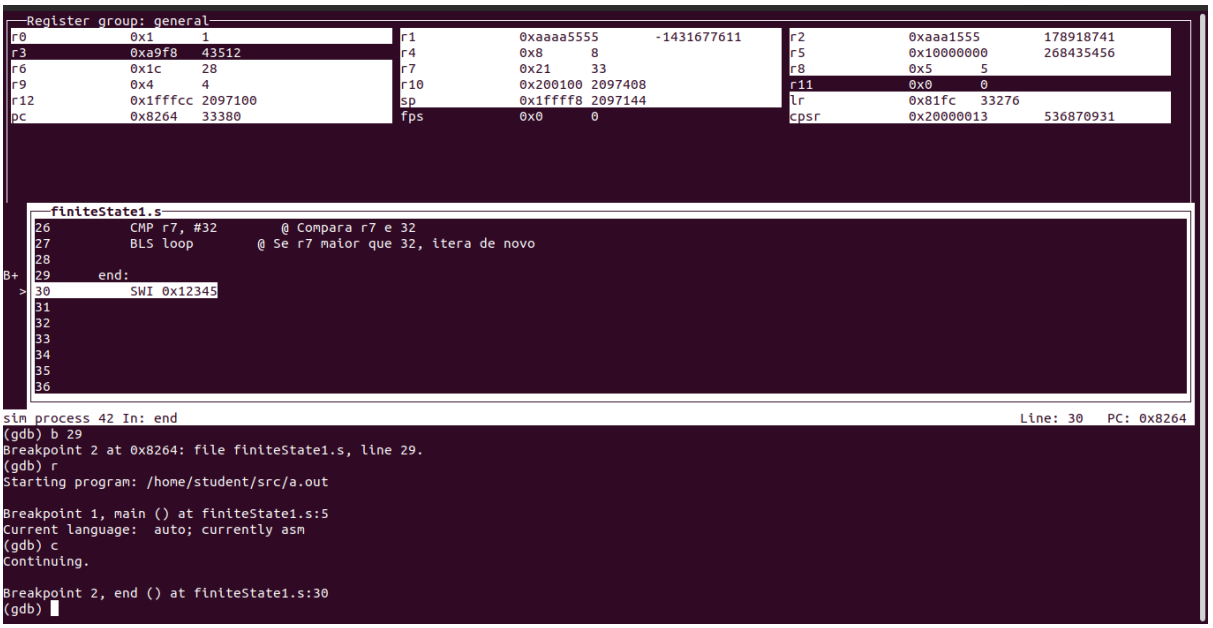
Observa-se que o valor 35 foi armazenado em *r1* ao final da execução, que é o maior valor entre a lista de 13 inteiros declarada.



Exercício 5.5.4 - Finite State Machine

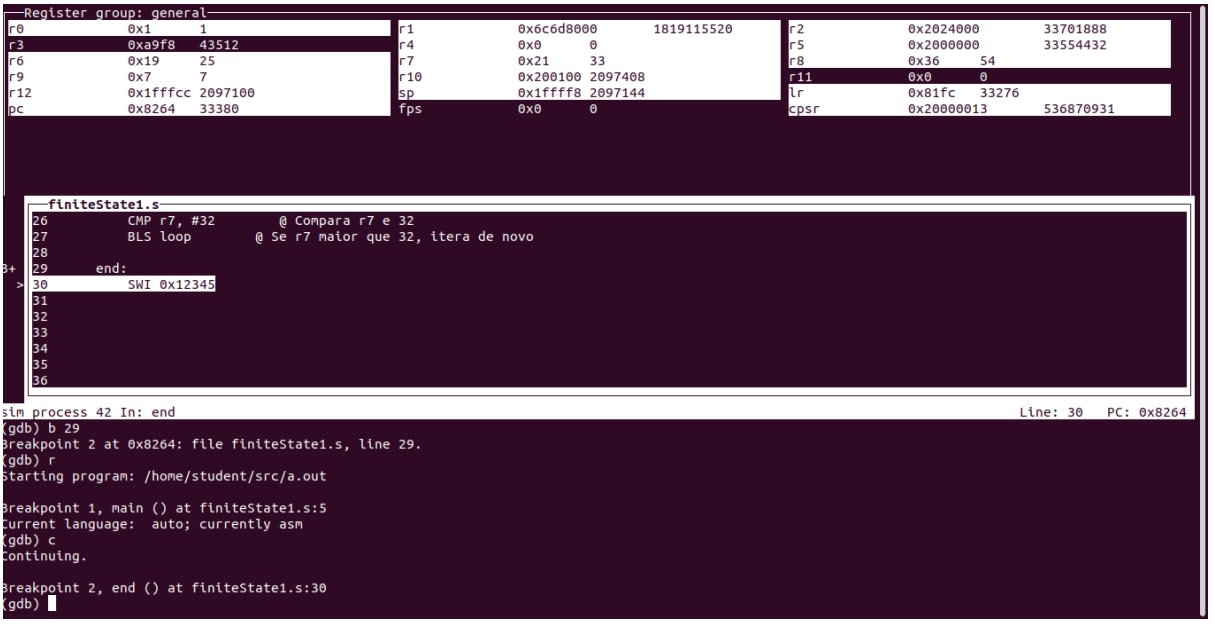
No sub exercício 1:

Para o input r1 = *0xAAAA5555* e padrão r8 = 0x*0101*, o resultado foi como esperado: r2 = *0x0AAA1555.*



No sub exercício 2:

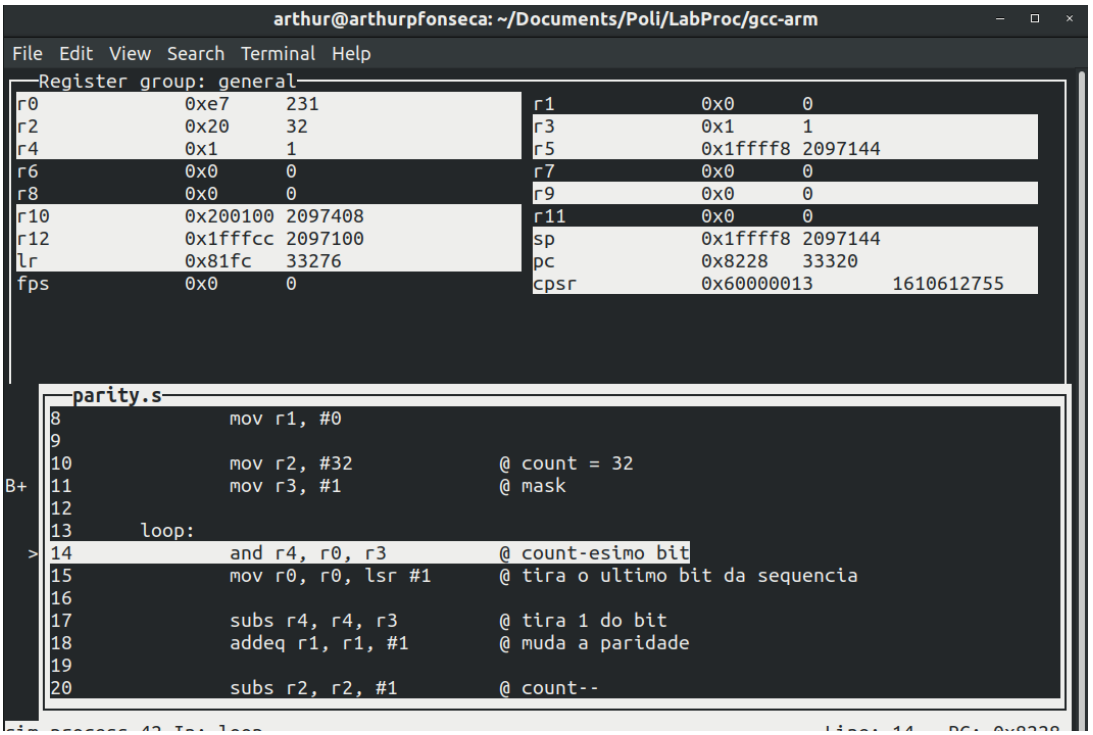
Para o input r1 = *0x6C6D8000* e padrão r8 = 0x*0110110*, o resultado foi como esperado: r2 = *0x2024000.*

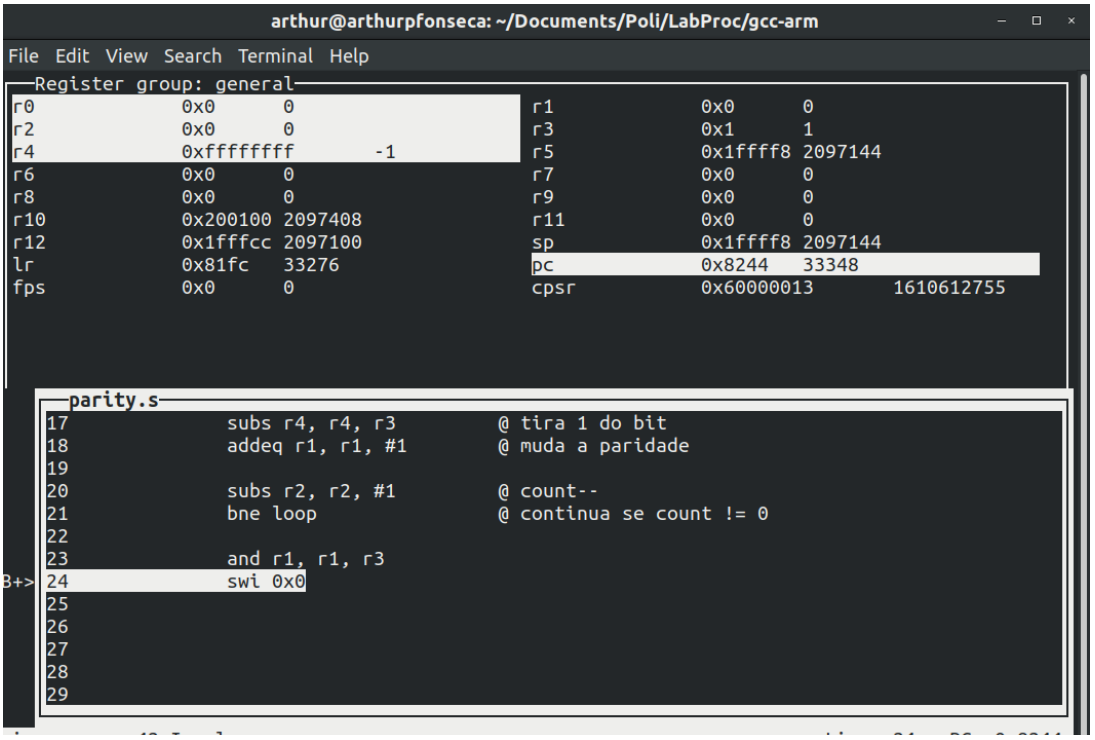


Exercício 5.5.5 - Sequential Parity Checker

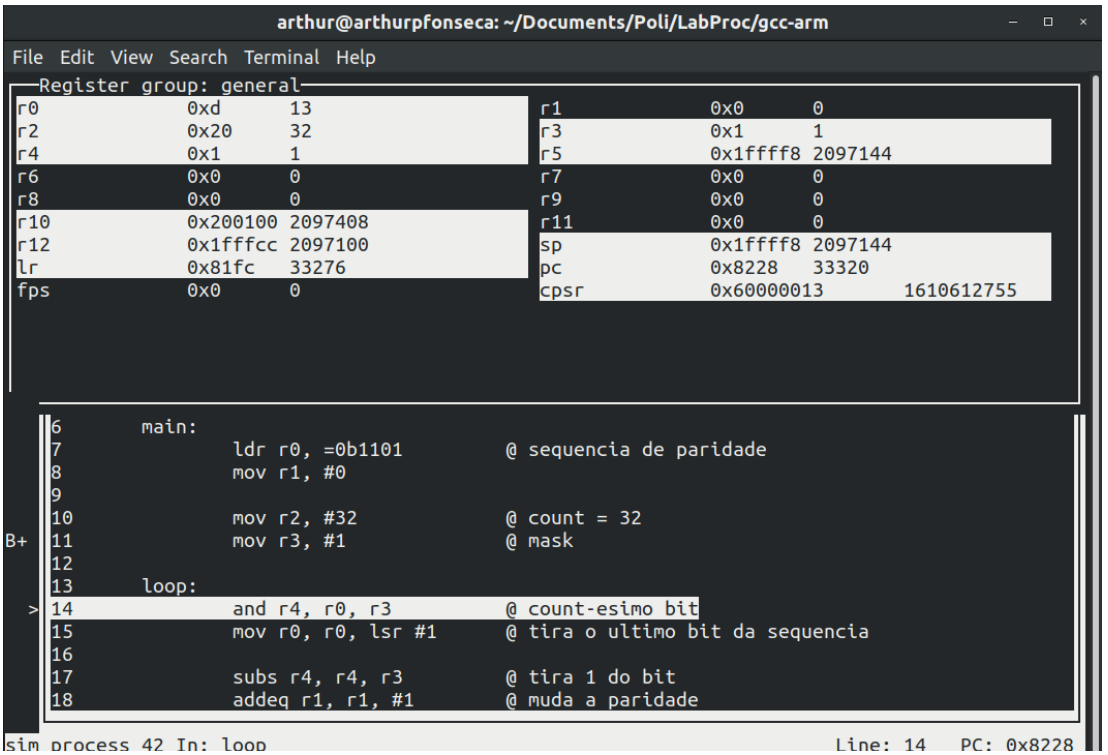
A sequência a ser analisada fica no registrador *r0* e o resultado, no *r1*.

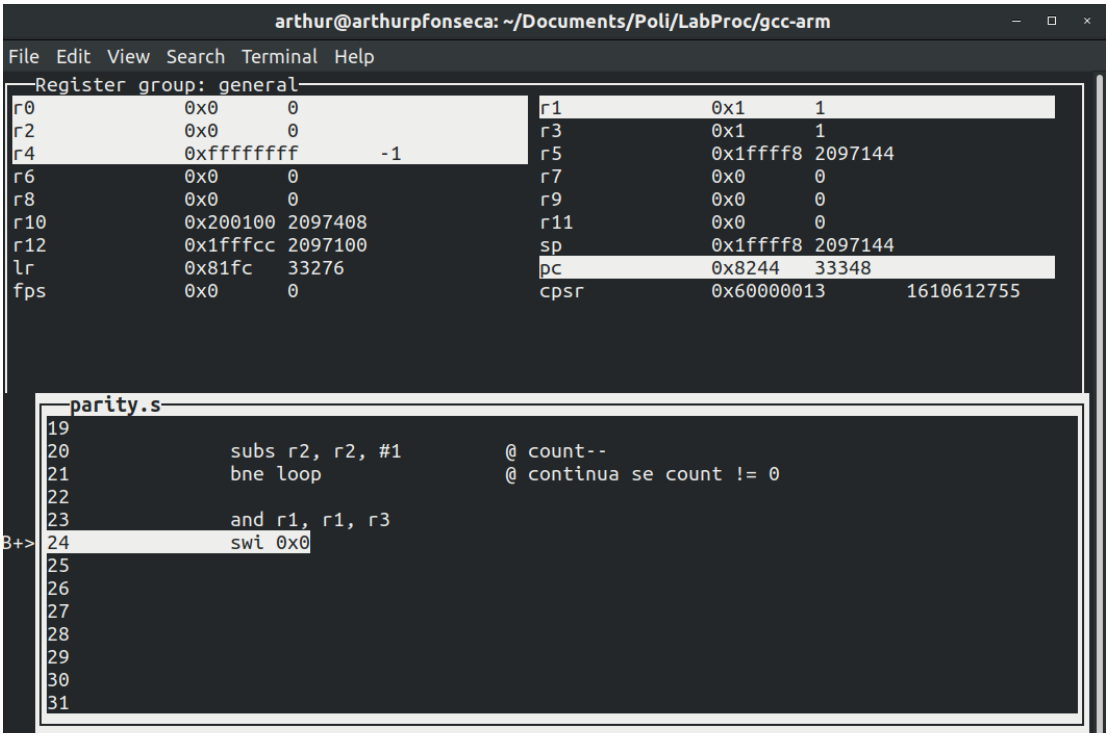
Primeiramente, uma sequência de paridade par, 0x00e7 (11100111):





Também, o resultado com paridade ímpar 0xd (1101):





Apêndice

Exercício 5.5.1

```
@ Exercicio 5.5.1 do livro
@ Para debugar este codigo:
@ gcc forloop.s && gdb a.out
   .text
   .globl main

main:
   LDR    r0, =0         @ contador inicia em 0
   LDR    r1, =0x4000    @ r1 - array a
   LDR    r2, =0x5000    @ r2 - array b

loop:
   CMP    r0, #7         @ compara r0 com 7
   BHI    fim            @ se for maior, acaba o loop
   RSB    r3, r0, #7     @ r3 = 7 - i
   LDRB   r4, [r1, r3]   @ carrega b[r3] ou seja b[7-1]
   STRB   r4, [r2, r0]   @ armazena em a[r0] ou seja a[i]
   ADD    r0, r0, #1     @ incrementa r0
   BAL    loop

fim:
   SWI    0x12345
```

Exercício 5.5.2

```
@ Exercicio 5.5.2 do livro
@ Para debugar este codigo:
@ gcc fatorial.s && gdb a.out
        .text
        .globl  main
main:
        ldr r0, =0x0

factorial:
        MOV r6,#0xA         @ load 10 into r6

        MOV r4, r6          @ copy n into a temp register

loop:
        SUBS r4, r4, #1     @ decrement next multiplier


        MULNE r6, r4, r6    @ perform multiply
        MOVNE r0, r6        @ save off product for another loop
        BNE loop            @ go again if not complete

        swi 0x0
```

Exercício 5.5.3

```
@ Exercicio 5.5.3 do livro
@ Para debugar este codigo:
@ gcc maxval.s && gdb a.out
    .text
    .globl main

main:
    ADR     r2, dados       @ carrega array de inteiros em r2
    MOV     r5, #13         @ salva tamanho em r5
    SUB     r5, r5, #1      @ pega valor do ultimo indice com tamanho -1
    MOV     r7, r2          @ guarga endereço em r7
    LDR     r1, [r7], #4    @ r1 salva o maior elemento, e inicia com o primeiro
    MOV     r0, #0          @ contador começa em 0
loop:
    CMP     r0, r5          @ compara contador com valor do ultimo indice
    BGE     end             @ se maior, sai
    LDR     r4, [r7], #4    @ pega o próximo elemento de r7 e guarga em r4
    CMP     r1, r4          @ compara r1 com r4
    MOVLT   r1, r4          @ se for menor, armazena r4 em r1
    ADD     r0, r0, #1      @ incrmenta contador
    B       loop
end:
    STR     r1, [r7]
    SWI     0x12345
dados:
    .word 1, 30, 2, 5, 19, 21, 25, 29, 4, 20, 32, 6, 35
```

Exercício 5.5.4

1

```
@ Exercicio 5.5.4.1 do livro
@ Para debugar este codigo:
@ gcc finiteState1.s && gdb a.out
        .text
        .globl main

main:
   MOV r8, #0x5       @ = 0101
   MOV r9, #0x4       @ Tamanho
   LDR r1, =0xAAAA5555    @ = 10101010101010100101010101010101
   MOV r2, #0x0       @ 0 inicial, espera-se
00001010101010100001010101010101 ou AAA1555 na saída
   MOV r3, r1

begin:
   MOV r5, #0x1       @ r5 = vetor de adicao
   MOV r6, #32        @ r6 = 32
   SUB r6, r6, r9    @ r6 = 32 - r9
   MOV r7, #0x1       @ r7 = contador
   MOV r2, #0x0       @ r2 = resultado
   MOV r5, r5, ROR r9    @ Rotaciona r5 pelo tamanho de Y

loop:
   MOV r4, r3, LSR r6    @ Shifta r3 por r6 vezes e guarda em r4
   CMP r4, r8        @ Compara r4 com r8
   ADDEQ r2, r2, r5    @ Se r4 e r8 iguais, r5 + r2
   ADD r7, r7, #1    @ r7++
   MOV r5, r5, ROR #1    @ Rotaciona o vetor de adicao
   MOV r3, r3, LSL #1    @ Shifta X
   CMP r7, #32        @ Compara r7 e 32
   BLS loop        @ Se r7 maior que 32, itera de novo

end:
   SWI 0x12345
```

2

```
@ Exercicio 5.5.4.2 do livro
@ Para debugar este codigo:
@ gcc finiteState2.s && gdb a.out
        .text
        .globl main

main:
   MOV r8, #0x36          @ padrao procurado 0110110
   MOV r9, #0x7           @ Tamanho do padrao
```

```
   LDR r1, =0x6c6d8000     @ palavra onde será feita a busca
   MOV r2, #0x0            @ registrador resultado
   MOV r3, r1              @ auxiliar da palavra

begin:
   MOV r5, #0x1            @ r5 flag de match
   MOV r6, #32             @ r6 shift maximo da flag de match
   SUB r6, r6, r9          @ r6 recebe 32 menos o tamanho do padrao
   MOV r7, #0x1            @ r7 = contador
   MOV r5, r5, ROR r9      @ Rotaciona flag de match pelo tamanho de Y ->
equivalente a shiftar para alinhar o padrao ao comeco da palavra

loop:
   MOV r4, r3, LSR r6      @ Guarda em r4 o pedaço da palavra a ser
comparado

   CMP r4, r8              @ Compara r4 com r8 (pedaço da palavra com padrao)
   ADDEQ r2, r2, r5        @ Se o padrão corresponder ao pedaço, adiciona
adiciona flag de match ao resultado
   MOV r5, r5, ROR #1      @ Rotaciona a flag de match em 1 bit p direita
   MOV r3, r3, LSL #1      @ Shifta palavra em 1 bit pra esquerda

   ADD r7, r7, #1          @ contador++
   CMP r7, #32             @ Compara contador e 32
   BLS loop                @ Se contador maior que 32, itera de novo

end:
   SWI 0x12345
```

Exercício 5.5.5

```
@ Exercicio 5.5.5 do livro
@ Para debugar este codigo:
@ gcc parity.s && gdb a.out
        .text
        .globl  main
main:
        ldr r0, =0b11100111   @ sequencia de paridade
        mov r1, #0

        mov r2, #32           @ count = 32
        mov r3, #1            @ mask

loop:
        and r4, r0, r3        @ count-esimo bit
        mov r0, r0, lsr #1    @ tira o ultimo bit da sequencia

        subs r4, r4, r3               @ tira 1 do bit
        addeq r1, r1, #1      @ muda a paridade

        subs r2, r2, #1               @ count--
        bne loop              @ continua se count != 0

        and r1, r1, r3
        swi 0x0
```