Atividades - Assembly RISC-V

- Autor: Rafael Grossi

Atividade 1

- Output

```
Teste de string
AAAAAAAAAAAAAAAAAAAAHello World
AAAAAAAAAAAAAAAAAA
Hello World
```

O programa original preenche `str2` com o caractere 'A' (65), mas não adiciona o terminador nulo (`\0`) imediatamente. Como `str2` e `str3` estão adjacentes na memória, a primeira impressão de `str2` "vaza" e imprime também o conteúdo de `str3` ("Hello World") até encontrar um nulo. Na segunda parte, o código insere o terminador nulo na última posição, corrigindo a impressão.

Atividade 2

```
.data
source: .string "Copy this string."
dest:   .space 30

.text
main:
    # Print
    li a0, 4
    la a1, source
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    # Load arguments for strcpy
    la a0, dest
    la a1, source
```

```
    call strcpy

    # Print
    li a0, 4
    la a1, dest
    ecall

    li a0, 10
    ecall

# char * strcpy(char *s1, char *s2)
# a0 = destination (s1), a1 = source (s2)
strcpy:
    mv t0, a0

strcpy_loop:
    # Load byte from source
    lbu t1, 0(a1)
    sb t1, 0(t0)
    # Check if null terminator
    beq t1, zero, strcpy_end

    # Increment pointers
    addi a1, a1, 1
    addi t0, t0, 1
    j strcpy_loop

strcpy_end:
    ret
```

Atividade 3

```
.data
buffer: .space 100
prompt: .string "Digite algo: "
written: .string "Digitado: "

.text
main:
    # Load prompt
    li a0, 4
    la a1, prompt
    ecall

    # this was bugging without this newline :)
```

```
    li a0, 11
    li a1, 13
    ecall

    # Load buffer
    la a0, buffer
    call gets

    # Print
    li a0, 4
    la a1, written
    ecall

    # Print read
    li a0, 4
    la a1, buffer
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    li a0, 10
    ecall

# char * gets(char *s)
gets:
    mv t0, a0

    li a0, 0x130
    ecall

gets_poll:
    # Check for char
    li a0, 0x131
    ecall

    # Check return status in a0
    # 0x00 = All read (finished/empty for now)
    # 0x01 = Nothing yet
    # 0x02 = Char available in a1

    li t1, 2

    beq a0, t1, gets_read
```

```
    beq a0, zero, gets_end_null

    # Keep polling
    j gets_poll

gets_read:
    # a1, c = char
    # Check for newline (ASCII 10) to stop
    li t2, 10
    beq a1, t2, gets_end

    # Store in buffer
    sb a1, 0(t0)

    addi t0, t0, 1
    j gets_poll

gets_end:
    # Add null terminator
    sb zero, 0(t0)
    ret

gets_end_null:
    # terminate
    sb zero, 0(t0)
    ret
```

Atividade 4

```
.data
buffer_safe: .space 10
prompt_safe: .string "Digite (max 9 chars): "
written: .string "Digitado (max 9 chars): "

.text
main:
    li a0, 4
    la a1, prompt_safe
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall
```

```
    la a0, buffer_safe
    # Max size N
    li a1, 10
    call fgets

    li a0, 4
    la a1, written
    ecall

    li a0, 4
    la a1, buffer_safe
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    li a0, 10
    ecall

# char * fgets(char *s, int N)
# a0 = buffer, a1 = N (size)
fgets:
    mv t0, a0
    # t3 = Max characters to read (N - 1) to leave room for null
    addi t3, a1, -1
    # Counter for chars read
    li t4, 0

    # Enable keyboard
    li a0, 0x130
    ecall

fgets_poll:
    # Check limit
    bge t4, t3, fgets_end

    li a0, 0x131
    ecall

    li t1, 2
    beq a0, t1, fgets_read
    beq a0, zero, fgets_end
    j fgets_poll
```

```
fgets_read:
    # Check newline
    li t2, 10
    beq a1, t2, fgets_end

    # Store char
    sb a1, 0(t0)

    # Increment pointer and counter
    addi t0, t0, 1
    addi t4, t4, 1
    j fgets_poll

fgets_end:
    # Null terminate
    sb zero, 0(t0)
    ret
```

Atividade 5

```
.data
strA: .string "Ola "
strB: .string "Mundo"
strComp1: .string "Abc"
strComp2: .string "Abd"
res_len: .string "Tamanho strA: "
res_lenb: .string " | Tamanho strB: "

res_cmp: .string "\nComparacao: "
res_cat: .string "\nConcatenacao: "

.text
main:
    # strlen A
    li a0, 4
    la a1, res_len
    ecall

    la a0, strA
    call strlen
    # Print result (integer)
    mv a1, a0
    li a0, 1
    ecall
```

```
    # strlen B
    li a0, 4
    la a1, res_lenb
    ecall

    la a0, strB
    call strlen
    # Print result (integer)
    mv a1, a0
    li a0, 1
    ecall

    # strcmp(a, b)
    li a0, 4
    la a1, res_cmp
    ecall

    la a0, strComp1
    la a1, strComp2
    call strcmp
    # Print result (negative means s1 < s2)
    mv a1, a0
    li a0, 1
    ecall

    # srtcat(a, b)
    li a0, 4
    la a1, res_cat
    ecall

    # We concat strB into strA
    # assumes we actually have enough space
    la a0, strA
    la a1, strB
    call strcat

    # Print result string
    li a0, 4
    la a1, strA
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall
```

```
    li a0, 10
    ecall

# int strlen(char *s)
strlen:
    li t0, 0
strlen_loop:
    add t1, a0, t0
    lbu t2, 0(t1)
    beq t2, zero, strlen_end
    addi t0, t0, 1
    j strlen_loop
strlen_end:
    mv a0, t0
    ret

# int strcmp(char *s1, char *s2)
strcmp:
strcmp_loop:
    lbu t0, 0(a0)
    lbu t1, 0(a1)
    # Compare bytes
    sub t2, t0, t1
    bne t2, zero, strcmp_diff

    # If both are zero (end of string), strings are equal
    beq t0, zero, strcmp_equal

    addi a0, a0, 1
    addi a1, a1, 1
    j strcmp_loop

strcmp_diff:
    mv a0, t2
    ret
strcmp_equal:
    li a0, 0
    ret

# char * strcat(char *s1, char *s2)
strcat:
    mv t0, a0

strcat_find_end:
    lbu t1, 0(t0)
    beq t1, zero, strcat_copy
```

```
    addi t0, t0, 1
    j strcat_find_end

strcat_copy:
    lbu t1, 0(a1)
    sb t1, 0(t0)
    beq t1, zero, strcat_end
    addi t0, t0, 1
    addi a1, a1, 1
    j strcat_copy

strcat_end:
    ret
```

Atividade 6

```
.data
prompt: .string "Digite algo: "
input_rev: .space 50
msg_rev: .string "Invertida: "

.text
main:
    # Load prompt
    li a0, 4
    la a1, prompt
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    # Use our previous fgets to read safely
    la a0, input_rev
    li a1, 50
    call fgets

    la a0, input_rev
    call strrev

    li a0, 4
    la a1, msg_rev
    ecall
```

```
    li a0, 4
    la a1, input_rev
    ecall

    # newline
    li a0, 11
    li a1, 13
    ecall

    li a0, 10
    ecall

fgets:
    mv t0, a0
    addi t3, a1, -1
    li t4, 0
    li a0, 0x130
    ecall
fgets_poll:
    bge t4, t3, fgets_end
    li a0, 0x131
    ecall
    li t1, 2
    beq a0, t1, fgets_read
    beq a0, zero, fgets_end
    j fgets_poll
fgets_read:
    li t2, 10
    beq a1, t2, fgets_end
    sb a1, 0(t0)
    addi t0, t0, 1
    addi t4, t4, 1
    j fgets_poll
fgets_end:
    sb zero, 0(t0)
    ret

strlen:
    li t0, 0
strlen_loop:
    add t1, a0, t0
    lbu t2, 0(t1)
    beq t2, zero, strlen_end
    addi t0, t0, 1
    j strlen_loop
strlen_end:
```

```
    mv a0, t0
    ret

# void strrev(char *s)
strrev:
    # Save return address and s0
    addi sp, sp, -8
    sw ra, 4(sp)
    sw s0, 0(sp)

    # start address in s0
    mv s0, a0

    # Calculate length: strlen(s) => a0
    call strlen

    mv t1, s0
    add t2, s0, a0
    addi t2, t2, -1

strrev_loop:
    bge t1, t2, strrev_end

    # Load bytes
    lbu t3, 0(t1)
    lbu t4, 0(t2)

    # Swap
    sb t4, 0(t1)
    sb t3, 0(t2)

    # Move pointers
    addi t1, t1, 1
    addi t2, t2, -1
    j strrev_loop

strrev_end:
    lw s0, 0(sp)
    lw ra, 4(sp)
    addi sp, sp, 8
    ret
```