Input Output Handling

Character Data

- Byte-encoded character sets
  - ASCII: 128 characters
    - 95 graphic, 33 control
  - Latin-1: 256 characters
    - ASCII, +96 more graphic characters

Character Data

- Unicode: 32-bit character set
  - Used in Java, C++ wide characters, …
  - Most of the world's alphabets, plus symbols
  - UTF-8, UTF-16: variable-length encodings

Byte/Halfword Operations

- ARM byte load/store
  - String processing is a common case

```
LDRB r0, [sp,#0]  ; Read byte from source
STRB r0, [r10,#0] ; Write byte to destination
```

  - Sign extend to 32 bits

```
LDRSB ; Sign extends to fill leftmost 24 bits
```

Byte/Halfword Operations

- ARM halfword load/store

```
LDRH r0, [sp,#0]  ; Read halfword (16 bits) from source
STRH r0,[r12,#0] ; Write halfword (16 bits) to destination
```

  - Sign extend to 32 bits

```
LDRSH ; Sign extends to fill leftmost 16 bits
```

String Copy Example

- C code (naïve):
  - Null-terminated string

```
void strcpy (char x[], char y[]){
  int i;
  i = 0;
  while ((x[i]=y[i])!='\0')
    i++;
}
```

  - *Addresses of x, y in registers r0, r1*
  - *i in register r4*

String Copy Example

- ARM code:

```
strcpy:
    SUB sp,sp, #4       ; adjust stack for 1 item
    STR r4,[sp,#0]      ; save r4
    MOV r4,#0           ; i = 0
L1: ADD r2,r4,r1        ; addr of y[i] in r2
    LDRB    r3, [r2, #0] ; r3 = y[i]
    ADD r12,r4,r0 ;     ; Addr of x[i] in r12
    STRB r3 [r12, #0]  ; x[i] = y[i]
    CMP r3,#0
        BEQ L2          ; exit loop if y[i] == 0
    ADD r4,r4,#1        ; i = i + 1
    B L1                ; next iteration of loop
L2: LDR r4, [sp,#0]     ; restore saved r4
    ADD sp,sp, #4       ; pop 1 item from stack
    MOV pc,lr           ; return
```

scanf and printf (example01.s)

- Read a number fron stdin and print to the stdout

```
sub     sp, sp, #4              @allocate stack for input
ldr r0, =formats            @scanf to get an integer
mov     r1, sp
bl  scanf                   @scanf("%d",sp)
ldr r1, [sp,#0]             @copy from stack to register
add     sp, sp, #4              @release stack
ldr r0, =formatp            @format for printf
bl  printf                  @printf

        .data       @ data memory
formats: .asciz "%d“
formatp: .asciz "The number is %d\n“
```

- The .asciz directive accepts string literals as arguments. String literal are a sequence characters in double quotes. The string literals are assembled into consecutive memory locations. The assembler automatically inserts a null character (\0 character) after each string.

Lab 4

- Write a program named exxyyy.s which reverses input strings as shown in the examples below.
  - For any negative input :
  - For 0 input :

```
hasindu@tesla:~/tmp$ ./a.out
Enter the number of strings :
-1
Invalid number
```

```
hasindu@tesla:~/tmp$ ./a.out
Enter the number of strings :
0
```

Lab 4

- For any other :

```
hasindu@tesla:~/tmp$ ./a.out
Enter the number of strings :
3
Enter the input string 0 :
hey this is interesting!
Output string 0 is :
!gnitseretni si siht yeh
Enter the input string 1 :
another sentence.
Output string 1 is :
.ecnetnes rehtona
Enter the input string 2 :
abc123 is a bad password :P
Output string 2 is :
P: drowssap dab a si 321cba
```

- You are allowed to use any library function
- Assume any string is less than 200 chars