1. # "Hello World" in MIPS assembly

```
# hello.asm


        .text
        .globl  main
main:
        li      $v0,4           # code for print_str
        la      $a0, msg        # point to string
        syscall
        li      $v0,10          # code for exit
        syscall

        .data
msg:    .asciiz "Hello World!\n"
```

Explanation :

```
      # All program code is placed after the
      # .text assembler directive
      .text

      # Declare main as a global function
      .globl   main

# The label 'main' represents the starting point
main:
      # Run the print_string syscall which has code 4
      li       $v0,4            # Code for syscall: print_string
      la       $a0, msg         # Pointer to string (load the address of msg)
      syscall
      li       $v0,10           # Code for syscall: exit
      syscall

      # All memory structures are placed after the
      # .data assembler directive
      .data

      # The .asciiz assembler directive creates
      # an ASCII string in memory terminated by
      # the null character. Note that strings are
      # surrounded by double-quotes
msg:  .asciiz  "Hello World!\n"
```

2. # Simple input/output in MIIPS assembly

```
.text
        .globl  main
main:
        li      $v0,4           # output msg1
        la      $a0, msg1
        syscall
```

```
        li      $v0,5           # input A and save
        syscall
        move    $t0,$v0
        li      $v0,4           # output msg2
        la      $a0, msg2
        syscall
        li      $v0,5           # input B and save
        syscall
        move    $t1,$v0
        add     $t0, $t0, $t1   # A = A + B
        li      $v0, 4          # output msg3
        la      $a0, msg3
        syscall
        li      $v0,1           # output sum
        move    $a0, $t0
        syscall
        li      $v0,4           # output lf
        la      $a0, cflf
        syscall

        li      $v0,10          # exit
        syscall
        .data
msg1:   .asciiz "\nEnter A:   "
msg2:   .asciiz "\nEnter B:   "
msg3:   .asciiz "\nA + B =  "
cflf:   .asciiz "\n"
```

Explanation :

```
        # Start .text segment (program code)
        .text

        .globl  main
main:
        # Print string msg1
        li      $v0,4           # print_string syscall code = 4
        la      $a0, msg1       # load the address of msg
        syscall

        # Get input A from user and save
        li      $v0,5           # read_int syscall code = 5
        syscall
        move    $t0,$v0                 # syscall results returned in $v0

        # Print string msg2
        li      $v0,4           # print_string syscall code = 4
        la      $a0, msg2       # load the address of msg2
        syscall

        # Get input B from user and save
        li      $v0,5           # read_int syscall code = 5
        syscall
        move    $t1,$v0                 # syscall results returned in $v0

        # Math!
        add     $t0, $t0, $t1   # A = A + B
```

```
# Print string msg3
li      $v0, 4
la      $a0, msg3
syscall

# Print sum
li      $v0,1           # print_int syscall code = 1
move    $a0, $t0        # int to print must be loaded into $a0
syscall

# Print \n
li      $v0,4           # print_string syscall code = 4
la      $a0, newline
syscall

li      $v0,10          # exit
syscall

# Start .data segment (data!)
        .data
msg1:   .asciiz  "Enter A:   "
msg2:   .asciiz  "Enter B:   "
msg3:   .asciiz  "A + B = "
newline:   .asciiz       "\n"
```

3. A Simple Expression

C code:

```
i = N*N + 3*N
```

"Unoptimized":

(Note: There are some small disagreements in the syntax of assembler between SPIM, which is used in the book, and Cebollita, which is the tool we will be using. I have tried to follow the conventions of Cebollita here.)

```
lw      $t0, 4($gp)        # fetch N
mult    $t0, $t0, $t0      # N*N
lw      $t1, 4($gp)        # fetch N
ori     $t2, $zero, 3      # 3
mult    $t1, $t1, $t2      # 3*N
add     $t2, $t0, $t1      # N*N + 3*N
sw      $t2, 0($gp)        # i = ...
```

"Optimized":

```
lw      $t0, 4($gp)        # fetch N
add     $t1, $t0, $zero    # copy N to $t1
addi    $t1, $t1, 3        # N+3
mult    $t1, $t1, $t0      # N*(N+3)
sw      $t1, 0($gp)        # i = ...
```