

MODUL MIPS: Project 0

Oleh : Rico Tadjudin bersama Asisten Dosen
Pengantar Organisasi Komputer
Semester Genap 2022/2023

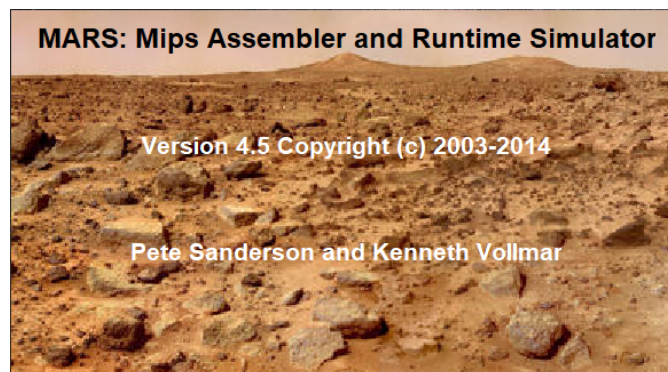
Apa itu MIPS?

MIPS (Microprocessor without Interlocked Pipelined Stages) adalah sebuah ISA yang berdasarkan RISC. Namun saat kita membicarakan MIPS pada lab POK, kita akan lebih banyak membicarakan tentang MIPS Assembly Language. Bahasa assembly ini akan mirip dengan bahasa-bahasa assembly lainnya, tetapi karena MIPS berdasarkan RISC, maka tentunya bahasanya kan lebih mudah dan simpel dari bahasa assembly lainnya.

Contoh alat yang menggunakan MIPS: PlayStation!

Apa itu MARS?

MARS (MIPS Assembler and Runtime Simulator) adalah sebuah IDE yang akan kita gunakan pada lab di POK. IDE ini khusus didesain untuk MIPS Assembly Language.



Tutorial program MARS dari sumber lain:

<https://courses.missouristate.edu/KenVollmar/mars/tutorial.htm>

[MARS MIPS simulator - Missouri State University](#)

Catatan: Untuk lab/tugas, gunakan MARS yang ada di scele!

referensi: slide "MIPS Module" oleh NADHIF ADYATMA PRAYOGA (1806205501), nadhif.adyatma@ui.ac.id

#POKoknyaPOK ❤️

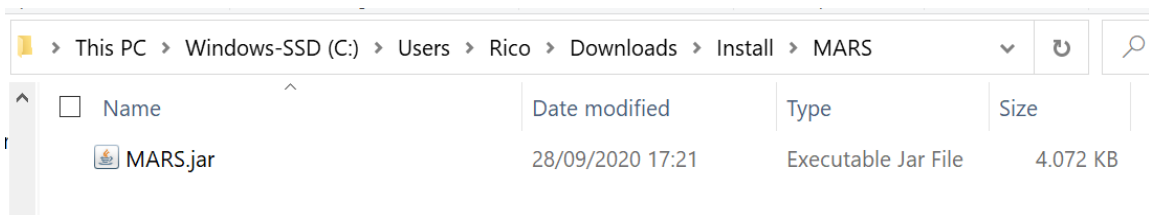
1. Membuka MARS.jar

Pertama-tama, download file jar MARS.jar dari scele. Jika Anda sudah mempunyai JRE, maka cukup membuka MARS.jar dengan klik dua kali. Jika tidak bisa dengan klik dua kali, jalankan perintah berikut di cmd **pada folder yang memiliki file MARS.jar**:

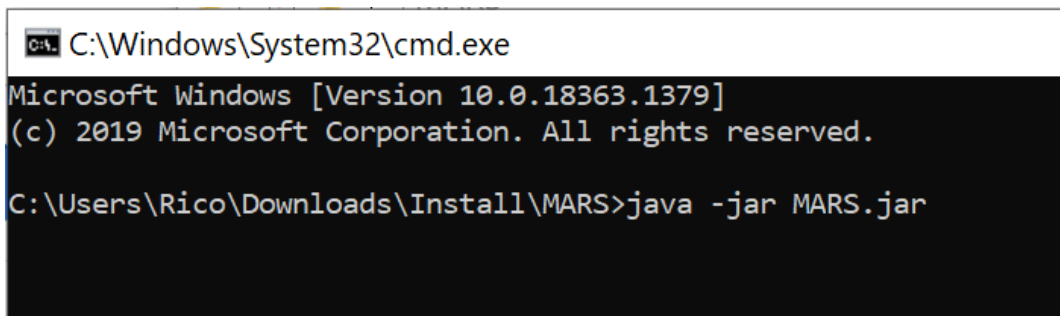
```
java -jar MARS.jar
```

Contoh (Hal yang sama berlaku juga pada macOS):

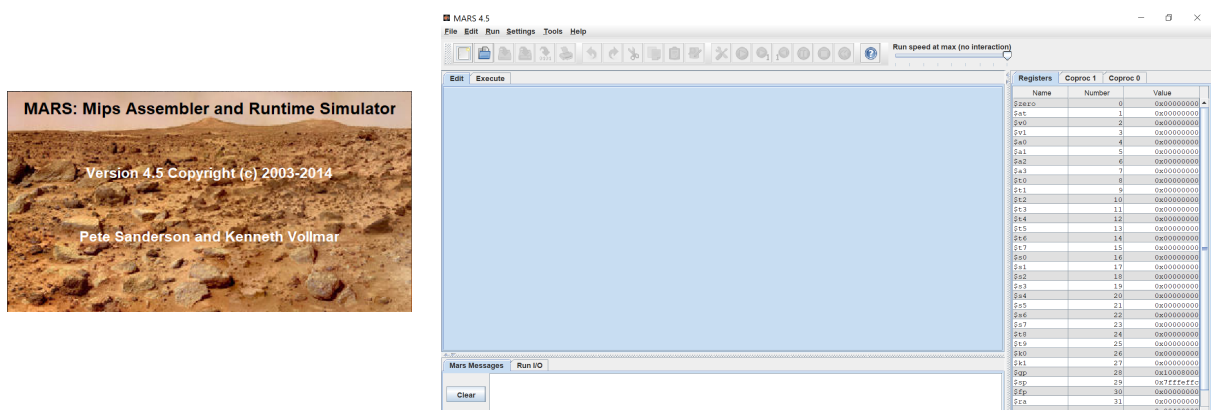
- File MARS.jar ada di direktori C:\Users\Rico\Downloads\Install\MARS (contoh direktori)
- **Gunakan direktori kalian masing-masing! (Tempat kalian menaruh file MARS.jar)**



- Maka pada cmd juga harus berada di direktori C:\Users\Rico\Downloads\Install\MARS



- Tuliskan perintah sesuai dengan gambar dan tekan ENTER.



- Jika Anda tidak bisa membuka MARS dengan mengklik dua kali, maka setiap kali Anda ingin membuka MARS harus menggunakan cara di atas.

Jika Ada Masalah

Jika Anda belum download dan install JRE/JDK, maka install terlebih dahulu JRE atau JDK (Java Development Kit) secara mandiri.

Jika setelah di-install secara mandiri masih tidak bisa menjalankan MARS.jar (terkadang ada masalah versi), maka download dan install langsung JRE yang ada di scele (JRE-8u281). Versi JRE tersebut seharusnya sudah dipastikan dapat menjalankan MARS.

Jika masih tidak bisa juga, hubungi salah satu asisten dosen melalui Discord POK.

2. Refresh Materi MIPS Assembly Language

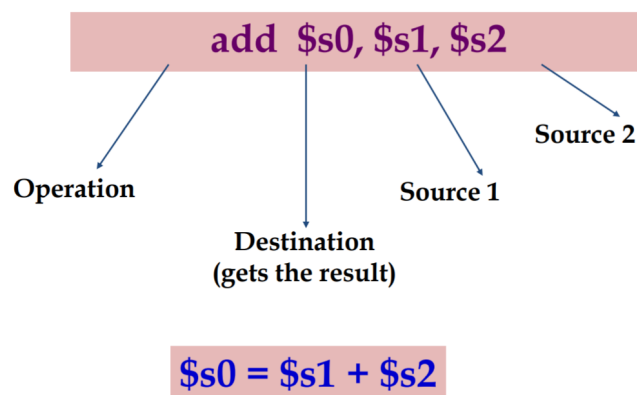
Register yang dipakai:

| Name | Register number | Usage |
|-----------|-----------------|--|
| \$zero | 0 | Constant value 0 |
| \$v0-\$v1 | 2-3 | Values for results and expression evaluation |
| \$a0-\$a3 | 4-7 | Arguments |
| \$t0-\$t7 | 8-15 | Temporaries |
| \$s0-\$s7 | 16-23 | Program variables |

| Name | Register number | Usage |
|-----------|-----------------|------------------|
| \$t8-\$t9 | 24-25 | More temporaries |
| \$gp | 28 | Global pointer |
| \$sp | 29 | Stack pointer |
| \$fp | 30 | Frame pointer |
| \$ra | 31 | Return address |

\$at (register 1) is reserved for the assembler.

\$k0-\$k1 (registers 26-27) are reserved for the operation system.



Berbagai macam instruksi MIPS bisa dilihat di [MIPS Green Sheet](#).

- **Moving/Memindahkan value dari \$v0 ke \$t0:**

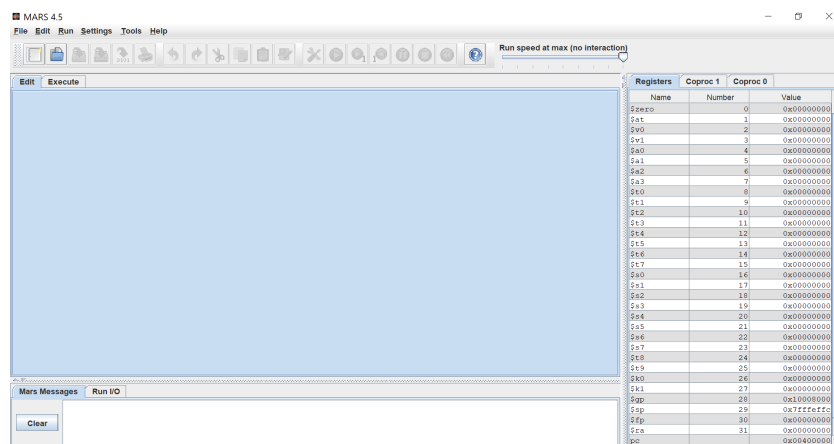
add \$t0, \$v0, \$zero

Sama saja dengan, $\$t0 = \$v0 + \$zero$.
 $\$t0 = \$v0$

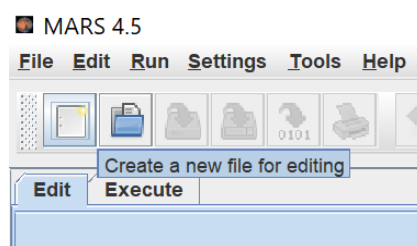
- **Tanda # pada MARS menandakan baris komentar/comment.**

3. Membuat Program Baru dalam MARS

Tampilan pertama saat membuka MARS:

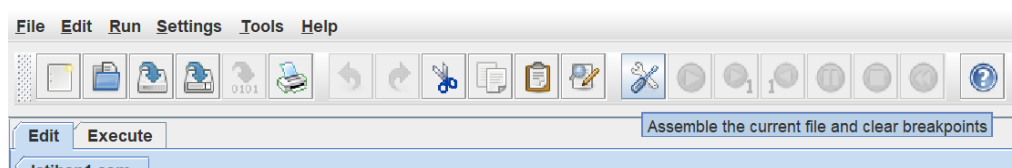


Klik "Create a new file for editing" untuk mulai membuat program

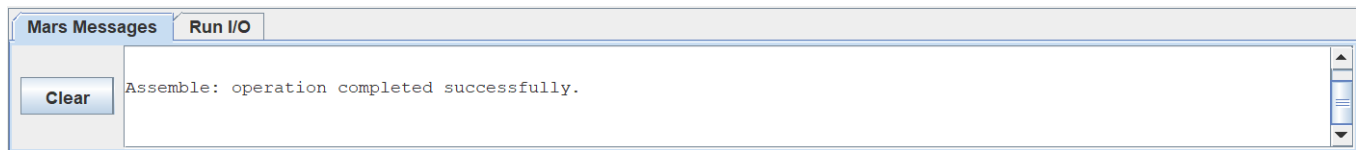


Setelah itu kalian dapat mulai menuliskan kode yang diinginkan.

Jika ingin menjalankan program, **simpan dulu file** lalu klik pada tombol "Assemble the current file and clear breakpoints"



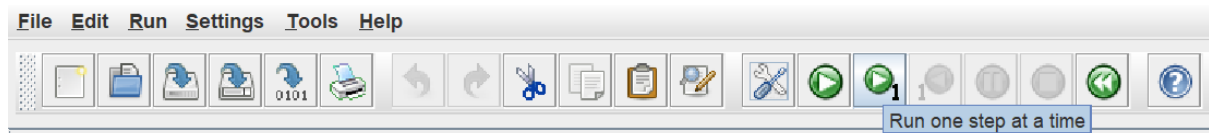
Jika program Anda berhasil di-assemble, maka akan muncul tulisan seperti ini di bagian “Mars Messages” di bawah.



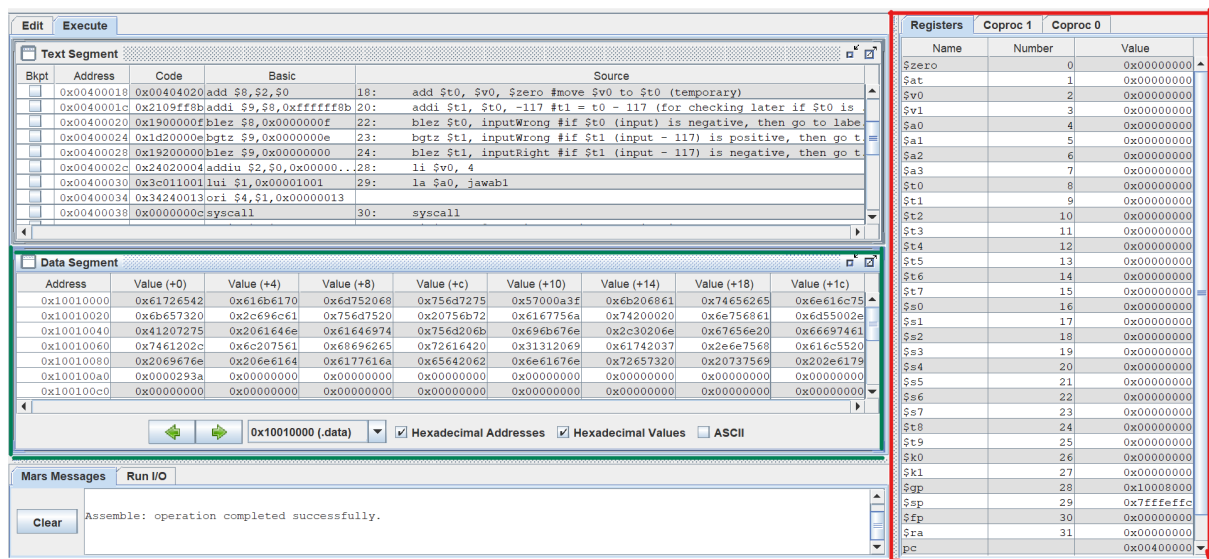
Klik “Run the current program” untuk menjalankan semua baris program sekaligus.



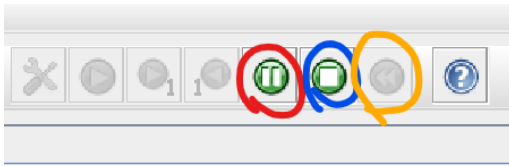
Klik “Run one step at a time” untuk menjalankan baris program satu per satu. Tombol ini dapat digunakan untuk debugging dengan melihat setiap baris program dan perubahan apa saja yang terjadi.



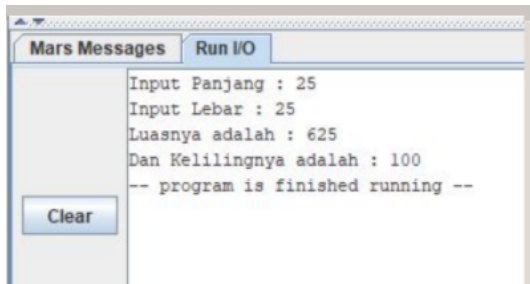
Bagian di kanan yang digarisi dengan **garis merah** adalah list dari register dan value yang ada di dalam register tersebut. Ada register dari \$zero (\$0) sampai \$ra (\$31). Pada kotak yang di bawah yang digarisi dengan **garis hijau**, adalah memory dari MIPS dan value-valuenya. Hal ini dapat digunakan untuk debugging.



Merah: pause jalannya program, **biru:** hentikan run program, **oranye:** reset atau mundur



Setelah program selesai berjalan akan ada pesan "-- program is finished running --"



4. Dasar-Dasar MIPS

Pada MIPS, ada dua bagian dalam programnya, yaitu bagian **data** dan **text**. Pada bagian data (.data) dapat dimasukkan variabel-variabel seperti string, integer, dan juga array. Pada bagian text (.text) dapat dimasukkan kode-kode program, fungsi, label, dan lain-lainnya.

```
1  .data
2  masukan data di sini
3
4  .text
5  masukan kode, fungsi, dan label disini
6
7
8
```

Syscall

Syscall digunakan untuk mengurus I/O ataupun untuk *exit* dari MIPS. Syscall memiliki berbagai fungsi berbeda-beda. Syscall akan melihat *service codes* pada register \$v0 untuk mengetahui apa yang harus dilakukan. Dan jika syscall tersebut membutuhkan argumen (seperti print string butuh argumen berupa string yang ingin diprint) maka argumen tersebut bisa dipindahkan ke register \$a0). Contoh pemanggilan Syscall dengan *service code* 4 dan 5 (4 dan 5 dimasukkan ke register \$v0 agar saat Syscall dipanggil, Syscall akan mengetahui perintah apa yang harus dilakukan):

```
.text
.globl main
main:

    li $v0, 4 #syscall 4: load instruction to print string
    la $a0, tanya #loads tanya to $a0 (anything in variable tanya will be printed)
    syscall #execute (print anything in tanya)

    li $v0, 5 #syscall 5: load instruction to read integer
    syscall #execute (read input as integer and save it to $v0)
```

3 baris pertama: Print String

2 baris di bawah: Meminta input integer

Beberapa *service code* dan gunanya untuk syscall:

| Service | Code | Arguments | Result |
|---------------|------|--------------------------|-------------------|
| print integer | 1 | \$a0 = value | (none) |
| print float | 2 | \$f12 = float value | (none) |
| print double | 3 | \$f12 = double value | (none) |
| print string | 4 | \$a0 = address of string | (none) |
| read integer | 5 | (none) | \$v0 = value read |

Selebihnya akan ada di file terpisah [Guide to MIPS.pdf](#).

Variabel

Untuk men-*define* sebuah variabel dapat dilakukan seperti ini (nama dari variabel yang berada di kiri tanda titik dua (:)) dapat dipanggil di bagian text):

```

1  .data
2
3  data1: .ascii "Ini data1"
4  data2: .ascii "Haloo"
5  inputmsg: .ascii "Masukkan input: "
6  outputmsg: .ascii "Output: "
7
8  myNumber: .word 20 # interger 20 disimpan dalam 1 word (32 bit)
9
10 array: .byte 21, 42, 56, 57, 13, 12, 13, 41, 15, 12 # array
11

```

Label

Suatu label adalah sebuah address yang telah di-*define* dalam program. Dengan address ini, maka program dapat melompat (jump) atau pergi ke bagian label tersebut saat dibutuhkan. Agar program 'melompat' ke bagian kode tertentu, Anda cukup memberikan nama dari label tersebut kepada suatu command jump ataupun branch.

```

11 .text
12 .globl main
13 main:
14
15 lenArray: #to get len of array
16     la $a1,array #get the base address of array and move it to $a1
17     li $t1,0 #load immediate 0 to $t1
18
19 loopLenArray:
20     lb $t2,0($a1) #load byte from array[$a1 + 0] to $t2
21     beq $t2,$zero,endLoop #if $t2 is equal to zero, go to branch endLoop
22     addi $t1,$t1,1 #increment $t1 by 1 ($t1 is len of array)
23     addi $a1,$a1,1 #increment $a1 (address of array) by 1. (because every data is a byte)
24     j loopLenArray #jump back to loopLenArray for looping
25
26 endLoop:
27     move $t7,$t1 #after all data in array is counted, move result of len to $t7
28

```

Labels : Yang digarisi merah

#POKoknyaPOK ❤️

Jump Command :

Yang ditandai dengan **panah biru**. Saat program mendapatkan perintah "j loopLenArray", maka program akan pergi (lompat) dan melakukan instruction pada address label "loopLenArray". Hal ini akan membuat sebuah loop.

Branch

Branch digunakan untuk menentukan conditional dalam MIPS assembly language.

Contohnya:

```
beq $t0, $t1, suatuLabel
```

- beq = instruksi 'branch on equal'
- \$t0 dan \$t1 dibandingkan apakah value dari \$t0 == \$t1
- suatuLabel = Label yang akan dituju jika \$t0 == \$t1

Maka, saat program menjalankan instruksi di atas akan terjadi seperti berikut.

- Jika value dari \$t0 == \$t1, maka program akan pergi ke label suatuLabel. Jika tidak, program lanjut berjalan seperti biasa.

Contoh lain dalam MARS:

```
5
6      addi $t1, $zero, 20          #$t1 = 20
7      beq $t1, input, inputRight  #Jika input == $t1 atau input == 20, maka pergi ke branch inputIs20
8
9 inputIsNot20:  #Jika input != 20, maka branch tidak akan terjadi dan program lanjut ke inputIsNot20
10              #INGAT! Jika input != 20, kita ingin program menjalankan Label inputIsNot20 dan
11      #kode    #tidak menjalankan Label inputIs20.
12
13 inputIs20:    #Jika dibiarkan seperti ini, program akan terus berjalan seperti biasa dan
14              #akan terus menjalankan sampai akhir program (Label inputIs20 juga akan dijalankan)
15      #kode    #Agar program hanya menjalankan Label inputIsNot20, masukkan exit command atau
16              #jump command sebelum label inputIs20.
```

Selain beq, ada beberapa instruksi branch lainnya yang memiliki perbandingan berbeda-beda. Dapat dilihat di file Guide to MIPS.pdf

Keluar dari Program

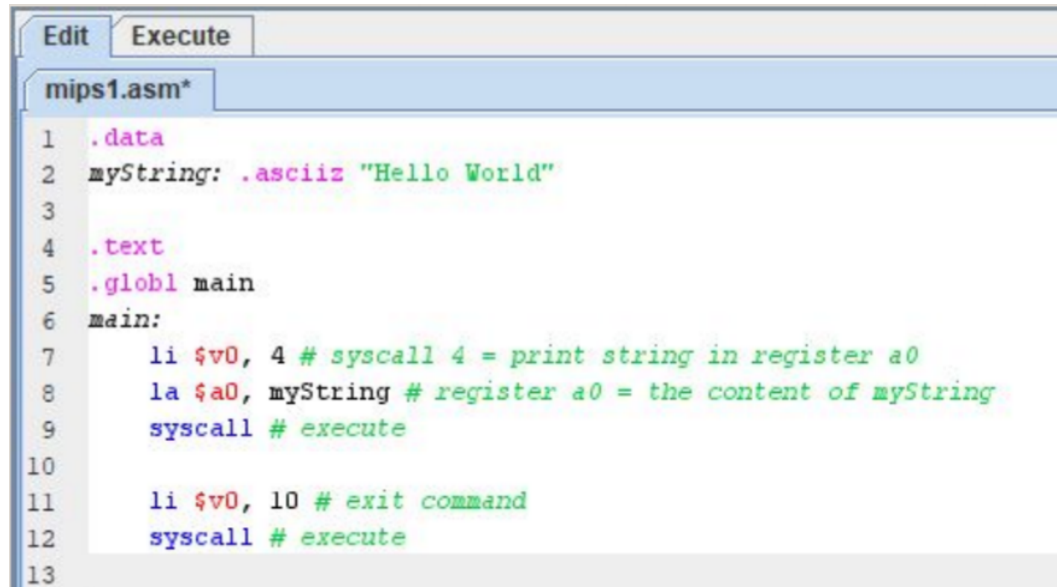
Untuk keluar dan menyelesaikan program, lakukan syscall dengan *service code* 10 pada akhir kode.

```
li $v0, 10 # exit command
syscall # execute
```


Contoh-Contoh program sederhana:

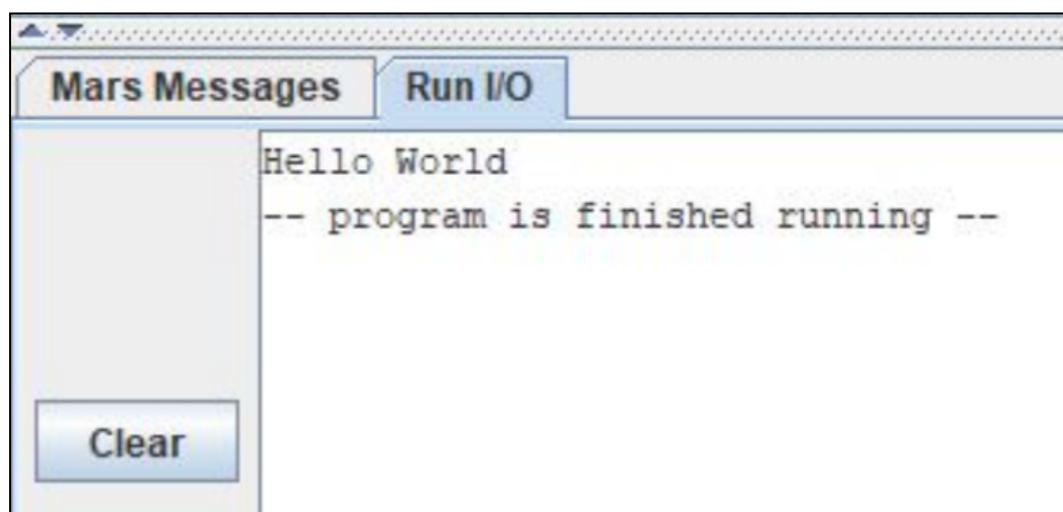
- Sebuah Hello World (PRINT STRING)

Sebuah contoh program yang akan mem-*print* sebuah string "Hello World".



```
1  .data
2  myString: .asciiz "Hello World"
3
4  .text
5  .globl main
6  main:
7      li $v0, 4 # syscall 4 = print string in register a0
8      la $a0, myString # register a0 = the content of myString
9      syscall # execute
10
11     li $v0, 10 # exit command
12     syscall # execute
13
```

Jika program tersebut di-assemble dan dijalankan, hasilnya adalah sebagai berikut.



```
Mars Messages  Run I/O
Hello World
-- program is finished running --

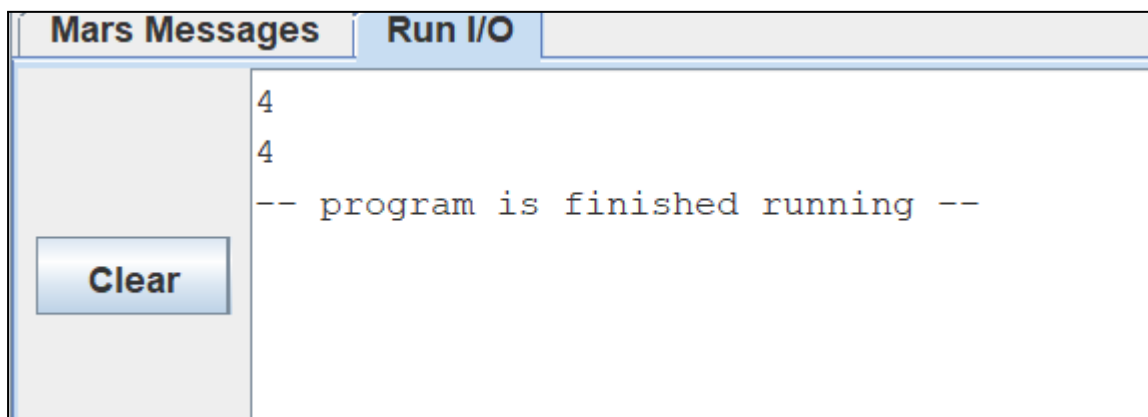
Clear
```

- Sebuah Input (MEMINTA INPUT INTEGER dan PRINT INTEGER)

Sebuah contoh program yang akan meminta input integer dan *print* integer tersebut.

```
1  .data
2
3  .text
4  .globl main
5  main:
6
7      li $v0, 5    #load instruction to READ integer (Service Code 5)
8      syscall      #execute (READ or ASK for integer input)
9
10     add $t0, $v0, $zero    #memindahkan value dari $v0 (input) ke $t0 (temporary)
11
12     li $v0, 1    #load instruction to print integer (Service Code 1)
13     add $a0, $t0, $zero    #move $t0 (value of input) to $a0 (parameter) to be printed
14     syscall      #execute (print the value of $t0 (input))
15
16     li $v0, 10    #load instruction to exit command (Service Code 10)
17     syscall      #execute (exit)
```

Jika program tersebut di-assemble dan dijalankan, hasilnya adalah sebagai berikut.



Integer 4 pertama adalah input yang diberikan oleh user saat input diminta (program akan menunggu user untuk memberikan input pada terminal, seperti method input() pada python ataupun scanner pada java). Sedangkan integer 4 kedua adalah hasil print integer yang didapatkan dari permintaan input.

----- Semangat Labnya! -----