

Thực hành ICT4

TS. Nguyễn Thị Thanh Nga
Bộ môn KTMT
Viện CNTT&TT

Bài 1

- Giới thiệu về công cụ MARS
- Cài đặt
- Cơ bản về giao diện lập trình IDE
- Lập trình và tìm hiểu công cụ lập trình với chương trình Helloworld
- Chạy giả lập

Mục tiêu

- Có khả năng cài đặt được công cụ MARS
- Viết thử chương trình đơn giản để thử nghiệm công cụ MARS như:
 - Lập trình hợp ngữ
 - Chạy giả lập
 - Gỡ rối
 - ...

để hiểu rõ hơn về bản chất và các hoạt động thực sự xảy ra trong bộ xử lý MIPS

Tài liệu

- Tài liệu Kiến trúc MIPS
- MARS features
- MARS tutorial

Công cụ giả lập MIPS IT

- Microprocessor without Interlocked Pipeline Stages
- Là một kiến trúc tập lệnh RISC (Reduced Instruction Set Computer) được phát triển bởi MIPS Technologies.
- Năm 1981, John L.Hennessy bắt đầu nghiên cứu về bộ xử lý MIPS đầu tiên tại Stanford University
- Yêu cầu các câu lệnh phải hoàn thành trong 1 chu kỳ máy

Công cụ giả lập MIPS IT

Một số ứng dụng:

- Pioneer DVR-57-H
- Kenwood HDV-810 Car Navigation System
- HP Color Laser Jet 2500 Printer
- 3COM IP phone, cordless phone
- EOS 10D digital camera
- Sony Play station PSX and High Definition Television
- Samsung Digital Photo Frame
- Sony Media Server Vaio VGX-X90P
- Pioneer Plasma Television

Cài đặt

- Tải về Java Runtime Enviroment, JRE, để chạy công cụ MARS

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

- Cài đặt JRE

- Tải về công cụ MARS, bao gồm:

- Phiên bản mới nhất của MARS, và nên lấy thêm 2 tài liệu
- MARS features
- MARS tutorial

ở URL <http://courses.missouristate.edu/KenVollmar/MARS/download.htm>

Công cụ MARS có thể thực hiện ngay mà không cần cài đặt. Click đúp vào file Mars.jar để chạy

Cơ bản về giao diện lập trình IDE

The screenshot shows the MARS 4.5 IDE window. The title bar reads "W:\KTMT\OneDrive\ICT4\Mars\Fibonacci.asm - MARS 4.5". The menu bar includes "File", "Edit", "Run", "Settings", "Tools", and "Help". A red circle with the number "1" highlights the "Help" menu item. Below the menu bar, the assembly code is displayed in a text editor. The code includes comments in green and assembly instructions in blue. A status bar at the bottom left shows "Line: 2 Column: 12" and a checked "Show Line Numbers" option. Below the text editor is a "Mars Messages" panel with a "Run I/O" tab. The message area displays "Assemble: operation completed successfully." and a "Clear" button. On the right side of the IDE, a register window shows the state of registers \$a0 through \$a7 and \$t0 through \$t7, all containing the value 0x00000000.

W:\KTMT\OneDrive\ICT4\Mars\Fibonacci.asm - MARS 4.5

File Edit Run Settings Tools Help 1

1 Menu

- Hầu hết các mục trong menu đều có các icon tương ứng
- Di chuyển chuột lên trên của icon, chức năng tương ứng sẽ hiển thị.
- Các mục trong menu cũng có phím tắt tương ứng.

```
4 size: .word 12           # size of "array"
5 .text
6 la $t0, fibs             # load address of array
```

Line: 2 Column: 12 ☒ Show Line Numbers

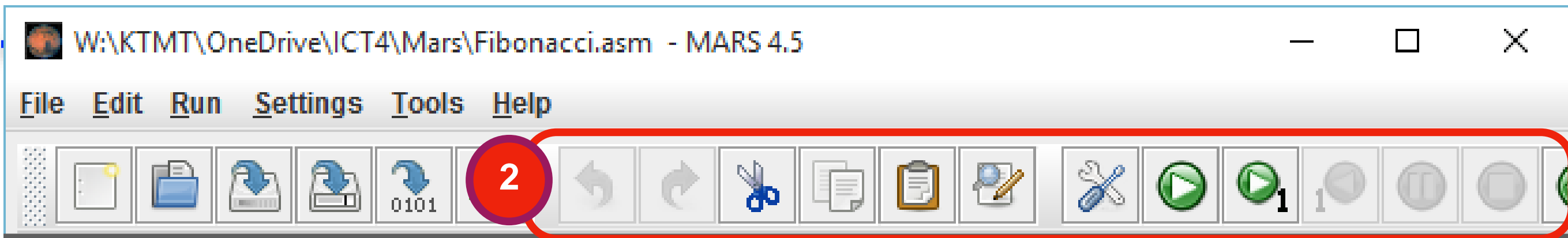
Mars Messages Run I/O

Assemble: operation completed successfully.

Clear

\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$a0	16	0x00000000

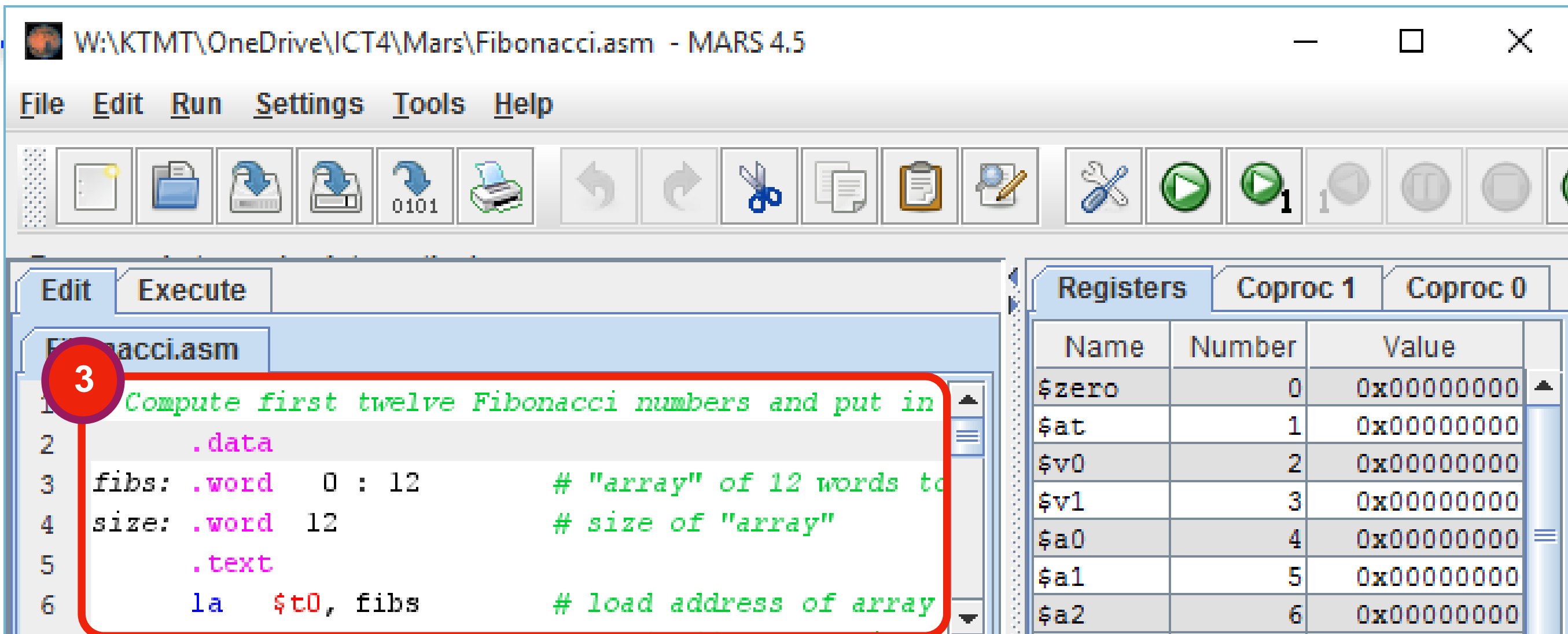
Cơ bản về giao diện lập trình IDE



2 Toolbar

- Chứa một vài tính năng soạn thảo cơ bản như: copy, paste, open.
- Các tính năng gỡ rối (trong hình chữ nhật màu đỏ)
 - Run: chạy toàn bộ chương trình
 - Run one step at a time: chạy từng lệnh và dừng (rất hữu ích)
 - Undo the last step: khôi phục lại trạng thái ở lệnh trước đó (rất hữu ích)
 - Pause: tạm dừng quá trình chạy toàn bộ (Run)
 - Stop: kết thúc quá trình gỡ rối
 - Reset MIPS memory and register: Khởi động lại

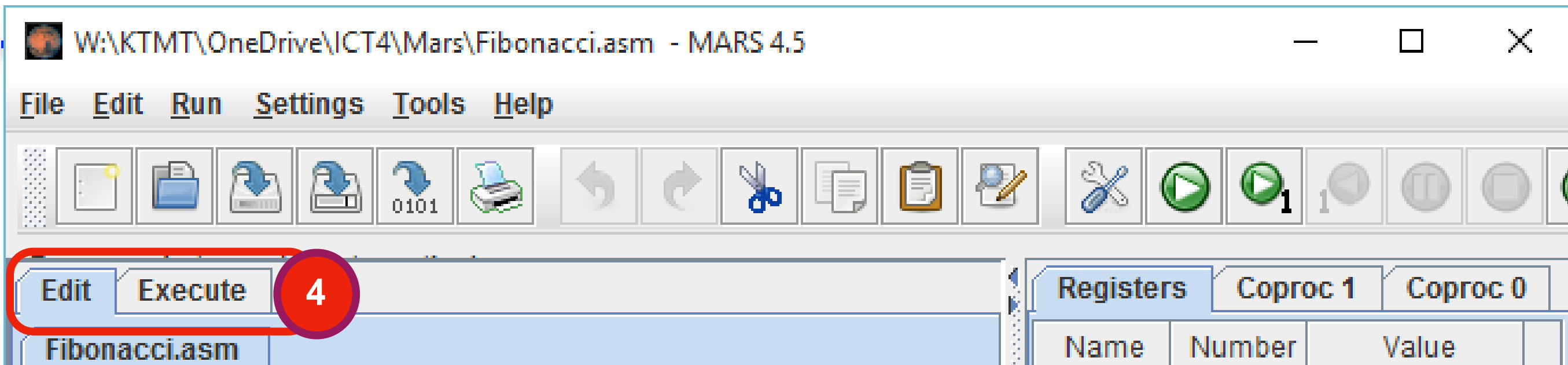
Cơ bản về giao diện lập trình IDE



3 Edit tab

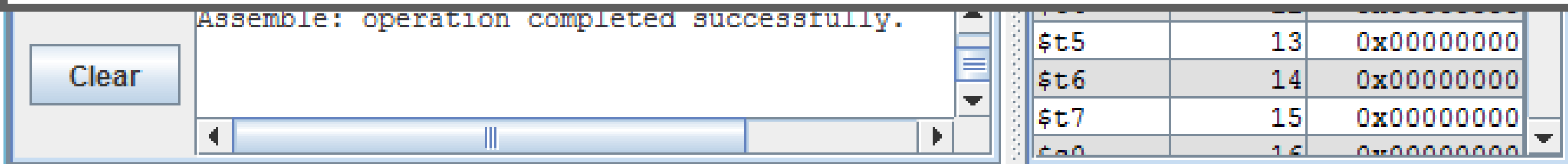
- Soạn thảo văn bản tích hợp sẵn với tính năng **tô màu theo cú pháp**, giúp người dùng dễ theo dõi mã nguồn.
- Khi gõ lệnh mà chưa hoàn tất, một popup sẽ hiện ra để trợ giúp.
- Vào menu Settings / Editor... để thay đổi màu sắc, font...

Cơ bản về giao diện lập trình IDE



4 Edit/Execute

- Mỗi file mã nguồn ở giao diện soạn thảo có 2 cửa sổ - 2 tab: Edit và Execute
 - Edit tab:** viết chương trình hợp ngữ với tính năng **tô màu theo cú pháp**.
 - Execute tab:** biên dịch chương trình hợp ngữ đã viết ở Edit tab thành mã máy, **chạy và gỡ rối**.



Cơ bản về giao diện lập trình IDE

5 Message Areas: Có 2 cửa sổ message ở cạnh dưới của giao diện IDE

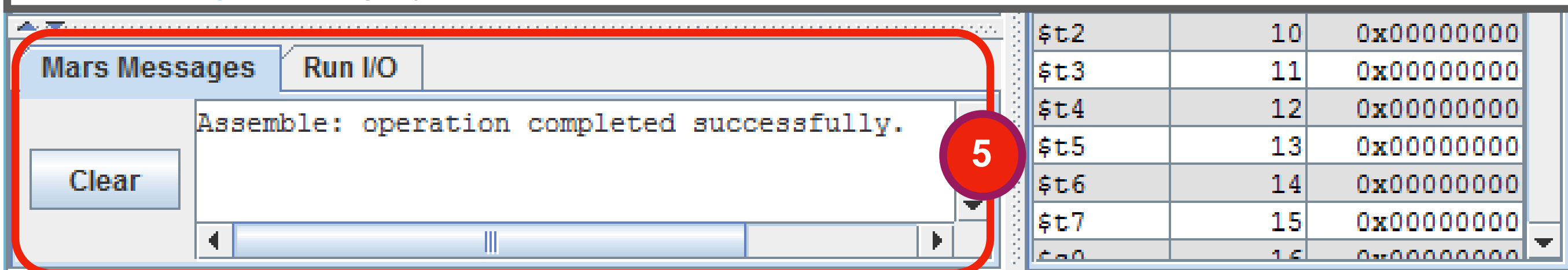
- **The *Run I/O* tab** chỉ có tác dụng khi đang chạy run-time

- Hiển thị các kết quả xuất ra console, và
- Nhập dữ liệu vào cho chương trình qua console.

MARS có tùy chọn để mọi thông tin nhập liệu vào qua console sẽ được hiển thị lại ra message area.

- ***MARS Messages* tab** được dùng để hiển thị cho các thông báo còn lại như là các báo lỗi trong quá trình biên dịch hay trong quá trình thực hiện run-time.

- Có thể **click vào thông báo lỗi để chương trình tự động nhảy tới dòng lệnh** gây ra lỗi.

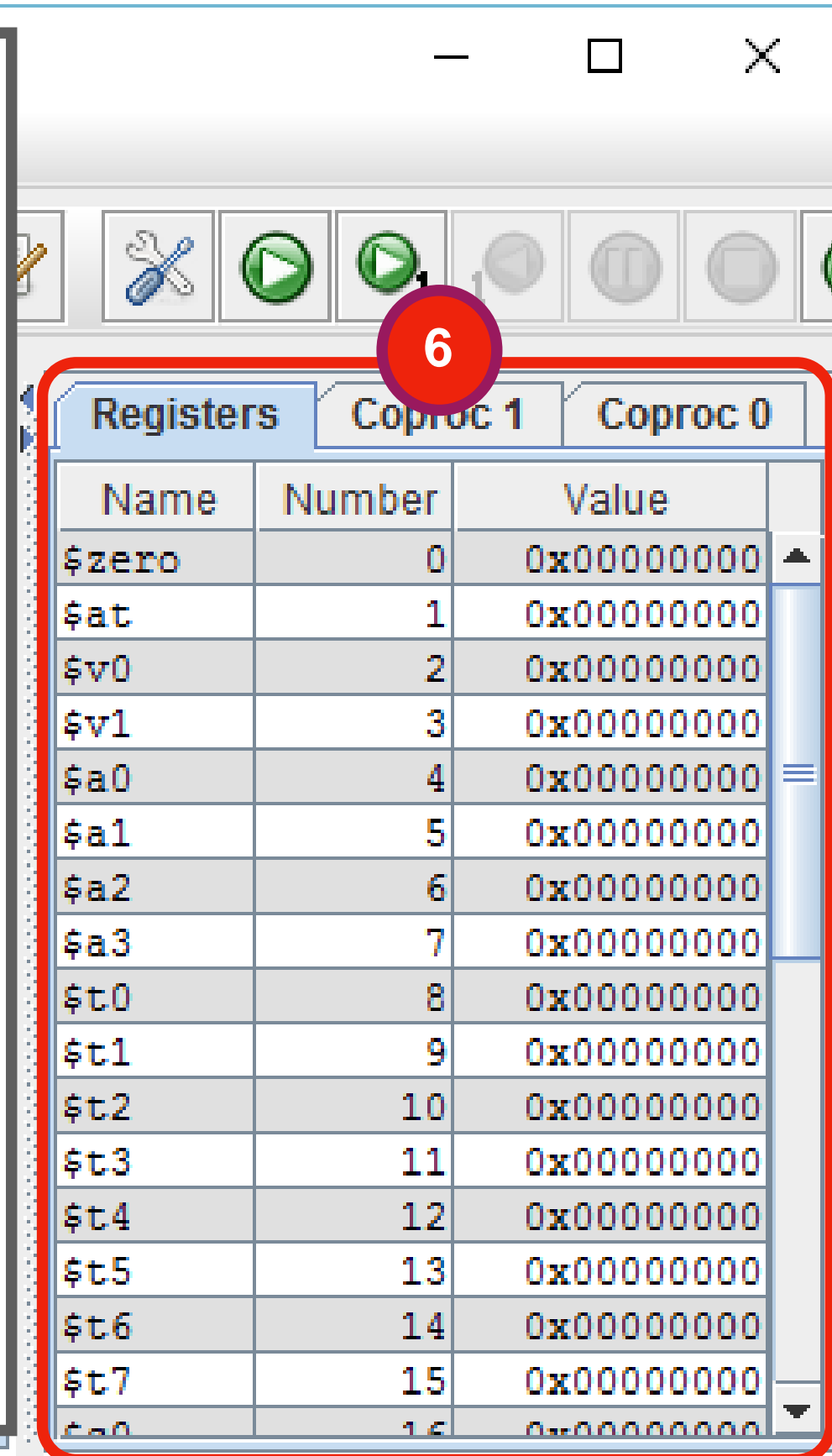


Cơ bản về giao diện lập trình IDE

⑥ MIPS Registers: Bảng hiển thị giá trị của các thanh ghi của bộ xử lý MIPS, luôn luôn được hiển thị, bất kể chương trình hợp ngữ có được chạy hay không. Khi viết chương trình, bảng này sẽ giúp người dùng nhớ tên của các thanh ghi và địa chỉ của chúng.

Có 3 tab trong bảng này:

- **the Register File:** các thanh ghi số nguyên với địa chỉ từ \$0 tới \$31, và cả 3 thanh ghi đặc LO, HI và thanh ghi Program Counter
- **the Coprocessor 0 registers:** các thanh ghi của bộ đồng xử lý C0, phục vụ cho xử lý ngắt
- **the Coprocessor 1 registers:** các thanh ghi số dấu phẩy động

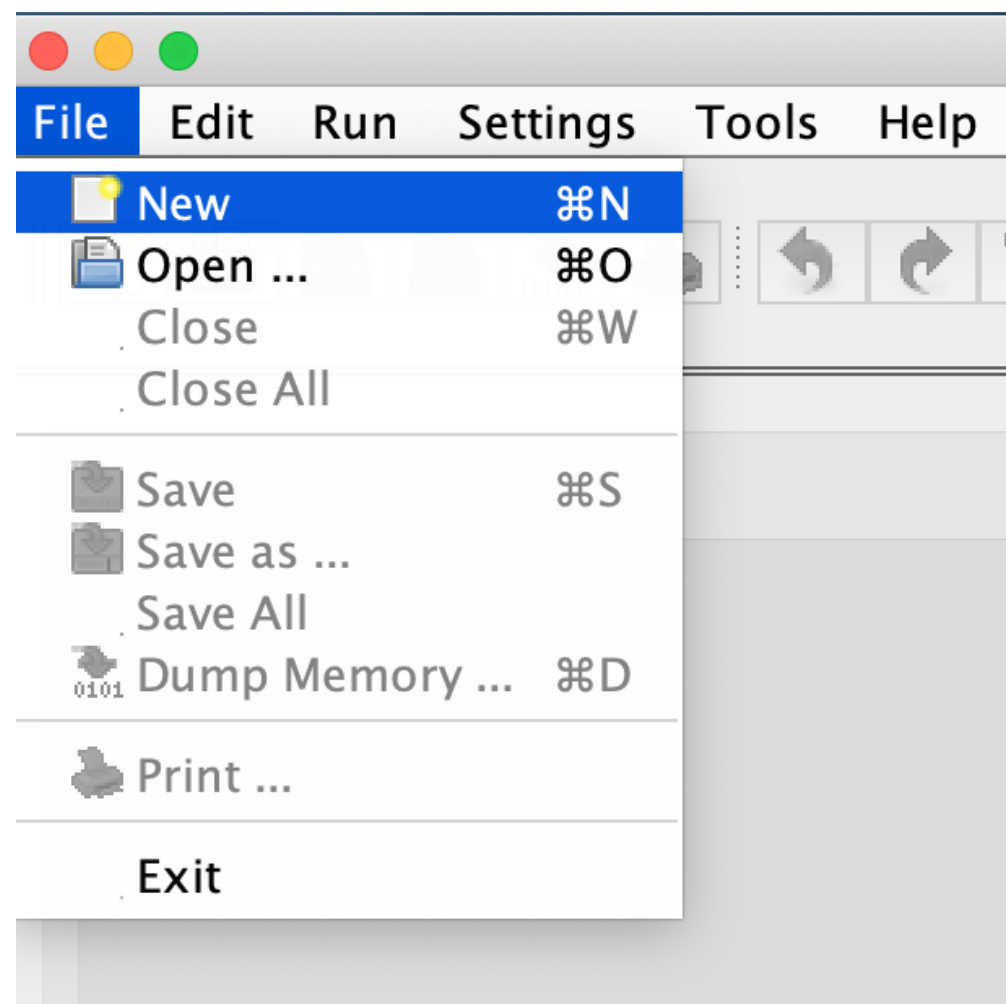


The screenshot shows the MIPS IDE interface. A red circle with the number 6 highlights the 'Registers' tab. The table below displays the values of the registers.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$f0	16	0x00000000

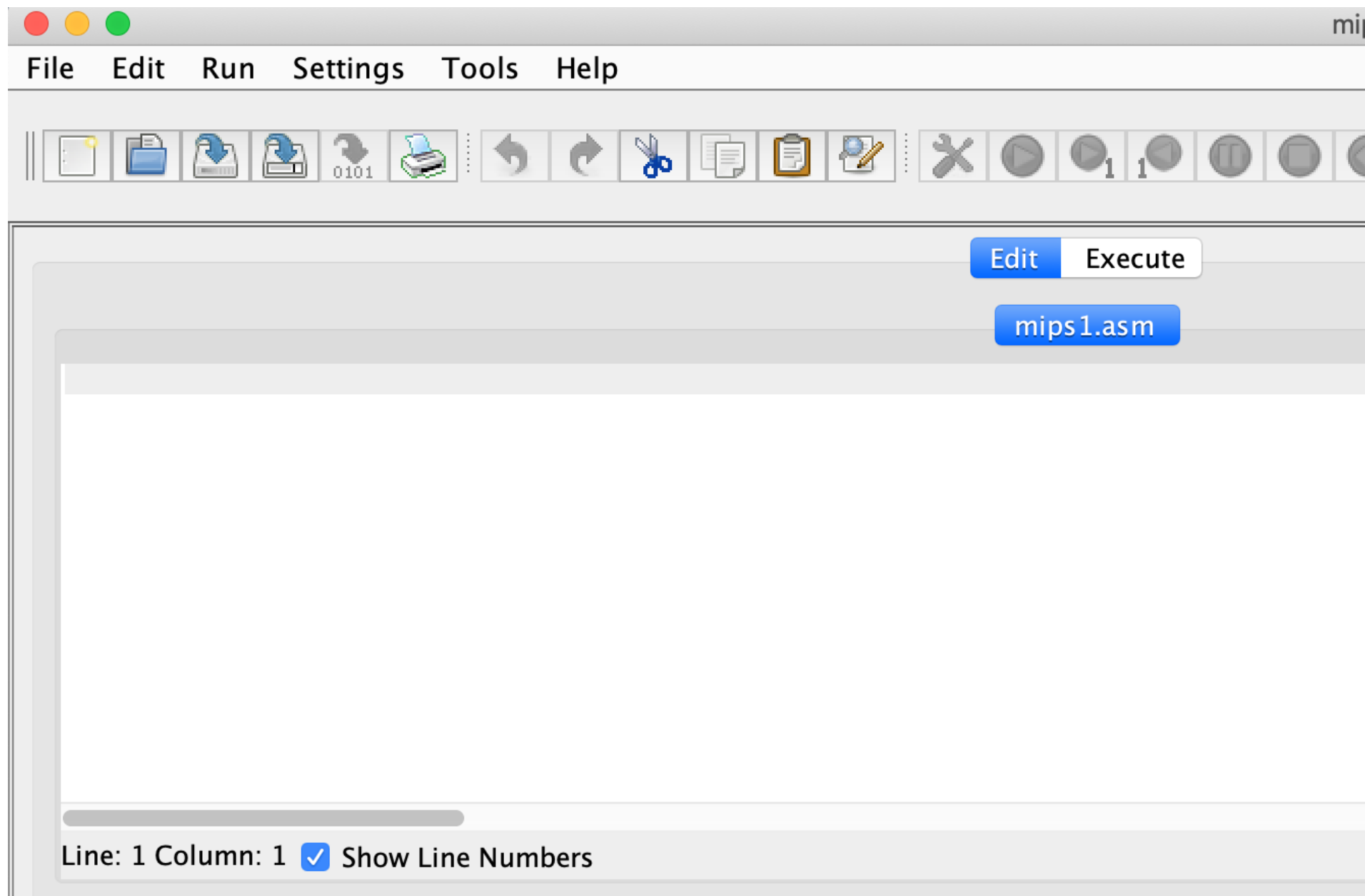
Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

1. Click vào file mars.jar để bắt đầu chương trình
2. Ở thanh menu, chọn File/New để tạo một file hợp ngữ mới



Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

3. Cửa sổ soạn thảo file hợp ngữ sẽ hiện ra như hình bên. Bắt đầu lập trình.



Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

4. Hãy gõ đoạn lệnh sau vào cửa sổ soạn thảo

```
.data                # Vung du lieu, chua cac khai bao bien
x:      .word  0x01020304  # bien x, khoi tao gia tri
message: .asciiz "Bo mon Ky thuat May tinh"

.text                # Vung lenh, chua cac lenh hop ngu
    la $a0, message    #Dua dia chi bien mesage vao thanh ghi a0
    li $v0, 4           #Gan thanh ghi $v0 = 4
    syscall             #Goi ham so v0, ham so 4, la ham print

    addi $t1, $zero, 2   #Thanh ghi $t1 = 2
    addi $t2, $zero, 3   #Thanh ghi $t2 = 3
    add  $t0, $t1, $t2   #Thanh ghi t- = $t1 + $t2
```


Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

Kết quả như sau:

Edit Execute

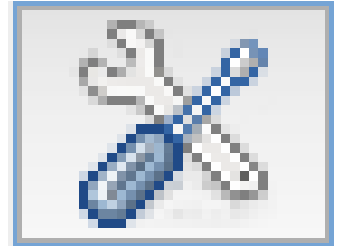
mips1.asm*

```
1  .data                # Vung du lieu, chua cac khai bao bien
2  x:                  .word    0x01020304    # bien x, khoi tao gia tri
3  message:           .asciiz  "Bo mon Ky thuat May tinh"
4
5  .text               # Vung lenh, chua cac lenh hop ngu
6      la    $a0, message    #Dua dia chi bien mesage vao thanh ghi a0
7      li    $v0, 4          #Gan thanh ghi $v0 = 4
8      syscall              #Goi ham so v0, ham so 4, la ham print
9
10     addi $t1,$zero,2      #Thanh ghi $t1 = 2
11     addi $t2,$zero,3      #Thanh ghi $t2 = 3
12     add  $t0, $t1, $t2    #Thanh ghi t- = $t1 + $t2
```

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

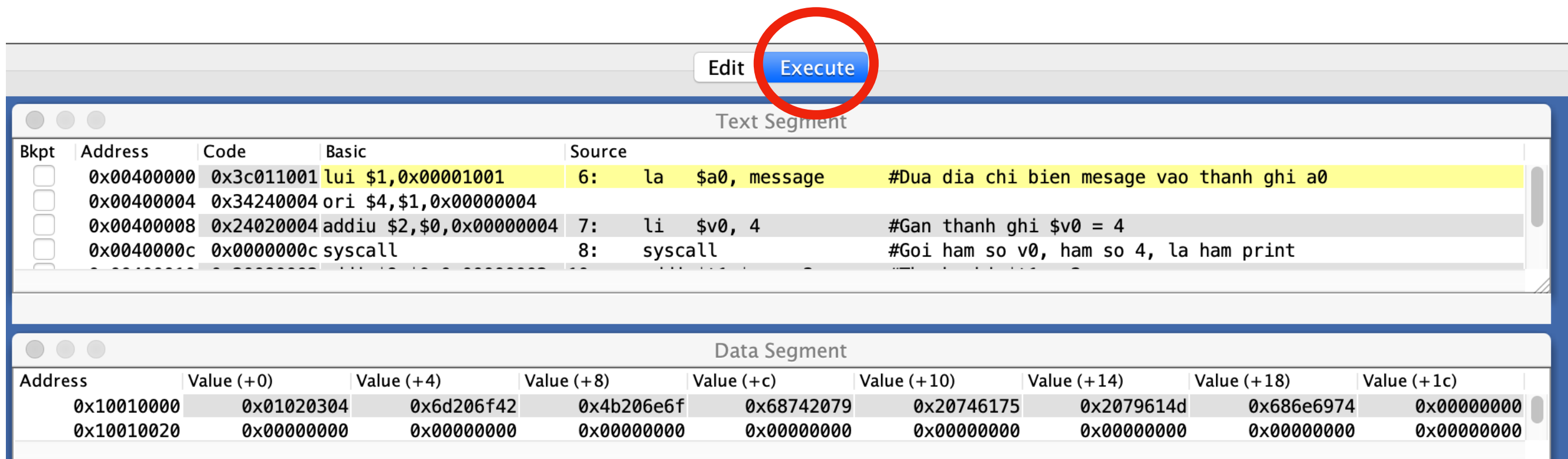
5. Để biên dịch chương trình hợp ngữ trên thành mã máy, thực hiện một trong các cách sau:

- vào menu Run / Assemble, hoặc
- trên thanh menu, bấm vào biểu tượng
- hoặc bấm phím tắt F3.



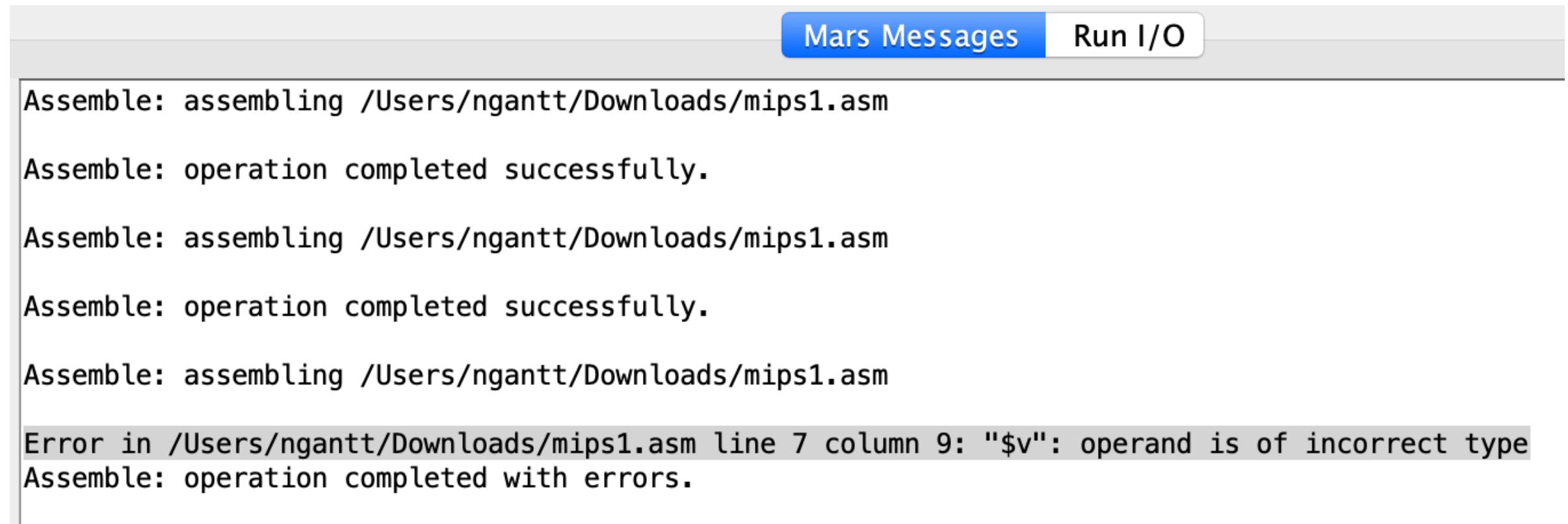
Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

6. Nếu đoạn hợp ngữ đúng, MARS sẽ chuyển từ Edit tab sang Execute tab



Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

Chú ý: nếu đoạn hợp ngữ có lỗi, cửa sổ Mars Messages sẽ hiển thị chi tiết lỗi. Bấm vào dòng thông báo lỗi để trình soạn thảo tự động nhảy tới dòng code bị lỗi, rồi tiến hành sửa lại cho đúng.



The screenshot shows a window titled "Mars Messages" with a "Run I/O" button. The window contains the following text:

```
Assemble: assembling /Users/ngantt/Downloads/mips1.asm
Assemble: operation completed successfully.
Assemble: assembling /Users/ngantt/Downloads/mips1.asm
Assemble: operation completed successfully.
Assemble: assembling /Users/ngantt/Downloads/mips1.asm
Error in /Users/ngantt/Downloads/mips1.asm line 7 column 9: "$v": operand is of incorrect type
Assemble: operation completed with errors.
```

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

7. Ở Execute tab, có 2 cửa sổ chính là **Text Segment**, và **Data Segment**

The screenshot displays the Execute tab of a debugger, showing two main windows: the Text Segment and the Data Segment.

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$a0, message #Dua dia chi bien mesage vao thanh ghi a0
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,0x00000004	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Goi ham so v0, ham so 4, la ham print

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Below the Data Segment window, there is a control bar with a left arrow, a right arrow, a dropdown menu showing "0x10010000 (.data)", and checkboxes for "Hexadecimal Addresses" (checked), "Hexadecimal Values" (checked), and "ASCII" (unchecked).

The assembly code for the Data Segment is shown below:

```
1 .data # Vung du lieu, chua cac khai bao bien
2 x: .word 0x01020304 # bien x, khoi tao gia tri
3 message: .asciiz "Bo mon Ky thuat May tinh"
4
5 .text # Vung lenh, chua cac lenh hop ngu
6 la $a0, message #Dua dia chi bien mesage vao thanh ghi a0
7 li $v0, 4 #Gan thanh ghi $v0 = 4
8 syscall #Goi ham so v0, ham so 4, la ham print
9
10 addi $t1,$zero,2 #Thanh ghi $t1 = 2
11 addi $t2,$zero,3 #Thanh ghi $t2 = 3
12 add $t0, $t1, $t2 #Thanh ghi t- = $t1 + $t2
```

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

7. Ở Execute tab, có 2 cửa sổ chính là **Text Segment**, và **Data Segment**

The screenshot displays the Execute tab of a debugger, showing two main windows: the Text Segment and the Data Segment.

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$a0, message #Dua dia chi bien mesage vao thanh ghi a0
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,0x00000004	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Goi ham so v0, ham so 4, la ham print

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Navigation controls: 0x10010000 (.data) ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

Assembly Code:

```
1 .data # Vung du lieu, chua cac khai bao bien
2 x: .word 0x01020304 # bien x, khoi tao gia tri
3 message: .asciiz "Bo mon Ky thuat May tinh"
4
5 .text # Vung lenh, chua cac lenh hop ngu
6 la $a0, message #Dua dia chi bien mesage vao thanh ghi a0
7 li $v0, 4 #Gan thanh ghi $v0 = 4
8 syscall #Goi ham so v0, ham so 4, la ham print
9
10 addi $t1,$zero,2 #Thanh ghi $t1 = 2
11 addi $t2,$zero,3 #Thanh ghi $t2 = 3
12 add $t0, $t1, $t2 #Thanh ghi t- = $t1 + $t2
```


Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

7. Ở Execute tab, có 2 cửa sổ chính là **Text Segment**, và **Data Segment**

- **Text Segment:** là vùng không gian bộ nhớ chứa các mã lệnh hợp ngữ. Tương ứng với mã nguồn hợp ngữ, các dòng nào viết sau chỉ thị **.TEXT** tức là lệnh và sẽ thuộc Text Segment.
- **Data Segment:** là vùng không gian bộ nhớ chứa các biến. Tương ứng với mã nguồn hợp ngữ, các dòng nào viết sau chỉ thị **.DATA** tức là lệnh và sẽ thuộc Text Segment.

*Chú ý: vì lý do nào đó, nếu ta khai báo biến sau chỉ thị **.TEXT** hoặc ngược lại thì trình biên dịch sẽ báo lỗi hoặc bỏ qua khai báo sai đó.*

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

8. Ở Execute tab, sử dụng checkbox bên dưới để thay đổi cách hiển thị dữ liệu cho dễ quan sát
- ☒ Hexadecimal Addresses : hiển thị địa chỉ ở dạng số nguyên hệ 16
 - ☒ Hexadecimal Values : hiển thị giá trị thanh ghi ở dạng số nguyên hệ 16.
 - ☒ ASCII : hiển thị giá trị trong bộ nhớ ở dạng kí tự ASCII

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

8. Ở Execute tab, sử dụng checkbox bên dưới để thay đổi cách hiển thị dữ liệu cho dễ quan sát

The screenshot shows the 'Execute' tab of a debugger. The top panel, titled 'Text Segment', displays assembly code with columns for Bkpt, Address, Code, Basic, and Source. The bottom panel, titled 'Data Segment', shows memory addresses and their values. A red box highlights the left sidebar, which contains checkboxes for each instruction in the assembly list. Another red box highlights the 'Hexadecimal Addresses' checkbox at the bottom of the interface, which is currently checked.

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,4097	6: la \$a0, message #Dua dia chi bien message vào thanh ghi a0
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,4	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,4	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Goi ham so v0, ham so 4, la ham print

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	16909060	1830842178	1260416623	1752440953	544498037	544825677	1752066420	0
0x10010020	0	0	0	0	0	0	0	0

0x10010000 (.data) ☒ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

8. Ở Execute tab, sử dụng checkbox bên dưới để thay đổi cách hiển thị dữ liệu cho dễ quan sát

The screenshot shows the 'Execute' tab of a debugger. The top panel displays assembly code with columns for 'Bkpt', 'Address', 'Code', 'Basic', and 'Source'. The bottom panel displays the 'Data Segment' with columns for 'Address' and 'Value (+0)' through 'Value (+28)'. Both panels have red boxes highlighting specific areas. The bottom panel also has a control bar with checkboxes for 'Hexadecimal Addresses', 'Hexadecimal Values', and 'ASCII'.

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	4194304	0x3c011001	lui \$1, 0x00001001	6: la \$a0, message #Dua dia chi bien mesage vao thanh ghi a0
<input type="checkbox"/>	4194308	0x34240004	ori \$4, \$1, 0x00000004	
<input type="checkbox"/>	4194312	0x24020004	addiu \$2, \$0, 0x00000004	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	4194316	0x0000000c	syscall	8: syscall #Goi ham so v0, ham so 4, la ham print

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
268501024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☐ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

8. Ở Execute tab, sử dụng checkbox bên dưới để thay đổi cách hiển thị dữ liệu cho dễ quan sát

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	4194304	0x3c011001	lui \$1,4097	6: la \$a0, message #Dua dia chi bien mesage vao thanh ghi a0
<input type="checkbox"/>	4194308	0x34240004	ori \$4,\$1,4	
<input type="checkbox"/>	4194312	0x24020004	addiu \$2,\$0,4	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	4194316	0x0000000c	syscall	8: syscall #Goi ham so v0, ham so 4, la ham print

Data Segment


Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	.	m	o	B	K	n	o	h
268501024	\0 \0 \0 \0	t	y	t	a	u	y	a
							M	h
								n
								i
								t
								\0 \0 \0 \0

0x10010000 (.data) ☐ Hexadecimal Addresses ☐ Hexadecimal Values ☒ ASCII

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

9. Ở Execute tab, trong cửa sổ Text Segment, bảng có 5 cột.


Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$a0, message #Dua dia chi bien mesage vao thanh ghi a0
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,0x00000004	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Goi ham so v0, ham so 4, la ham print

- **Bkpt:** Breakpoint, điểm dừng khi chạy toàn bộ chương trình chương trình bằng nút 

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

9. Ở Execute tab, trong cửa sổ Text Segment, bảng có 5 cột.


Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$a0, message #Dua dia chi bien mesage vao thanh ghi a0
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,0x00000004	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Goi ham so v0, ham so 4, la ham print

- **Bkpt:** Breakpoint, điểm dừng khi chạy toàn bộ chương trình chương trình bằng nút 
- **Address:** địa chỉ của lệnh ở dạng số nguyên (*xem thêm hướng dẫn về cửa sổ Label*)

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

9. Ở Execute tab, trong cửa sổ Text Segment, bảng có 5 cột.


Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$a0, message #Dua dia chi bien mesage vao thanh ghi a0
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,0x00000004	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Goi ham so v0, ham so 4, la ham print

- **Bkpt:** Breakpoint, điểm dừng khi chạy toàn bộ chương trình chương trình bằng nút 
- **Address:** địa chỉ của lệnh ở dạng số nguyên (*xem thêm hướng dẫn về cửa sổ Label*)
- **Code:** lệnh ở dạng mã máy

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

9. Ở Execute tab, trong cửa sổ Text Segment, bảng có 5 cột.

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$a0, message #Đưa địa chỉ biến message vào thanh ghi a0
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,0x00000004	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 #Gán thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Gọi hàm số v0, hàm số 4, là hàm print

- **Bkpt:** Breakpoint, điểm dừng khi chạy toàn bộ chương trình chương trình bằng nút 
- **Address:** địa chỉ của lệnh ở dạng số nguyên (*xem thêm hướng dẫn về cửa sổ Label*)
- **Code:** lệnh ở dạng mã máy
- **Basic:** lệnh ở dạng hợp ngữ thuần, **giống như qui định trong tập lệnh**. Ở đây, tất cả các nhãn, tên gọi nhớ.. đều đã được chuyển đổi thành hằng số.

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

9. Ở Execute tab, trong cửa sổ Text Segment, bảng có 5 cột.

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$a0, message #Dua dia chi bien mesage vao thanh ghi a0
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,0x00000004	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Goi ham so v0, ham so 4, la ham print

- **Source:** lệnh ở dạng hợp ngữ có bổ sung các macro, nhãn.. giúp lập trình nhanh hơn, dễ hiểu hơn, **không còn giống như tập lệnh** nữa. Trong ảnh minh họa bên dưới:
 - Lệnh **la** trong cột Source là lệnh giả, không có trong tập lệnh được dịch tương ứng thành 2 lệnh **lui** và **ori** trong cột Basic.
 - Nhãn message trong lệnh **la \$a0, message** trong cột Source được dịch thành hằng số 0x00001001 (xem thêm hướng dẫn về cửa sổ Label)

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

9. Ở Execute tab, trong cửa sổ Data Segment, bảng có 9 cột.

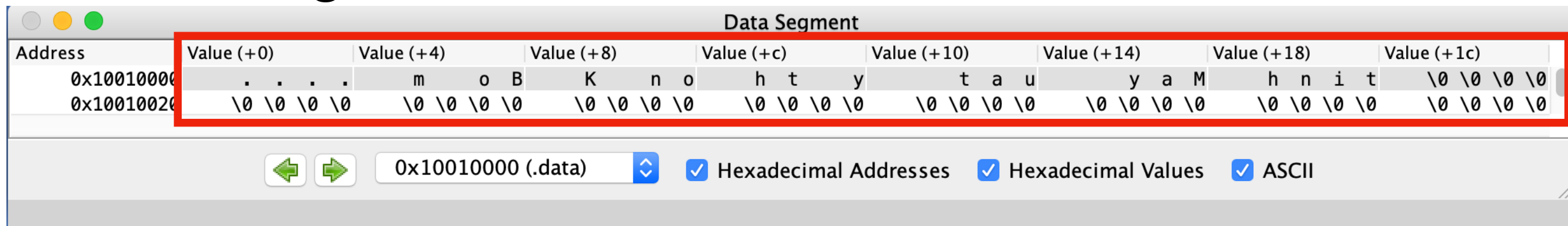
Data Segment									
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)	
0x10010000	.	m	K	h	t	y	a	M	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

☒ Hexadecimal Addresses
 ☒ Hexadecimal Values
 ☒ ASCII

- **Address:** địa chỉ của dữ liệu, biến ở dạng số nguyên. Giá trị mỗi dòng tăng 32 đơn vị (ở hệ 10, hoặc $20_{(16)}$) bởi vì mỗi dòng sẽ trình bày 32 byte ở các địa chỉ liên tiếp nhau

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

9. Ở Execute tab, trong cửa sổ Data Segment, bảng có 9 cột.



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	m o B	K n o	h t y	t a u	y a M	h n i t	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

- **Address:** địa chỉ của dữ liệu, biến ở dạng số nguyên. Giá trị mỗi dòng tăng 32 đơn vị (ở hệ 10, hoặc $20_{(16)}$) bởi vì mỗi dòng sẽ trình bày 32 byte ở các địa chỉ liên tiếp nhau
- **Các cột Value:** mỗi cột Value chứa 4 byte, và có 8 cột Value, tương ứng với 32 byte liên tiếp nhau.

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

9. Ở Execute tab, trong cửa sổ Data Segment, bảng có 9 cột.

The top screenshot shows the Data Segment window with the following table:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	.	m	o	B	K	n	o	h
0x10010020	\0	\0	\0	\0	\0	\0	\0	\0

The assembly code below shows:

```
1 .data # Vung du lieu, chua cac khai bao bien
2 x: .word 0x01020304 # bien x, khoi tao gia tri
3 message: .ascii "Bo mon Ky thuat May tinh"
```

The bottom screenshot shows the Data Segment window after execution, with the following table:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

The assembly code is the same as in the top screenshot.

- Có thể thấy rõ giá trị của biến **x = 0x01020304** được hiển thị chính xác trong Data Segment khi hiển thị dữ liệu ở dạng số ☒ **Hexadecimal Values**, và giá trị của chuỗi “Bo mon Ky thuat May tinh” khi hiển thị ở dạng kí tự ☒ **ASCII**

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

9. Ở Execute tab, trong cửa sổ Data Segment, bảng có 9 cột.

The top screenshot shows the Data Segment window with the following table:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	.	m	o	B	K	n	o	h
0x10010020	\0	\0	\0	\0	\0	\0	\0	\0

The middle section shows assembly code:



```
1 .data # Vung du lieu, chua cac khai bao bien
2 x: .word 0x01020304 # bien x, khoi tao gia tri
3 message: .asciiz "Bo mon Ky thuat May tinh"
```

The bottom screenshot shows the Data Segment window after execution, with the following table:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

- Lưu ý rằng việc lưu trữ chuỗi trong bộ nhớ ở dạng little-endian là do cách lập trình hàm phần mềm syscall, chứ không phải do bộ xử lý MIPS qui định. Có thể thấy, ở công cụ giả lập MIPS IT, hàm print lại qui định chuỗi theo kiểu big-endian.

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

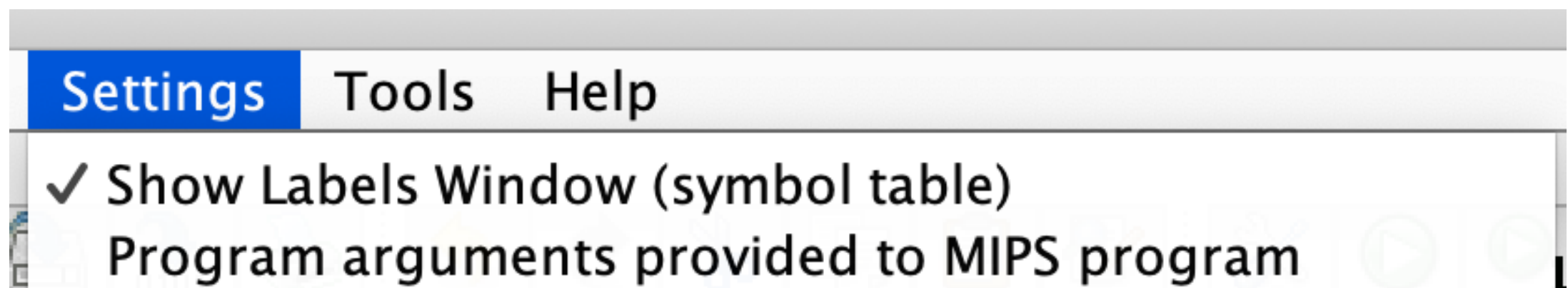
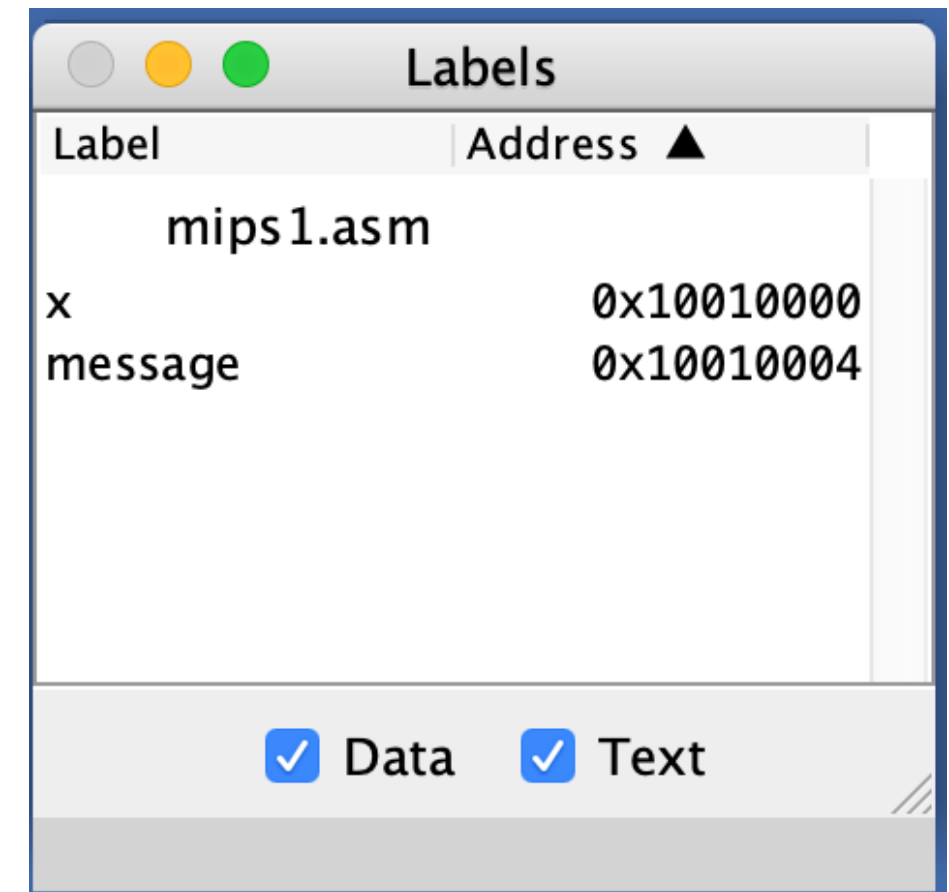
- Bấm vào cặp nút   để dịch chuyển tới vùng địa chỉ lân cận
- Bấm vào combobox để dịch tới vùng bộ nhớ chứa loại dữ liệu được chỉ định. Trong đó lưu ý
 - .data: vùng dữ liệu
 - .text: vùng lệnh
 - \$sp: vùng ngăn xếp

0x10000000 (.extern)
✓ 0x10010000 (.data)
0x10040000 (heap)
current \$gp
current \$sp
0x00400000 (.text)
0x90000000 (.kdata)
0xffff0000 (MMIO)

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

11. Cửa sổ Label: hiển thị tên nhãn và hằng số địa chỉ tương ứng với nhãn khi được biên dịch ra mã máy.

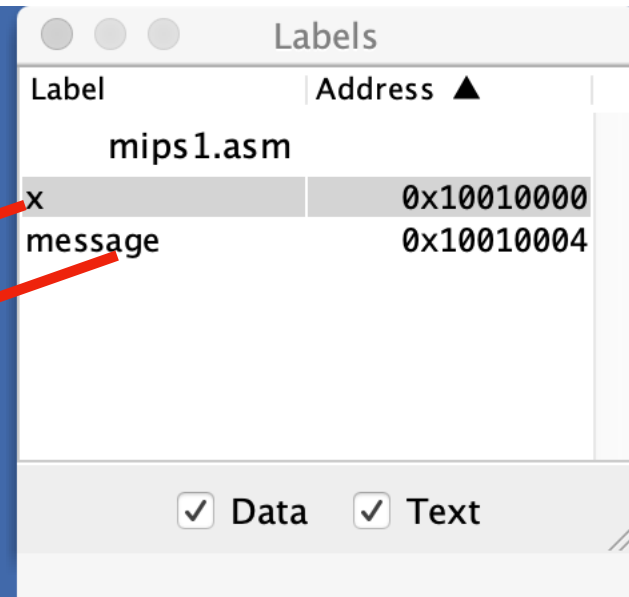
- Cửa sổ Label không tự động hiển thị. Phải vào menu Settings / chọn Show Labels Windows.



Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

Trong cửa sổ Labels cho biết :

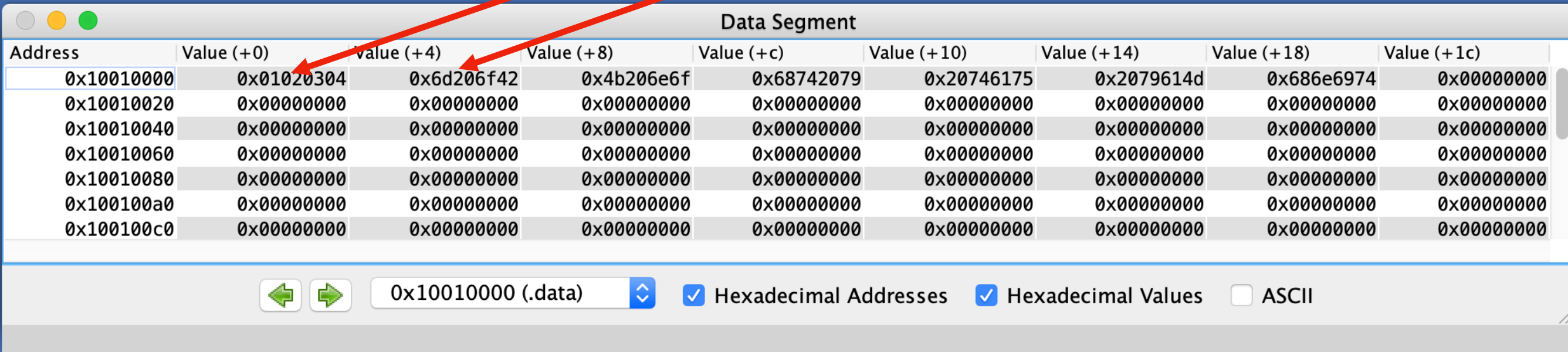
- **x** chỉ là tên gọi nhớ, **x** sẽ được qui đổi thành hằng số 0x10010000.
- **message** cũng chỉ là tên gọi nhớ, sẽ được qui đổi thành hằng số 0x10010004



Labels window showing memory addresses for 'x' and 'message'.

Label	Address ▲
x	0x10010000
message	0x10010004

Buttons: ☒ Data ☒ Text



Data Segment window showing memory addresses and values. Red arrows point from the 'x' and 'message' labels in the Labels window to the corresponding values in the Data Segment window.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Navigation: 0x10010000 (.data)

Options: ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

- Click đúp vào tên biến, sẽ tự động chuyển sang vị trí tương ứng trong cửa sổ Data Segment.

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

The screenshot displays a MIPS assembler interface with two main windows: 'Text Segment' and 'Data Segment'.

Text Segment Window:

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$a0, message #Dua dia chi bien m...
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,0x00000004	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Goi ham so v0, ham...
<input type="checkbox"/>	0x00400010	0x20090002	addi \$9,\$0,0x00000002	10: addi \$t1,\$zero,2 #Thanh ghi \$t1 = 2
<input type="checkbox"/>	0x00400014	0x200a0003	addi \$10,\$0,0x00000003	11: addi \$t2,\$zero,3 #Thanh ghi \$t2 = 3
<input type="checkbox"/>	0x00400018	0x012a4020	add \$8,\$9,\$10	12: add \$t0, \$t1, \$t2 #Thanh ghi t- = \$t1...

Labels Window:

Label	Address ▲
mips1.asm	
x	0x10010000
message	0x10010004

☒ Data ☒ Text

Data Segment Window:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Navigation: 0x10010000 (.data) ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

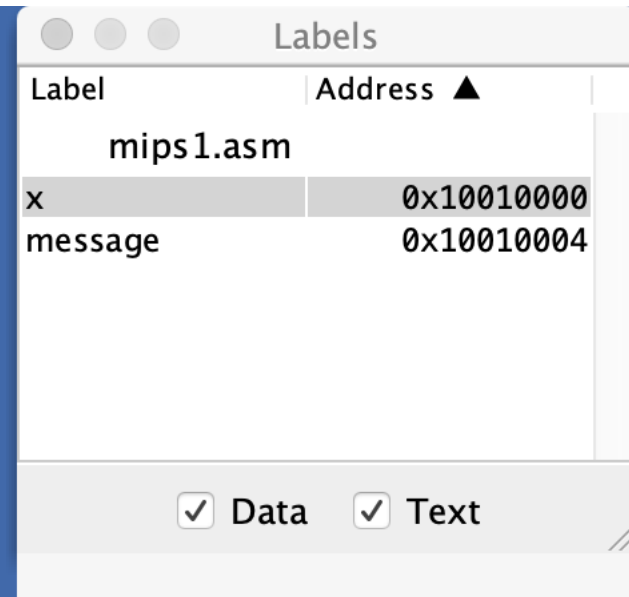
Trong cửa sổ Text Segment cho biết:

- Ở lệnh gán **la \$a0, message** tên gọi nhớ **message** đã được chuyển thành hằng số 0x10010004 thông qua cặp lệnh **lui, ori**

Bắt đầu lập trình và tìm hiểu các công cụ với chương trình Helloworld

Trong cửa sổ Data Segment cho biết:

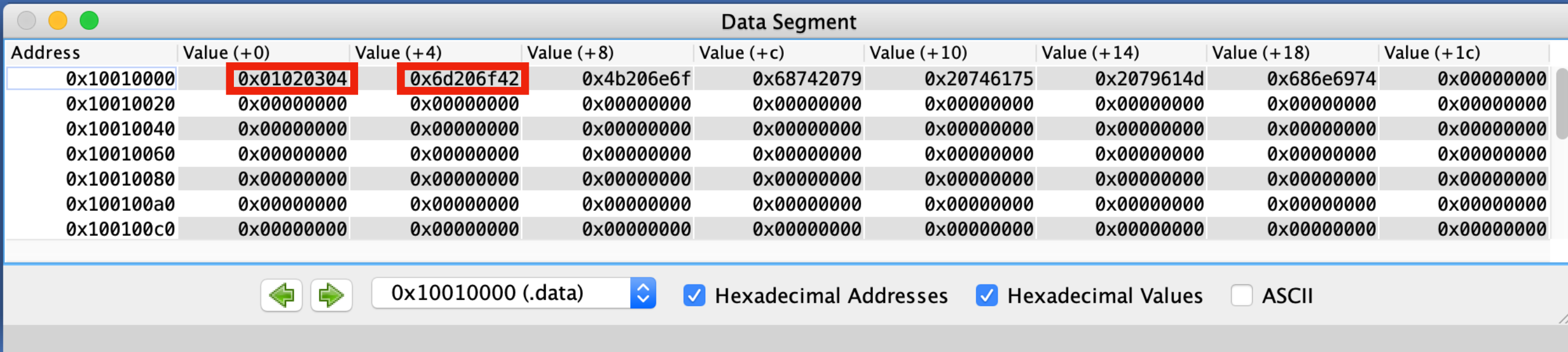
- Để giám sát giá trị của biến **x**, mở Data Segment ở hằng số 0x10010000 sẽ nhìn thấy giá trị của **x**.
- Để giám sát giá trị của biến **message**, mở Data Segment ở hằng số 0x10010004 sẽ nhìn thấy giá trị của **message**.



A window titled 'Labels' showing a list of labels and their addresses. The label 'x' is at address 0x10010000 and 'message' is at address 0x10010004. At the bottom, there are checkboxes for 'Data' and 'Text', both of which are checked.

Label	Address ▲
mips1.asm	
x	0x10010000
message	0x10010004

☒ Data ☒ Text



A window titled 'Data Segment' showing a table of memory values. The first two columns are 'Address' and 'Value (+0)'. The value at address 0x10010000 is 0x01020304, and the value at address 0x10010004 is 0x6d206f42. Both values are highlighted with red boxes. The table has columns for values at intervals of 4 bytes (+0, +4, +8, +c, +10, +14, +18, +1c). At the bottom, there are navigation arrows, a dropdown menu showing '0x10010000 (.data)', and checkboxes for 'Hexadecimal Addresses' (checked), 'Hexadecimal Values' (checked), and 'ASCII' (unchecked).

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

Chạy giả lập

1. Tiếp tục chạy chương trình HelloWorld ở trên.

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: la \$a0, message #Dua dia chi bien m...
<input type="checkbox"/>	0x00400004	0x34240004	ori \$4,\$1,0x00000004	
<input type="checkbox"/>	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 #Gan thanh ghi \$v0 = 4
<input type="checkbox"/>	0x0040000c	0x0000000c	syscall	8: syscall #Goi ham so v0, ham...
<input type="checkbox"/>	0x00400010	0x20090002	addi \$9,\$0,0x00000002	10: addi \$t1,\$zero,2 #Thanh ghi \$t1 = 2
<input type="checkbox"/>	0x00400014	0x200a0003	addi \$10,\$0,0x00000003	11: addi \$t2,\$zero,3 #Thanh ghi \$t2 = 3
<input type="checkbox"/>	0x00400018	0x012a4020	add \$8,\$9,\$10	12: add \$t0, \$t1, \$t2 #Thanh ghi t- = \$t1...

Labels

Label	Address ▲
mips1.asm	
x	0x10010000
message	0x10010004

☒ Data☒ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

←→

0x10010000 (.data)

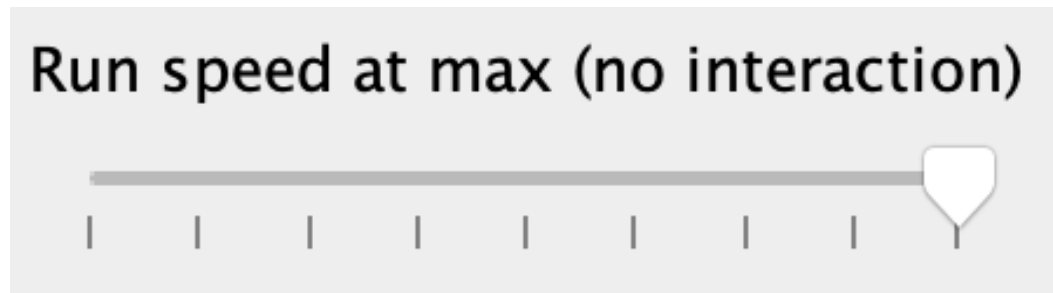
Hexadecimal Addresses

Hexadecimal Values

ASCII

Chạy giả lập





2. Sử dụng slider bar để thay đổi tốc độ thực thi lệnh hợp ngữ.



- Mặc định, tốc độ thực thi là tối đa, và ở mức này, người dùng không thể can thiệp được nhiều vào quá trình hoạt động của các lệnh và kiểm soát chúng.
- Có thể dịch chuyển slider bar về khoảng 2 lệnh/giây để dễ quan sát.

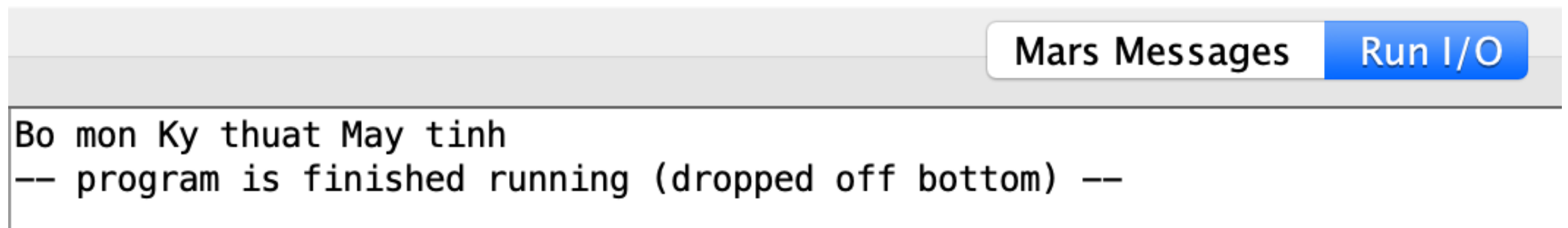
Chạy giả lập

2. Ở Execute tab, chọn cách để thực thi chương trình

- Bấm vào icon  Run, để thực hiện toàn bộ chương trình. Khi sử dụng Run, quan sát dòng lệnh được tô màu vàng cho biết chương trình hợp ngữ đang được xử lý tới chỗ nào. Đồng thời, quan sát sự biến đổi dữ liệu trong cửa sổ Data Segment và cửa sổ Register.
- Bấm vào icon  Reset, để khởi động lại trình giả lập về trạng thái ban đầu. Tất cả các ngăn nhớ và các thanh ghi đều được gán lại về 0.
- Bấm vào icon  Run one step, để thực thi chỉ duy nhất 1 lệnh rồi chờ bấm tiếp vào icon đó, để thực hiện lệnh kế tiếp.
- Bấm vào icon  Run one step backwards, để khôi phục lại trạng thái và quay trở lại lệnh đã thực thi trước đó.

Chạy giả lập

2. Ở Execute tab, chọn cách để thực thi chương trình
 - Sau khi chạy xong tất cả các lệnh của chương trình Hello Word, sẽ thấy cửa sổ Run I/O hiển thị chuỗi.



The screenshot shows a window titled "Mars Messages" with a "Run I/O" button. The output text is as follows:

```
Bo mon Ky thuat May tinh
-- program is finished running (dropped off bottom) --
```

Giả lập & gỡ rối

Quan sát sự thay đổi của các biến

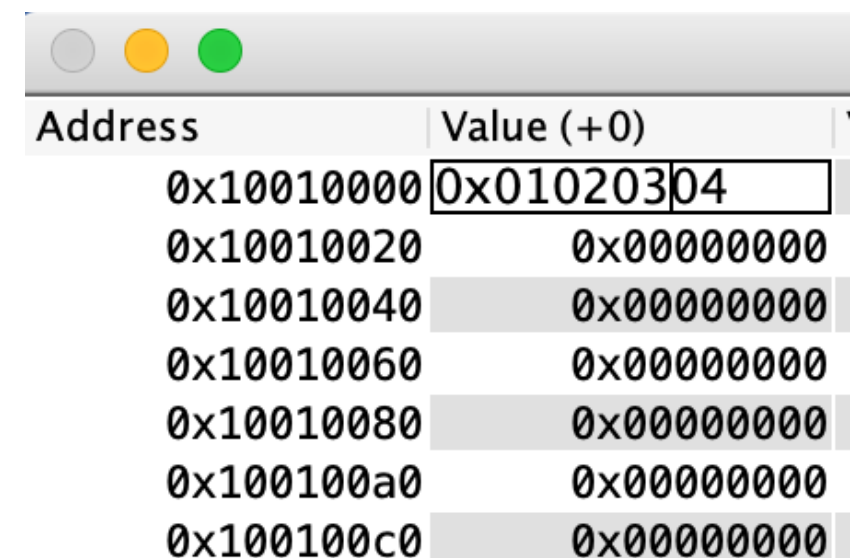
- Trong quá trình chạy giả lập, hãy chạy từng lệnh với chức năng Run one step.
- Ở mỗi lệnh, quan sát sự thay đổi giá trị trong cửa sổ Data Segment và cửa sổ Register, và hiểu rõ ý nghĩa của sự thay đổi đó.

Giả lập & gỡ rối

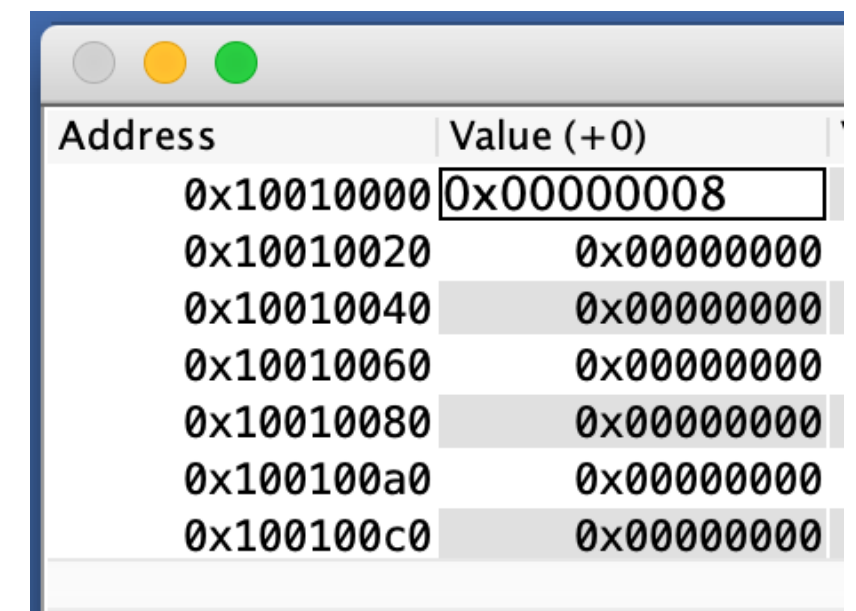
Thay đổi giá trị biến khi đang chạy run-time

Trong khi đang chạy giả lập, có thể thay đổi giá trị của một ngăn nhớ bất kì bằng cách:

- Trong Data Segment, click đúp vào một ngăn nhớ bất kì
- Nhập giá trị mới
- Tiếp tục chạy chương trình



Address	Value (+0)
0x10010000	0x01020304
0x10010020	0x00000000
0x10010040	0x00000000
0x10010060	0x00000000
0x10010080	0x00000000
0x100100a0	0x00000000
0x100100c0	0x00000000



Address	Value (+0)
0x10010000	0x00000008
0x10010020	0x00000000
0x10010040	0x00000000
0x10010060	0x00000000
0x10010080	0x00000000
0x100100a0	0x00000000
0x100100c0	0x00000000


Giả lập & gỡ rối

Thay đổi giá trị thanh ghi khi đang chạy run-time

Cách làm tương tự như thay đổi giá trị của biến, áp dụng cho cửa sổ Registers

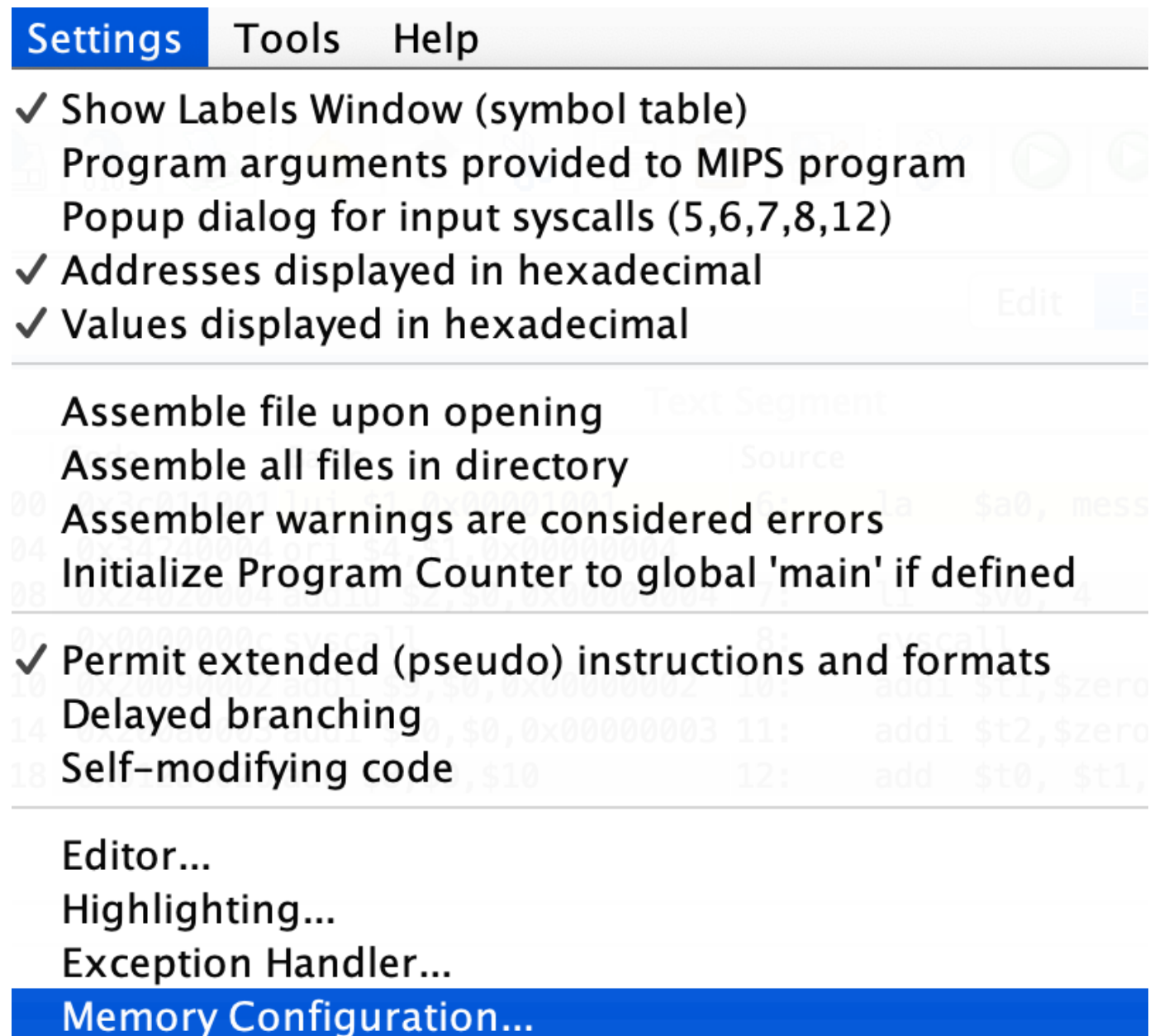
Registers			Coproc 1	Coproc 0
Name	Number	Value		
\$zero	0	0x00000000		
\$at	1	0x00000000		
\$v0	2	0x00000000		
\$v1	3	0x00000000		
\$a0	4	0x00000000		
\$a1	5	0x00000000		

Tra cứu Help

- Bấm nút Help  để xem giải thích các lệnh của MIPS, các giả lệnh, chỉ dẫn biên dịch và các hàm của syscalls.

Các hằng địa chỉ

- Chọn menu Settings / Memory Configuration



Các hằng địa chỉ

- Cửa sổ MIPS Memory Configuration chứa bảng qui định các hằng địa chỉ mà công cụ MARS sử dụng.
- Theo bảng này, có thể thấy các mã lệnh luôn bắt đầu từ địa chỉ 0x00400000, còn dữ liệu luôn bắt đầu từ địa chỉ 0x10000000.

MIPS Memory Configuration

0xffffffff	memory map limit address
0xffffffff	kernel space high address
0xffff0000	MMIO base address
0xffffefff	kernel data segment limit address
0x90000000	.kdata base address
0x8fffffff	kernel text limit address
0x80000180	exception handler address
0x80000000	kernel space base address
0x80000000	.ktext base address
0x7fffffff	user space high address
0x7fffffff	data segment limit address
0x7ffffffc	stack base address
0x7fffeffc	stack pointer \$sp
0x10040000	stack limit address
0x10040000	heap base address
0x10010000	.data base address
0x10008000	global pointer \$gp
0x10000000	data segment base address
0x10000000	.extern base address
0x0ffffffc	text limit address
0x00400000	.text base address

Configuration

☒ Default

☐ Compact, Data at Address 0

☐ Compact, Text at Address 0

Apply and Close Apply Cancel Reset

Bài 2

- Giới thiệu về các loại bộ nhớ
- Instruction Set
- Basic Instructions
- Directives

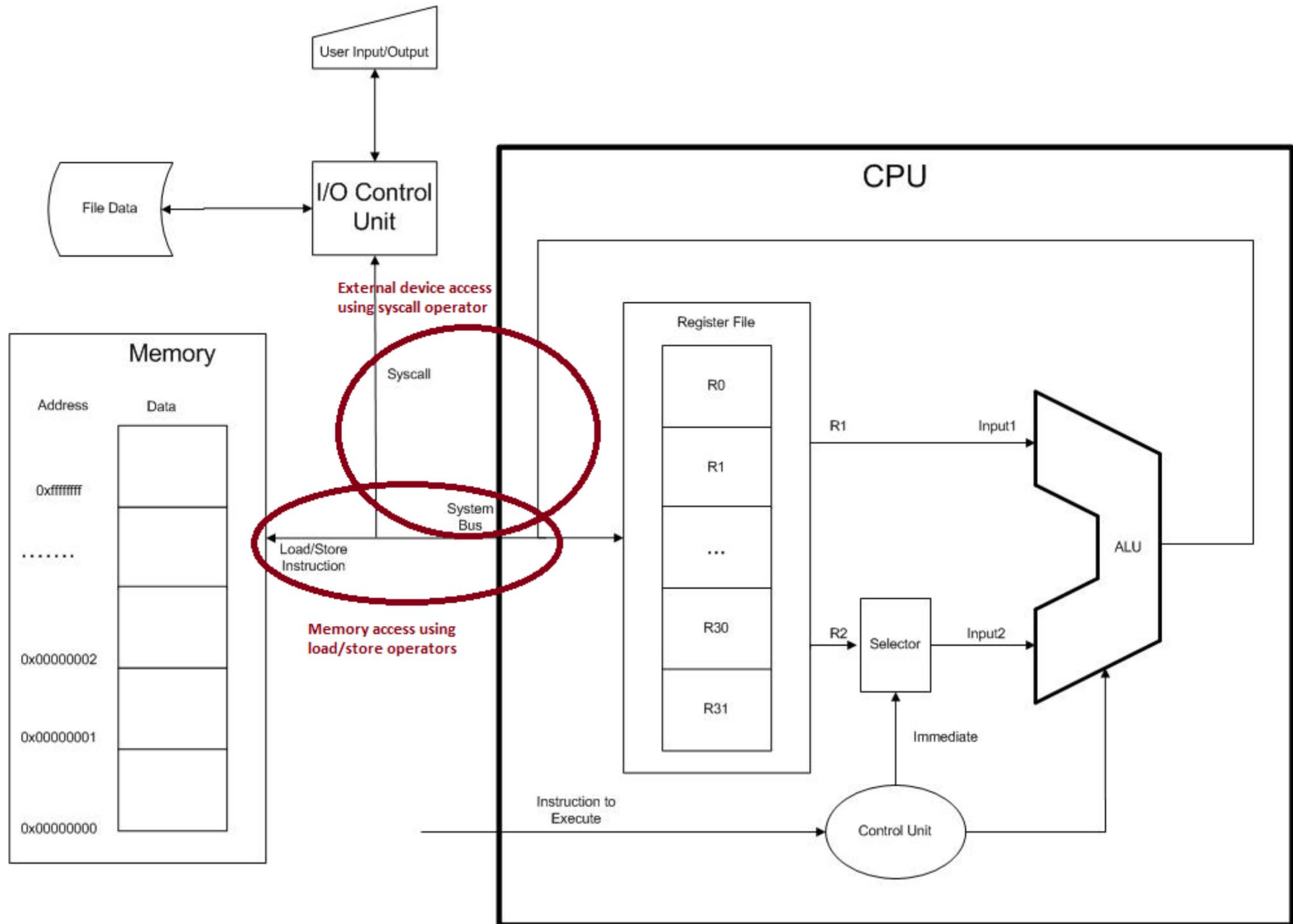
Mục tiêu

- Nguyên lý cơ bản về tập lệnh của bộ xử lý MIPS.
- Sử dụng được các lệnh hợp ngữ cơ bản và sử dụng công cụ gỡ rối để kiểm nghiệm lại các kiến thức về tập lệnh và hợp ngữ.
- Thành thạo với các chỉ thị biên dịch (Directives) để công cụ MARS có thể dịch hợp ngữ thành mã máy một cách đúng đắn.

Tài liệu

- Tài liệu tóm tắt về Kiến trúc MIPS
- Bảng tra cứu tập lệnh MIPS

Kiến trúc CPU MIPS



32 thanh ghi

\$zero (\$0):

- Dành cho mục đích sử dụng đặc biệt
- Luôn chứa giá trị 0
- Có thể được đọc nhưng không thể ghi.

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

\$at (\$1):

- Dành cho bộ hợp ngữ.
- Không dành cho người lập trình sử dụng.

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

\$v0-\$v1 (\$2-\$3)

- Sử dụng cho các giá trị trả về của các chương trình con.
- \$v0 cũng được sử dụng để đưa vào dịch vụ yêu cầu cho syscall.

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

\$a0-\$a3 (\$4-\$7)

- Được sử dụng để truyền đổi số (hoặc tham số) vào các chương trình con.

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

\$t0-\$t9 (\$8-\$15,
\$24-\$25)

- Được sử dụng để lưu các giá trị tạm thời.
- Các giá trị của các biến tạm thời có thể thay đổi khi các chương trình con bị gọi.

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

\$s0-\$s7 (\$16-\$23)

- Được sử dụng để lưu các giá trị nhớ.
- Các giá trị của những thanh ghi này được duy trì qua các lời gọi hàm con.

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

\$k0-\$k1 (\$26-\$27)

- Được sử dụng bởi hệ điều hành
- Không khả dụng đối với người lập trình.

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

\$gp (\$28)

- Con trỏ tới bộ nhớ toàn cục, sử dụng với cấp phát động (heap allocation)

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

\$sp (\$29):

- Con trỏ ngăn xếp
- Được sử dụng để trỏ đến điểm bắt đầu của dữ liệu trong ngăn xếp

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

\$fp (\$30)

- Con trỏ khung
- Sử dụng với \$sp để duy trì thông tin về ngăn xếp.

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

32 thanh ghi

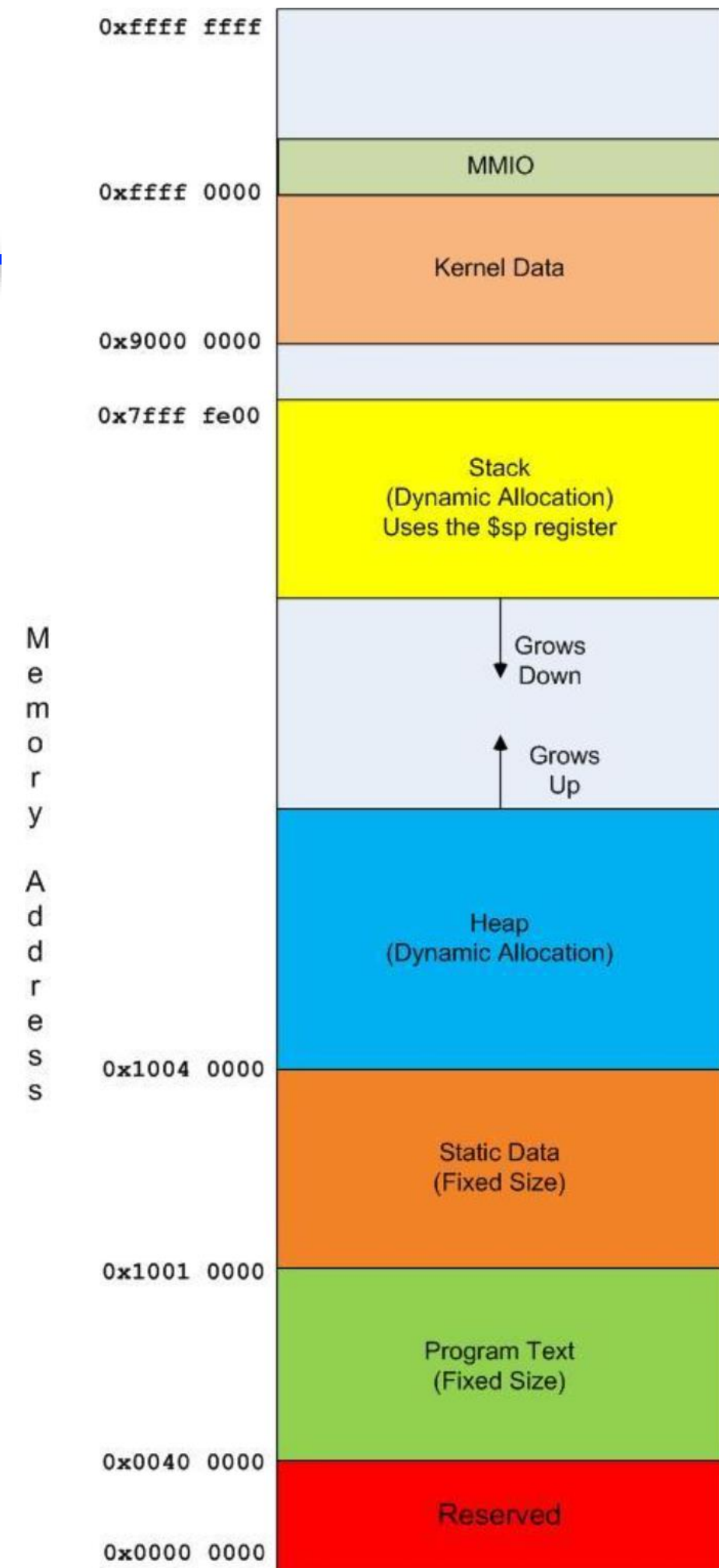
\$ra (\$31):

- Địa chỉ trả về
- Con trỏ trỏ tới địa chỉ sử dụng khi quay về từ 1 chương trình con.

Mnemonic	Number	Mnemonic	Number	Mnemonic	Number
\$zero	\$0		\$t3	\$11		\$s6	\$22
\$at	\$1		\$t4	\$12		\$s7	\$23
\$v0	\$2		\$t5	\$13		\$t8	\$24
\$v1	\$3		\$t6	\$14		\$t9	\$25
\$a0	\$4		\$t7	\$15		\$k0	\$26
\$a1	\$5		\$s0	\$16		\$k1	\$27
\$a2	\$6		\$s1	\$17		\$gp	\$28
\$a3	\$7		\$s2	\$18		\$sp	\$29
\$t0	\$8		\$s3	\$19		\$fp	\$30
\$t1	\$9		\$s4	\$20		\$ra	\$31
\$t2	\$10		\$s5	\$21			

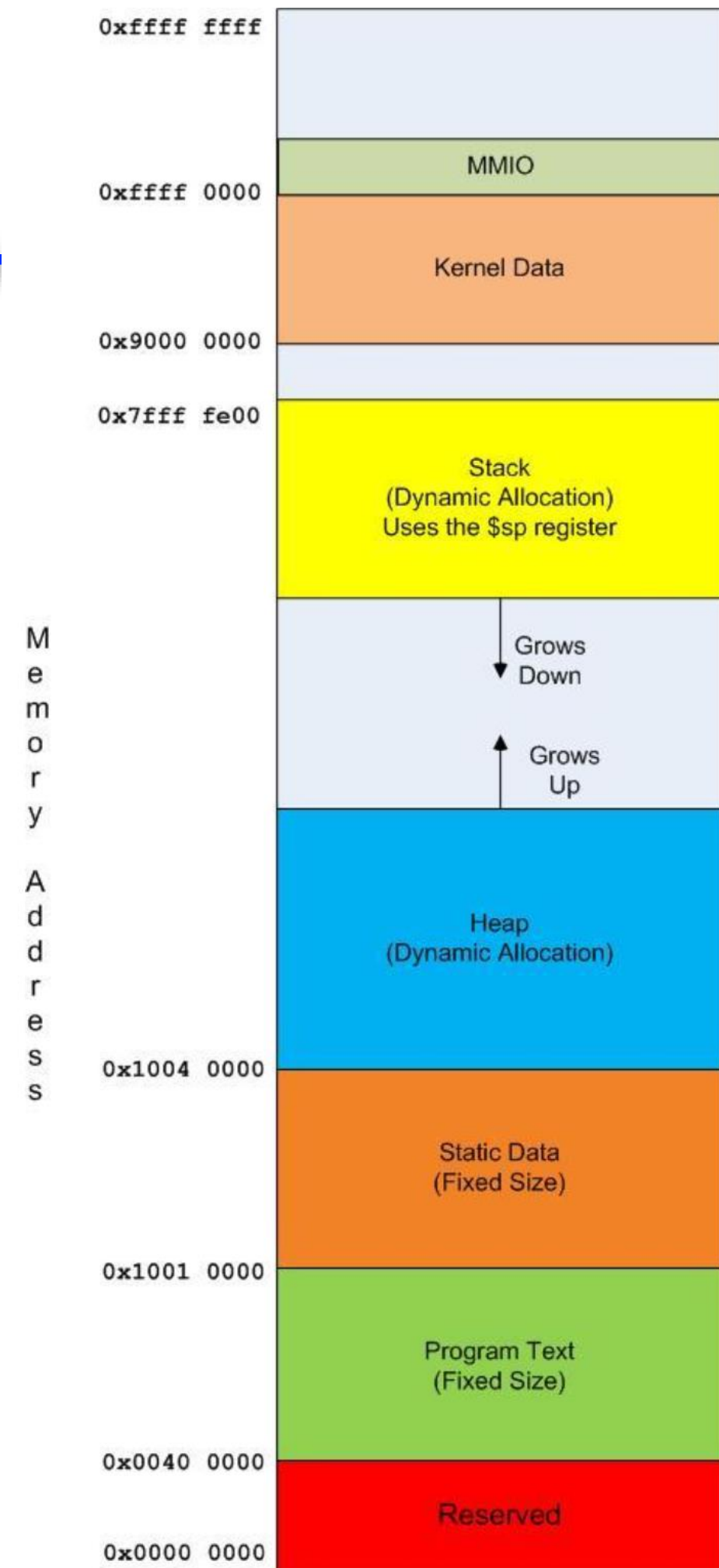
Bộ nhớ

- Mô hình địa chỉ 32 bit phẳng
- Có thể đánh địa chỉ cho 4GB dữ liệu
- Bắt đầu từ địa chỉ 0x00000000 đến 0xffffffff
- Không phải tất cả bộ nhớ đều khả dụng



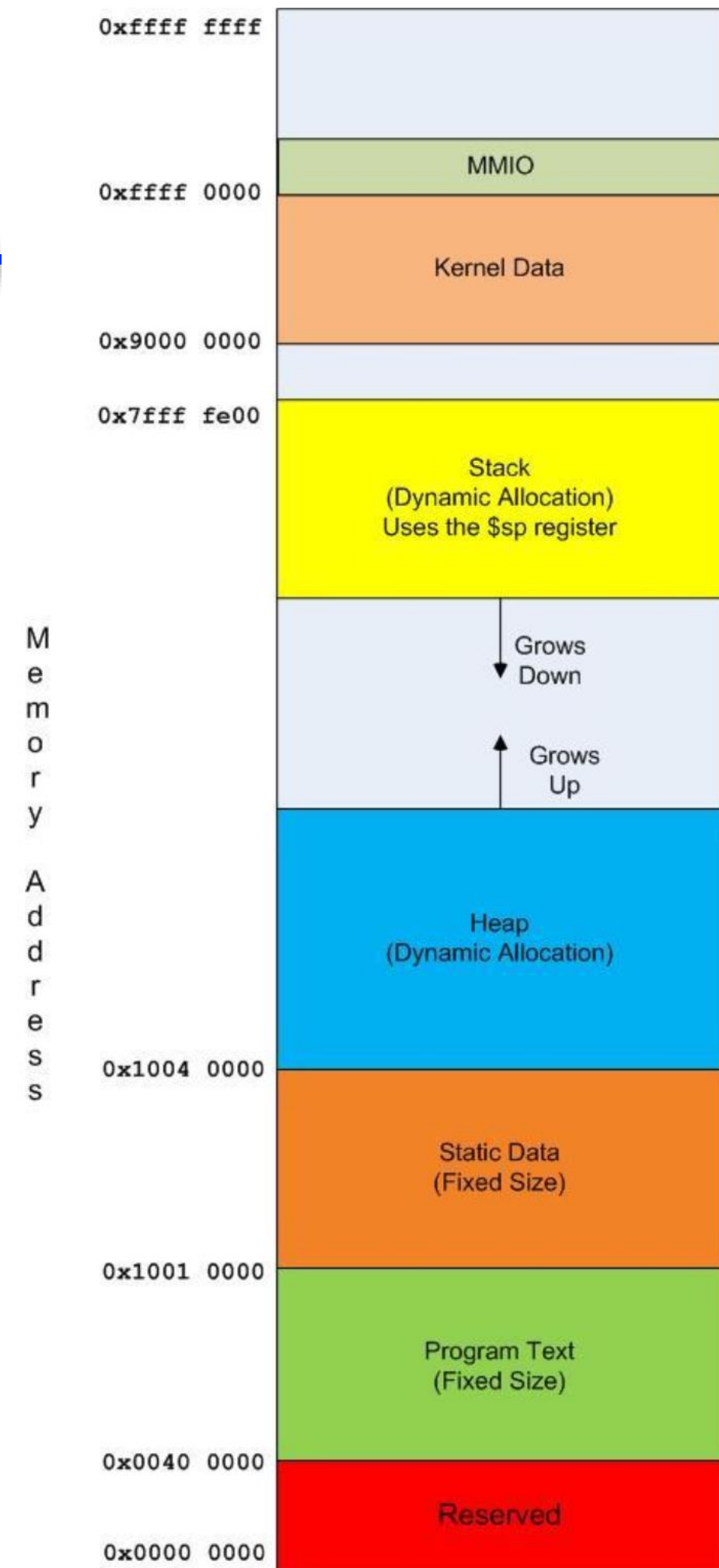
Bộ nhớ

- Reversed: Dự phòng:
- Bộ nhớ được dành cho nền tảng MIPS.
- Bộ nhớ ở những địa chỉ này không khả dụng đối với chương trình.



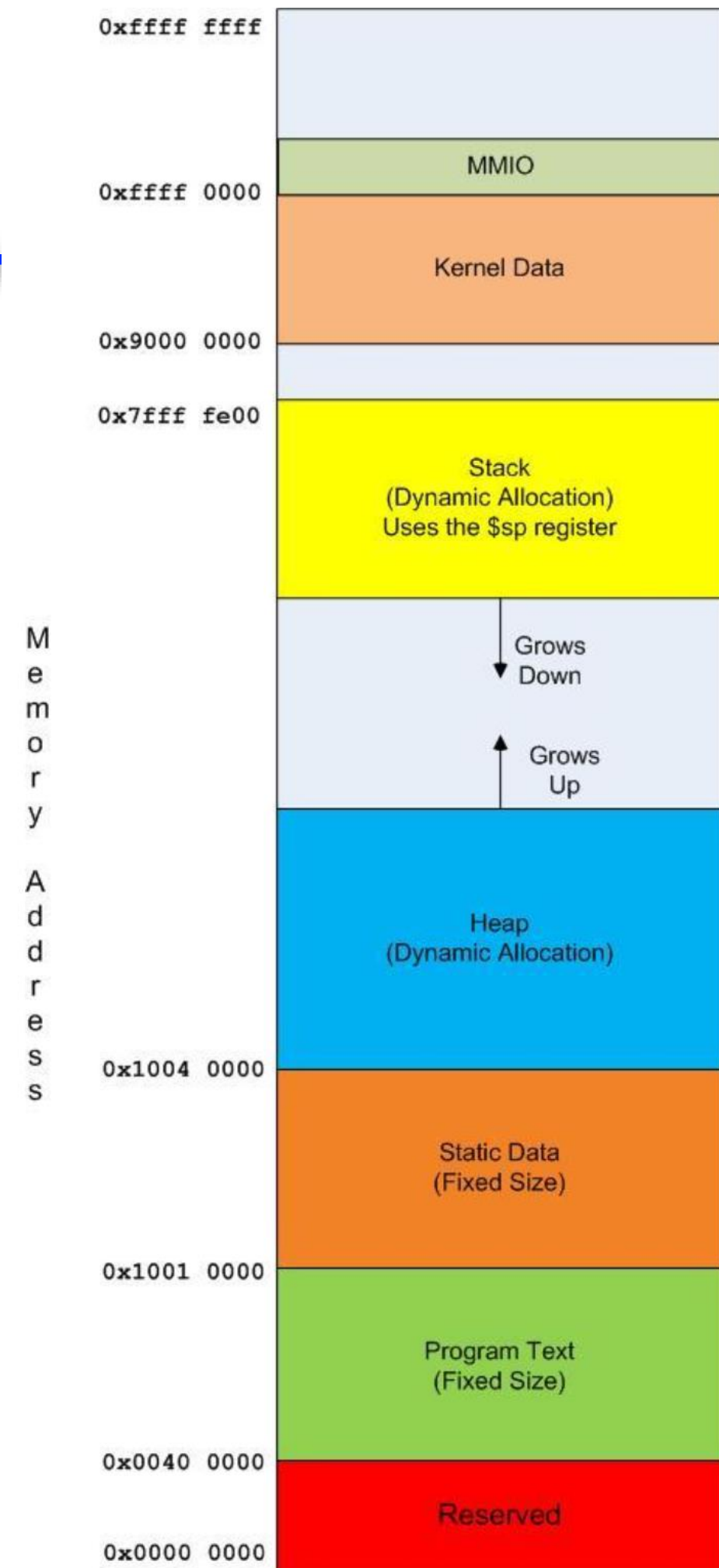
Bộ nhớ

- Program text: (các địa chỉ từ 0x0040 0000 - 0x1000 0000)
- Lưu trữ biểu diễn mã máy của một chương trình.
- Mỗi lệnh được lưu như là 1 từ (word 32 bit hay 4 byte) trong bộ nhớ.
- Tất cả các lệnh đều có giới hạn từ, là bội số của 4 (0x0040 0000, 0x0040 0004, 0x0040 0080, 0x0040 00B0...)



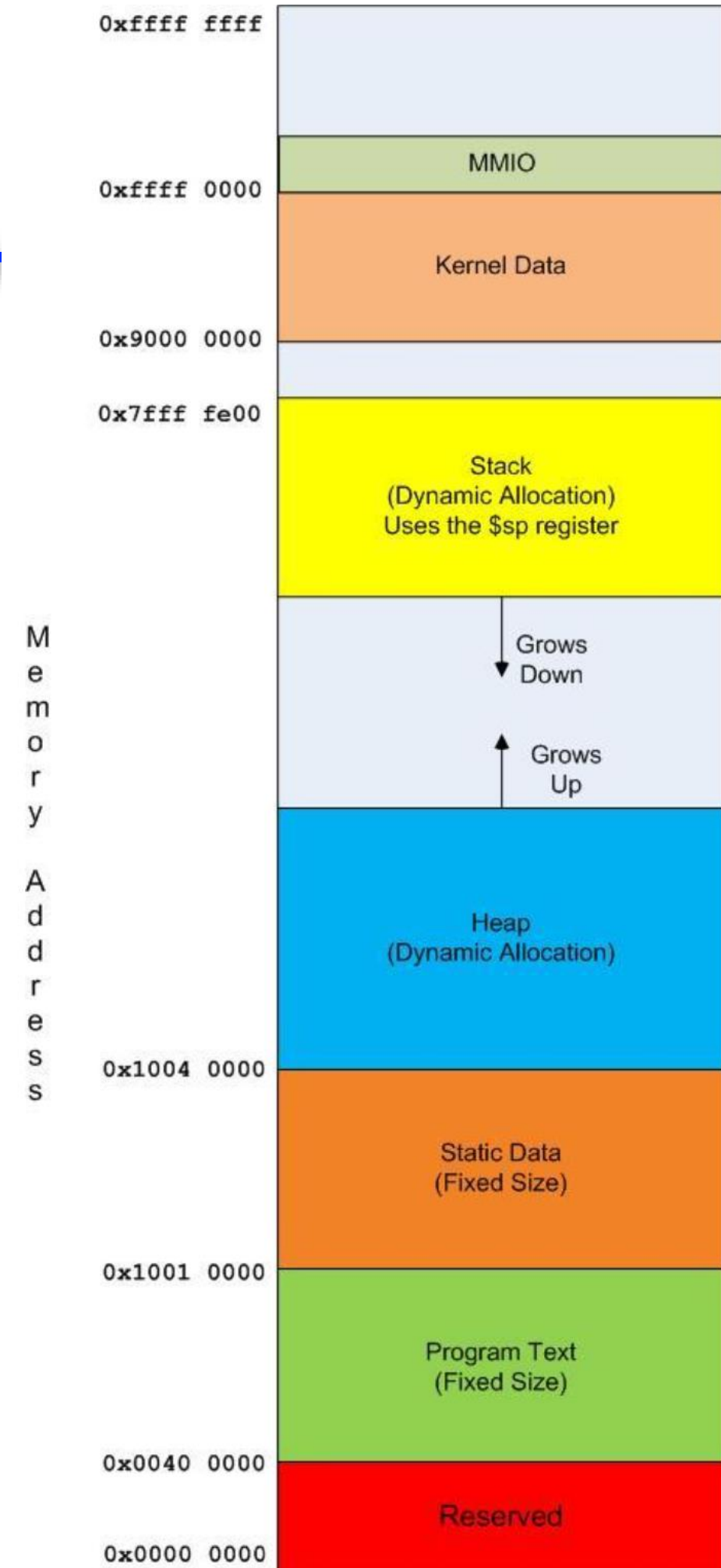
Bộ nhớ

- Static data: dữ liệu tĩnh (các địa chỉ từ 0x1001 0000 - 0x1004 0000)
- Dữ liệu đến từ data segment (phân đoạn dữ liệu) của chương trình.
- Kích thước các phần tử trong vùng này được gán khi chương trình được tạo ra (hợp dịch và liên kết) và không thể thay đổi trong suốt quá trình thực thi của chương trình.



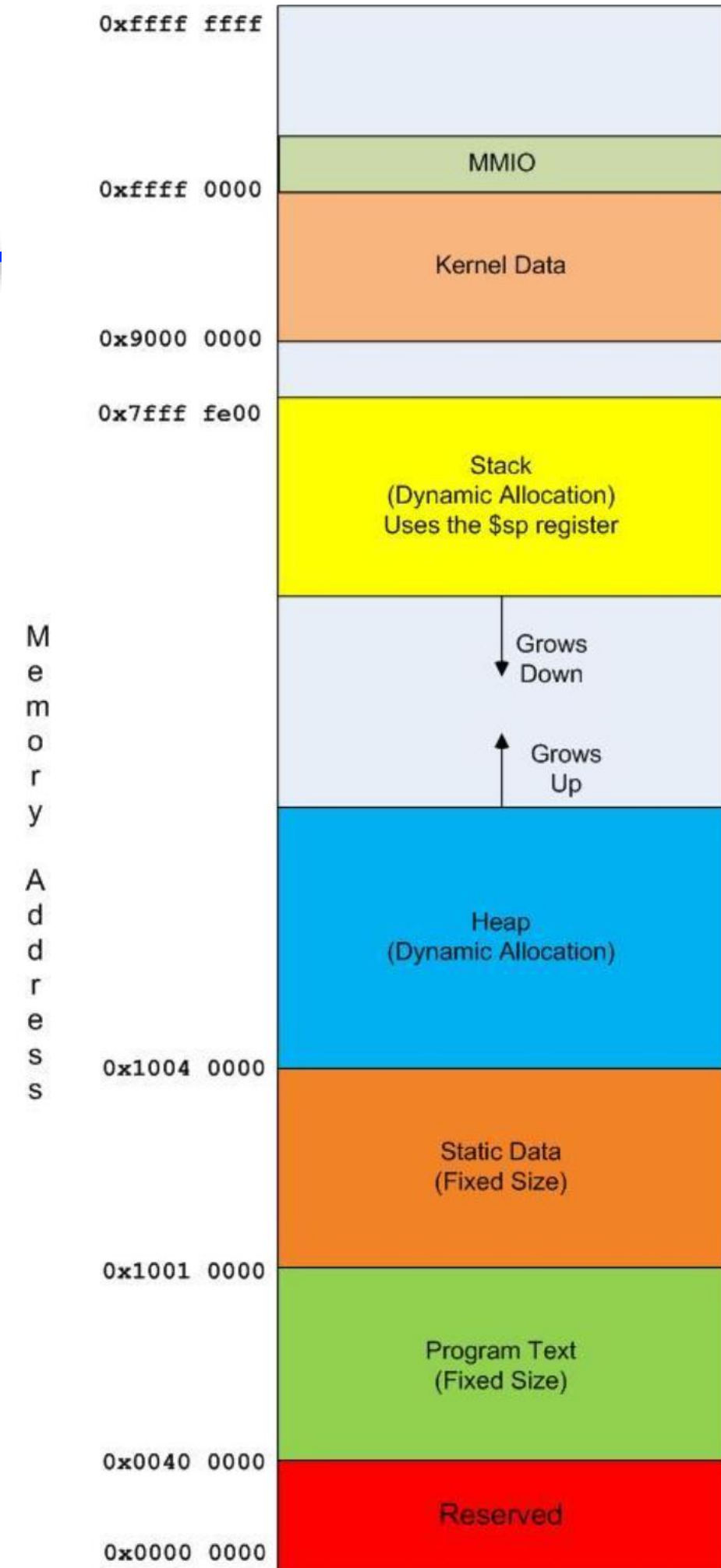
Bộ nhớ

- Heap – (địa chỉ từ 0x1004 0000 – cho đến khi tới được dữ liệu stack, đi lên)
- Là dữ liệu động được cấp phát nếu cần thiết trong quá trình chạy chương trình.
- Cách bộ nhớ này được cấp phát và khai báo tùy thuộc vào ngôn ngữ cụ thể.
- Dữ liệu trong heap luôn luôn là dữ liệu toàn thể.



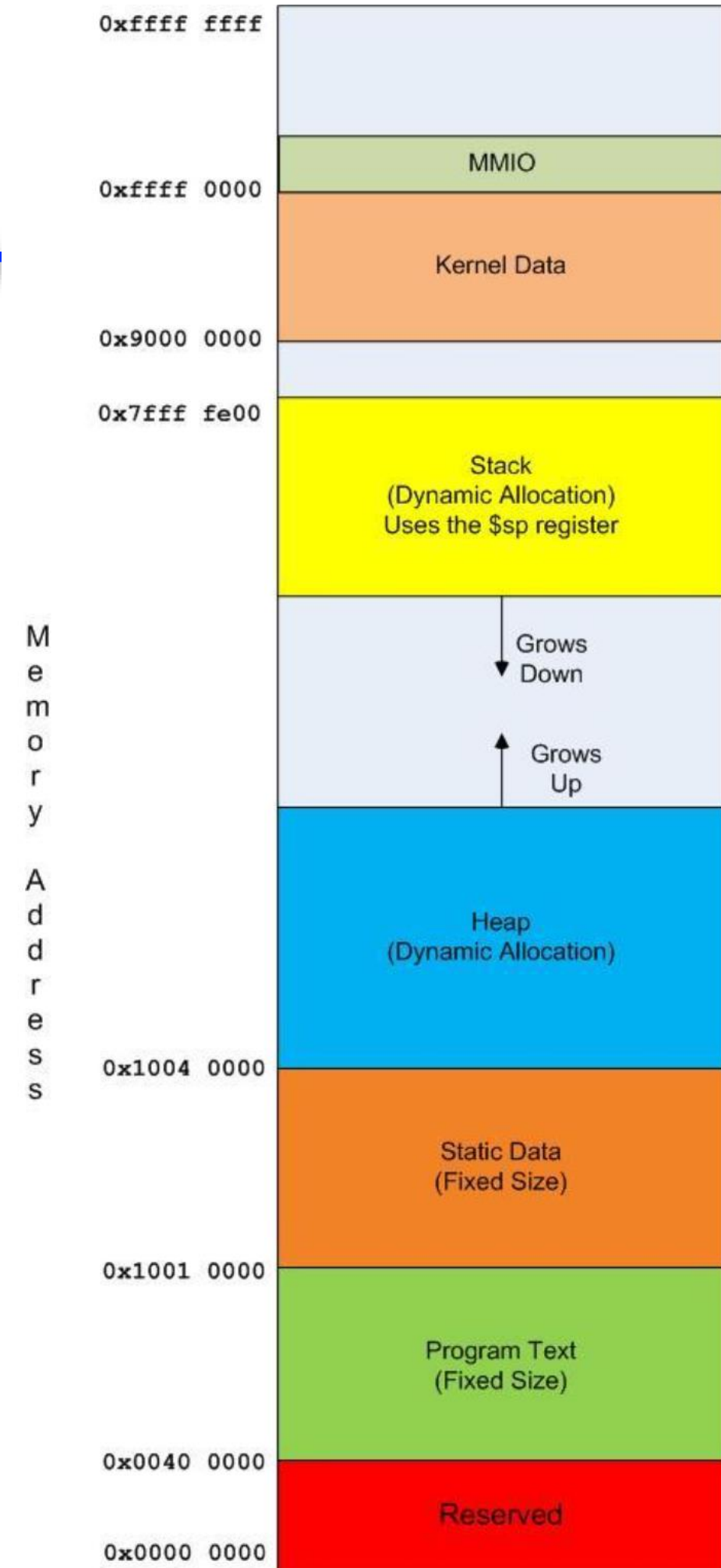
Bộ nhớ

- Stack - (địa chỉ từ 0x7fff fe00 – cho đến khi tới được dữ liệu heap, đi xuống).
- Ngăn xếp chương trình là dữ liệu động được cấp phát cho các chương trình con theo toán tử push và pop.
- Tất cả các phương thức biến cục bộ được lưu trữ tại đây. Bởi vì tính tự nhiên của toán tử pop và push, kích thước của bản ghi ngăn xếp khi được tạo ra cần phải được biết trước khi chương trình được hợp dịch.



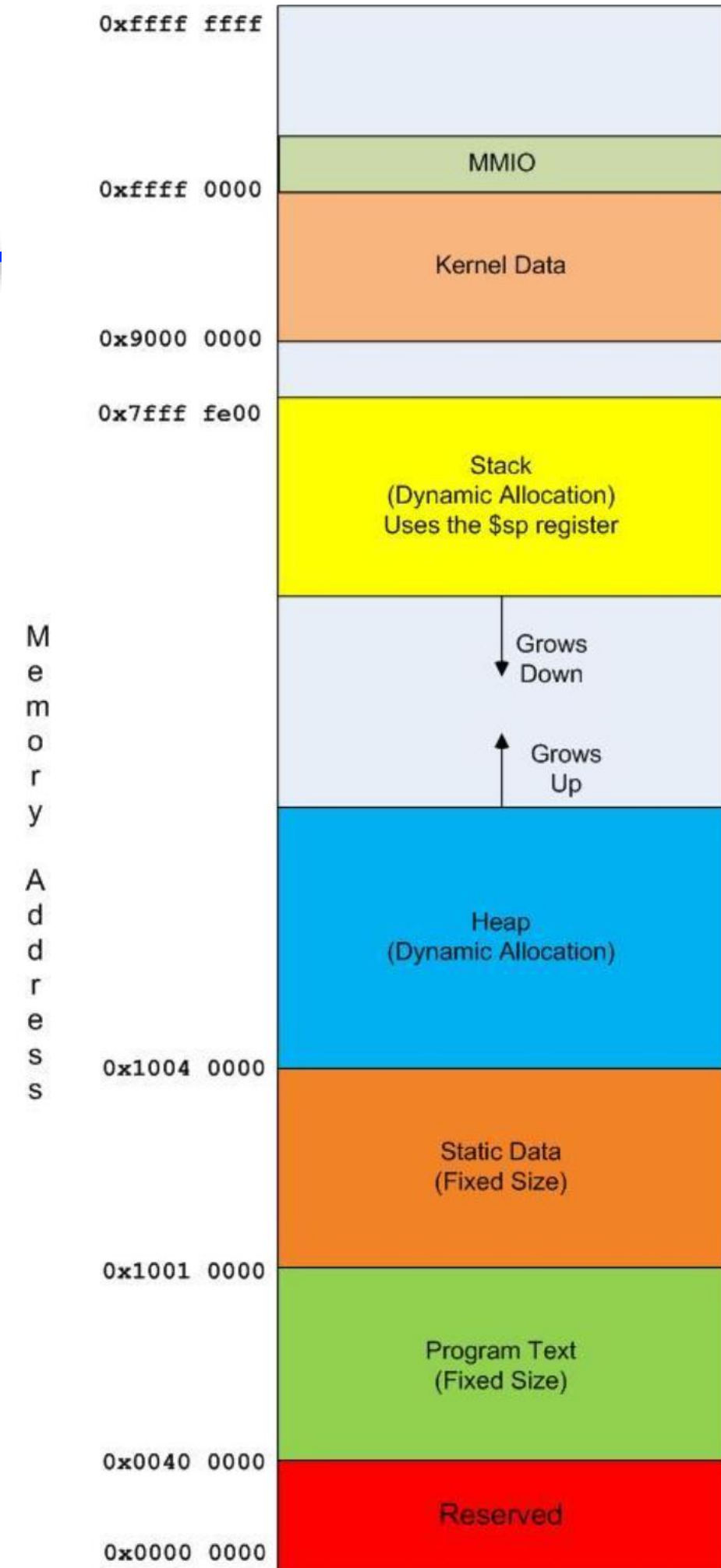
Bộ nhớ

- Kernel: (các địa chỉ từ 0x9000 0000 - 0xffff 0000)
- Bộ nhớ nhân kernel được sử dụng bởi hệ điều hành và do đó người sử dụng không được quyền truy cập.



Bộ nhớ

- MMIO: (các địa chỉ từ 0xffff 0000 - 0xffff 0010):
- Bộ nhớ ánh xạ ra/vào: được sử dụng cho bất kỳ loại dữ liệu bên ngoài nào không ở trong bộ nhớ như là màn hình, ổ đĩa, console...



Chương trình đầu tiên

- Cách chú thích một chương trình MIPS
- Thanh ghi và bộ nhớ trong máy tính MIPS
- Các chỉ thị hợp ngữ như **.text**, **.data**, **.asciiz** và **.word**
- Nhãn trong MIPS
- Các toán tử hợp ngữ MIPS như **li**, **la**, **lw** và **move**
- Các dịch vụ hệ thống để giao tiếp với console người dùng, cụ thể là các dịch vụ 1, 4, 5 và 8
- Sự khác nhau giữa giá trị tham chiếu và tham trị của dữ liệu

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Mã trình biên dịch MIPS có thể được thực lệ và bỏ trống khoảng trắng trên 1 dòng.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Tất cả câu lệnh phải nằm trên cùng 1 dòng duy nhất.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Dấu # có nghĩa là bất kỳ ký tự nào từ dấu # cho đến hết dòng là chú thích và có thể được bỏ qua.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Chuỗi được biểu thị bằng dấu “” xung quanh chuỗi.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                                # Label to define the main program.
9      li $v0,4                          # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                  # Load the address of the greeting into $a0.
11     syscall                          # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                        # Load a 10 (halt) into $v0.
15     syscall                          # The program ends.
```

- Ghi chú lại các chú thích ở đầu mỗi tệp tin. Các ghi chú này được gọi là phần mở đầu trong chương trình này.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Các chương trình hợp ngữ không được **biên dịch**, chúng được **hợp dịch**.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Dấu “.” trước một chuỗi văn bản có nghĩa là mã (chuỗi) tiếp theo là một chỉ thị của bộ hợp ngữ.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .ascii "Hello World"      #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                                # Label to define the main program.
9      li $v0,4                          # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                  # Load the address of the greeting into $a0.
11     syscall                          # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                        # Load a 10 (halt) into $v0.
15     syscall                          # The program ends.
```

- Chỉ thị **.text** có nghĩa là các lệnh tiếp theo thuộc về văn bản chương trình (gọi tắt là chương trình), được tập hợp thành một chương trình và lưu trữ trong vùng văn bản của bộ nhớ.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Chỉ thị **.data** có nghĩa là những gì tiếp theo là dữ liệu chương trình và được lưu trữ trong vùng dữ liệu tĩnh của bộ nhớ.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .ascii "Hello World"      #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                                # Label to define the main program.
9      li $v0, 4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                          # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                       # Load a 10 (halt) into $v0.
15     syscall                          # The program ends.
```

- Chỉ thị **.ascii** yêu cầu trình biên dịch dịch dữ liệu tiếp theo như là một chuỗi ký tự ASCII.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                                # Label to define the main program.
9      li $v0,4                          # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                  # Load the address of the greeting into $a0.
11     syscall                          # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                        # Load a 10 (halt) into $v0.
15     syscall                          # The program ends.
```

- Chuỗi văn bản nào theo sau là dấu “:” là nhãn.
- Một nhãn chỉ là một điểm đánh dấu trong chương trình mà có thể được sử dụng trong các câu lệnh khác.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                                # Label to define the main program.
9      li $v0,4                          # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                  # Load the address of the greeting into $a0.
11     syscall                          # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                        # Load a 10 (halt) into $v0.
15     syscall                          # The program ends.
```

- Nhãn **main**: không cần phải có trong chương trình vì MARS giả thiết rằng chương trình bắt đầu từ dòng đầu tiên trong chương trình hợp ngữ. Tuy nhiên, sẽ tốt hơn nếu ta gán nhãn cho điểm bắt đầu vì phần lớn thời gian chạy sẽ tìm kiếm một tên biểu trưng toàn thể **main** như là nơi để bắt đầu thực thi chương trình.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World" #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Bất kỳ khi nào một hằng số có trong một lệnh, nó được gọi là giá trị tức thời (Immediate I).

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Chỉ có các lệnh và nhãn có thể được định nghĩa trong một vùng văn bản và chỉ có dữ liệu và nhãn có thể được định nghĩa trong một vùng dữ liệu.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Các toán tử là các chuỗi ký tự giống như **li**, **la**, và **syscall**.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Lệnh bao gồm toán tử và đối số của nó.

Chương trình đầu tiên

```
1  # Program File: Program1-1.asm
2  # Author: NTTNga
3  # Purpose: First program, Hello World
4  .data                                # Define the program data.
5  greeting: .asciiz "Hello World"    #The string to print.
6
7  .text                                # Define the program instructions.
8  main:                               # Label to define the main program.
9      li $v0,4                        # Load 4 into $v0 to indicate a print string.
10     la $a0, greeting                # Load the address of the greeting into $a0.
11     syscall                         # Print greeting. The print is indicated by
12                                     # $v0 having a value of 4, and the string to
13                                     # print is stored at the address in $a0.
14     li $v0, 10                      # Load a 10 (halt) into $v0.
15     syscall                         # The program ends.
```

- Toán tử syscall được sử dụng để gọi các dịch vụ hệ thống.

Tra cứu trong HELP

The screenshot shows the MARS 4.5 Help window. The title bar is "MARS 4.5 Help". The menu bar includes "MIPS", "MARS", "License", "Bugs/Comments", "Acknowledgements", and "Instruction Set Song". The "MIPS" menu is open, showing a list of topics: "Basic Instructions", "Extended (pseudo) Instructions", "Directives", "Syscalls" (highlighted), "Exceptions", and "Macros".

Below the menu bar, there is a section titled "Operand Key for Example Instructions" with a light green background. It lists the following keys and their meanings:

- label, target: any textual label
- \$t1, \$t2, \$t3: any integer register
- \$f2, \$f4, \$f6: even-numbered floating point register
- \$f0, \$f1, \$f3: any floating point register

Below this section, there is a code block showing the following assembly code:

```
li $v0, 1          # service 1 is print integer
add $a0, $t0, $zero # load desired value into argument register $a0, using pseudo-op
syscall
```

Below the code block, there is a section titled "Table of Available Services" with a table containing the following data:

Service	Code in \$v0	Arguments	Result
print integer	1	\$a0 = integer to print	
print float	2	\$f12 = float to print	
print double	3	\$f12 = double to print	
print string	4	\$a0 = address of null-terminated string to print	
read integer	5		\$v0 contains integer read
read float	6		\$f0 contains float read
read double	7		\$f0 contains double read
read string	8	\$a0 = address of input buffer \$a1 = maximum number of characters to read	See note below table

At the bottom of the window, there is a "Close" button.

Chương trình nhắc và đọc vào một số nguyên từ người dùng

```
1  # Program File: Program1-2.asm
2  # Author: NTTNga
3  # Program to read an integer number from a user, and
4  # print that number back to the console.
5
6  .data
7  prompt: .asciiz "Hay nhap vao mot so nguyen: "
8  output: .asciiz "\nBan da nhap vao so: "
9
10 .text
11 main:
12     # Prompt for the integer to enter
13     li $v0, 4
14     la $a0, prompt
15     syscall
16
17     # Read the integer and save it in $s0
18     li $v0, 5
19     syscall
20     move $s0, $v0
21
22     # Output the text
23     li $v0, 4
24     la $a0, output
25     syscall
26
27     # Output the number
28     li $v0, 1
29     move $a0, $s0
30     syscall
31
32     # Exit the program
33     li $v0, 10
34     syscall
```

- Các khối lệnh được chú thích, không phải từng lệnh.
- Mỗi khối lệnh nên được chú thích theo mục đích hoặc kết quả.

Chương trình nhắc và đọc vào một số nguyên từ người dùng

```
1  # Program File: Program1-2.asm
2  # Author: NTTNga
3  # Program to read an integer number from a user, and
4  # print that number back to the console.
5
6  .data
7  prompt: .asciiz "Hay nhap vao mot so nguyen: "
8  output: .asciiz "\nBan da nhap vao so: "
9
10 .text
11 main:
12     # Prompt for the integer to enter
13     li $v0, 4
14     la $a0, prompt
15     syscall
16
17     # Read the integer and save it in $s0
18     li $v0, 5
19     syscall
20     move $s0, $v0
21
22     # Output the text
23     li $v0, 4
24     la $a0, output
25     syscall
26
27     # Output the number
28     li $v0, 1
29     move $a0, $s0
30     syscall
31
32     # Exit the program
33     li $v0, 10
34     syscall
```

- Toán tử move
- Chuyển văn bản từ một thanh ghi này sang một thanh ghi khác.

Chương trình nhắc và đọc vào một số nguyên từ người dùng

```
1  # Program File: Program1-2.asm
2  # Author: NTTNga
3  # Program to read an integer number from a user, and
4  # print that number back to the console.
5
6  .data
7  prompt: .asciiz "Hay nhap vao mot so nguyen: "
8  output: .asciiz "\nBan da nhap vao so: "
9
10 .text
11 main:
12     # Prompt for the integer to enter
13     li $v0, 4
14     la $a0, prompt
15     syscall
16
17     # Read the integer and save it in $s0
18     li $v0, 5
19     syscall
20     move $s0, $v0
21
22     # Output the text
23     li $v0, 4
24     la $a0, output
25     syscall
26
27     # Output the number
28     li $v0, 1
29     move $a0, $s0
30     syscall
31
32     # Exit the program
33     li $v0, 10
34     syscall
```

- Dịch vụ 5 chờ đồng bộ để người dùng nhập vào một số nguyên trên console
- Khi số nguyên được nhập sẽ trả về trong thanh ghi \$v0.
- Kiểm tra xem giá trị nhập có phải là một giá trị nguyên không
- Đưa ra một ngoại lệ nếu không phải.

Chương trình nhắc và đọc vào một số nguyên từ người dùng

```
1  # Program File: Program1-2.asm
2  # Author: NTTNga
3  # Program to read an integer number from a user, and
4  # print that number back to the console.
5
6  .data
7  prompt: .asciiz "Hay nhap vao mot so nguyen: "
8  output: .asciiz "\nBan da nhap vao so: "
9
10 .text
11 main:
12     # Prompt for the integer to enter
13     li $v0, 4
14     la $a0, prompt
15     syscall
16
17     # Read the integer and save it in $s0
18     li $v0, 5
19     syscall
20     move $s0, $v0
21
22     # Output the text
23     li $v0, 4
24     la $a0, output
25     syscall
26
27     # Output the number
28     li $v0, 1
29     move $a0, $s0
30     syscall
31
32     # Exit the program
33     li $v0, 10
34     syscall
```

- Dịch vụ 1 in ra giá trị số nguyên trong thanh ghi **\$a0**
- Chú ý:
 - Với dịch vụ 4, chuỗi ký tự lưu tại địa chỉ trong thanh ghi **\$a0** được in ra.
 - Với dịch vụ 1, giá trị trong thanh ghi **\$a0** cũng được in ra.

Chương trình nhắc và đọc vào một số nguyên từ người dùng

```
1  # Program File: Program1-2.asm
2  # Author: NTTNga
3  # Program to read an integer number from a user, and
4  # print that number back to the console.
5
6  .data
7  prompt: .asciiz "Hay nhap vao mot so nguyen: "
8  output: .asciiz "\nBan da nhap vao so: "
9
10 .text
11 main:
12     # Prompt for the integer to enter
13     li $v0, 4
14     la $a0, prompt
15     syscall
16
17     # Read the integer and save it in $s0
18     li $v0, 5
19     syscall
20     move $s0, $v0
21
22     # Output the text
23     li $v0, 4
24     la $a0, output
25     syscall
26
27     # Output the number
28     li $v0, 1
29     move $a0, $s0
30     syscall
31
32     # Exit the program
33     li $v0, 10
34     syscall
```

- Ký tự “\n” được sử dụng trong chuỗi ký tự được đặt tên là **output**.
- Được gọi là ký tự dòng mới, làm cho chuỗi **output** bắt đầu trên một dòng mới.

Chương trình nhắc và đọc vào một chuỗi từ người dùng

```
1  # Program File: Program1-3.asm
2  # Author: NTTNga
3  # Program to read a string from a user, and
4  # print that string back to the console.
5
6  .data
7  input: .space 81
8  inputSize: .word 80
9  prompt: .asciiz "Hay nhap vao mot chuoai: "
10 output: .asciiz "\nBan da nhap vao chuoai: "
11
12 .text
13 main:
14     # Prompt for the string to enter
15     li $v0, 4
16     la $a0, prompt
17     syscall
18
19     # Read the string.
20     li $v0, 8
21     la $a0, input
22     lw $a1, inputSize
23     syscall
24
25     # Output the text
26     li $v0, 4
27     la $a0, output
28     syscall
29
30     # Output the number
31     li $v0, 4
32     la $a0, input
33     syscall
34
35     # Exit the program
36     li $v0, 10
37     syscall
```

- Chỉ thị **.space** phân bổ **n bytes** của bộ nhớ trong vùng dữ liệu của chương trình với **n=81** trong chương trình này.
- Khi kích thước của 1 ký tự là 1 byte, thì điều này tương đương với lưu 80 ký tự dữ liệu.
- Tại sao 81 được sử dụng?

Chương trình nhắc và đọc vào một chuỗi từ người dùng

```
1  # Program File: Program1-3.asm
2  # Author: NTTNga
3  # Program to read a string from a user, and
4  # print that string back to the console.
5
6  .data
7  input: .space 81
8  inputSize: .word 80
9  prompt: .asciiz "Hay nhap vao mot chuoai: "
10 output: .asciiz "\nBan da nhap vao chuoai: "
11
12 .text
13 main:
14     # Prompt for the string to enter
15     li $v0, 4
16     la $a0, prompt
17     syscall
18
19     # Read the string.
20     li $v0, 8
21     la $a0, input
22     lw $a1, inputSize
23     syscall
24
25     # Output the text
26     li $v0, 4
27     la $a0, output
28     syscall
29
30     # Output the number
31     li $v0, 4
32     la $a0, input
33     syscall
34
35     # Exit the program
36     li $v0, 10
37     syscall
```

- Chỉ thị **.word** phân bổ 4 bytes không gian bộ nhớ trong vùng dữ liệu.
- Có thể được gán một giá trị số nguyên, và nó sẽ khởi tạo một vùng không gian bộ nhớ cho giá trị số nguyên đó.

Chương trình nhắc và đọc vào một chuỗi từ người dùng

```
1  # Program File: Program1-3.asm
2  # Author: NTTNga
3  # Program to read a string from a user, and
4  # print that string back to the console.
5
6  .data
7  input: .space 81
8  inputSize: .word 80
9  prompt: .asciiz "Hay nhap vao mot chuoai: "
10 output: .asciiz "\nBan da nhap vao chuoai: "
11
12 .text
13 main:
14     # Prompt for the string to enter
15     li $v0, 4
16     la $a0, prompt
17     syscall
18
19     # Read the string.
20     li $v0, 8
21     la $a0, input
22     lw $a1, inputSize
23     syscall
24
25     # Output the text
26     li $v0, 4
27     la $a0, output
28     syscall
29
30     # Output the number
31     li $v0, 4
32     la $a0, input
33     syscall
34
35     # Exit the program
36     li $v0, 10
37     syscall
```

- Toán tử **la** nạp địa chỉ của nhãn vào trong một thanh ghi.
- Được gọi là tham chiếu đến dữ liệu
- Được thể hiện bởi

\$a0 <= label

có nghĩa là giá trị của nhãn (địa chỉ bộ nhớ) được nạp vào trong một thanh ghi.

Chương trình nhắc và đọc vào một chuỗi từ người dùng

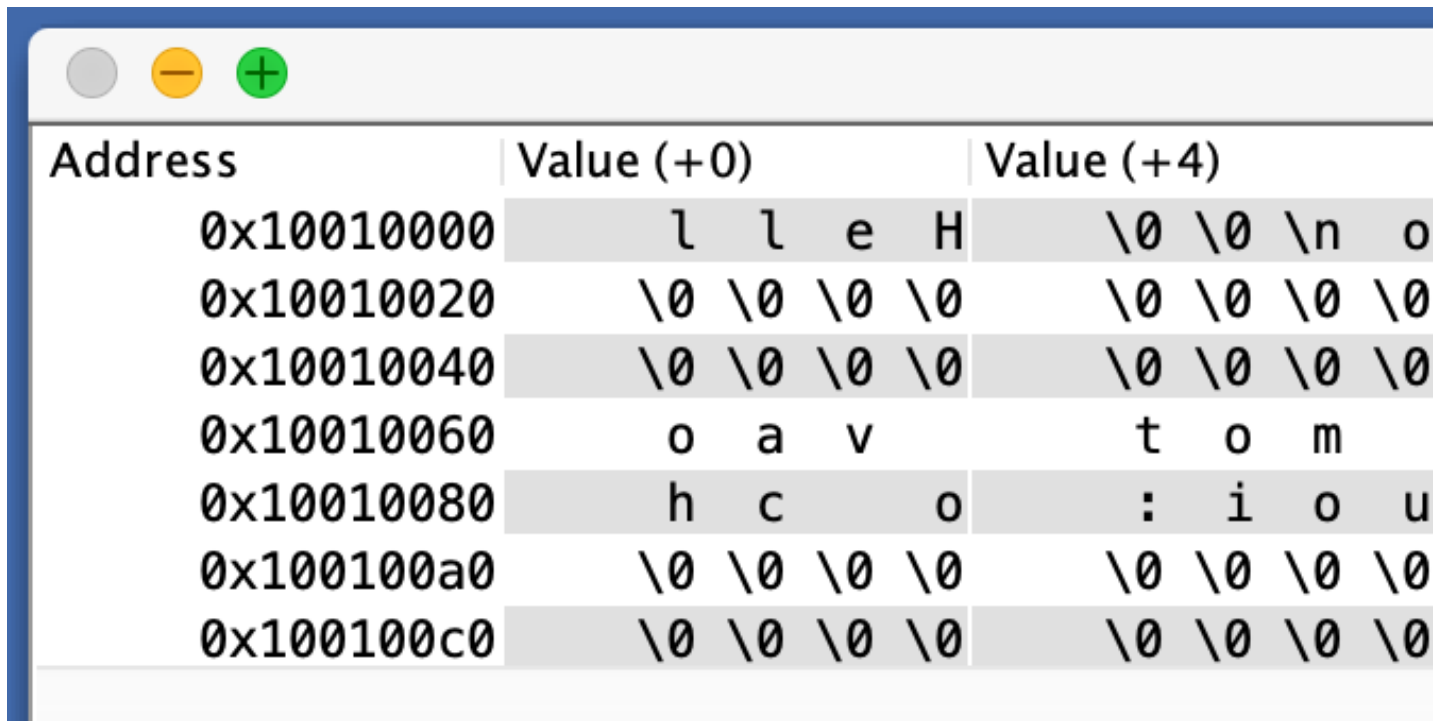
```
1  # Program File: Program1-3.asm
2  # Author: NTTNga
3  # Program to read a string from a user, and
4  # print that string back to the console.
5
6  .data
7  input: .space 81
8  inputSize: .word 80
9  prompt: .asciiz "Hay nhap vao mot chuoai: "
10 output: .asciiz "\nBan da nhap vao chuoai: "
11
12 .text
13 main:
14     # Prompt for the string to enter
15     li $v0, 4
16     la $a0, prompt
17     syscall
18
19     # Read the string.
20     li $v0, 8
21     la $a0, input
22     lw $a1, inputSize
23     syscall
24
25     # Output the text
26     li $v0, 4
27     la $a0, output
28     syscall
29
30     # Output the number
31     li $v0, 4
32     la $a0, input
33     syscall
34
35     # Exit the program
36     li $v0, 10
37     syscall
```

- Toán tử **lw** nạp giá trị chứa trong nhãn vào thanh ghi.
- Việc nạp giá trị vào trong thanh ghi được thể hiện như sau:

\$a1 <= M[label]

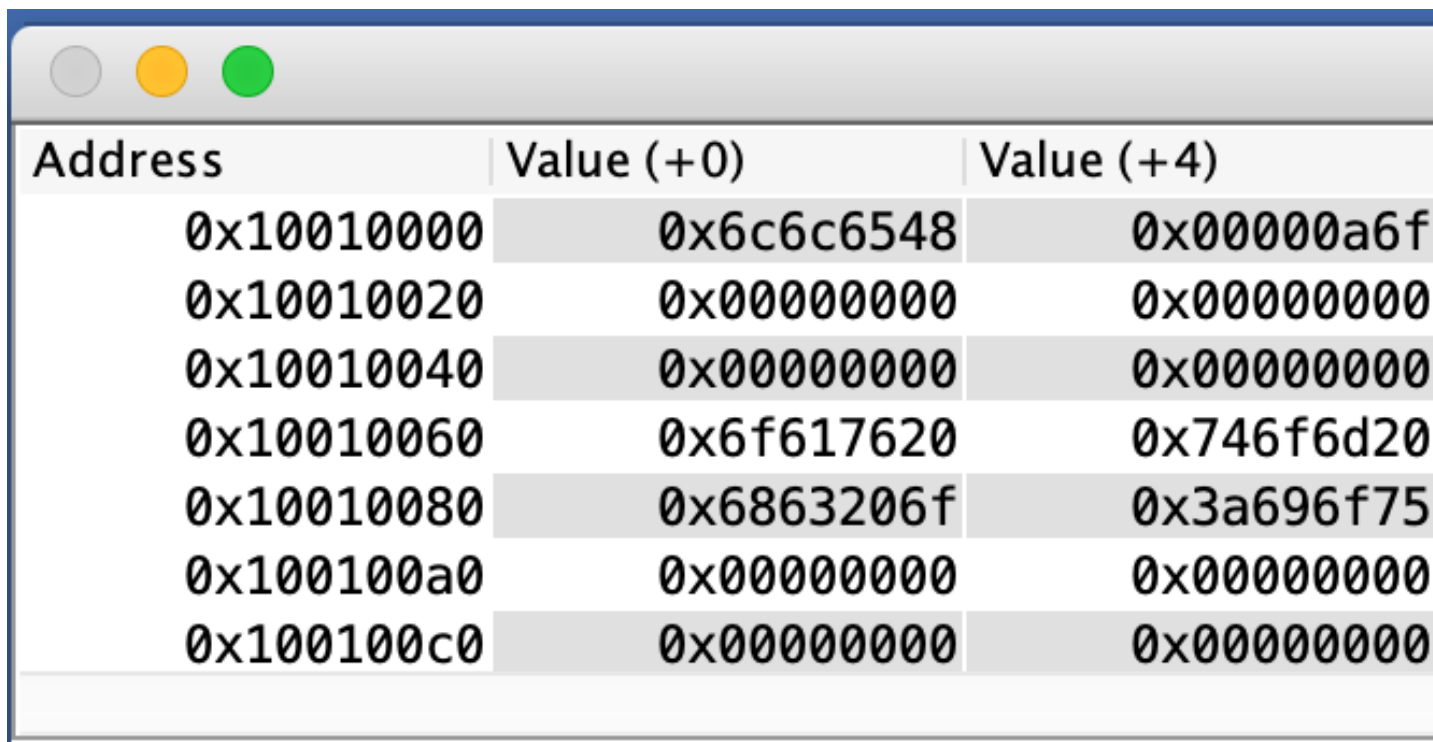
có nghĩa giá trị tại nhãn được nạp vào trong thanh ghi **\$a1**.

Chương trình nhắc và đọc vào một chuỗi từ người dùng



Address	Value (+0)	Value (+4)
0x10010000	l l e H	\0 \0 \n o
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	o a v	t o m
0x10010080	h c o	: i o u
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0

- Một chuỗi ký tự là một chuỗi tuần tự các ký tự ASCII được kết thúc bởi một giá trị rỗng.



Address	Value (+0)	Value (+4)
0x10010000	0x6c6c6548	0x00000a6f
0x10010020	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000
0x10010060	0x6f617620	0x746f6d20
0x10010080	0x6863206f	0x3a696f75
0x100100a0	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000

- Chuỗi 5 ký tự cần 6 bytes để lưu trữ.

Chương trình nhắc và đọc vào một chuỗi từ người dùng

```
1  # Program File: Program1-3.asm
2  # Author: NTTNga
3  # Program to read a string from a user, and
4  # print that string back to the console.
5
6  .data
7  input: .space 81
8  inputSize: .word 80
9  prompt: .asciiz "Hay nhap vao mot chuoai: "
10 output: .asciiz "\nBan da nhap vao chuoai: "
11
12 .text
13 main:
14     # Prompt for the string to enter
15     li $v0, 4
16     la $a0, prompt
17     syscall
18
19     # Read the string.
20     li $v0, 8
21     la $a0, input
22     lw $a1, inputSize
23     syscall
24
25     # Output the text
26     li $v0, 4
27     la $a0, output
28     syscall
29
30     # Output the number
31     li $v0, 4
32     la $a0, input
33     syscall
34
35     # Exit the program
36     li $v0, 10
37     syscall
```

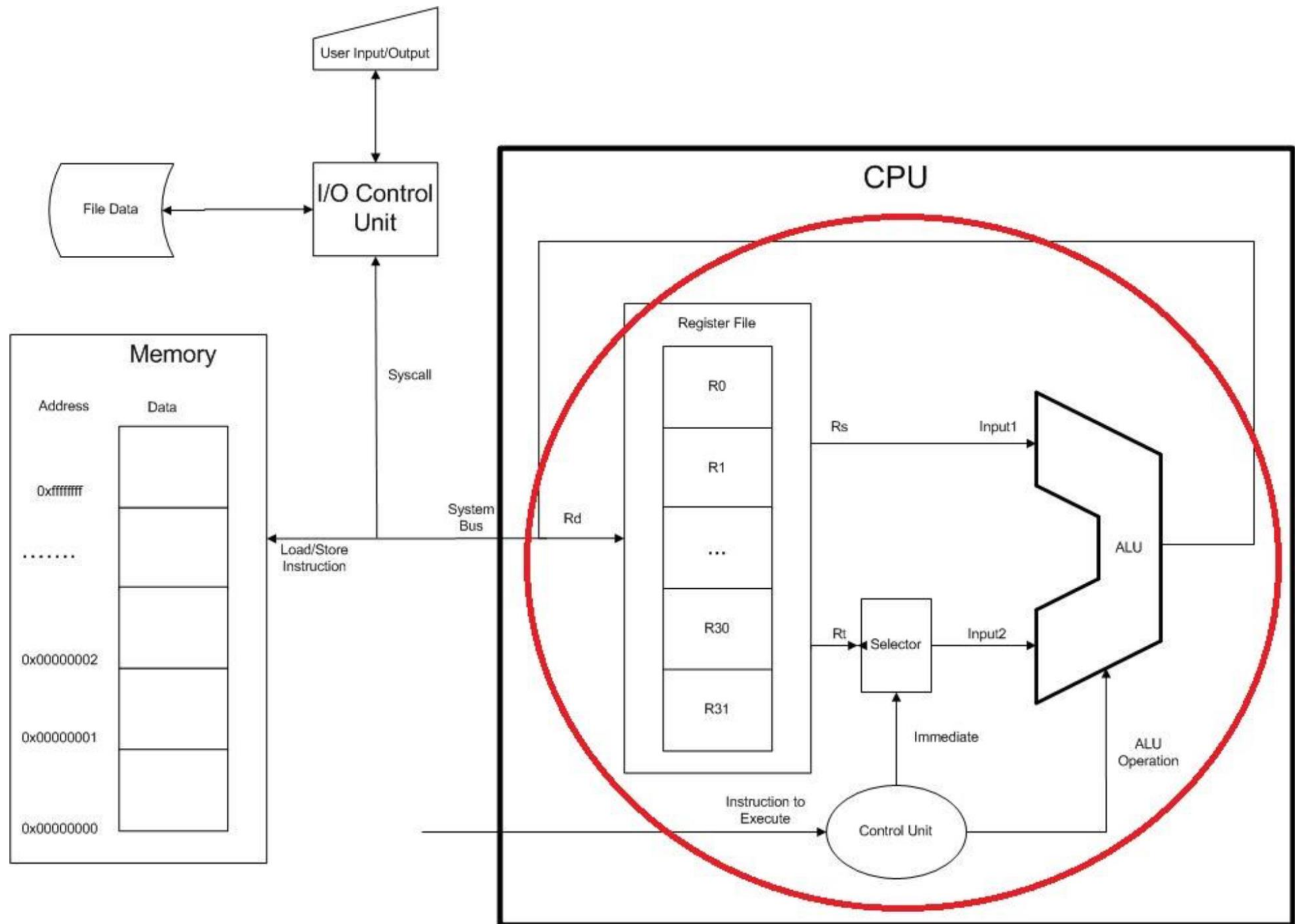
- Dịch vụ **syscall 8** đọc một chuỗi từ bàn điều khiển console.
- Gọi đến 2 tham số:
 - Bộ nhớ sử dụng để lưu chuỗi (lưu trong thanh ghi **\$a0**)
 - Kích thước lớn nhất của chuỗi đọc vào (lưu trong thanh ghi **\$a1**)

Chương trình nhắc và đọc vào một chuỗi từ người dùng

Address	Value (+0)	Value (+4)
0x10010000	\0 \0 \0 \0	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	o a v	t o m
0x10010080	h c o	: i o u
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0

Address	Value (+0)	Value (+4)
0x10010000	l l e H	\0 \0 \n o
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	o a v	t o m
0x10010080	h c o	: i o u
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0

Máy 3 địa chỉ



Phân loại toán tử

- Toán tử thanh ghi (Register: R)
- Toán tử tức thì (Immediate: I)
- Toán tử nhảy (Jump: J)

Toán tử thanh ghi R

- Cú pháp:

[toán tử] R_d, R_s, R_t

- Trong đó:
 - R_d : thanh ghi đích, dùng để ghi kết quả
 - R_s : thanh ghi nguồn thứ nhất
 - R_t : thanh ghi nguồn thứ 2

Toán tử tức thì I

- Cú pháp:

[toán tử] R_d , R_s , Giá trị tức thì

- Trong đó:
 - R_d : thanh ghi đích, dùng để ghi kết quả
 - R_s : thanh ghi nguồn thứ nhất
 - Giá trị tức thì: nguồn thứ 2

Các phép toán số học

- Cộng
- Trừ
- Nhân
- Chia

Phép cộng

- Toán tử **add**
- Toán tử **addi**
- Toán tử **addu**
- Toán tử **addiu**

Toán tử add

- Nhận giá trị của các thanh ghi R_s và R_t chứa các số nguyên, cộng giá trị các số và lưu giá trị tổng vào thanh ghi R_d .
- Định dạng và ý nghĩa như sau:

Định dạng: **add R_d, R_s, R_t**

Ý nghĩa: **$R_d \leftarrow R_s + R_t$**

Toán tử addi

- Lấy giá trị của thanh ghi R_s và cộng với một giá trị 16 bit tức thì trong lệnh, lưu trữ giá trị trở lại vào thanh ghi R_t .
- Định dạng và ý nghĩa như sau:

Định dạng: **addi R_d , R_s , Giá trị tức thì**

Ý nghĩa: **$R_d \leftarrow R_s + \text{Giá trị tức thì}$**

Toán tử addu

- Tương tự như toán tử add, ngoại trừ giá trị trong thanh ghi là số nhị phân không dấu.
- Không có giá trị âm, vì thế giá trị sẽ nằm trong khoảng $0 \dots 2^{32}-1$. Định dạng và ý nghĩa tương tự như toán tử add:

Định dạng: **addu R_d , R_s , R_t**

Ý nghĩa: **$R_d \leftarrow R_s + R_t$**

Toán tử addi

- Tương tự như toán tử addi nhưng giá trị tức thời là số nguyên không dấu.
- Định dạng và ý nghĩa như sau:

Định dạng: **addiu R_d , R_s , Giá trị tức thì**

Ý nghĩa: **$R_d \leftarrow R_s + \text{Giá trị tức thì}$**

Toán tử add giả

- Sử dụng giá trị tức thì 16 bit.
- Đây là cách viết tắt cho toán tử add để thực hiện toán tử addi. Nguyên lý áp dụng tương tự cho addu nếu một giá trị tức thời được sử dụng, và toán tử được chuyển thành một addiu.
- Định dạng và ý nghĩa và phiên dịch như sau:

Định dạng: **add R_t, R_s , Giá trị tức thì**

Ý nghĩa: **$R_t \leftarrow R_s + \text{Giá trị tức thì}$**

Thực hiện bởi : **addi R_t, R_s , Giá trị tức thì**

Toán tử cộng với giá trị 32 bits

- Nếu lệnh **I** chứa một số > 16 bit, số đó phải được nạp vào theo 2 bước.
 - **Bước 1:** nạp 16 bits cao vào một thanh ghi sử dụng toán tử **lui** (Load Upper Immediate: Nạp nửa trên tức thời)
 - **Bước 2:** nạp 16 bits thấp sử dụng toán tử **ori** (Or Immediate: hoặc tức thì). Phép cộng sau đó được thực hiện sử dụng toán tử cộng thanh ghi **R**.
- Lệnh **addi R_t, R_s, Giá trị tức thì (32 bits)** được biên dịch thành:

lui \$at, tức thời (nửa cao 16 bits) # nạp nửa cao 16 bits vào thanh ghi \$at

ori \$at, \$at, tức thời (nửa thấp 16 bits) # nạp nửa thấp 16 bits vào thanh ghi \$at

add R_t, R_s, \$at

Ví dụ thực hiện phép cộng

```
1  # File: Program2-1.asm
2  # Author: NTTNga
3  # Purpose: To illustrate some addition operators
4
5  # illustrate R format add operator
6  li $t1, 100
7  li $t2, 50
8  add $t0, $t1, $t2
9
10 # illustrate add with an immediate. Note that
11 # an add with a pseudo instruction translated
12 # into an addi instruction
13 addi $t0, $t0, 50
14 add $t0, $t0, 50
15
16 # using an unsign number. Note that the
17 # result is not what is expected
18 # for negative numbers.
19 addiu $t0, $t2, -100
20
21 # addition using a 32 immediate. Note that 5647123
22 # base 10 is 0x562b13
23 addi $t1, $t2, 5647123
```


Ví dụ thực hiện phép cộng

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24090064	addiu \$9,\$0,0x00000064	6: li \$t1, 100
<input type="checkbox"/>	0x00400004	0x240a0032	addiu \$10,\$0,0x0000...	7: li \$t2, 50
<input type="checkbox"/>	0x00400008	0x012a4020	add \$8,\$9,\$10	8: add \$t0, \$t1, \$t2
<input type="checkbox"/>	0x0040000c	0x21080032	addi \$8,\$8,0x00000032	13: addi \$t0, \$t0, 50
<input type="checkbox"/>	0x00400010	0x21080032	addi \$8,\$8,0x00000032	14: add \$t0, \$t0, 50
<input type="checkbox"/>	0x00400014	0x2548ff9c	addiu \$8,\$10,0xffff...	19: addiu \$t0, \$t2, -100
<input type="checkbox"/>	0x00400018	0x3c010056	lui \$1,0x00000056	23: addi \$t1, \$t2, 5647123
<input type="checkbox"/>	0x0040001c	0x34212b13	ori \$1,\$1,0x00002b13	
<input type="checkbox"/>	0x00400020	0x01414820	add \$9,\$10,\$1	

- Cột đầu tiên là Source
- Chứa chương trình chính xác như những gì được gõ vào.

Ví dụ thực hiện phép cộng

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24090064	addiu \$9,\$0,0x00000064	6: li \$t1, 100
<input type="checkbox"/>	0x00400004	0x240a0032	addiu \$10,\$0,0x0000...	7: li \$t2, 50
<input type="checkbox"/>	0x00400008	0x012a4020	add \$8,\$9,\$10	8: add \$t0, \$t1, \$t2
<input type="checkbox"/>	0x0040000c	0x21080032	addi \$8,\$8,0x00000032	13: addi \$t0, \$t0, 50
<input type="checkbox"/>	0x00400010	0x21080032	addi \$8,\$8,0x00000032	14: add \$t0, \$t0, 50
<input type="checkbox"/>	0x00400014	0x2548ff9c	addiu \$8,\$10,0xffff...	19: addiu \$t0, \$t2, -100
<input type="checkbox"/>	0x00400018	0x3c010056	lui \$1,0x00000056	23: addi \$t1, \$t2, 5647123
<input type="checkbox"/>	0x0040001c	0x34212b13	ori \$1,\$1,0x00002b13	
<input type="checkbox"/>	0x00400020	0x01414820	add \$9,\$10,\$1	

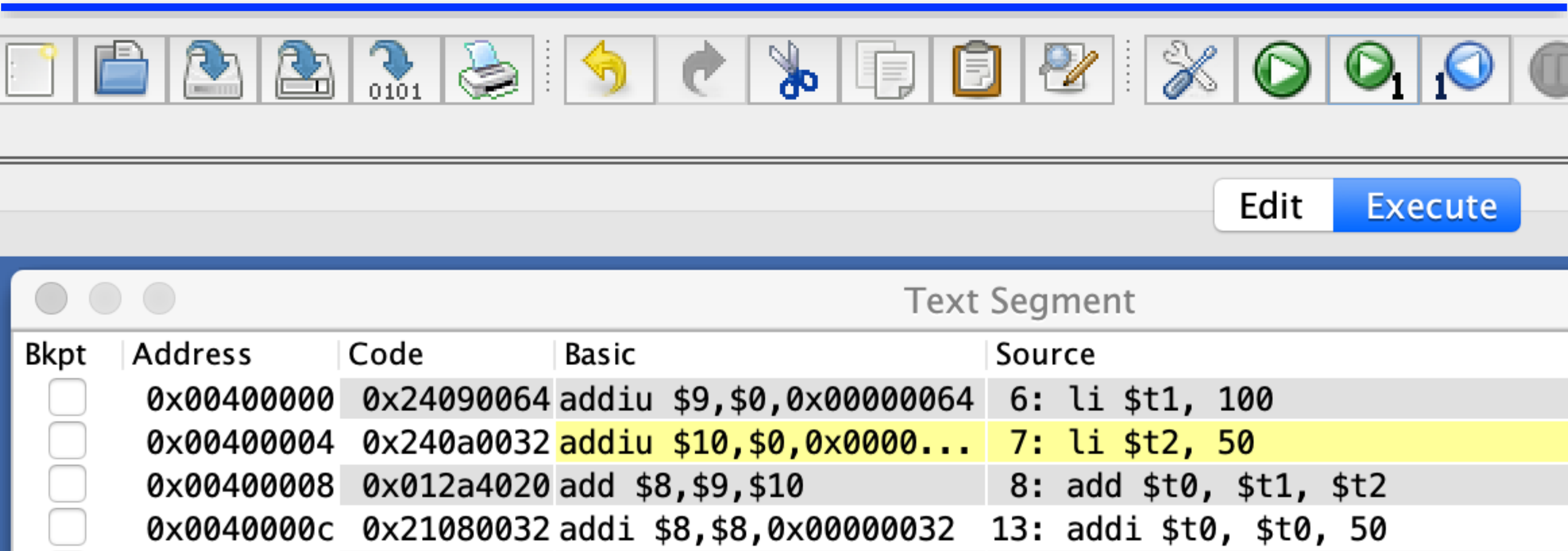
- Cột thứ hai là Basic
- Chứa mã nguồn như là những gì được truyền vào bộ hợp ngữ.

Ví dụ thực hiện phép cộng

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24090064	addiu \$9,\$0,0x00000064	6: li \$t1, 100
<input type="checkbox"/>	0x00400004	0x240a0032	addiu \$10,\$0,0x0000...	7: li \$t2, 50
<input type="checkbox"/>	0x00400008	0x012a4020	add \$8,\$9,\$10	8: add \$t0, \$t1, \$t2
<input type="checkbox"/>	0x0040000c	0x21080032	addi \$8,\$8,0x00000032	13: addi \$t0, \$t0, 50
<input type="checkbox"/>	0x00400010	0x21080032	addi \$8,\$8,0x00000032	14: add \$t0, \$t0, 50
<input type="checkbox"/>	0x00400014	0x2548ff9c	addiu \$8,\$10,0xffff...	19: addiu \$t0, \$t2, -100
<input type="checkbox"/>	0x00400018	0x3c010056	lui \$1,0x00000056	23: addi \$t1, \$t2, 5647123
<input type="checkbox"/>	0x0040001c	0x34212b13	ori \$1,\$1,0x00002b13	
<input type="checkbox"/>	0x00400020	0x01414820	add \$9,\$10,\$1	

- Chương trình đã sẵn sàng thực thi ngay từ dòng đầu tiên của chương trình.

Ví dụ thực hiện phép cộng



- Chương trình được thực thi dòng đầu tiên và đang chờ thực thi dòng thứ 2.
- Kết quả khi chạy dòng đầu tiên, thanh ghi **\$t1 (\$9)** đã được cập nhật để chứa giá trị **100 (0x64)**.

Ví dụ thực hiện phép cộng

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24090064	addiu \$9,\$0,0x00000064	6: li \$t1, 100
<input type="checkbox"/>	0x00400004	0x240a0032	addiu \$10,\$0,0x0000...	7: li \$t2, 50
<input type="checkbox"/>	0x00400008	0x012a4020	add \$8,\$9,\$10	8: add \$t0, \$t1, \$t2
<input type="checkbox"/>	0x0040000c	0x21080032	addi \$8,\$8,0x00000032	13: addi \$t0, \$t0, 50

- Tiếp tục chạy chương trình
- Dòng tiếp theo để thực thi luôn được đánh dấu bằng màu vàng
- Thanh ghi cuối cùng thay đổi được bôi màu xanh.

Registers			Coproc 1	Coproc 0
Name	Number	Value		
\$zero	0	0x00000000		
\$at	1	0x00000000		
\$v0	2	0x00000000		
\$v1	3	0x00000000		
\$a0	4	0x00000000		
\$a1	5	0x00000000		
\$a2	6	0x00000000		
\$a3	7	0x00000000		
\$t0	8	0x00000000		
\$t1	9	0x00000064		
\$t2	10	0x00000032		
\$t3	11	0x00000000		

Ví dụ thực hiện phép cộng

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24090064	addiu \$9,\$0,0x00000064	6: li \$t1, 100
<input type="checkbox"/>	0x00400004	0x240a0032	addiu \$10,\$0,0x0000...	7: li \$t2, 50
<input type="checkbox"/>	0x00400008	0x012a4020	add \$8,\$9,\$10	8: add \$t0, \$t1, \$t2
<input type="checkbox"/>	0x0040000c	0x21080032	addi \$8,\$8,0x00000032	13: addi \$t0, \$t0, 50

- Khi dòng 7 được chạy thì dòng 8 được bôi màu vàng
- Thanh ghi **\$t2 (\$10)** chứa giá trị **0x32 (50₁₀)** và được bôi màu xanh.

Registers			Coproc 1	Coproc 0
Name	Number	Value		
\$zero	0	0x00000000		
\$at	1	0x00000000		
\$v0	2	0x00000000		
\$v1	3	0x00000000		
\$a0	4	0x00000000		
\$a1	5	0x00000000		
\$a2	6	0x00000000		
\$a3	7	0x00000000		
\$t0	8	0x00000000		
\$t1	9	0x00000064		
\$t2	10	0x00000032		
\$t3	11	0x00000000		

Ví dụ thực hiện phép cộng

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24090064	addiu \$9,\$0,0x00000064	6: li \$t1, 100
<input type="checkbox"/>	0x00400004	0x240a0032	addiu \$10,\$0,0x0000...	7: li \$t2, 50
<input type="checkbox"/>	0x00400008	0x012a4020	add \$8,\$9,\$10	8: add \$t0, \$t1, \$t2
<input type="checkbox"/>	0x0040000c	0x21080032	addi \$8,\$8,0x00000032	13: addi \$t0, \$t0, 50
<input type="checkbox"/>	0x00400010	0x21080032	addi \$8,\$8,0x00000032	14: add \$t0, \$t0, 50
<input type="checkbox"/>	0x00400014	0x2548ff9c	addiu \$	

Registers

Coproc 1

Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000096
\$t1	9	0x00000064

- Sau phép cộng ở dòng 8, dòng 13 được đánh dấu màu vàng
- Thanh ghi **\$t0 (\$8)** chứa giá trị **0x96** (hay **150₁₀**).

Ví dụ thực hiện phép cộng

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24090064	addiu \$9,\$0,0x00000064	6: li \$t1, 100
<input type="checkbox"/>	0x00400004	0x240a0032	addiu \$10,\$0,0x0000...	7: li \$t2, 50
<input type="checkbox"/>	0x00400008	0x012a4020	add \$8,\$9,\$10	8: add \$t0, \$t1, \$t2
<input type="checkbox"/>	0x0040000c	0x21080032	addi \$8,\$8,0x00000032	13: addi \$t0, \$t0, 50
<input type="checkbox"/>	0x00400010	0x21080032	addi \$8,\$8,0x00000032	14: add \$t0, \$t0, 50
<input type="checkbox"/>	0x00400014	0x2548ff9c	addiu \$8,\$10,0xffff...	19: addiu \$t0, \$t2, -100
<input type="checkbox"/>	0x00400018	0x3c010056	lui \$1,0x00000056	23: addi \$t1, \$t2, 5647123

- Dòng 19 được bôi vàng và sẵn sàng để chạy.
- Thanh ghi **\$t0** có giá trị **0xfa (250₁₀)**.

Registers			Coproc 1	Coproc 0
Name	Number	Value		
\$zero	0	0x00000000		
\$at	1	0x00000000		
\$v0	2	0x00000000		
\$v1	3	0x00000000		
\$a0	4	0x00000000		
\$a1	5	0x00000000		
\$a2	6	0x00000000		
\$a3	7	0x00000000		
\$t0	8	0x000000fa		
\$t1	9	0x00000064		

Ví dụ thực hiện phép cộng

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x24090064	addiu \$9,\$0,0x00000064	6: li \$t1, 100
<input type="checkbox"/>	0x00400004	0x240a0032	addiu \$10,\$0,0x0000...	7: li \$t2, 50
<input type="checkbox"/>	0x00400008	0x012a4020	add \$8,\$9,\$10	8: add \$t0, \$t1, \$t2
<input type="checkbox"/>	0x0040000c	0x21080032	addi \$8,\$8,0x00000032	13: addi \$t0, \$t0, 50
<input type="checkbox"/>	0x00400010	0x21080032	addi \$8,\$8,0x00000032	14: add \$t0, \$t0, 50
<input type="checkbox"/>	0x00400014	0x2548ff9c	addiu \$8,\$10,0xffff...	19: addiu \$t0, \$t2, -100
<input type="checkbox"/>	0x00400018	0x3c010056	lui \$1,0x00000056	23: addi \$t1, \$t2, 5647123
<input type="checkbox"/>	0x0040001c	0x34212b13	ori \$1,	
<input type="checkbox"/>	0x00400020	0x01414820	add \$9,	

- Thực thi dòng 19
- Giá trị trong thanh ghi **\$t0** thay đổi từ **0xfa** thành **0xfffffcea** (-50₁₀), chứ không phải là **0x96** (150₁₀) như mong đợi

Registers			Coproc 1	Coproc 0
Name	Number	Value		
\$zero	0	0x00000000		
\$at	1	0x00000000		
\$v0	2	0x00000000		
\$v1	3	0x00000000		
\$a0	4	0x00000000		
\$a1	5	0x00000000		
\$a2	6	0x00000000		
\$a3	7	0x00000000		
\$t0	8	0xfffffcea		
\$t1	9	0x00000064		

Các phép toán số học

- Cộng
- Trừ
- Nhân
- Chia

Phép trừ

- Toán tử **sub**
- Toán tử **subi**
- Toán tử **subu**
- Toán tử **subiu**

Toán tử sub

- Nhận giá trị của các thanh ghi R_s và R_t chứa các số nguyên, thực hiện phép trừ và lưu giá trị vào thanh ghi R_d .
- Định dạng và ý nghĩa như sau:

Định dạng: **sub R_d, R_s, R_t**

Ý nghĩa: **$R_d \leftarrow R_s - R_t$**

Toán tử giả subi

- Lấy giá trị của thanh ghi R_s trừ giá trị 16 bit tức thì trong lệnh, lưu trữ giá trị trở lại vào thanh ghi R_t .
- Định dạng và ý nghĩa như sau:

Định dạng: **subi R_d , R_s , Giá trị tức thì**

Ý nghĩa: **$R_d \leftarrow R_s - \text{Giá trị tức thì}$**

Thực hiện bởi : **addi \$at, \$zero, Giá trị tức thì**
sub R_t , R_s , \$at

Toán tử subu

- Tương tự như toán tử add, ngoại trừ giá trị trong thanh ghi là số nhị phân không dấu, vì thế giá trị sẽ nằm trong khoảng $0 \dots 2^{32}-1$.
- Định dạng và ý nghĩa tương tự như toán tử add ở trên:

Định dạng: **subu R_d, R_s, R_t**

Ý nghĩa: **$R_d \leftarrow R_s - R_t$**

Toán tử subiu

- Tương tự như toán tử addi nhưng giá trị tức thời là số nguyên không dấu.
- Định dạng và ý nghĩa như sau:

Định dạng: **subiu R_d , R_s , Giá trị tức thì**

Ý nghĩa: **$R_d \leftarrow R_s - \text{Giá trị tức thì}$**

Thực hiện bởi : **addi \$at, \$zero, Giá trị tức thì**
subu R_t , R_s , \$at

Các phép toán số học

- Cộng
- Trừ
- Nhân
- Chia

Phép nhân

- Toán tử **sub**
- Toán tử **subi**
- Toán tử **subu**
- Toán tử **subiu**

Phép nhân

- Thực hiện nhân hai số 32-bit, kết quả yêu cầu không gian nhớ 64 bit
- Lưu vào 32 bit cao và thấp vào hai thanh ghi **hi** và **lo**
- Thực hiện phép nhân sử dụng toán tử **mult**:

mult \$t1, \$t2 $\#t1 * t2$

mflo \$t0 $\#lưu\ 32\ bit\ thấp\ vào\ \$t0$

Nhận xét

- Nếu kết quả phép nhân đúng thì phải được chứa trong 32 bits thấp.
- Điều gì xảy ra nếu kết quả của phép nhân quá lớn để lưu trong một thanh ghi 32 bit?

Nhận xét

- Ví dụ $3*2=06$ và $3*6=18$

$$\begin{array}{r} 0011 \\ * \quad \underline{0010} \\ 0000 \quad 0110 \end{array}$$

- Không tràn

$$\begin{array}{r} 0011 \\ * \quad \underline{0110} \\ 0001 \quad 0010 \end{array}$$

Tràn

Nhận xét

- Ví dụ $2^*(-3)=-6$ và $2^*(-8)=-16$

$$\begin{array}{r} 0010 \\ * \quad \underline{1101} \\ \hline 1111 \quad 1010 \end{array}$$

- Bit dấu

$$\begin{array}{r} 0010 \\ * \quad \underline{1010} \\ \hline 1110 \quad 1110 \end{array}$$

Không phải bit dấu

Nhận xét

- Ví dụ $6^{*}(-2)$

0010

* 1010

1111 0010

- Không phải bit dấu

Nhận xét

- Để chỉ ra tràn số trong 1 phép nhân chứa trong thanh ghi **hi** cần phải đáp ứng các bit đều bằng 0 hoặc 1,
- Và bit dấu trong thanh ghi **lo** phải đúng.

Toán tử nhân

- Toán tử **mult**
- Toán tử **mflo**
- Toán tử **mfhi**
- Toán tử **mult**
- Toán tử mã giả **mulo**

Toán tử mult

- Nhân giá trị của thanh ghi R_s và R_t , lưu vào thanh ghi **lo** và **hi**.
- Định dạng và ý nghĩa như sau:

Định dạng: **mult R_s, R_t**

Ý nghĩa: **$[hi, lo] \leftarrow R_s * R_t$**

Toán tử mflo

- chuyển giá trị từ thanh ghi **lo** vào thanh ghi R_d .
- Định dạng và ý nghĩa như sau:

Định dạng: **mflo R_d**

Ý nghĩa: **$R_d \leftarrow lo$**

Toán tử mfhi

- chuyển giá trị từ thanh ghi **hi** vào thanh ghi R_d .
- Định dạng và ý nghĩa như sau:

Định dạng: **mfhi R_d**

Ý nghĩa: **$R_d \leftarrow hi$**

Toán tử mult

- Nhân giá trị của thanh ghi R_s và R_t , lưu vào thanh ghi R_d .
- Định dạng và ý nghĩa như sau:

Định dạng: **mult R_d, R_s, R_t**

Ý nghĩa: **$R_d \leftarrow R_s * R_t$**

Toán tử giả mulo

- Nhân giá trị của thanh ghi R_s và R_t , lưu vào thanh ghi R_d , và kiểm tra tràn số. Nếu tràn số xảy ra thì một ngoại lệ sẽ được đưa ra và chương trình sẽ dừng lại do xảy ra lỗi.
- Định dạng và ý nghĩa như sau:

Định dạng: **mult R_d, R_s, R_t**

Ý nghĩa: **$R_d \leftarrow R_s * R_t$**

Toán tử nhân với giá trị tức thì

- Cả hai toán tử `mult` và `mulo` có hỗ trợ toán tử giả cho giá trị tức thời.
- Định dạng, ý nghĩa và phiên dịch như sau:

Định dạng: **`mult Rd, Rs, Giá trị tức thì`**

Ý nghĩa: **$R_d \leftarrow R_s * \text{Giá trị tức thì}$**

Thực hiện bởi: **`addi $Rt, $zero, Giá trị tức thì`**

`mult Rd, Rs, Rt`

Toán tử nhân với giá trị tức thì

- Cả hai toán tử `mult` và `mulo` có hỗ trợ toán tử giả cho giá trị tức thời.
- Định dạng, ý nghĩa và phiên dịch như sau:

Định dạng: **`mulo R_d , R_s , Giá trị tức thì`**

Ý nghĩa: **$R_d \leftarrow R_s * \text{Giá trị tức thì}$**

Thực hiện bởi: **`addi $\$R_t$, $\$zero$, Giá trị tức thì`**

`mult R_d , R_s , R_t`

Các phép toán số học

- Cộng
- Trừ
- Nhân
- Chia

Toán tử chia

- Chia hai số 32 bit, cần không gian địa chỉ 64 bit để lưu kết quả
- Thương được lưu trong thanh ghi **lo**
- Phần dư được lưu trong thanh ghi **hi**

Toán tử div

- Định dạng thứ nhất là định dạng thực duy nhất của toán tử này. Toán tử chia R_s cho R_t và lưu kết quả vào cặp thanh ghi **[hi, lo]** với thương lưu trong **lo** và phần dư lưu trong **hi**.

- Định dạng, ý nghĩa như sau:

Định dạng: **div R_s, R_t**

Ý nghĩa: **$[hi, lo] \leftarrow R_s / R_t$**

Toán tử div

- Định dạng thứ 2 của toán tử div là một lệnh giả. Đó là 1 định dạng 3 địa chỉ nhưng vẫn là 1 lệnh giả. Hơn nữa, để thực thi lệnh chia, lệnh giả này cũng kiểm tra phép chia 0. Phép kiểm tra cụ thể không nằm trong phạm vi này vì liên quan đến lệnh **bne** và **break**
- Định dạng, ý nghĩa và thực hiện như sau:

Định dạng **div R_d, R_s, R_t**

Ý nghĩa: **[if $R_t \neq 0$] $R_d \leftarrow R_s / R_t$**

else break

Toán tử div

Thực hiện: **bne R_t , \$zero, 0x00000001**

break

div R_s , R_t

mflo R_d

Toán tử div

- Định dạng thứ 3 là một lệnh giả.
- Định dạng, ý nghĩa và thực hiện như sau:

Định dạng **div R_d , R_s , Giá trị tức thì**

Ý nghĩa: **$R_d \leftarrow R_s / \text{Giá trị tức thì}$**

addi $\$R_t$, $\$zero$, Giá trị tức thì

div R_s , R_t

mflo R_d

Toán tử rem

- Chỉ có các định dạng mã giả cho lệnh này. Định dạng đầu tiên là toán tử rem là một lệnh giả.
- Định dạng, ý nghĩa và thực hiện như sau:

Định dạng **div R_d , R_s , R_t**

Ý nghĩa: **[if $R_t \neq 0$] $R_d \leftarrow R_s \% R_t$**

else break

Toán tử rem

Thực hiện: **bne R_t , \$zero, 0x00000001**

break

div R_s , R_t

mfhi R_d

Toán tử rem

- Định dạng thứ hai của toán tử **rem** cũng là một lệnh giả.
- Định dạng, ý nghĩa và thực hiện như sau:

Định dạng **rem R_d , R_s , Giá trị tức thì**

Ý nghĩa: **$R_d \leftarrow R_s / \text{Giá trị tức thì}$**

addi $\$R_t$, $\$zero$, Giá trị tức thì

div R_s , R_t

mfhi R_d

Giải các phép toán trong MIPS

- Sử dụng các phép toán số học trong MIPS, một chương trình có thể được tạo ra để giải các hàm số.
- Ví dụ với chương trình giả ngữ sau, người dùng nhập vào 1 giá trị của x và chương trình sẽ in ra kết quả của hàm $5x^2 + 2x + 3$.

```
main
{
    int x = prompt("Enter a value for x: ");
    int y = 5 * x * x + 2 * x + 3;
    print("The result is: " + y);
}
```

Tính hàm số

```
1  # File: Program2-3.asm
2  # Author: NTTNga
3  # Purpose: To calculate the result of 5*x*x+2*x+3
4
5  .data
6  prompt: .asciiz "Hay nhap vao gia tri x: "
7  result: .asciiz "Ket qua la: "
8
9  .text
10 .globl main
11 main:
12     # Get input value, x
13     addi $v0, $zero, 4
14     la $a0, prompt
15     syscall
16     addi $v0, $zero, 5
17     syscall
18     move $s0, $v0
19
20     # Calculate the result of 5*x*x+2*x+3 and store it in $s1
21     mul $t0, $s0, $s0
22     mul $t0, $t0, 5
23     mul $t1, $s0, 2
24     add $t0, $t0, $t1
25     addi $s1, $t0, 3
26
27     # Print output
28     addi $v0, $zero, 4           # Print result string
29     la $a0, result
30     syscall
31     addi $v0, $zero, 1           # Print result
32     move $a0, $s1
33     syscall
34
35     #Exit program
36     addi $v0, $zero, 10
37     syscall
```