MARS – chương trình mô phỏng hợp ngữ (assembly) MIPS

Hướng dẫn sử dụng nhanh

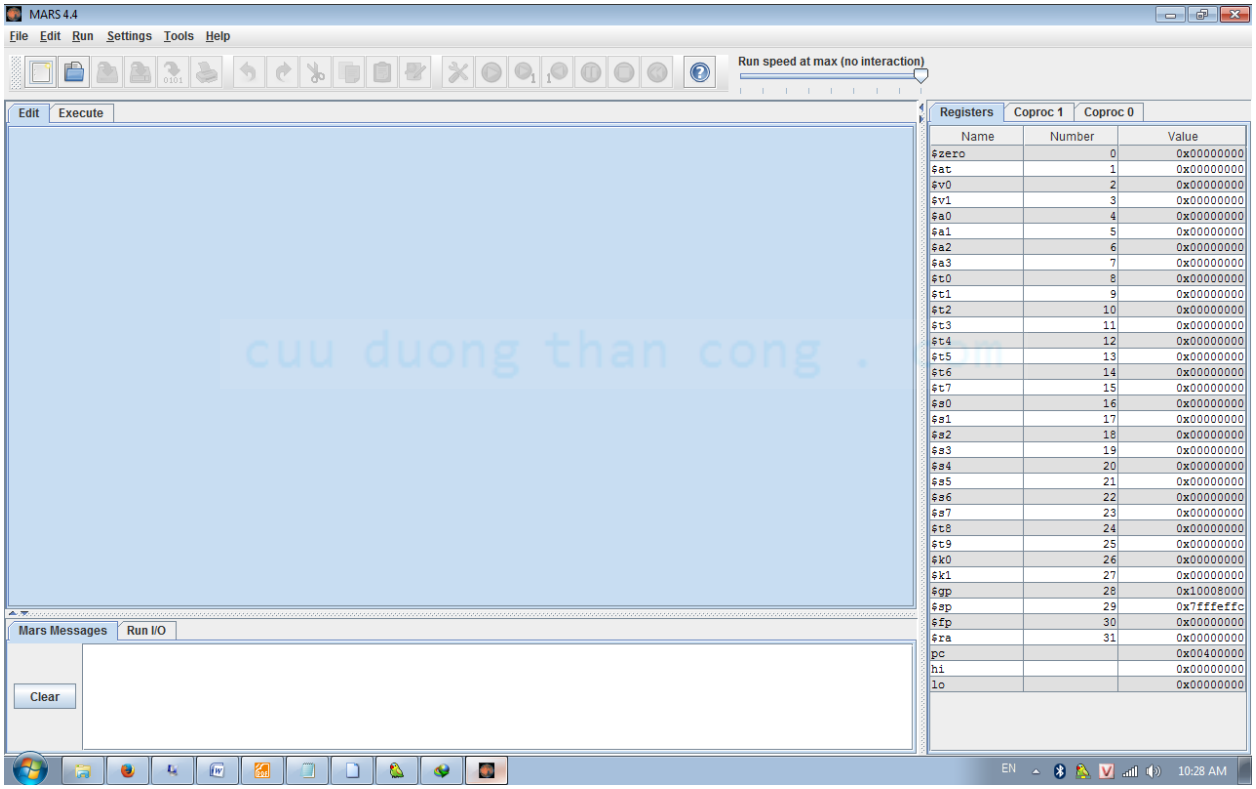
---o0o---

Phần A: Cài đặt, chạy chương trình đầu tiên và thao tác trên thanh ghi

1. Cài đặt

Download chương trình tại: http://courses.missouristate.edu/kenvollmar/mars/download.htm

Giao diện chương trình như hình 1



Hình 1. Giao diện đầu tiên của MARS

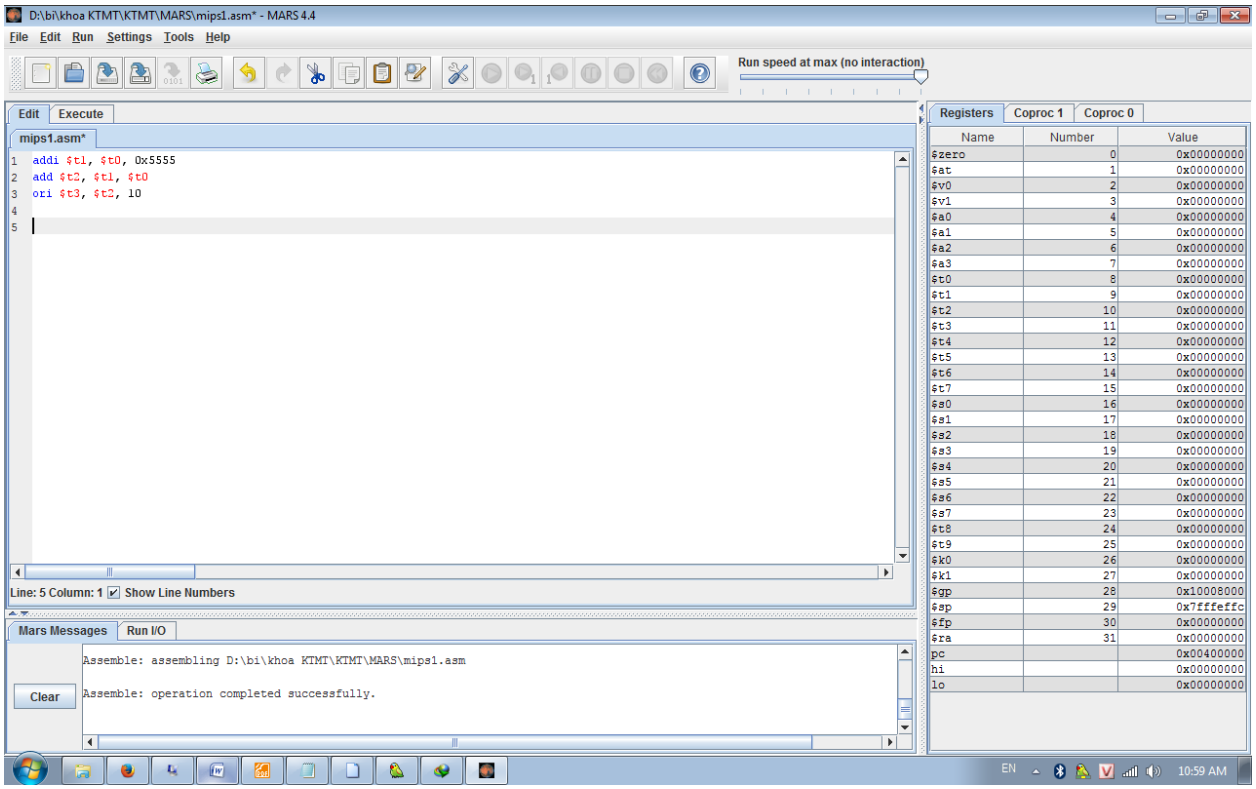
2. Bắt đầu lập trình assembly và chạy chương trình (run)

Ví dụ: *thực hiện đoạn lệnh sau, biết lúc đầu thanh ghi $t0 = 0x1111*

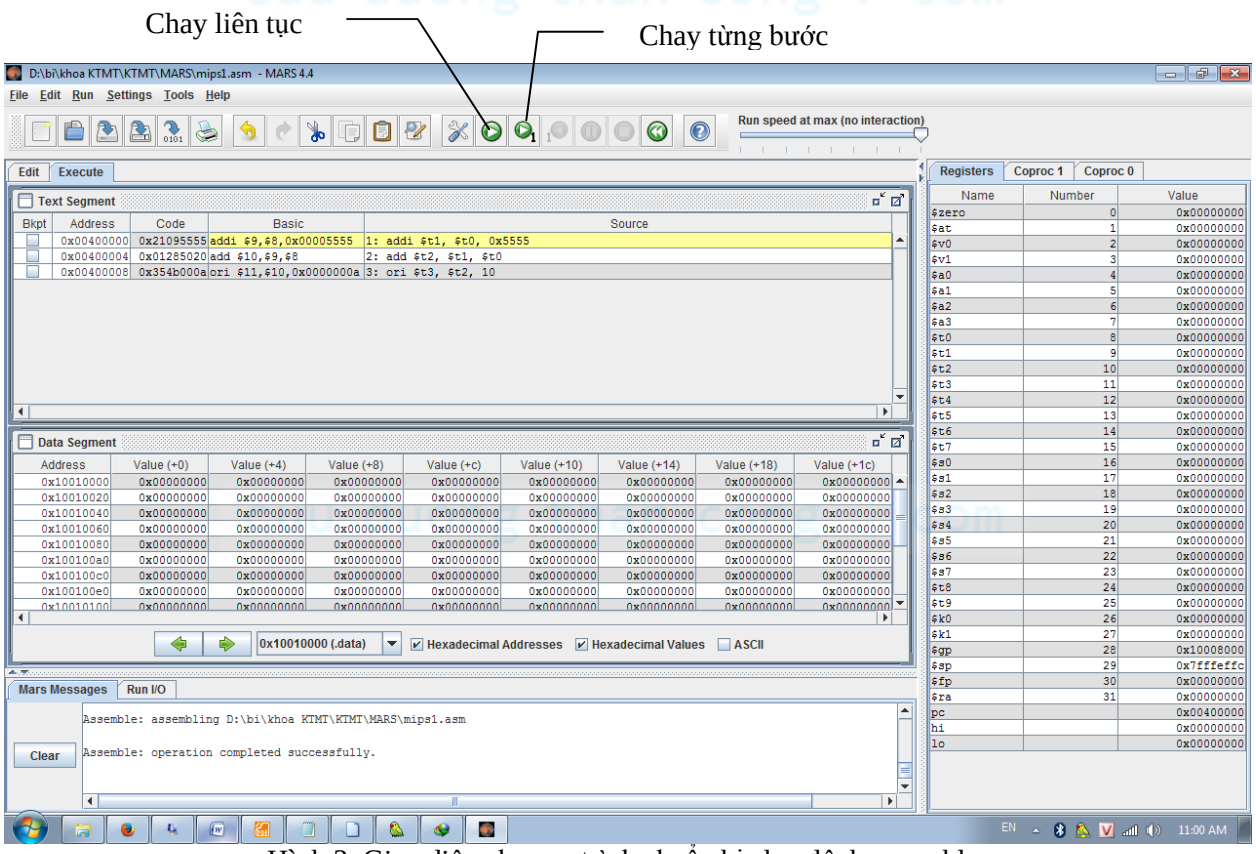
```
addi $t1, $t0, 0x5555
add $t2, $t1, $t0
ori $t3, $t2, 10
```

Lưu ý: trong MARS, 0x5555 tức là 5555 trong hệ 16, còn 10 viết bình thường sẽ được hiểu là trong hệ 10

- Vào File → New → gõ đoạn code trên vào chương trình soạn thảo như hình 2 → save file
- Click Run → Assemble → được giao diện như hình 3

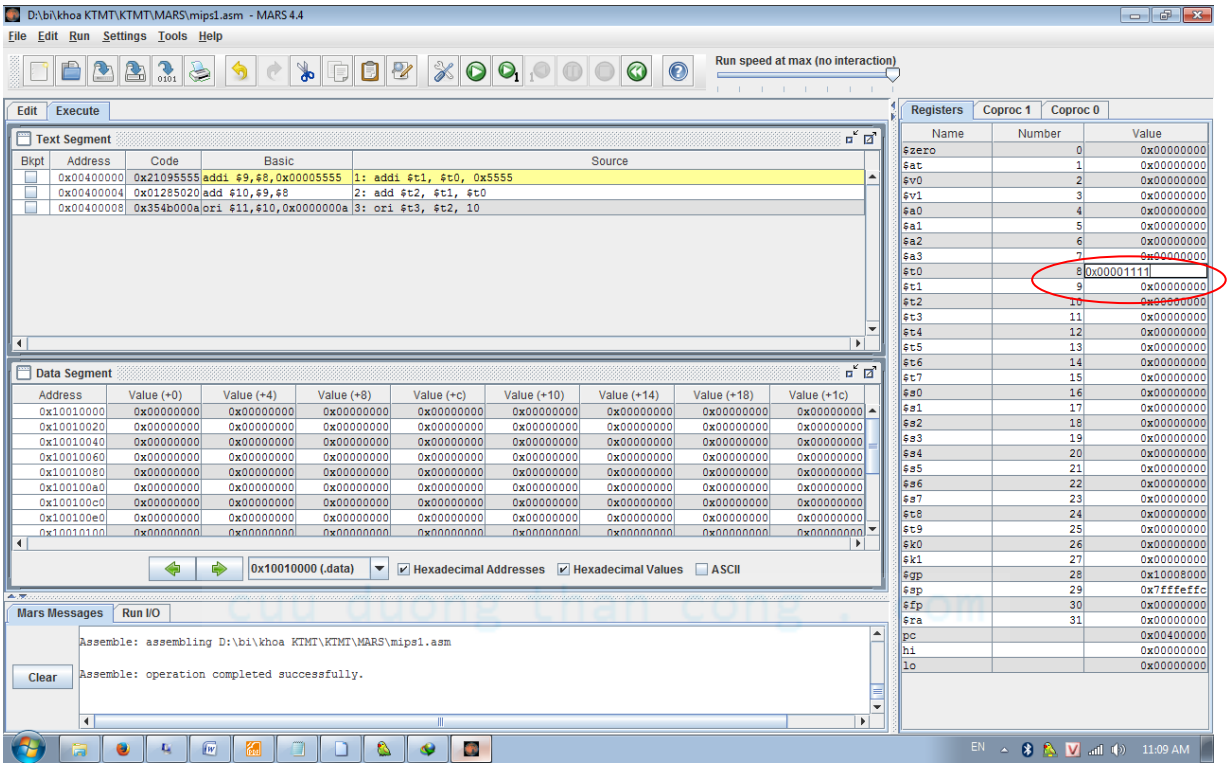


Hình 2. Soạn thảo lệnh assembly

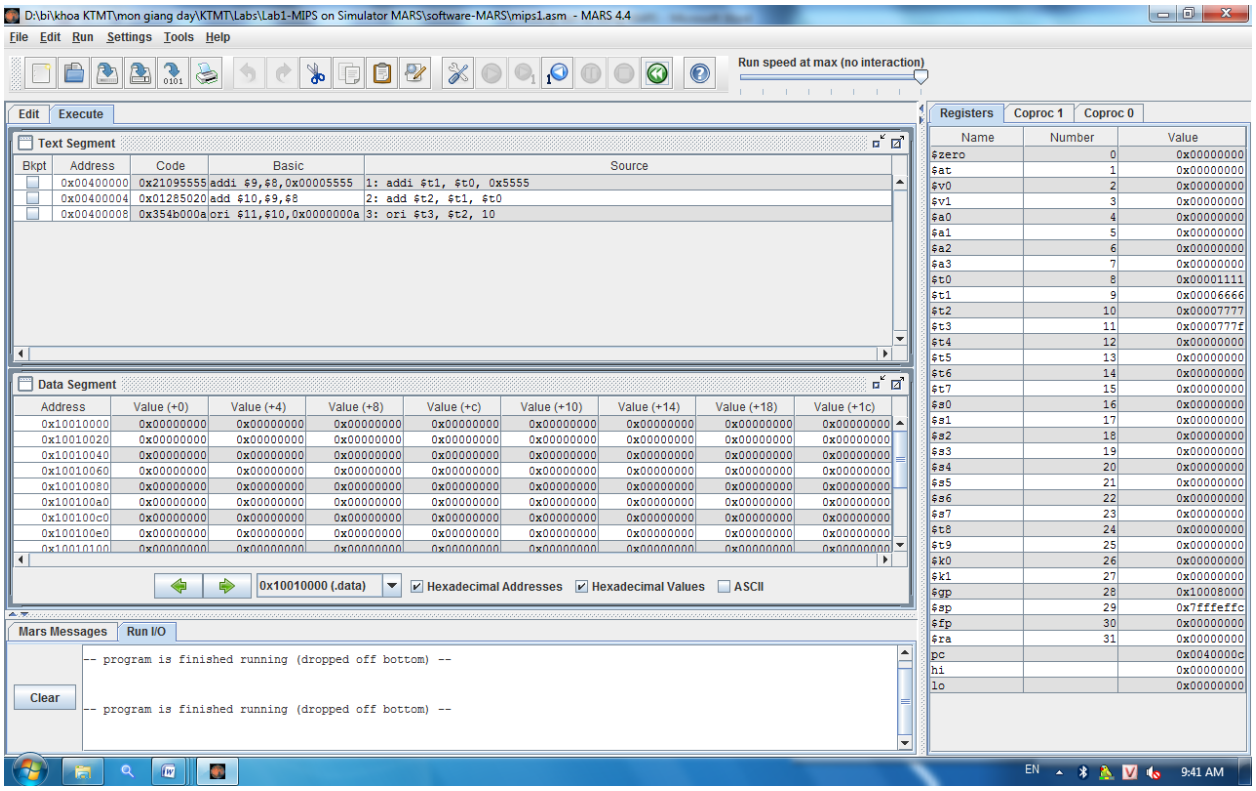


Hình 3. Giao diện chương trình chuẩn bị chay lệnh assembly

- Việc chạy chương trình có thể thực hiện chạy từng từng lệnh hoặc chay toàn bộ một lần như trong hình 3.
- Trước khi chạy đoạn code trong ví dụ, giá trị thanh ghi $t0 đang là 0, chỉnh sửa giá trị thanh ghi này thành 0x1111 như hình 4.
- Chạy từng bước và quan sát sự thay đổi của các thanh ghi. Kết quả cuối cùng như hình 5



Hình 4. Chỉnh sửa giá trị thanh ghi



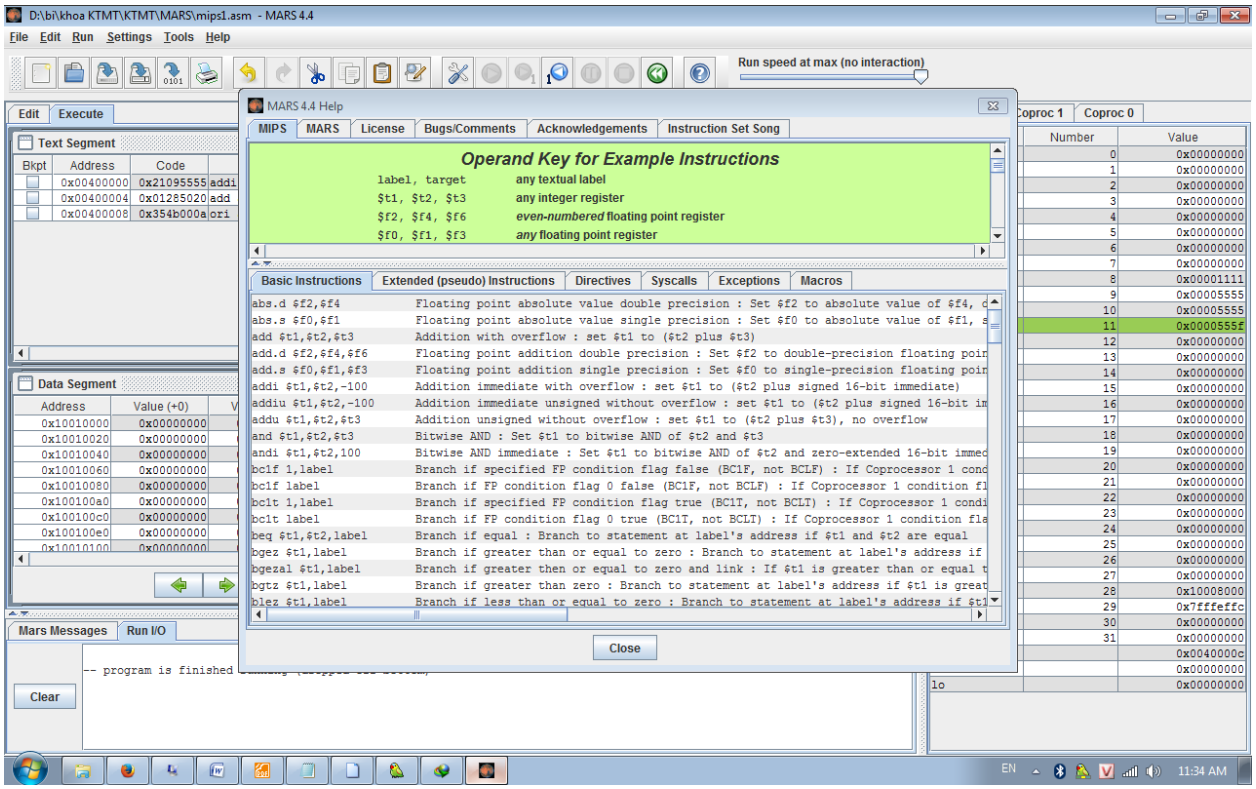
Hình 5. Giao diên chương trình sau khi thực hiên xong đoạn code ví dụ trên

Phần B: Phân biệt giữa lệnh assembly cơ bản của MIPS (basic instruction) và các lệnh mở rộng, hoặc giả (extended/pseudo instruction)

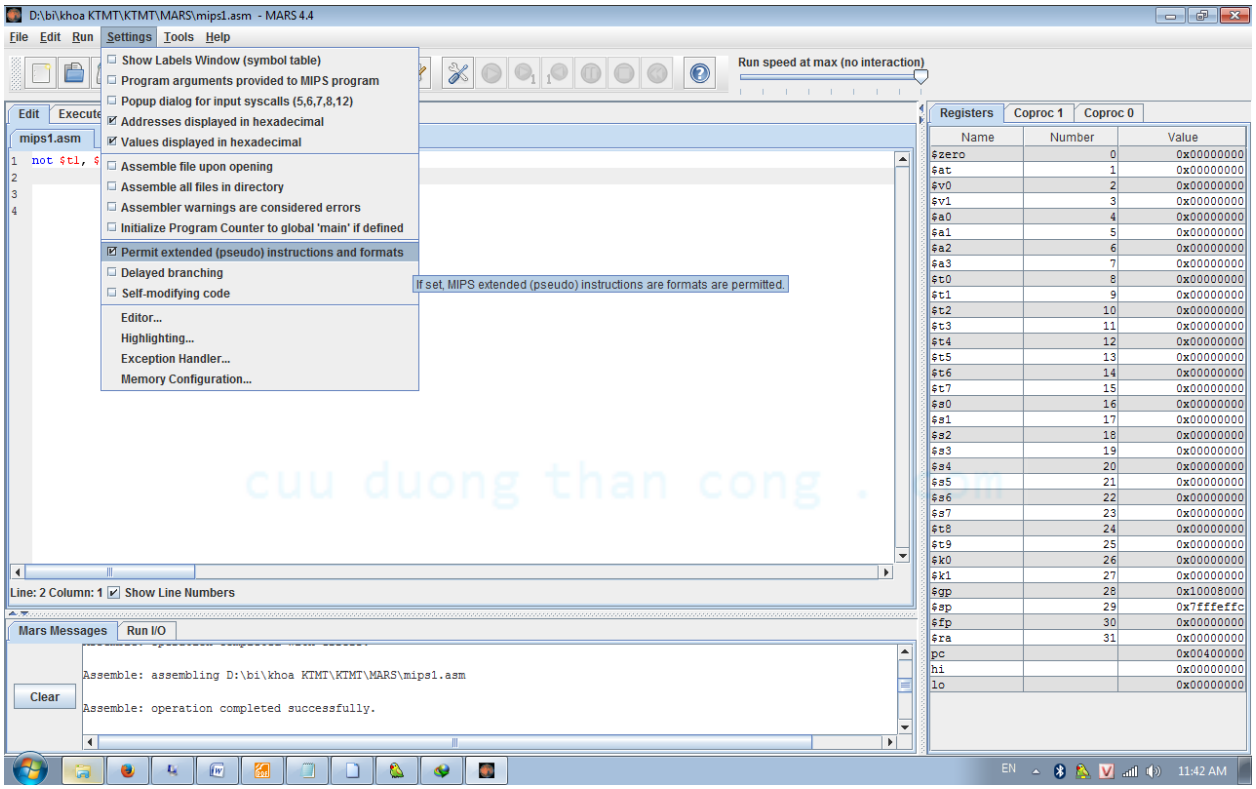
- ✓ Nhóm lệnh cơ bản là những lệnh chính mà processor thật sự thực hiện
- ✓ Nhóm lệnh mở rộng hoặc giả là những lệnh người lập trình vẫn có thể sử dụng, nhưng khi chạy thật sự nó là sự kết hợp của các lệnh trong nhóm lệnh cơ bản

Vào menu Help → Help, nhóm các lệnh cơ bản và lệnh giả lập gồm những lệnh nào được liệt kê như hình 6.

Trong MARS, khi chạy một chương trình cần lưu ý có đang cho phép nhóm lệnh mở rộng/giả hay không, không cho phép tức là simulator này chỉ chạy đúng những lệnh cơ bản. Vào Setting → chọn tùy chọn "Permit extended (pseudo) instructions and formats" như trong hình 7



Hình 6. Danh sách nhóm lệnh cơ bản và nhóm lệnh mở rộng/giả trong MARS



Hình 7. Tùy chọn cho phép sử dụng nhóm lệnh mở rộng (giả) hay không

Ví dụ:

Thực hiện đoạn lệnh sau (với thanh ghi $t0 ban đầu chứa giá trị 0x00000000)

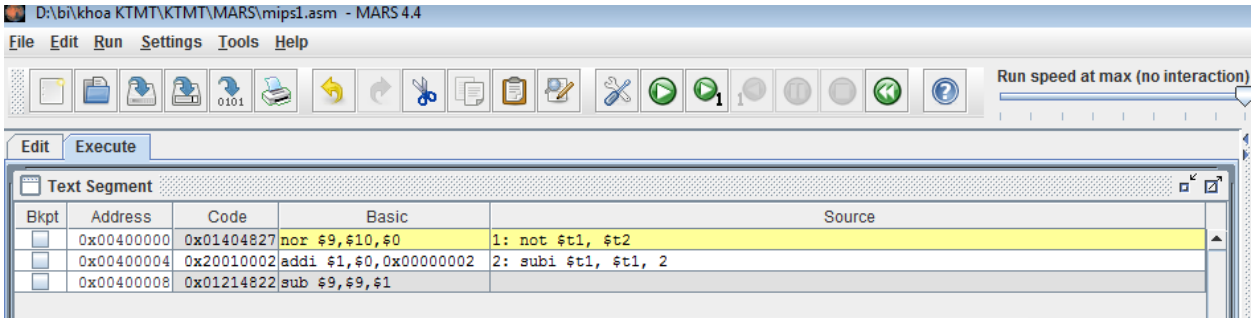
not $t1 $t2
subi $t1, $t1, 2

Xét ví dụ này với hai trường hợp, cho phép nhóm lệnh giả và chỉ sử dụng những lệnh cơ bản:

- **Cho phép nhóm lệnh mở rộng/giả:**

Chay đoạn code ví dụ trên và được kết quả như trong hình 9, ta thấy:

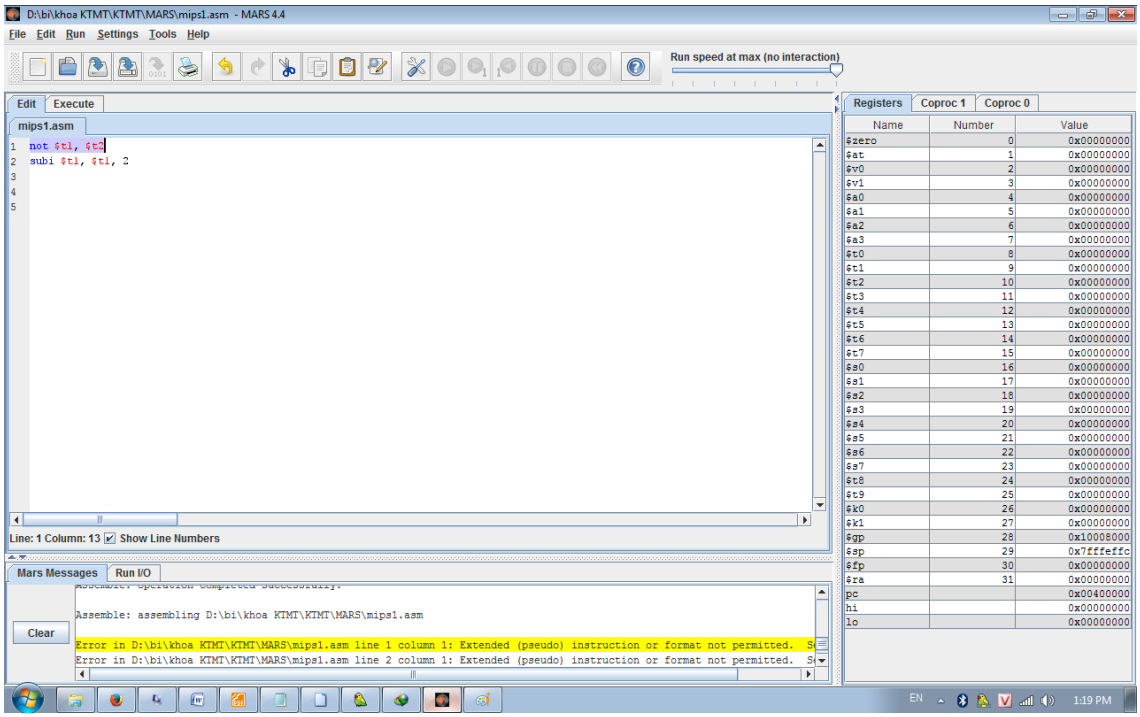
- lệnh 'not' là lệnh giả, thật sự được thực hiện bằng lệnh 'nor'
- lệnh 'subi' thật sự được thực hiện bằng hai lệnh 'addi' và 'sub'



Hình 9. Hai lệnh 'not' và 'subi' trước khi thực thi với nhóm lệnh giả được cho phép

- **Không cho phép nhóm lệnh mở rộng/giả, tức chỉ sử dụng những lệnh cơ bản:**

Chay đoạn code ví dụ trên và bị lỗi như hình 10



Hình 10. Lỗi khi không đươc phép sử dụng những lệnh giả

Phần C: Làm việc với dữ liệu trên bộ nhớ

1. Lệnh lw/sw

Ví dụ:

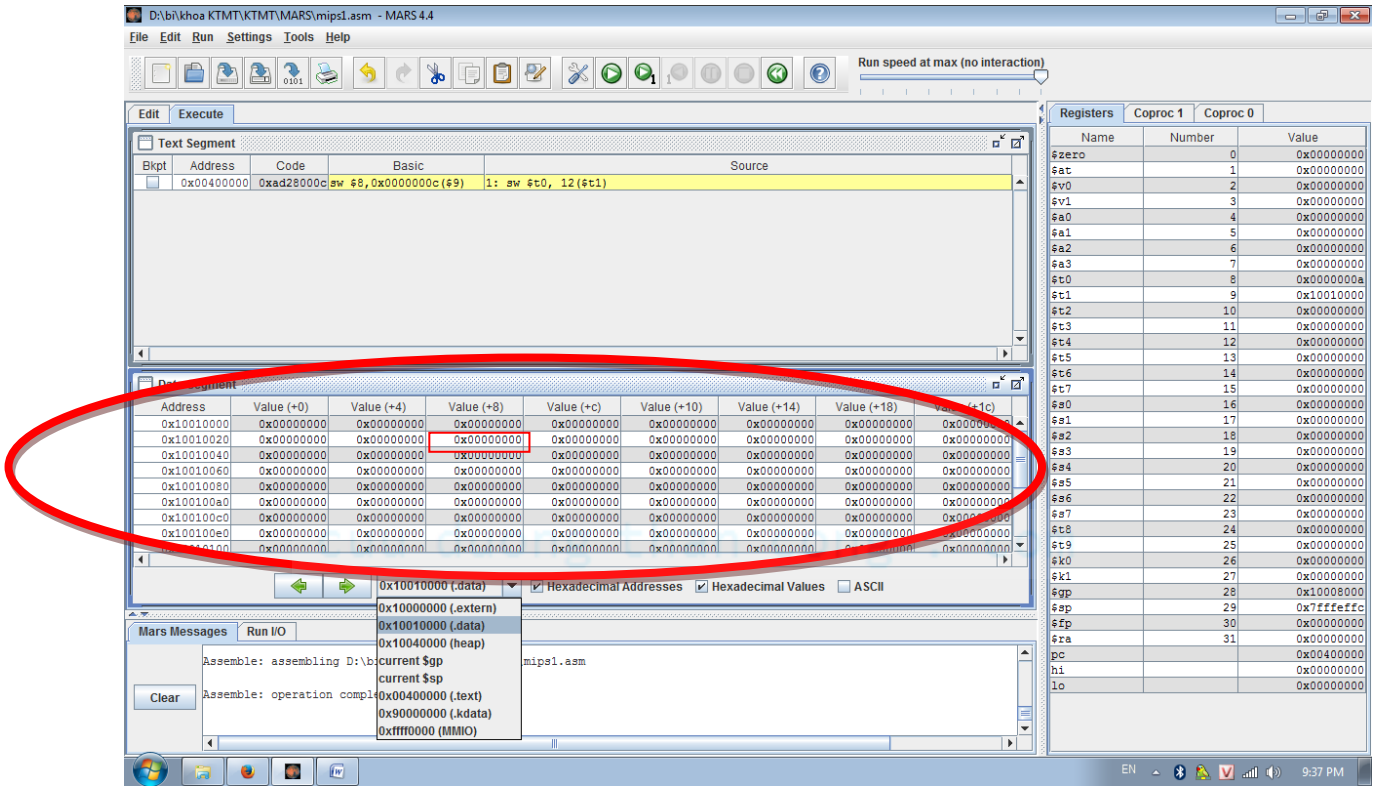
Thực hiện lệnh sau giả sử thanh ghi $t1 lúc đầu đang chứa giá trị 0x10010000, thanh ghi $t0 đang chứa giá trị 0x0000000a

sw $t0, 12($t1)

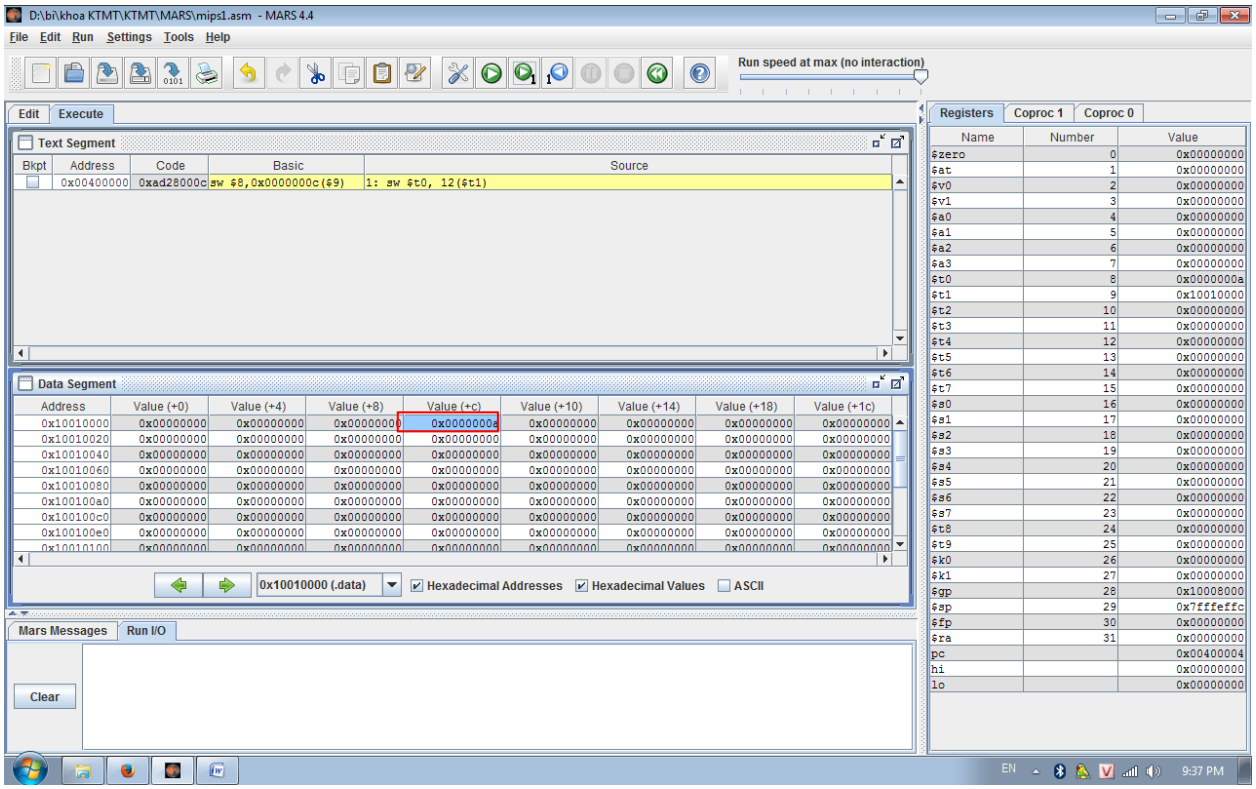
Từng bước thực hiện:

- Gõ lệnh trên vào edit → run Assemble → chỉnh sửa giá trị thanh ghi $t1 = 0x10010000 và $t0 = 0x0000000a trước khi run
- Vùng nhớ đang làm việc được hiển thị tại vị trí khoanh tròn như hình 11, hiển thị giống như một bảng hoặc ma trận gồm các hàng và cột. Mỗi ô trong bảng là 1 từ (word) – 4bytes. Địa chỉ của mỗi word đang ở cột nào được tính bằng cách lấy 'Value' của cột đó cộng với 'Address' tương ứng. Ví dụ trong hình 11, từ nhớ ở hàng 2, cột 'Value (+8)' (khoanh đỏ) có địa chỉ là 0x10010028 và giá trị chứa là 0x00000000
- Chú ý, bộ nhớ của máy tính được chia làm nhiều phân vùng (.data, .text, .extern, heap, …) mỗi phân vùng thực hiện những chức năng khác nhau, chọn .data như hình 11.

- Run và quan sát từ nhớ có địa chỉ 0x1001000c. Như hình 12, nội dung của ô nhớ đã chuyển từ 0 sang 0x0000000a vì lệnh *'sw $t0, 12($t1)'* thực hiện lưu giá trị của thanh ghi $t0 vào word có địa chỉ 0x1001000c ($1001000c_{(16)} = 12_{(10)} + 10010000_{(16)}$), mà giá trị của $t0 đang bằng 0x0000000a, nên sau khi thực hiện lệnh này, giá trị của từ nhớ 0x1001000c lập tức chuyển thành 0x0000000a
- Tương tự, thử và quan sát sự thay đổi bộ nhớ của lệnh 'lw'



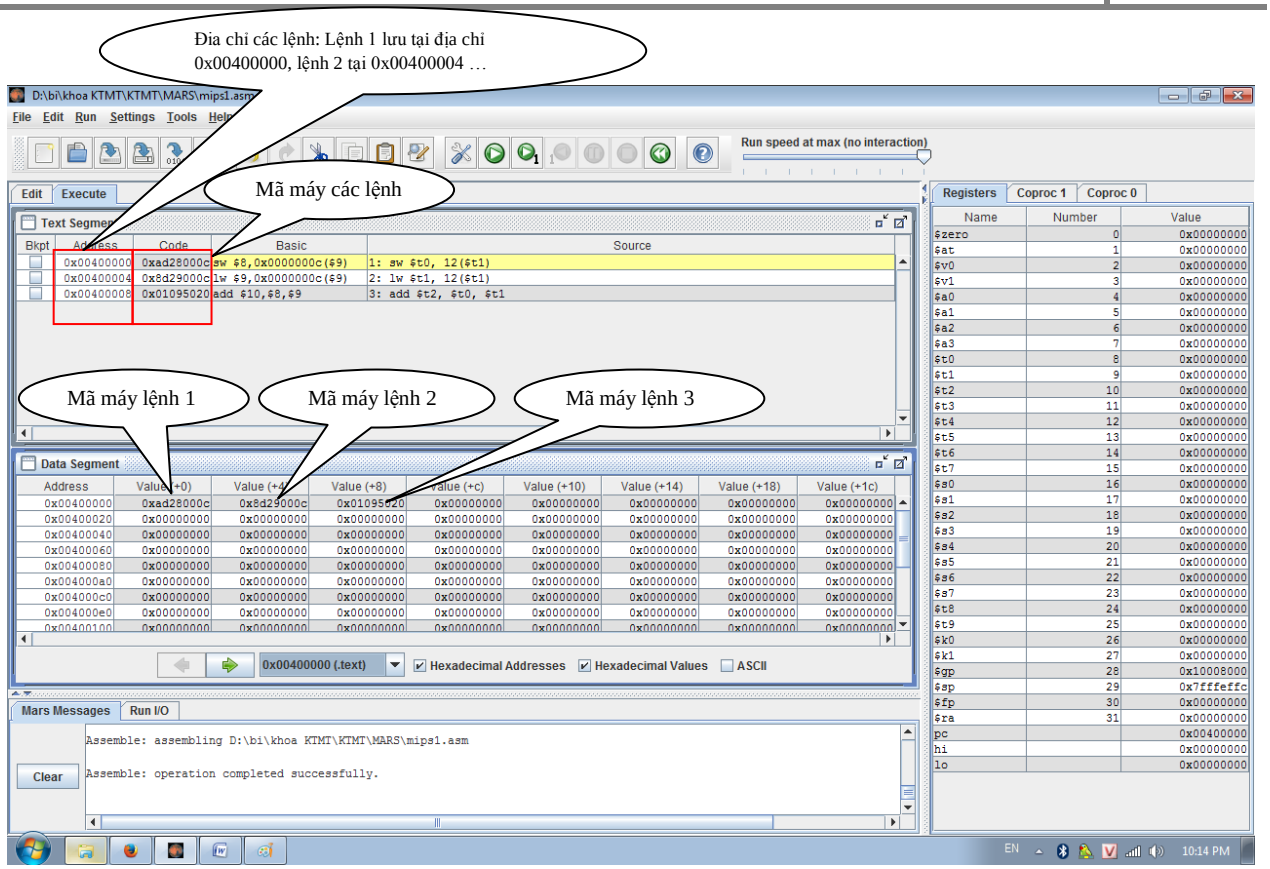
Hình 11. Vùng nhớ làm việc



Hình 12. Kết quả sau khi thực hiện lệnh sw, word tại địa chỉ 0x1001000c thay đổi thành 0x0000000a

2. Vùng nhớ lưu trữ mã máy (machine/binary code) của chương trình assembly

Sau chương trình assembly được biên dịch, tất cả các lệnh mã máy của nó sẽ được đưa vào vùng nhớ tại .text (mặc định trong MARS là 0x400000). Mỗi lệnh sẽ được lưu trong một word, bắt đầu từ 0x400000, như trong hình 13



Hình 13

Phần D. Xuất nhập và gọi hệ thống (system calls)

System calls được sử dụng để đọc vào hoặc in ra giá trị của chuỗi từ cửa sổ input/output, hoặc chỉ thị kết thúc chương trình.

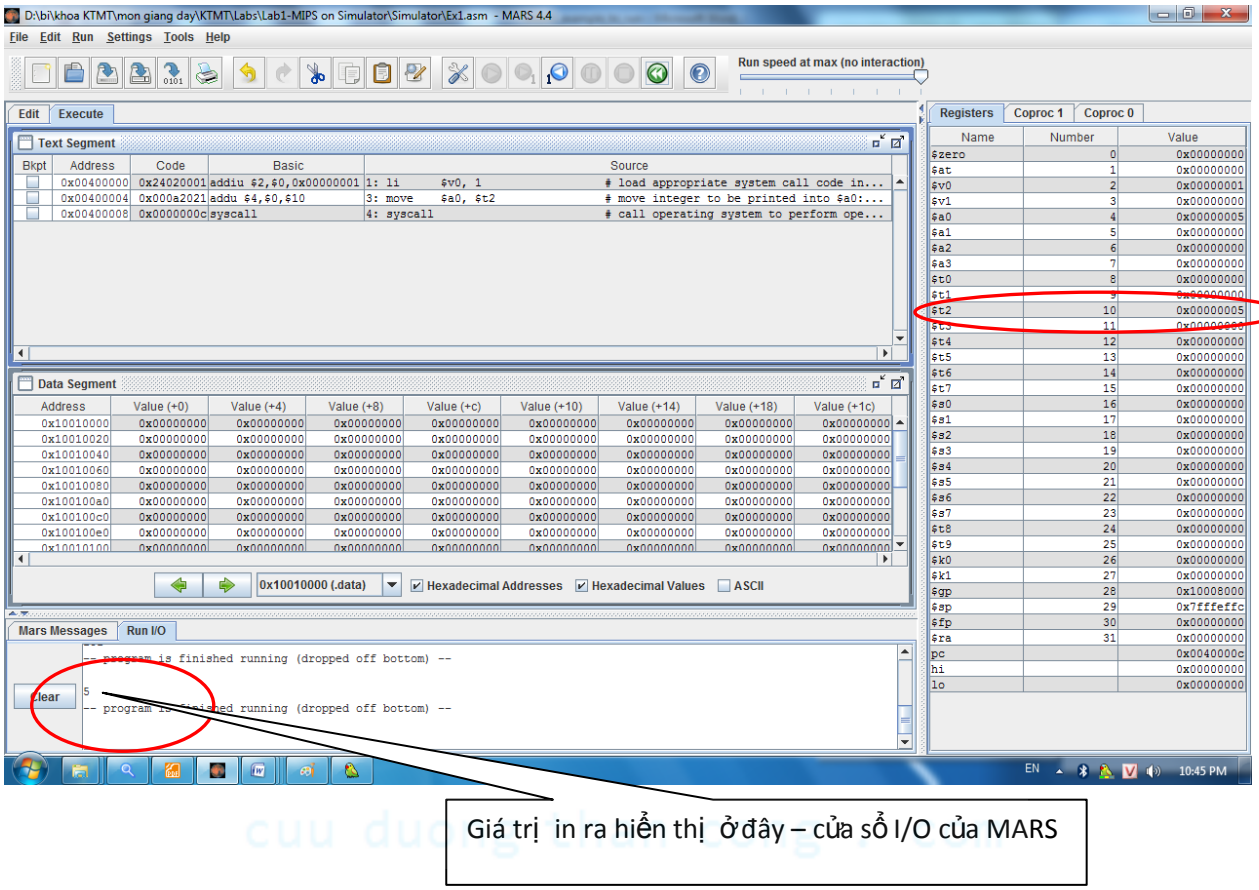
Ví dụ sau in ra cửa sổ giá trị đang chứa trong thanh ghi $t2

```
li   $v0, 1          #đưa mã mong muốn của chức năng vào thanh ghi $v0,
                     #trong trường hợp này mã là 1, tức muốn in số
                     #nguyên ra cửa sổ

move $a0, $t2        #đưa giá trị cần in vào thanh ghi $a0
                     #trong trường này giá trị cần in ra đang chứa
                     #trong thanh ghi $t2

syscall              #Gọi hệ điều hành để thực hiện thao tác in ra cửa
                     #sổ số nguyên chứa trong thanh ghi $a0
```

Khi $v0 đang chứa giá trị 1 và lệnh syscall được thực hiện, chương trình sẽ in số nguyên đang trong thanh ghi $a0 ra cửa sổ, như hình 14 (ví dụ $t2 = 5)



Hình 14. Ví dụ in một số ra cửa sổ hiển thị

Lưu ý: Trong ví dụ trên, thanh ghi $v0 chứa 1 được hiểu là in số nguyên ra cửa sổ. Tương tự, các thao tác in khác (in số thực dang float hay double, in chuỗi, v.v.) với các mã tương ứng được trình bày trong hình 15:

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

Hình 16. Các lệnh hệ thống và mã tương ứng