LẬP TRÌNH HỢP NGỮ MIPS

Mục đích

- Làm quen với hợp ngữ MIPS.
- Biết cách viết, biên dịch và chạy chương trình hợp ngữ MIPS với công cụ MARS.

Tóm tắt lý thuyết

Hợp ngữ (Assembly) là ngôn ngữ lập trình bậc thấp, nó gồm tập các từ khóa và từ gợi nhớ rất gần với ngôn ngữ máy (machine code).

Mỗi kiến trúc vi xử lý đều có tập lệnh (instruction set) riêng, do đó sẽ có hợp ngữ riêng dành cho kiến trúc đó. Ở đây, ta tập trung nghiên cứu về hợp ngữ dành cho kiến trúc MIPS. Môi trường lập trình được sử dụng là chương trình MARS. MARS là môi trường lập trình giả lập giúp ta viết, biên dịch và chạy hợp ngữ MIPS trên các máy x86.

➢ Cấu trúc của một chương trình hợp ngữ MIPS

```
        .data           # khai báo biến sau chỉ thị này
...
        .text           # viết các lệnh sau chỉ thị này
main:                   # điểm bắt đầu của chương trình
…
```

➢ Cách khai báo biến

tên_biến: kiểu_lưu_trữ giá_trị

Các kiểu lưu trữ hỗ trợ: .word, .byte, .ascii, .asciiz, .space

Lưu ý: tên_biến (nhãn) phải theo sau bởi dấu hai chấm (:)

Ví dụ:

```
var1:   .word       3       # số nguyên 4-byte có giá trị khởi tạo là 3
var2:   .byte       'a','b' # mảng 2 phần tử, khởi tạo là a và b
var3:   .space      40      # cấp 40-byte bộ nhớ, chưa được khởi tạo
char_array:  .byte  'A':10  # mảng 10 ký tự được khởi tạo là 'A', có thể thay 'A' bằng 65
int_array:   .word  0:30    # mảng 30 số nguyên được khởi tạo là 0
```

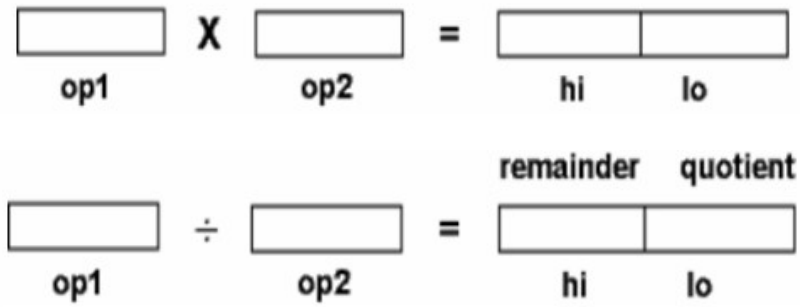
- **Các thanh ghi trong MIPS**

Thanh ghi đa năng

Số	Tên	Ý nghĩa
$0	$zero	Hằng số 0
$1	$at	Assembler Temporary
$2-$3	$v0-$v1	Giá trị trả về của hàm hoặc biểu thức
$4-$7	$a0-$a3	Các tham số của hàm
$8-$15	$t0-$t7	Thanh ghi tạm (không giữ giá trị trong quá trình gọi hàm)
$16-$23	$s0-$s7	Thanh ghi lưu trữ (giữ giá trị trong suốt quá trình gọi hàm)
$24-$25	$t8-$t9	Thanh ghi tạm
$26-27	$k0-$k1	Dự trữ cho nhân HĐH
$28	$gp	Con trỏ toàn cục (global pointer)
$29	$sp	Con trỏ stack
$30	$fp	Con trỏ frame
$31	$ra	Địa chỉ trả về

Thanh ghi HI và LO

Thao tác nhân của MIPS có kết quả chứa trong 2 thanh ghi HI và LO. Bit 0-31 thuộc LO và 32-63 thuộc HI.



Thanh ghi dấu phẩy động

MIPS sử dụng 32 thanh ghi dấu phẩy động để biểu diễn độ chính xác đơn của số thực. Các thanh ghi này có tên là : **$f0 – $f31.**

Để biểu diễn độ chính xác kép (double precision) thì MIPS sử dụng sự ghép đôi của 2 thanh ghi có độ chính xác đơn.

➢ Cú pháp tổng quát lệnh MIPS

<tên-lệnh> <r1>, <r2>, <r3>

- r1: thanh ghi chứa kết quả
- r2: thanh ghi
- r3: thanh ghi hoặc hằng số

➢ Một số lệnh MIPS cơ bản

Ghi chú:

- Rd: thanh ghi đích, Rs, Rt: thanh ghi nguồn.
- các lệnh màu xanh là các lệnh giả (pseudo instructions).

Lệnh Load / Store

Đây là các lệnh duy nhất được phép truy xuất bộ nhớ RAM trong tập lệnh của MIPS.

Cú pháp		Ý nghĩa
lw	Rd, RAM_src	Chép 1 word (4 byte) tại vị trí trong bộ nhớ RAM vào thanh ghi
lb	Rd, RAM_src	Chép 1 byte tại vị trí trong bộ nhớ RAM vào byte thấp của thanh ghi
sw	Rs, RAM_dest	Lưu 1 word trong thanh ghi vào vị trí trong bộ nhớ RAM
sb	Rs, RAM_dest	Lưu 1 byte thấp trong thanh ghi vào vị trí trong bộ nhớ RAM
li	Rd, value	Khởi tạo thanh ghi với giá trị
la	Rd, label	Khởi tạo thanh ghi với địa chỉ của nhãn

Nhóm lệnh số học:

Cú pháp		Ý nghĩa
add	Rd, Rs, Rt	Rd = Rs + Rt (kết quả có dấu)
addi	Rd, Rs, imm	Rd = Rs + imm
addu	Rd, Rs, Rt	Rd = Rs + Rt (kết quả không dấu)
sub	Rd, Rs, Rt	Rd = Rs - Rt
subu	Rd, Rs, Rt	Rd = Rs - Rt (kết quả không dấu)
mult	Rs, Rt	(Hi,Lo) = Rs * Rt
div	Rs, Rt	Lo = Rs / Rt (thương), Hi = Rs % Rt (số dư)
mfhi	Rd	Rd = Hi
mflo	Rd	Rd = Lo
move	Rd, Rs	Rd = Rs

Nhóm lệnh nhảy

Cú pháp	Ý nghĩa
j label	Nhảy không điều kiện đến nhãn 'label'
jal label	Lưu địa chỉ trở về vào $ra và nhảy đến nhãn 'label' (dùng khi gọi hàm)
jr Rs	Nhảy đến địa chỉ trong thanh ghi Rs (dùng để trở về từ lời gọi hàm)
bgez Rs, label	Nhảy đến nhãn 'label' nếu Rs >= 0
bgtz Rs, label	Nhảy đến nhãn 'label' nếu Rs > 0
blez Rs, label	Nhảy đến nhãn 'label' nếu Rs <= 0
bltz Rs, label	Nhảy đến nhãn 'label' nếu Rs < 0
beq Rs, Rt, label	Nhảy đến nhãn 'label' nếu Rs = Rt
bne Rs, Rt, label	Nhảy đến nhãn 'label' nếu Rs != Rt

System Call:

Lệnh syscall làm treo sự thực thi của chương trình và chuyển quyền điều khiển cho HĐH (được giả lập bởi MARS). Sau đó, HĐH sẽ xem giá trị thanh ghi $v0 để xác định xem chương trình muốn nó làm việc gì.

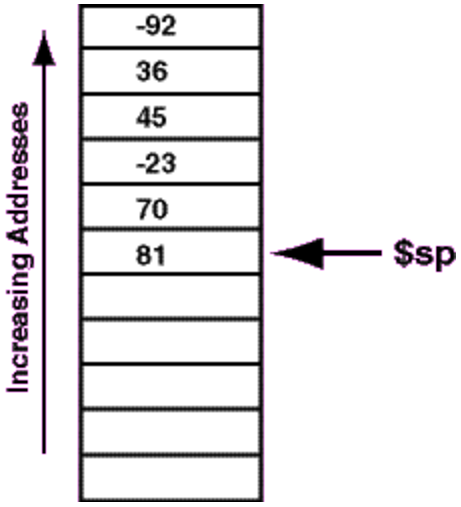
Bảng các system call

Dịch vụ	Giá trị trong $v0	Đối số	Kết quả
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (trong $v0)
read_float	6		float (trong $f0)
read_double	7		double (trong $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (trong $v0)
exit	10		
print_character	11	$a0 = char	
read_character	12		char (trong $v0)

Ví dụ:

```
.data          # khai báo data segment
```

```
str:    .asciiz "hello world"
        .text
        .globl main
main:  # nhãn main cho vi xử lý biết nơi thực thi lệnh đầu tiên
        la $a0, str     # tải địa chỉ của nhãn str vào thanh ghi $a0
        addi $v0, $zero, 4   # đưa giá trị 4 vào thanh ghi $v0
        syscall
        addi   $v0, $zero, 10
        syscall
```



➢ Stack

Stack (ngăn xếp) là vùng nhớ đặc biệt được truy cập theo cơ chế "vào trước ra sau" (LIFO – Last In First Out), nghĩa là dữ liệu nào đưa vào sau sẽ được lấy ra trước.

Hình bên là cấu trúc stack trong bộ nhớ, mỗi phần tử có kích thước một word (32-bit).

Thanh ghi $sp đóng vai trò là con trỏ ngăn xếp (stack pointer), luôn chỉ đến đỉnh của stack. Stack phát triển theo chiều giảm của địa chỉ vùng nhớ (đỉnh của stack luôn có địa chỉ thấp).

Hai thao tác cơ bản trong stack là push (đưa một phần tử vào stack) và pop (lấy một phần tử ra khỏi stack). Cơ chế như sau:

- push: giảm $sp đi 4, lưu giá trị vào ô nhớ mà $sp chỉ đến.

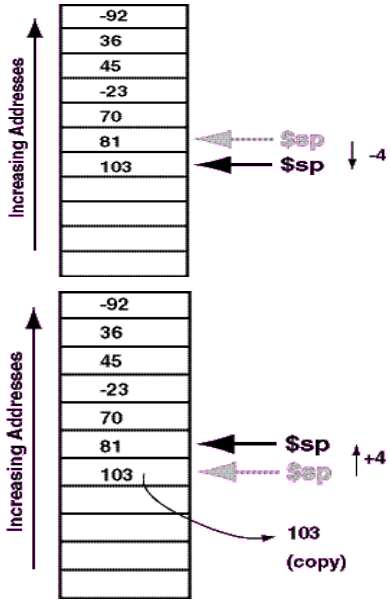
  Ví dụ: push vào stack giá trị trong $t0

  ```
  subu   $sp, $sp, 4
  sw     $t0, ($sp)
  ```

- pop: copy giá trị trong vùng nhớ được chỉ đến bởi $sp, cộng 4 vào $sp.

  Ví dụ: pop từ stack ra $t0

  ```
  lw     $t0, ($sp)
  addu   $sp, $sp, 4
  ```



➢ Thủ tục

MIPS hỗ trợ một số thanh ghi để lưu trữ các dữ liệu phục vụ cho thủ tục:

- Đối số $a0, $a1, $a2, $a3
- Kết quả trả về $v0, $v1
- Biến cục bộ $s0, $s1, ... , $s7
- Địa chỉ quay về $ra

Cấu trúc của một thủ tục:

Đầu thủ tục

```
entry_label:
addi $sp,$sp, -framesize        # khai báo kích thước cho stack
sw $ra, framesize-4($sp)        # cất địa chỉ trở về của thủ tục trong $ra vào ngăn xếp
                                (dùng khi gọi hàm lồng nhau)
Lưu tạm các thanh ghi khác (nếu cần)
```

Thân thủ tục ...

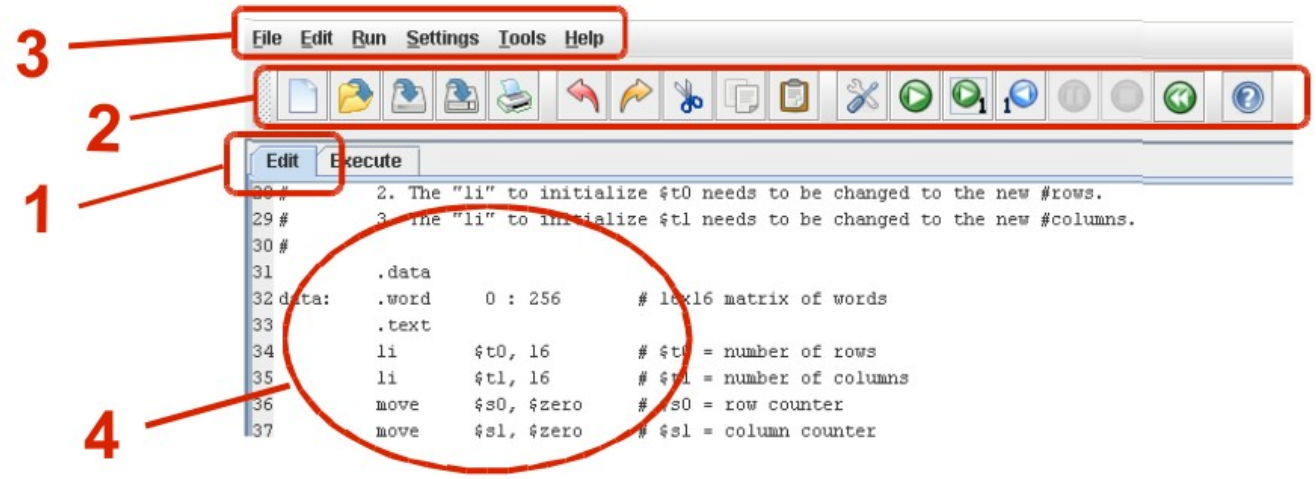
```
(có thể gọi các thủ tục khác...)
```

Cuối thủ tục

```
Phục hồi các thanh ghi khác (nếu cần)
lw $ra, framesize-4($sp)        # lấy địa chỉ trở về ra $ra
addi $sp,$sp, framesize
jr $ra
```

Gọi thủ tục: jal entry_label

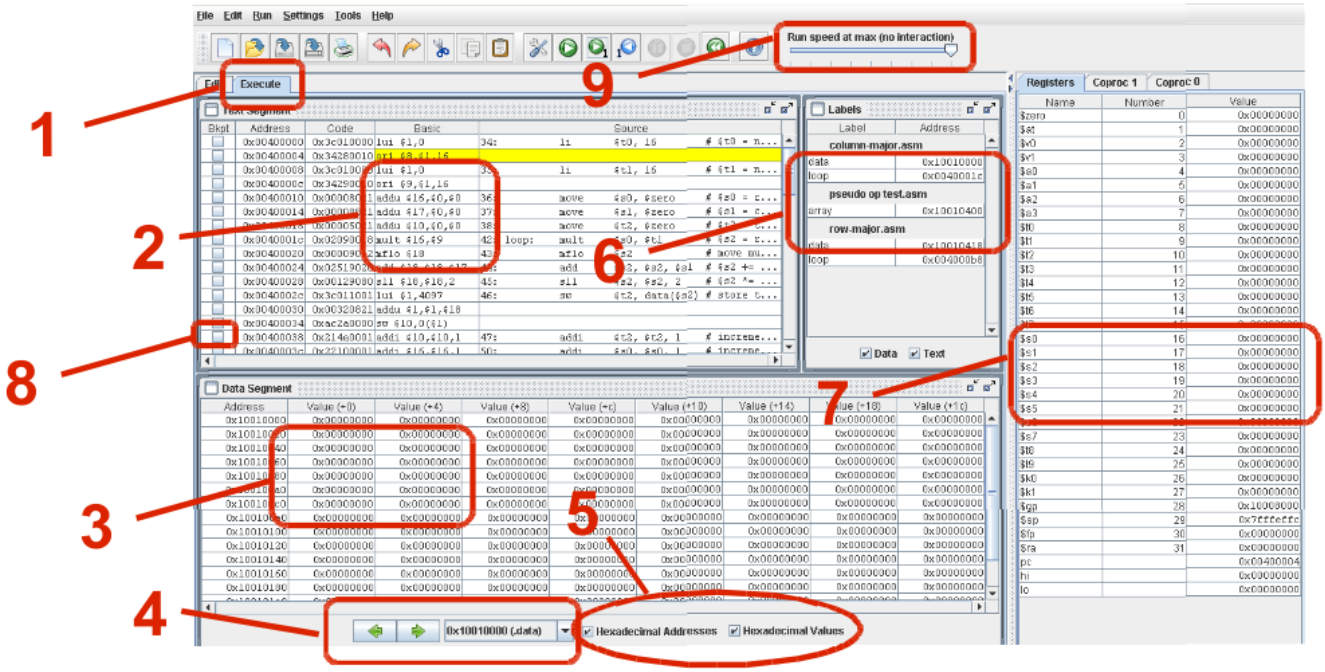
- **Giới thiệu chương trình MARS**



1. Cho biết ta đang ở chế độ soạn thảo

2,3. Thanh menu và thanh công cụ hỗ trợ các chức năng của chương trình.

4. Nơi soạn thảo chương trình hợp ngữ MIPS



1. Cho biết ta đang ở chế độ thực thi
2. Khung thực thi cho ta biết địa chỉ lệnh (Address), mã máy (Code), lệnh hợp ngữ MIPS (Basic), dòng lệnh trong file source tương ứng (Source).
3. Các giá trị trong bộ nhớ, có thể chỉnh sửa được.
4. Cho phép ta duyệt bộ nhớ (2 nút mũi tên) và đi đến các phân đoạn bộ nhớ thông dụng.