TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG & TIN HỌC

Bài giảng

KIẾN TRÚC MÁY TÍNH

Giảng viên : Phạm Huyền Linh
Bộ môn : Toán Tin

CHƯƠNG 5

HỢP NGỮ MIPS

(MIPS Assembly Language)

Giảng viên: Phạm Huyền Linh

Chương 5

5.1. Tập thanh ghi (Register Set)

5.2. Các lớp lệnh (Instruction Classes)

5.3. Cấu trúc lập trình

Tập thanh ghi

- 32 thanh ghi cơ bản
- Bắt đầu bởi $, có 2 kiểu cho địa chỉ
  - Số: $0÷$31
  - Tên: ví dụ $sp, $v1
- 32 thanh ghi dấu phẩy động: $f0 ÷$f31
  - Độ chính xác đơn: Dùng 1 thanh ghi
  - Độ chính xác đôi: Dùng 2 thanh ghi gộp lại
- Hai thanh ghi đặc biệt: PC, Hi, Lo

Các thanh ghi cơ bản

- $Zero: Luôn có giá trị bằng không
  - Add $t1,$0,1
  - Add $t1,$t2,$0
- $2,$3 ($v0,$v1): Chứa các giá trị trả về của thủ tục
- $4-$7 ($a0-$a3): Tham số đầu vào của thủ tục, có thể bị thay đổi bởi thủ tục con
- $16--$23($s0-$s7): Không được thay đổi bởi thủ tục con ->save trước khi thủ con thay đổi giá trị của nó
- $31($ra): Lưu địa chỉ trở về của thủ tục
  - jr $ra

Tên thanh ghi	Số hiệu thanh ghi	Công dụng
$zero	0	the constant value 0, chứa hằng số = 0
$at	1	assembler temporary, giá trị tạm thời cho hợp ngữ
$v0-$v1	2-3	procedure return values, các giá trị trả về của thủ tục
$a0-$a3	4-7	procedure arguments, các tham số vào của thủ tục
$t0-$t7	8-15	temporaries, chứa các giá trị tạm thời
$s0-$s7	16-23	saved variables, lưu các biến
$t8-$t9	24-25	more temporarie, chứa các giá trị tạm thời
$k0-$k1	26-27	OS temporaries, các giá trị tạm thời của OS
$gp	28	global pointer, con trỏ toàn cục
$sp	29	stack pointer, con trỏ ngăn xếp
$fp	30	frame pointer, con trỏ khung
$ra	31	procedure return address, địa chỉ trở về của thủ tục

Các thanh ghi dấu phẩy động

- $f0, $f2: Lưu kết quả trả về của hàm
- $f12-$f14: Dùng cho các đối số của thủ tục
- $f20-$f31: Là thanh ghi phải được luu trừ trước khi gọi thủ tục

Register Name	Use and Linkage
$f0..f3	Used to hold floating-point type function results ($f0) and complex type function results ($f0 has the real part, $f2 has the imaginary part).
$f4..f10	Temporary registers, used for expression evaluation, whose values are not preserved across procedure calls.
$f12..$f14	Used to pass the first 2 single or double precision actual arguments, whose values are not preserved across procedure calls.
$f16..$f18	Temporary registers, used for expression evaluations, whose values are not preserved across procedure calls.
$f20..$f30	Saved registers, whose values must be preserved across procedure calls.

Các thanh ghi đặc biệt

Name	Description
PC	Program Counter
HI	Multiply/Divide special register holds the most-significant 32 bits of multiply, remainder of divide
LO	Multiply/Divide special register holds the least-significant 32 bits of multiply, quotient of divide

- Lệnh cho phép lấy giá trị trong thanh ghi hi và lo, 2 thanh ghi này ẩn trong câu lệnh
  - mfhi rd ≠ Move nội dung thanh ghi hi vào rd
  - mflo rd ≠ Move nội dung thanh ghi lo vào rd

Chương 5

5.1. Tập thanh ghi (Register Set)

5.2. Các lớp lệnh (Instruction Classes)

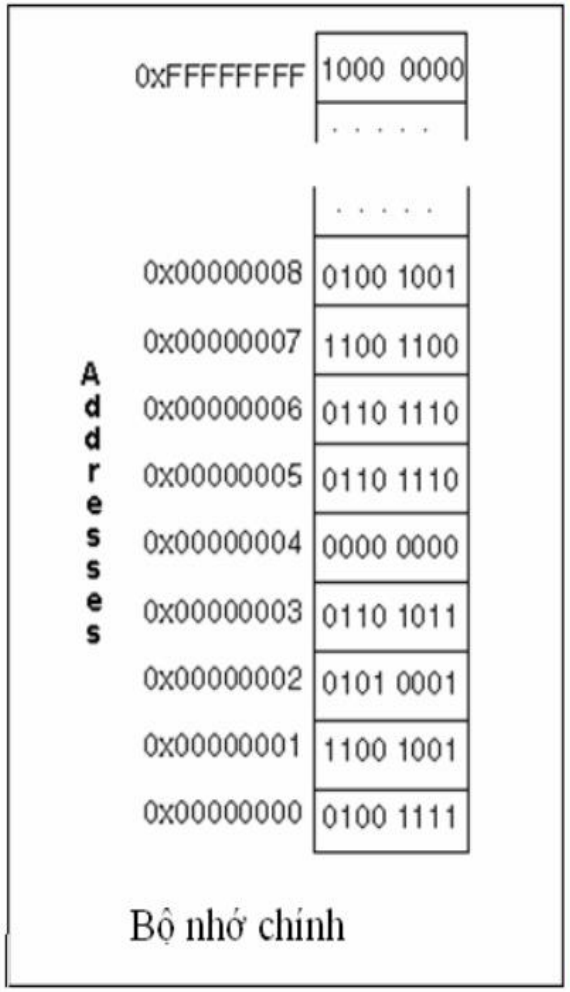
5.3. Cấu trúc lập trình

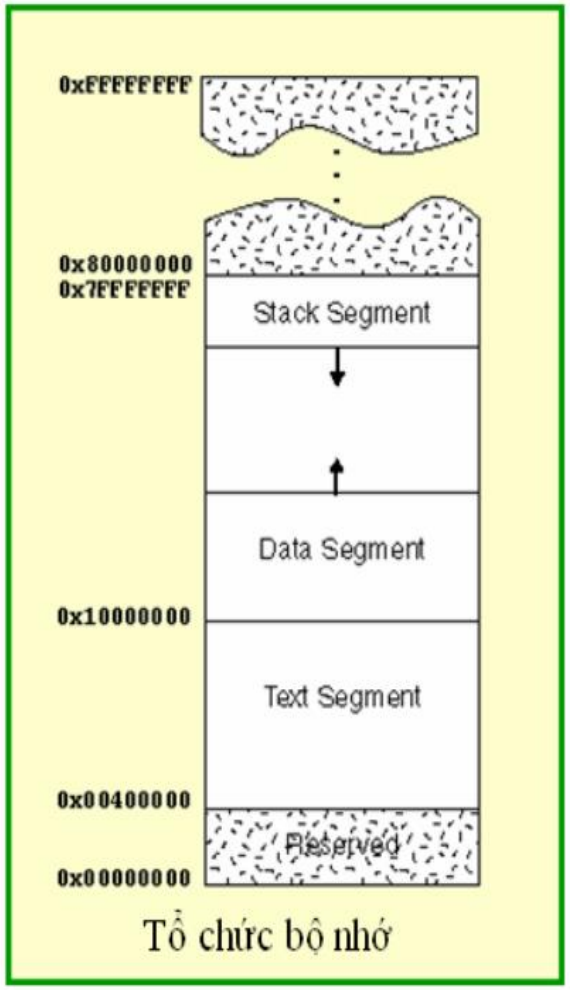
5.2. Instruction classes

- Tổ chức bộ nhớ
- Lệnh load/store
- Các lệnh tính toán
- Các lệnh so sánh
- Các lệnh rẽ nhánh/ lệnh nhảy
- Các lệnh khác

Tổ chức bộ nhớ

- Là mảng 2^{32} ô nhớ 8bit
  - 0x00000000-0xFFFFFFFF
- Người lập trình sử dụng:
  - 0x00400000-0x7FFFFFFF
- 3 Phần: text, data, stack
- Chú ý: Stack phát triển theo chiều giảm của địa chỉ
- Mips:
  - Word: 32 bit
  - Haftword: 16 bit
  - Byte: 8 bit





5.2. Instruction classes

- Tổ chức bộ nhớ
- Lệnh load/store
- Các lệnh tính toán
- Các lệnh so sánh
- Các lệnh rẽ nhánh/ lệnh nhảy
- Các lệnh khác

Lệnh load

- **Load một nhóm byte liên tục theo chiều tăng của địa chỉ, bắt đầu từ ô nhớ được chỉ định**
- **Các lệnh hợp ngữ**

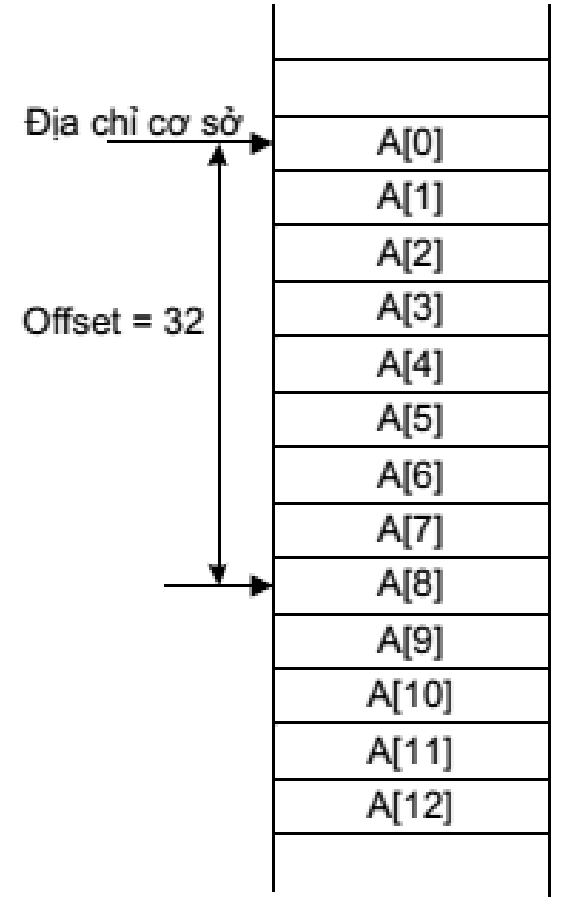
```
lw rt, imm(rs)     # Load word
lh rt, imm(rs)     # Load half_word và mở rộng dấu
lb rt, imm(rs)     # Load byte và mở rộng dấu
lhu rt, imm(rs)    # Load half_word và mở rộng số 0 đầu
lbu rt, imm(rs)    # Load byte và mở rộng số 0 đầu
```

- **Các lệnh giả mã**

```
li rd, imm         # Load giá trị 2 hằng sau khi mở rộng 0
la rd, exp         # Load d/c của nhãn, thường exp là nhãn
lui rd, imm        # Load giá trị vào 2 byte, 2 byte thấp là 0x0000
```

VD lệnh load

- a là mảng các phần tử 32-bit
  - Lấy word dữ liệu ở ô nhớ a[10],a[0] đặt vào thanh ghi $t1, $t2
  - Lấy byte dữ liệu ở ô nhớ a[1] đặt vào $t3



- Mã hợp ngữ MIPS:

```
la    $t0,a
lw    $t1, 40($t0)
lw    $t2, ($t0)
lb    $t3, 4($t0)
```

VD lệnh lui

- Dùng khởi tạo thanh ghi 32 bit

VD: Nạp vào thanh ghi $s0 giá trị 32-bit sau

0010 0001 1010 0000 0100 0000 0011 1011 = 0x21A0 403B

```
lui $s0,0x21A0          #nạp 0x21A0 vào nửa cao
ori $s0,$s0,0x403B      # nạp 0x403B vào nửa thấp
```

Nội dung $s0 sau khi thực hiện lệnh lui

0010	0001	1010	0000	0000	0000	0000	0000

Nội dung $s0 sau khi thực hiện lệnh ori

0010	0001	1010	0000	0400	0000	0011	1011

Lệnh store

- **Store một nhóm byte liên tục theo chiều tăng của địa chỉ, bắt đầu từ ô nhớ được chỉ định**
- **Các lệnh hợp ngữ**

```
sw rs, imm(rt)    # store word
sh rs, imm(rt)    # store 2 byte thấp vào bộ nhớ
sb rs, imm(rt)    # store byte thấp vào bộ nhớ
```

VD lệnh store

- a,b là 2 mảng các phần tử 32-bit
  - Lấy half_word dữ liệu ở ô nhớ a[1],a[2] lưu vào b[0]

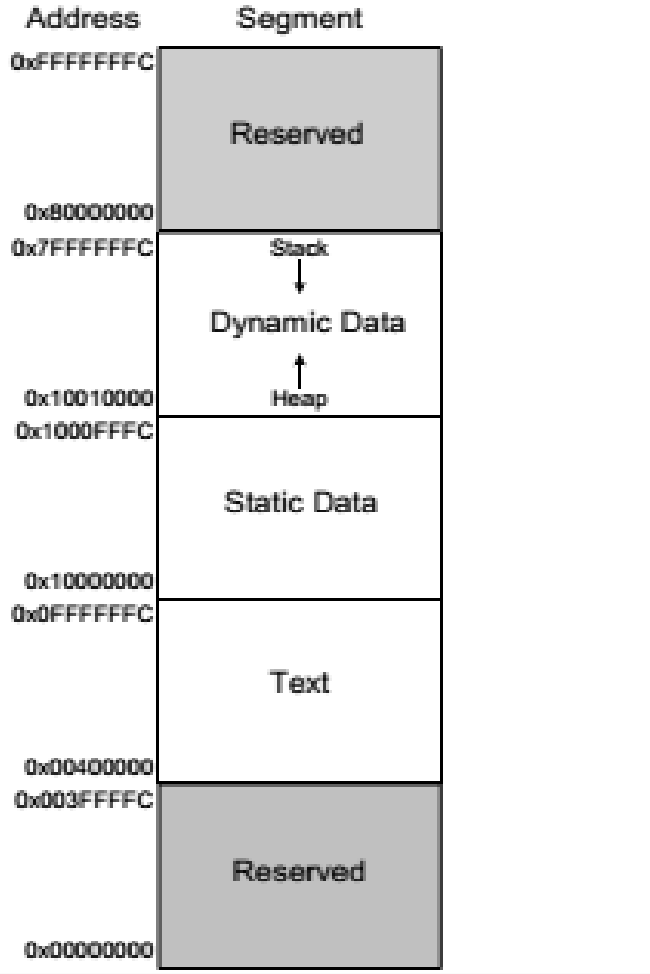
- Mã hợp ngữ MIPS:

```
la    $t0, a
lh    $t1, 4($t0)
lh    $t2, 8($t0)
la    $t3, b
sh    $t1, ($t3)
sh    $t2, 2($t3)
```

Ví dụ

- Lưu giá trị của thanh ghi $s1,$s2 vào stack, gán giá trị 0 cho 2 thanh ghi rồi khôi phục.
- Mã hợp ngữ MIPS:

```
add   $sp, $sp,-8
sw    $s1, 4($sp)
sw    $s2, ($sp)
add   $s1,$0,0
add   $s2,$0,0
lw    $s1, 4($sp)
lw    $s2, ($sp)
add   $sp, $sp,8
```



5.2. Instruction classes

- Tổ chức bộ nhớ
- Lệnh load/store
- Các lệnh tính toán
- Các lệnh so sánh
- Các lệnh rẽ nhánh/ lệnh nhảy
- Các lệnh khác

Các lệnh số học

- **Các lệnh cộng**

```
add rd,rs,rt        # rd<-rs+rt, cộng 2 số bù 2
addu rd,rs,rt       # rd<-rs+rt, cộng 2 số ko dấu
addi rd,rs,imm      # rd<-rs+imm, imm 16 bit bù 2
addiu rd,rs,imm     # rd<-rs+imm, imm 16 bit ko dấu
```

- **Các lệnh trừ**

```
sub rd,rs,rt        # rd<-rs+rt, trừ 2 số bù 2
subu rd,rs,rt       # rd<-rs+rt, trừ 2 số ko dấu
subi rd,rs,imm      # rd<-rs+imm, imm 16 bit bù 2
subiu rd,rs,imm     # rd<-rs+imm, imm 16 bit ko dấu
```

Ví dụ các lệnh số học

- Ví dụ mã C:

```
f = (g + h) - (i + j);
```

giả thiết: f, g, h, i, j nằm ở $s0, $s1, $s2, $s3, $s4

- Mã hợp ngữ MIPS:

```
add $t0, $s1, $s2     # $t0 = g + h
add $t1, $s3, $s4     # $t1 = i + j
sub $s0, $t0, $t1     # f = (g+h)-(i+j)
```

- Ví dụ mã C:

// A là mảng các phần tử 32-bit

```
A[10] = h + A[8];
```

Giả thiết: h ở $s1, $s2 chứa địa chỉ cơ sở của mảng A

- Mã hợp ngữ MIPS:

```
lw $t0, 32($s2)   # $t0 = A[8]
add $t0, $t0, $s2  # $t0 =h+A[8]
sw $t0,40($s2)    # A[10] =h+A[8]
```

Các lệnh số học

- **Các lệnh nhân/chia**

```
mult rs,rt      # (Hi,Lo) <-rs*rt, nhân 2 số bù 2
multu rs,rt     # (Hi,Lo) <-rs*rt, nhân 2 số ko dấu
div rs,rt       # Lo <-rs/rt (thương), Hi: số dư
divu rs,rt      # Lo <-rs/rt (thương), Hi: số dư
```

- **Các lệnh khác**

```
mfhi rd         # rd<-Hi
mflo rd         # rd<-Lo
move rd,rs      # rd=rs
```

Ví dụ các lệnh số học

- Ví dụ mã C:

```
f = 3x^2+5x-18;
```

Giả thiết: f, x, nằm ở $s0, $s1, các kết quả nằm vừa trong 32 bit

- Mã hợp ngữ MIPS:

```
mult $s1,$s1
mflo $s0                #       f=x^2
addi $t0,$0,3           #       $t0=3
mult $s0,$t0
mflo $s0                #       f=3x^2
ori $t0,$0,5            #       $t0=5
mult $s1,$t0
mflo $t0                #       $t0=5x
add  $s0,$s0,$t0        #        f=3x^2+5x
addi  $s0,$s0,-18       #        f=3x^2+5x-18
```

Các lệnh logic

```
and rd,rs,rt        # rd<- bitwise AND of rt, rs
andi rd,rs,imm      # rd<- bitwise AND of rt, imm

or rd,rs,rt         # rd<- bitwise OR of rt, rs
ori rd,rs,imm       # rd<- bitwise OR of rt, imm

nor rd,rs,$0        # rd<- bitwise NOT of rs
nor rd,rs,rt        # rd<- bitwise NOR of rs, rt

xor rd,rs,rt        # rd<- bitwise XOR of rs, rt
xori rd,rs,imm      # rd<- bitwise XOR of rs, imm
```

Note: imm được mở rộng 0

Ví dụ: lệnh logic

- Cho $s1, $s2

$s1	0100	0110	1010	0001	1100	0000	1011	0111
$s2	1111	1111	1111	1111	0000	0000	0000	0000

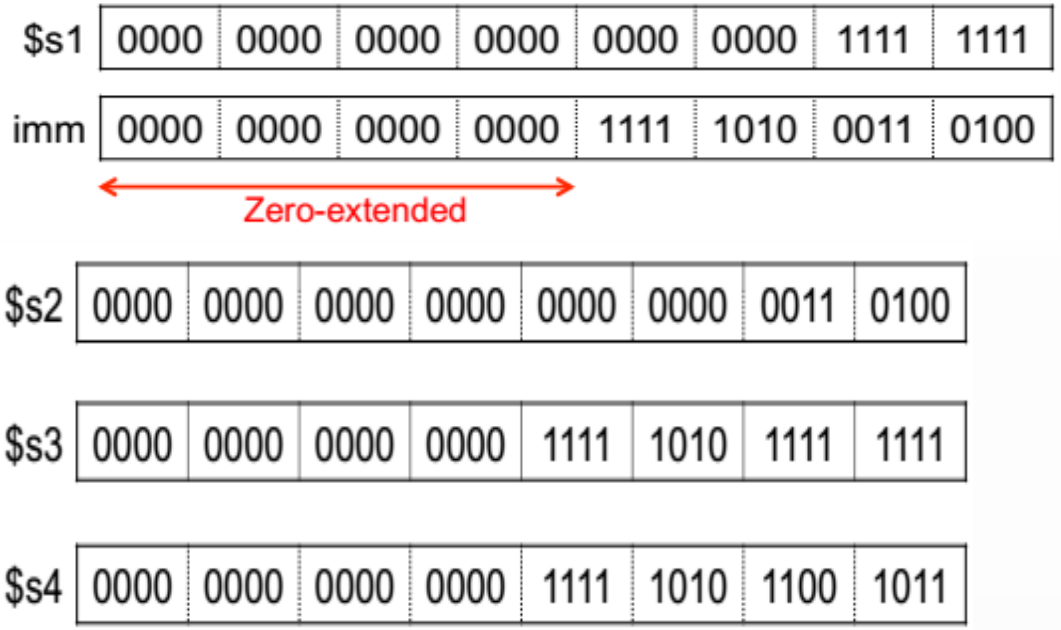
- Tính kết quả ở các thanh ghi đích

```
and $s3, $s1, $s2
or $s4, $s1, $s2
xor $s5, $s1, $s2
nor $s6, $s1, $s2
```

$s3	0100	0110	1010	0001	0000	0000	0000	0000
$s4	1111	1111	1111	1111	1100	0000	1011	0111
$s5	1011	1001	0101	1110	1100	0000	1011	0111
$s6	0000	0000	0000	0000	0011	1111	0100	1000

Ví dụ: lệnh logic kiểu

- Cho $s1, $s2



- Tính kết quả ở các thanh ghi đích

```
andi $s2,$s1,0xFA34
ori $s3,$s1,0xFA34
xori $s4,$s1,0xFA34
```

- Xóa 2 byte đầu của thanh ghi, lấy 2 byte thấp

```
andi $s2,$s2,0xFFFF
```

- Xóa 2 byte thấp của thanh ghi về 0, lấy 2 byte cao

```
lui $t0, 0xFFFF
and $s2,$s2,$t0
```

Các lệnh dịch bit

```
sll rd,rs,shft        # rd<-rs dịch trái shft bit

sll $0,$0,0           # tạo thời gian trễ, tương đương lệnh giả nop

sra rd,rs,shft        # rd<-rs dịch phải shft bit, rồi mở rộng dấu

srl rd,rt,shft        # rd<-rs dịch phải ko dấu, thêm bit 0 bên trái
```

- *Dịch trái i bit = nhân với 2^i (nếu kết quả trong phạm vi biểu diễn 32-bit)*
- *Dịch phải i bit = chia cho 2^i*

Ví dụ lệnh dịch trái sll

- Lệnh hợp ngữ:

```
sll $t2, $s0, 4 # $t2 =$s0 << 4
```

- Mã máy:

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0
000000	00000	10000	01010	00100	000000

(0x00105100)

- Ví dụ kết quả thực hiện lệnh:

$s0	0000	0000	0000	0000	0000	0000	0000	1101	= 13
$t2	0000	0000	0000	0000	0000	0000	1101	0000	= 208 (13x16)

Ví dụ lệnh dịch trái sll

- Mã C

```
int array[5]; // Mảng các phần tử 32 bit
array[0] = array[0] * 2;
array[1] = array[1] * 2;
```

- Mã hợp ngữ MIPS

 # Giả sử địa chỉ cs của mảng 0x12348000

```
lui $s0, 0x1234          # 0x1234 vào nửa cao của $S0
ori $s0,$s0,0x8000       #0x8000 vào nửa thấp của $s0
lw  $t1, 0($s0)          # $t1 = array[0]
sll $t1, $t1, 1          # $t1 = $t1 * 2
sw  $t1, 0($s0)          # array[0] = $t1
lw  $t1, 4($s0)          # $t1 = array[1]
sll $t1, $t1, 1          # $t1 = $t1 * 2
sw $t1, 4($s0)           # array[1] = $t1
```

Ví dụ lệnh dịch phải srl

- Lệnh hợp ngữ:

```
srl $s2, $s1, 2 # $s2 =$s1 >> 2
```

- Mã máy:

op	rs	rt	rd	shamt	funct
0	0	17	18	2	2
000000	00000	10001	10010	00010	000010

(0x00119082)

- Ví dụ kết quả:

$s1	0000	0000	0000	0000	0000	0000	0101	0110	= 86
$s2	0000	0000	0000	0000	0000	0000	0001	0101	= 21 [86/4]

Ví dụ lệnh dịch phải sra

- Lệnh hợp ngữ:

```
sra $s2, $s1, 2 # $s2 =$s1 >> 2
```

- Ví dụ kết quả:

$s1	1111	1111	1111	1111	1111	1111	1011	0100	-76
$s2	1111	1111	1111	1111	1111	1111	1110	1101	-19 =-76/4

VD lệnh dịch srl

- VD: Nạp 0x12345678 vào thanh ghi $t0, nạp địa chỉ nhãn *tape vào* $t1. Lưu 2 byte cao vào địa chỉ ($t1), 2 byte thấp vào 2+($t1),

```
.data
        tape: .space 1024
.text
.globl  main
main:
        lui     $t0, 0x1234
        ori     $t0, $t0, 0x5678
        la      $t1, tape
        sh      $t0, 2($t1)
        srl     $t0,$t0,16
        sh      $t0, ($t1)
```

5.2. Instruction classes

- Tổ chức bộ nhớ
- Lệnh load/store
- Các lệnh tính toán
- Các lệnh so sánh
- Các lệnh rẽ nhánh/ lệnh nhảy
- Các lệnh khác

Các lệnh so sánh

```
slt rd,rs,rt      # if rs<rt   rd<-1 else rd<-0, rt, rs dạng bù 2
sltu rd,rs,rt     # if rs<rt   rd<-1 else rd<-0, rt, rs dạng ko dấu
slti rd,rs,imm    # if rs<imm  rd<-1 else rd<-0, rs, imm  dạng bù 2
sltiu rd,rs,imm   # if rs<imm  rd<-1 else rd<-0, rs, imm dạng ko dấu
```

VD:

```
slt $s0,$t1,$t2
slti $s0,$t1,100
```

5.2. Instruction classes

- Tổ chức bộ nhớ
- Lệnh load/store
- Các lệnh tính toán
- Các lệnh so sánh
- Các lệnh rẽ nhánh/ lệnh nhảy
- Các lệnh khác

Các lệnh rẽ nhánh

```
beq rs,rt,label    # if rs=rt   nhảy đến label, cần theo sau lệnh nop
bne rs,rt,label    # if rs!=rt  nhảy đến label, cần theo sau lệnh nop
bgez rs,label      # if rs>=0   nhảy đến label, cần theo sau lệnh nop
bgtz rs,label      # if rs>0    nhảy đến label, cần theo sau lệnh nop
blez rs,label      # if rs<=0   nhảy đến label, cần theo sau lệnh nop
bltz rs,label      # if rs<0    nhảy đến label, cần theo sau lệnh nop
```

Note: *4 lệnh cuối là lệnh giả ngữ*

VD: if $s0 ==0 $s0=1; else $s0=2;

```
    beq $s0,$0,L1
          addi  $s0,$0,2
          j exit
    L1: addi $s0,$0,1
    exit
```

Các lệnh nhảy

```
j label          # Nhảy ko điều kiện đến label
jal label        # Lưu đ/c trở về vào $ra. Nhảy đến label (dùng khi gọi hàm)
jr rs            # Nhảy đến đ/c trong thanh ghi rs, dùng trở về từ lời gọi hàm
```

- **Ví dụ:**

  Đổi chỗ giá trị của x và y trong bộ nhớ

Ví dụ

```
.data           x:      .word
                y:      .word
.globl main
main:
# Nạp đ/c x,y thanh ghi $a0, $a1
        la      $a0, x
        la      $a1,y
# Nhảy đến hàm swap, đổi giá trị của x, y
        jal swap
# Return here, kết thúc chương trình
        li      $v0,10
        syscall
```

```
swap:
        lw      $t0, ($a0)
        lw      $t1, ($a1)
        sw      $t0, ($a1)
        sw      $t1, ($a0)
        jr $ra
```

5.2. Instruction classes

- Tổ chức bộ nhớ
- Lệnh load/store
- Các lệnh tính toán
- Các lệnh so sánh
- Các lệnh rẽ nhánh / lệnh nhảy
- Các lệnh khác

Lệnh syscall

- Làm treo chương trình và chuyển quyền điều khiển cho HĐH
- HĐH thực thi căn cứ giá trị *$v0*
- Mars cung cấp các hàm hệ thống để xử lý thao tác xuất nhập

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)

VD

- **Viết đoạn c/trình đọc họ tên dài 24 bit**

```
.data
prompt: .asciiz  " Nhập Họ tên:"
hoten: .space 24
. . .
li          $v0, 4      # gán giá trị cho $v0
la          $a0, prompt
syscall                 # In chuỗi prompt  được chỉ bởi $a0
li          $v0, 8
 la         $a0, hoten
li          $a1, 24
syscall                 # Nhập chuỗi ghi vào d/c $a0 với chiều dài 24 bit
li          $v0, 10
syscall                 # Kết thúc
```

Chương 5

5.1. Tập thanh ghi (Register Set)

5.2. Các lớp lệnh (Instruction Classes)

5.3. Cấu trúc lập trình

5.3 Cấu trúc lập trình

- Khung mẫu chương trình
- Khai báo dữ liệu
- Địa chỉ bắt đầu trong bộ nhớ
- Cấu trúc điều khiển
- Chương trình con
- Các lệnh với dấu phẩy động

Khung mẫu chương trình

```
# Title:                                    Filename:
# Author:                                   Date:
# Description:
# Input:
# Output:
################### Data segment ########################
.data
  . . .
################### Code segment ########################
.text
.globl main
main:                                       # main program entry
  . . .
li $v0, 10                                  # Exit program
syscall
```

Khung mẫu chương trình

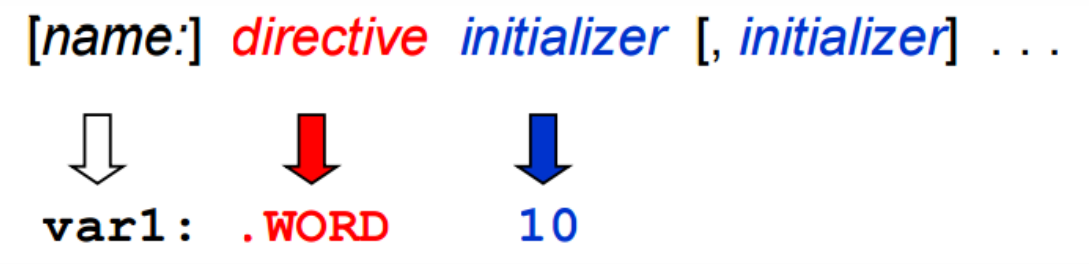
- **Chỉ thị**
  - *Chỉ thị .data*
    - *Định nghĩa phân đoạn dữ liệu*
    - *Các biến chương trình được định nghĩa tại vùng này*
    - *Assemler sẽ cấp phát và khởi tạo các biến này*
  - *Chỉ thị .text*
    - *Định nghĩa phân đoạn mã của chương trình, và chứa các lệnh*
  - *Chỉ thị .globl*
    - *Khai báo ký hiệu toàn cục*
    - *Ký hiệu main dùng chỉ thị toàn cục*
    - *Các ký hiệu toàn cục có thể tham khảo từ các file khác nhau*

5.3 Cấu trúc lập trình

- Khung mẫu chương trình
- Khai báo dữ liệu
- Địa chỉ bắt đầu trong bộ nhớ
- Cấu trúc điều khiển
- Chương trình con
- Các lệnh với dấu phẩy động

Khai báo dữ liệu

- **Cú pháp:**



  - Các giá trị khởi tạo chuyển sang dạng nhị phân trong vùng nhớ DL tương ứng

- **Tên:**
  - Là chuỗi ký tự, số, dấu gạch dưới (_) và dấu chấm
  - Không bắt đầu bằng số, không trùng từ dành riêng opcode

Khai báo dữ liệu

- **Các chỉ thị hỗ trợ:**

  .ascii str: **Lưu chuỗi vào bộ nhớ. Ko có ký tự kết thúc chuỗi**

  .asciiz str: **Lưu chuỗi vào bộ nhớ. Ký tự kết thúc chuỗi là Null**

  .byte b1,…,bn: **Lưu n byte b1,..bn vào bộ nhớ**

  .half h1,…,hn: **Lưu n haft word h1,..hn vào bộ nhớ**

  .word w1,…,wn: **Lưu n word w1,..wn vào bộ nhớ**

  .float f1,…,fn: **Lưu n số thực f1,..fn vào bộ nhớ (mỗi gtrị 32 bit)**

  .double d1,…,dn: **Lưu n số thực độ cxác đôi vào bộ nhớ**

  .space n: **Cấp phát ô nhớ n byte, ko khởi tạo giá trị**

  .align n: **Địa chỉ bắt đầu của ô nhớ kế tiếp chia hết 2^n**

VD khai báo biến

```
.data
var1: .byte     'A','b',-1,120,'\n'
var2: .half     -7, 0x1234
var3: .word     100:10
var4: .asciiz   "Họ và tên:"
var5: .ascii    "Nghề nghiệp:\n"
var6: .float    12.3, -5.7
var7: .space    40
```

Ví dụ

```
# In chuỗi
.data
        str1: .asciiz “Hello World”
.text
.globl          main
main:
        # In chuỗi str
        li      $v0, 4
        la      $a0, str1
        syscall
        # Thoát khỏi chương trình
        li      $v0, 10
        syscall
```

Ví dụ

 # Viết chương trình nhập Họ tên, tuổi

```
.data
        prompt1: .asciiz  " Nhập Họ tên:"
        prompt2: .asciiz  "Nhập tuổi:"
        hoten: .space 24
        tuoi: .word   0
.text
.globl  main
main:
        li          $v0, 4
        la          $a0, prompt1
        syscall                 #  In prompt1

# Nhập chuỗi ghi vào d/c $a0, chiều dài $a1
        li          $v0, 8
        la          $a0, hoten
        li          $a1, 24
        syscall
                    #  Nhập tuổi
        li          $v0, 4
        la          $a0, prompt2
        syscall                         #  In prompt2
        li          $v0, 5
        syscall
        la          $a0, tuoi
        sw          $v0, $a0
        li          $v0, 10
        syscall             #  Kết thúc
```

5.3 Cấu trúc lập trình

- Khung mẫu chương trình
- Khai báo dữ liệu
- Địa chỉ bắt đầu trong bộ nhớ
- Cấu trúc điều khiển
- Chương trình con
- Các lệnh với dấu phẩy động

Địa chỉ bắt đầu trong (Alignment) bộ nhớ

- Bộ nhớ được xem là một mảng các ô nhớ 1 byte, mỗi byte có một địa chỉ tương ứng
- Mỗi lệnh MIPS là một số nhị phân 4 byte, lưu liên tiếp trong bộ nhớ
- Alighnment: Địa chỉ bắt đầu phải chia hết cho kích thước
  - Địa chỉ của một word là số chia hết cho 4
  - Địa chỉ một half word là số chia hết cho 2
- Chỉ thị: *.align n*
  - Quy định: Địa chỉ bắt đầu của biến khai báo kế tiếp là số chia hết 2^n

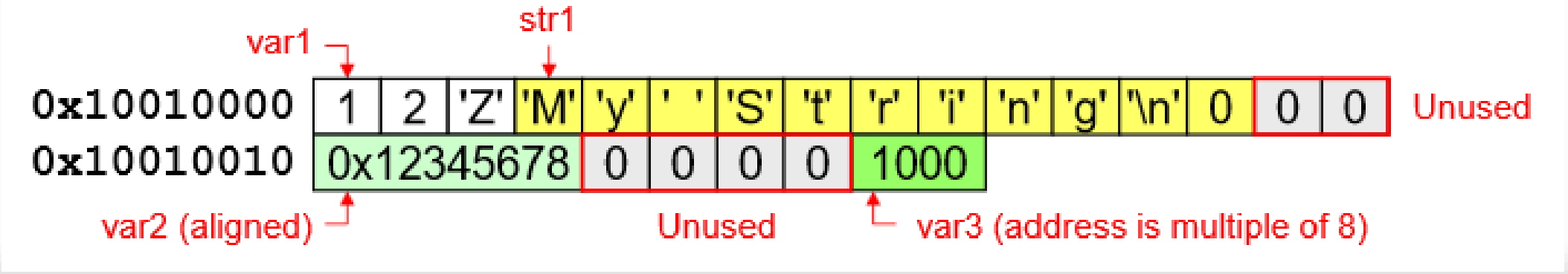
Bảng ký hiệu (Symbol table)

- Assembler tạo bảng ký hiệu cho các biến
- Ví dụ

```
.data
    var1:   .byte 1,2,'Z'
    str1:   .asciiz "My String\n"
    var2:   .word  0x12345678
    .align  3
    var3:   .half   1000
```

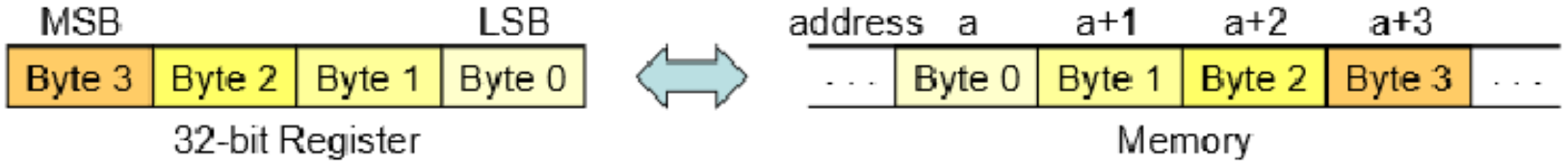
Symbol Table

Label	Address
var1	0x10010000
str1	0x10010003
var2	0x10010010
var3	0x10010018

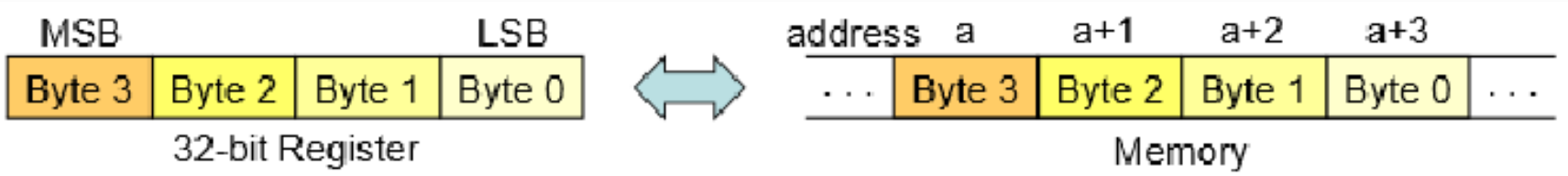


Thứ tự trọng số các ô nhớ

- BXL xác định thứ tự trọng số các ô nhớ trong một word theo
- Little Endian
  - Địa chỉ bắt đầu bằng địa chỉ byte trọng số nhỏ nhất LSB
  - Ví dụ: Intel IA-32, Alpha



- Big Endian
  - Địa chỉ bắt đầu bằng địa chỉ byte trọng số lớn nhất MSB
  - Ví dụ: SPARC, PA-RICS



- MIPS hỗ trợ cả 2 dạng định thứ tự byte ở trên

5.3 Cấu trúc lập trình

- Cấu trúc chương trình
- Khai báo biến
- Địa chỉ bắt đầu trong bộ nhớ
- Cấu trúc điều khiển
- Chương trình con
- Các lệnh với dấu phẩy động

Cấu trúc điều khiển

- Các cấu trúc rẽ nhánh
  - Cấu trúc **`if`**
  - Cấu trúc **`if/else`**
  - Cấu trúc **`switch/case`**
- Các cấu trúc lặp
  - Cấu trúc **`while`**
  - Cấu trúc **`do while`**
  - Cấu trúc **`for`**

Các lệnh rẽ nhánh/ nhảy (nhắc lại)

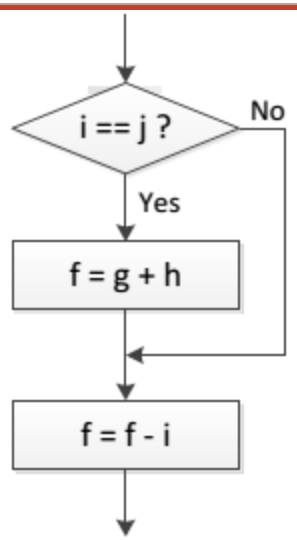
```
beq rs,rt,label    # if rs=rt   nhảy đến label, cần theo  sau lệnh nop
bne rs,rt,label    # if rs!=rt   nhảy đến label, cần theo sau lệnh nop
bgez rs,label      # if rs>=0  nhảy đến label, cần theo sau lệnh nop
bgtz rs,label      # if rs>0   nhảy đến label, cần theo sau lệnh nop
blez rs,label      # if rs<=0   nhảy đến label, cần theo sau lệnh nop
bltz rs,label      # if rs<0   nhảy đến label, cần theo sau lệnh nop
j label            # Nhảy ko điều kiện đến label

jal label          # Lưu đ/c trở về vào $ra. Nhảy đến label (dùng
                     khi gọi hàm)
jr rs              # Nhảy đến đ/c trong thanh ghi rs, dùng trở về từ
                     lời gọi hàm
```

Chuyển đổi cấu trúc if

- Mã C:

```
if (i==j)
    f = g+h;
f = f-i;
```



- Mã MIPS:

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j

bne $s3, $s4, L1   # Nhảy nếu i<>j
add $s0, $s1, $s2          # f=g+h
L1:  sub $s0, $s0, $s3     # f=f-i
```

- Mã C:

```
if (i==j)
    f = g+h;
else f = f-i;
```

- Mã MIPS:

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j

    bne $s3, $s4, L1
    add $s0, $s1, $s2
    j Exit
L1:  sub $s0, $s0, $s3
Exit:…
```

Chuyển đổi cấu trúc switch/case

- Mã C:

```
switch (amount) {
case 20: fee = 2; break;
case 30: fee = 3; break;
case 40: fee = 4; break;
default: fee = 0;
}
// tương đương với sử dụng các câu lệnh if/else

if(amount = = 20) fee = 2;
else if (amount = = 30) fee = 3;
else if (amount = = 40) fee = 4;
else fee = 0;
```

Chuyển đổi cấu trúc switch/case

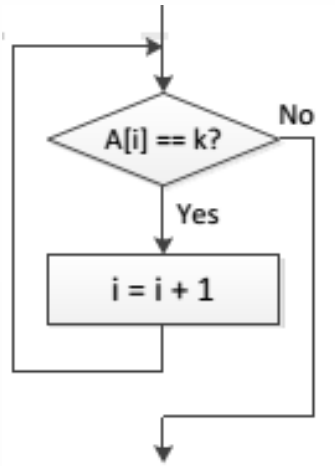
- Mã hợp ngữ MIPS

```
# $s0 = amount, $s1 = fee
case20:
    addi  $t0, $0, 20          # $t0 = 20
    bne   $s0, $t0, case30     # amount != 20, skip to case30
    addi  $s1, $0, 2           # fee = 2
    j   donecase               # break out of case
case30:
    addi  $t0, $0, 30          # $t0 = 30
    bne   $s0, $t0, case40     # amount != 30, skip to case40
    addi  $s1, $0, 3           # fee = 3
    j   donecase               # break out of case
case40:
    addi  $t0, $0, 40          # $t0 = 40
    bne   $s0, $t0, default    # amount != 40, go to default
    addi  $s1, $0, 4           # fee = 4
    j donecase                 # out of case
default:
addi   $s1 ,$0, 0              # fee = 0
donecase: …
```

Chuyển đổi cấu trúc while

- Mã C:

```
while (A[i] == k) i += 1;
```



- Mã hợp ngữ MIPS:

• *# $s1=i, $s2=k, đ/c cơ sở của mảng A ở $s3*

```
Loop:
    sll $t0, $s1, 2          # $t0 = 4*i
    add $t0, $t0, $s3        # $t0 = địa chỉ A[i]
    lw $t1, 0($t0)           # $t1 = A[i]
    bne $t1, $s2, Exit       # nếu A[i]<>k thì Exit
    addi $s1, $s1, 1         # nếu A[i]=k thì i=i+1
    j Loop                   # quay lại Loop
Exit: …
```

Chuyển đổi cấu trúc while

- Mã C:

```
sum=0; i=0;
while (i <= k) {i += 1;sum=sum+A[i]*8;}
printf("%d",sum);
```

A mảng các phần tử 32 bit

- Mã MIPS:

```
# $s1=i, $s2=k, $s3 chứa địa chỉ cơ sở của A, $s4= sum
      add $s4, $0,0
      add $s1, $0,0
Loop: subu $t2,$s1,$s2
      bgtz $t2,Exit              # nếu i>k thì Exit
      addi $s1, $s1, 1           # i=i+1
      sll $t0, $s1, 2            # $t0 = 4*i
      add $t0, $t0, $s3          # $t0 = địa chỉ A[i]
      lw  $t1, 0($t0)            # $t1 = A[i]
      sll $t1, $t1,3             # A[i]=A[i]*8
      add $s4,$s4,$t1            # sum=sum+A[i]*8
      j   Loop                   # quay lại Loop
Exit: move $a0, $s4
      li $v0, 1                  # In số nguyên
      syscall
      li $v0, 10
      syscall                    # Kết thúc
```

Chuyển đổi cấu trúc for

- Mã C:

```
// add the numbers from 0 to 9
   int sum = 0;
   int i;
   for (i=0; i!=10; i++) { sum = sum + i; }
```

- Mã hợp ngữ MIPS:

```
   # $s0 = i, $s1 = sum
         addi  $s1, $0, 0      # sum = 0
         add   $s0, $0, $0     # i = 0
         addi  $t0, $0, 10     # $t0 = 10
   for:  beq   $s0, $t0, done  # Nếu i=10, thoát
         add   $s1, $s1, $s0   # sum = sum+i
         addi  $s0, $s0, 1     # tăng i thêm 1
         j     for             # quay lại for
   done: ...
```

Chuyển đổi cấu trúc for

- VD: *A là mảng 20 phần tử 32 bit, tính tổng các phần tử của A và in tổng*
- Mã hợp ngữ MIPS:

```
# $s0 = i, $s1 = sum, $s2: địa chỉ cơ sở của A
        addi  $s1, $0, 0       # sum = 0
        add   $s0, $0, $0      # i = 0
        addi  $t0, $0, 20      # $t0 = 20
for:    beq   $s0, $t0, done   # Nếu i=20, thoát
        sll   $t1, $s0, 2      # $t1 = 4*i
        add   $t1, $t1, $s2    # $t1 = địa chỉ A[i]
        lw    $t2, 0($t1)      # $t2 = A[i]
        add   $s1, $s1, $t2    # Nếu i<20 thì sum = sum+A[i]
        addi  $s0, $s0, 1      # tăng i thêm 1
        j     for              # quay lại for
done:   move $a0, $s1
        li $v0, 1              # In số nguyên
        syscall
        li $v0, 10
        syscall
```

Bài tập

1. *Tính tổng các số từ : S= 1+2+3+…+n*
2. *Tính tổng các số chẵn từ : S= 0+2+4+…+2n*
3. *Tính tổng các số lẻ từ : S= 1+3+…+2n+1*
4. *Tính tổng : S= 1!+2!+…+n!*