Thực hành
Kiến trúc máy tính

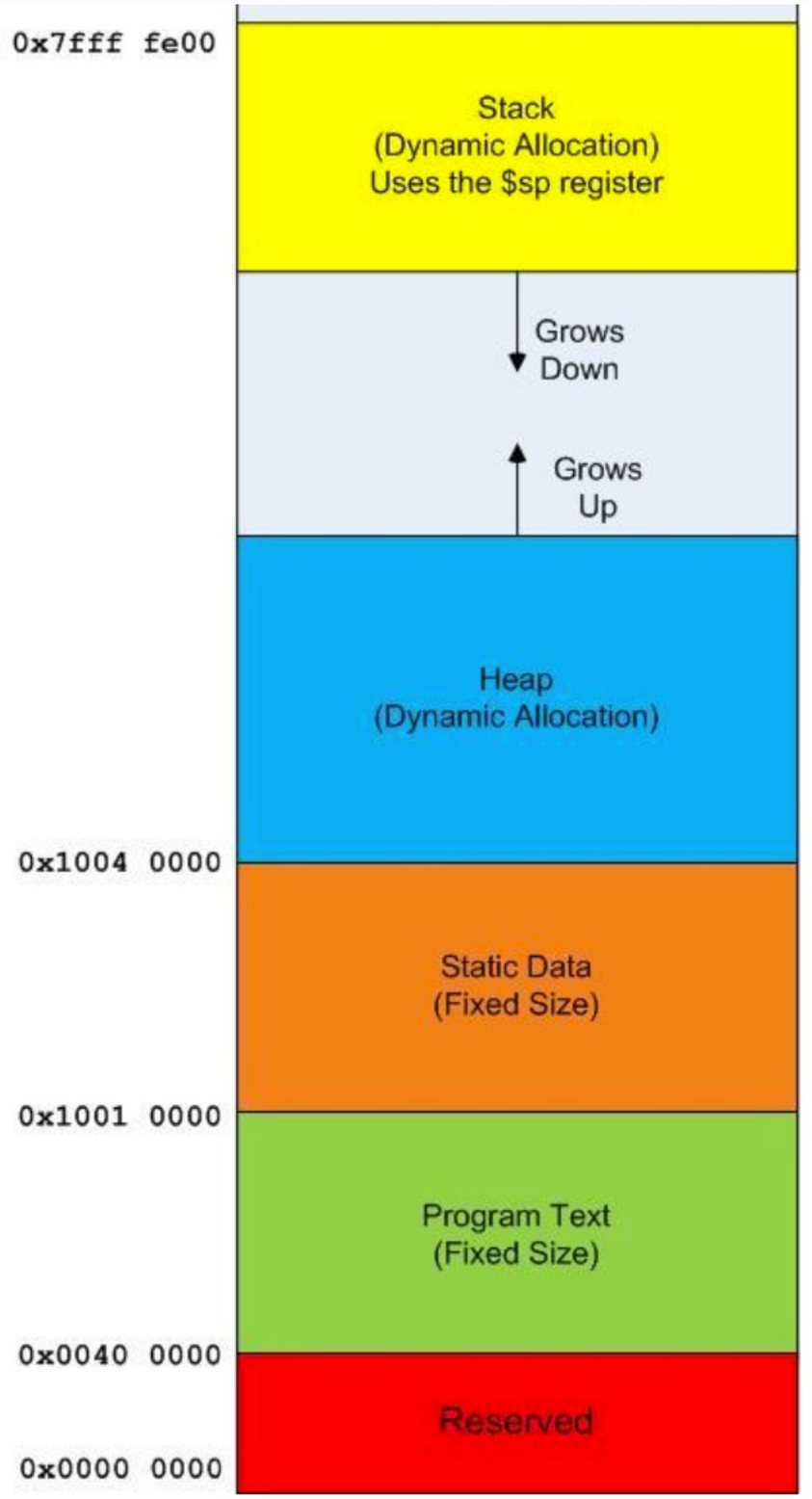
Giảng viên: Nguyễn Thị Thanh Nga
Khoa Kỹ thuật máy tính
Trường CNTT&TT

Tuần 8 Bộ nhớ tĩnh

- Phân đoạn dữ liệu
- Mô hình bộ nhớ phẳng
- Bô nhớ tĩnh
- Truy cập bộ nhớ
- Các phương pháp truy cập bộ nhớ

Bộ nhớ MIPS – Phân đoạn dữ liệu

- Có 3 loại bộ nhớ chính:
  - Bộ nhớ tĩnh
  - Bộ nhớ động stack
  - Bộ nhớ động heap
- Bộ nhớ tĩnh là bộ nhớ đơn giản nhất vì nó được định nghĩa khi chương trình được xây dựng và cấp phát khi chương trình bắt đầu thực thi.
- Bộ nhớ động được cấp phát trong khi chương trình đang chạy và được truy cập bằng các hiệu số địa chỉ. Điều này làm cho bộ nhớ động khó truy cập hơn trong một chương trình, nhưng hữu ích hơn nhiều.



Mô hình bộ nhớ phẳng

- Đối với một lập trình viên MIPS, bộ nhớ dường như là phẳng; không có cấu trúc nào cho nó.

- Bộ nhớ bao gồm một byte (8 bit) được lưu trữ nối tiếp nhau và tất cả các byte đều bằng nhau.

- Lập trình viên MIPS thấy một bộ nhớ nơi các byte được lưu trữ dưới dạng một mảng lớn và chỉ mục của mảng là địa chỉ byte.

- Bộ nhớ có thể định địa chỉ cho từng byte, và do đó được gọi là byte có thể định địa chỉ.

Mô hình bộ nhớ phẳng

Các byte trong MIPS được tổ chức thành các nhóm:

- Một byte đơn
- Một nhóm 2 byte, được gọi là nửa từ
- Một nhóm 4 byte, được gọi là một từ
- Một nhóm 8 byte, được gọi là một từ kép

Mô hình bộ nhớ phẳng

Tất cả các nhóm bắt đầu từ **0x10010000** và sau đó diễn ra đều đặn.

- Các nửa từ trong bộ nhớ sẽ bắt đầu tại các địa chỉ **0x10010000**, **0x10010002**, **0x10010004** và tiếp tục theo cách đó.

- Các từ bộ nhớ sẽ bắt đầu tại các địa chỉ **0x10010000**, **0x10010004**, **0x10010008**, **0x1001000c** và tiếp tục như vậy.

- Các từ kép trong bộ nhớ sẽ bắt đầu tại các địa chỉ **0x10010000**, **0x10010008**, **0x10010010**, **0x10010018** và tiếp tục.

Mô hình bộ nhớ phẳng

- Các nhóm bộ nhớ bắt đầu được gọi là một ranh giới.
- Không thể định địa chỉ cho một nhóm dữ liệu ngoại trừ ở ranh giới cho loại đó.
- Ví dụ, một từ bộ nhớ không thể được tải tại địa chỉ **0x10010002** vì nó không nằm trên ranh giới từ.
- Khi thảo luận về dữ liệu, một từ bộ nhớ có độ lớn 4 byte (32 bit), và nó cũng nằm trên một ranh giới từ.
- Nếu 32 bit không được căn chỉnh trên một ranh giới từ, thì việc coi nó là một từ là không chính xác.

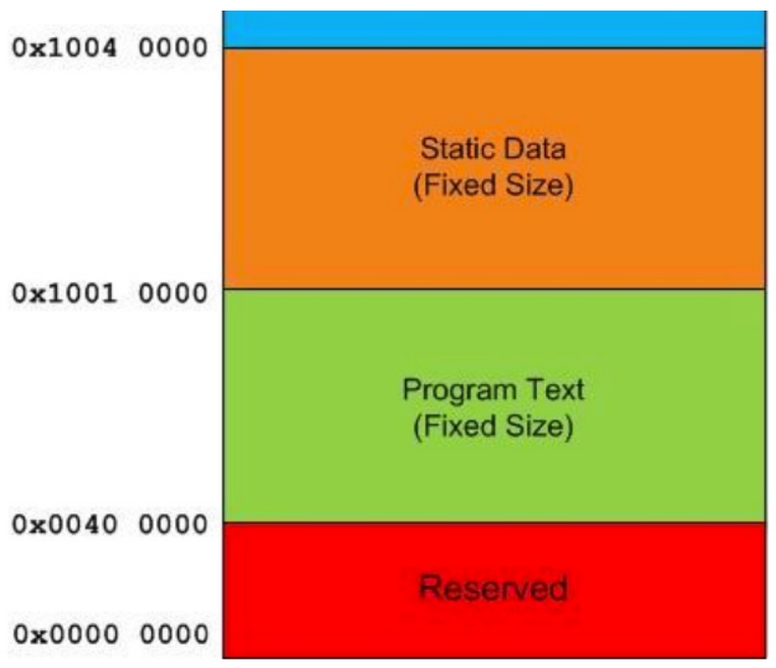
Bộ nhớ tĩnh

- Dữ liệu tĩnh là dữ liệu được xác định khi chương trình được xây dựng và cấp phát khi chương trình bắt đầu chạy.

- Kích thước và vi trí của dữ liệu tĩnh là cố đinh và không thể thay đổi.

- Nếu một mảng tĩnh được khai báo 10 phần tử, thì không thể thay đổi kích thước thành 20 phần tử.

- Tất cả các biến được định nghĩa là tĩnh phải được biết trước khi chương trình được chạy.



Bộ nhớ tĩnh

- Dữ liệu tĩnh được định nghĩa bằng cách sử dụng chỉ thị trình hợp dịch **.data**.

- Tất cả bộ nhớ được cấp phát trong chương trình trong khai báo **.data** là dữ liệu tĩnh.

- Đoạn dữ liệu tĩnh (hay đơn giản là dữ liệu) của bộ nhớ là phần bộ nhớ bắt đầu từ địa chỉ **0x10010000** đến địa chỉ **0x10040000**.

- Các giá trị dữ liệu có thể thay đổi trong quá trình thực hiện chương trình nhưng kích thước và địa chỉ dữ liệu không thay đổi.

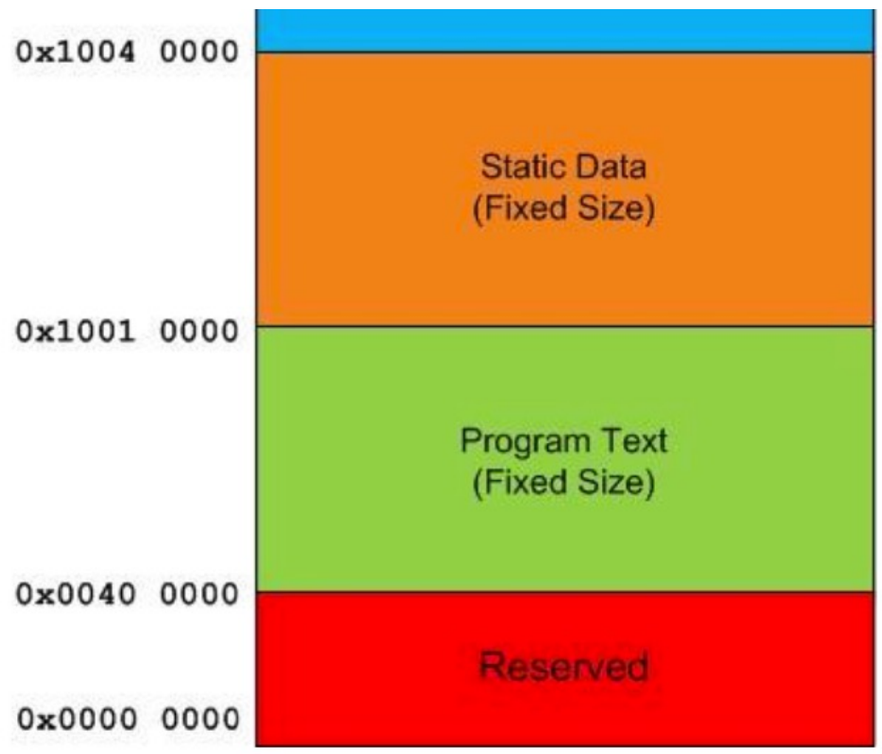
Bộ nhớ tĩnh

- Khi trình hợp dịch bắt đầu thực thi, nó sẽ theo dõi địa chỉ tiếp theo trong phân đoạn dữ liệu.

- Ban đầu, giá trị của vị trí khả dụng tiếp theo trong phân đoạn dữ liệu được thiết lập là **0x10010000**.

- Khi không gian được phân bổ trong phân đoạn dữ liệu, vị trí khả dụng tiếp theo sẽ tăng lên theo lượng không gian được yêu cầu.

- Điều này cho phép trình hợp dịch theo dõi nơi lưu trữ mục dữ liệu tiếp theo.



Bộ nhớ tĩnh

- Xét đoạn mã MIPS sau:

```
.data
a:  .word   0x1234567
    .space  14
b:  .word   0xFEDCBA98
```

Address	Value (+0)	Value (+4)
0x10010000	0x01234567	0x00000000
0x10010020	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000

- Nếu đây là chỉ thị **.data** đầu tiên được tìm thấy, thì địa chỉ để bắt đầu đặt dữ liệu là **0x10010000**.

- Một từ có 4 byte bộ nhớ, vì vậy nhãn **a:** trỏ đến phân bổ 4 byte bộ nhớ tại địa chỉ **0x10010000** và mở rộng đến **0x10010003**, và địa chỉ trống tiếp theo trong phân đoạn dữ liệu được cập nhật thành **0x10010004**.

Bộ nhớ tĩnh

- Xét đoạn mã MIPS sau:

```
.data
a:  .word   0x1234567
    .space 14
b:  .word   0xFEDCBA98
```

Data Segment

Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x00000000	0x00000000	0x00000000	0x0000000
0x00000000	0x00000000	0x00000000	0x0000000
0x00000000	0x00000000	0x00000000	0x0000000
0x00000000	0x00000000	0x00000000	0x0000000

- Next an area of memory is allocated that using the **.space 14** assembly directive. The **.space** directive sets aside 14 bytes of memory, starting at 0x10010004 and extending to 0x10010011.
- There is no label on this part of the data segment, which means that the programmer must access it directly through an address.
- Generally, there will be a label present for variables in the data segment.

Bộ nhớ tĩnh

- Xét đoạn mã MIPS sau:

```
.data
a:  .word   0x1234567
    .space  14
b:  .word   0xFEDCBA98
```

(+10)	Value (+14)	Value (+18)
0x00000000	0xfedcba98	0x00000000
0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000

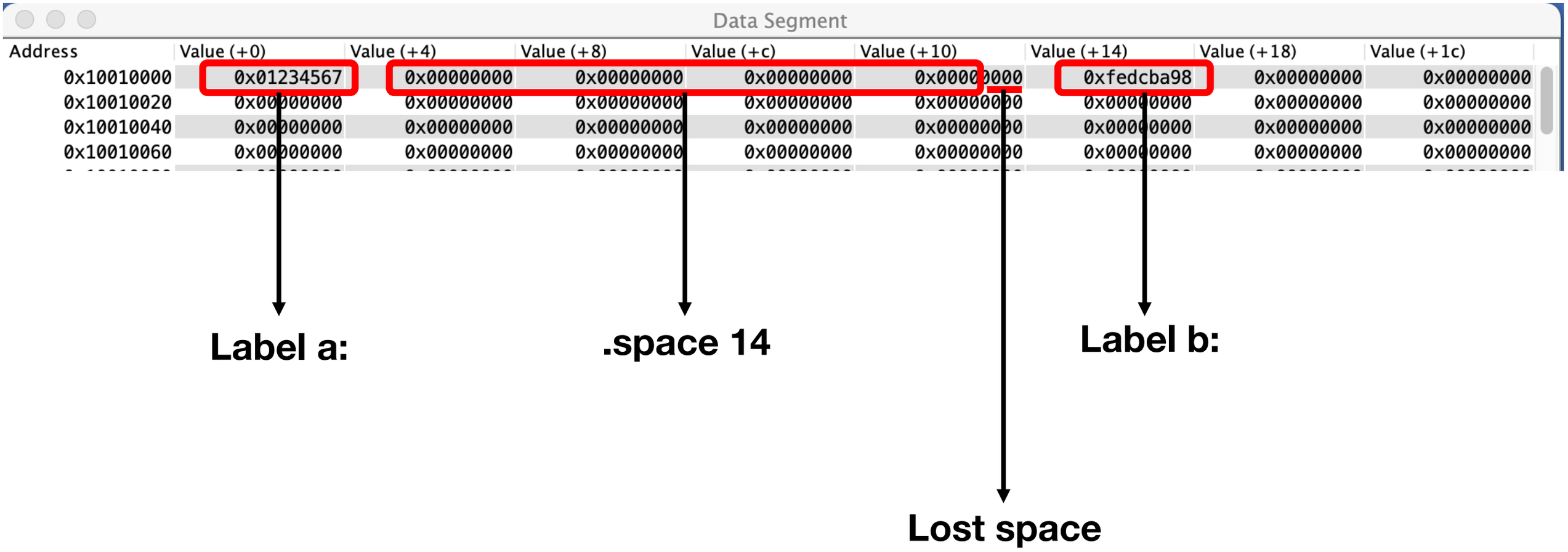
- Cuối cùng, một từ nhớ khác được cấp phát tại nhãn **b:**

- Từ nhớ này có thể đã được đặt ở **0x10010012**, vì đây là byte khả dụng tiếp theo. Tuy nhiên, dữ liệu này được khai báo là một từ có nghĩa là nó phải được đặt trên một ranh giới từ.

- Nếu địa chỉ có sẵn tiếp theo không nằm trên ranh giới từ khi yêu cầu cấp phát từ, trình hợp dịch sẽ di chuyển đến ranh giới từ tiếp theo và khoảng cách ở giữa đơn giản là bị bỏ qua.

Bộ nhớ tĩnh

- Bộ nhớ trông như thế này sau khi biên dịch đoạn mã trên.



Bộ nhớ tĩnh

- Địa chỉ cột cung cấp địa chỉ cơ sở cho một nhóm các địa chỉ 32 (0x20) byte.

- Mỗi cột tiếp theo là khoảng cách 4 byte (hoặc từ) từ địa chỉ cơ sở.

- Cột đầu tiên là địa chỉ cơ sở + 0 byte, vì vậy nó là địa chỉ **0x10010000** - **0x10010003**, cột thứ hai là địa chỉ **0x10010004** - **0x10010007**, v.v.

- Bộ nhớ ở nhãn **a:** lưu trữ **0x01234567**, sau đó 14 byte bộ nhớ chưa khởi tạo được cấp phát, hai byte bộ nhớ tiếp theo không được sử dụng và bị mất, và cuối cùng là một từ ở nhãn **b:** lưu trữ 0xfedcba98.

Accessing memory

- Tất cả truy cập bộ nhớ trong MIPS được thực hiện thông qua một số hình thức của toán tử load hoặc store.

- Các toán tử này bao gồm load/store một byte (lb, sb); nửa từ (lh, sh); từ (lw, sw); hoặc từ kép (ld, sd).

- Trong nội dung này chỉ xem xét các từ nhớ nên chỉ giới thiệu **lw** và **sw**.

Toán tử lw

- Định dạng thực duy nhất: một địa chỉ được lưu trữ trong **R_s** và một phần bù từ địa chỉ đó được lưu trữ trong giá trị **Immediate**. Giá trị của bộ nhớ tại [**R_s** + **Immediate**] được lưu trữ trong **R_t**.

- Định dạng và ý nghĩa:

  Định dạng: **lw R_t, Immediate (R_s)**

  Ý nghĩa: **R_t** ← **Memory** [**R_s** + **Immediate**]
  Sao chép từ bộ nhớ tới thanh ghi

Toán tử lw

- Toán tử giả, cho phép địa chỉ của nhãn được lưu trong **R_s** và sau đó toán tử **lw** thực được gọi để tải giá trị.

- Định dạng và ý nghĩa:

Định dạng: **lw R_s, label**

Ý nghĩa: **R_s ← Memory[Label]**

Thực thi: **lui $at 0x00001001**
lw R_s, offset($at)

offset là khoảng giá trị

trong phân đoạn dữ liệu

Toán tử lw

- Ví dụ:

```
.data
a:  .word   0x1234567
    .space 14
b:  .word   0xFEDCBA98

.text
lw $s0,b
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$t7	15	0x00000000
$s0	16	0xfedcba98

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	7: lw $s0,b
	0x00400004	0x8c300014	lw $16,0x00000014($1)	

sw operator

- Định dạng thực duy nhất: một địa chỉ được lưu trữ trong $\mathbf{R_s}$ và một phần bù từ địa chỉ đó được lưu trữ trong giá trị Immediate. Giá trị của $\mathbf{R_t}$ được lưu trong bộ nhớ ở [$\mathbf{R_s}$ + **Immediate**].

- Định dạng và ý nghĩa:

  Định dạng: **sw $\mathbf{R_t}$, Immediate** ($\mathbf{R_s}$)

  Ý nghĩa: **Memory** [$\mathbf{R_s}$ + **Immediate**] ← $\mathbf{R_t}$

  Sao chép từ thanh ghi tới bộ nhớ

sw operator

- Toán tử giả, cho phép nội dung trong thanh ghi Rs được lưu trữ trong địa chỉ của nhãn.
- Định dạng và ý nghĩa:

Định dạng: **sw R_s, label**

Ý nghĩa: **R_s ← Memory[Label]**

Thực thi: **lui $at 0x00001001**
sw R_s, offset($at)

offset là khoảng giá trị
trong phân đoạn dữ liệu

sw operator

- Ví dụ:

```
.data
a:  .word   0x1234567
    .space 14
b:  .word   0xFEDCBA98
```

$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0xfedcba98
$s1	17	0x00000000

```
.text
lw $s0,b
sw $s0,a
```

Address	Value (+0)	Value (+4)
0x10010000	0x01234567	0x00000000
0x10010020	0x00000000	0x00000000

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	7: lw $s0,b
	0x00400004	0x8c300014	lw $16,0x00000014($1)	
	0x00400008	0x3c011001	lui $1,0x00001001	8: sw $s0,a
	0x0040000c	0xac300000	sw $16,0x00000000($1)	

sw operator

- Ví dụ:

```
.data
a: .word   0x1234567
   .space 14
b: .word   0xFEDCBA98

.text
lw $s0,b
sw $s0,a
```

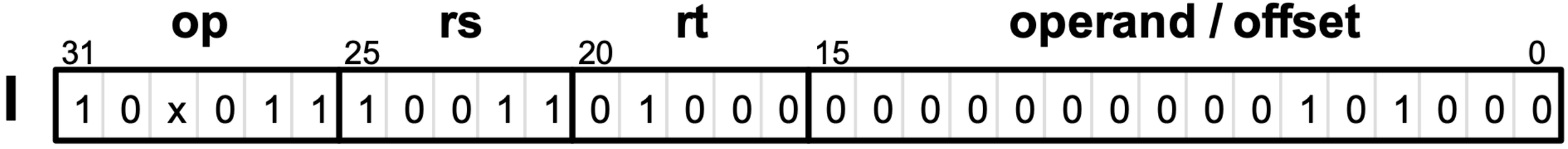
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0xfedcba98
$s1	17	0x00000000

Address	Value (+0)	Value (+4)
0x10010000	0xfedcba98	0x00000000
0x10010020	0x00000000	0x00000000

Text Segment

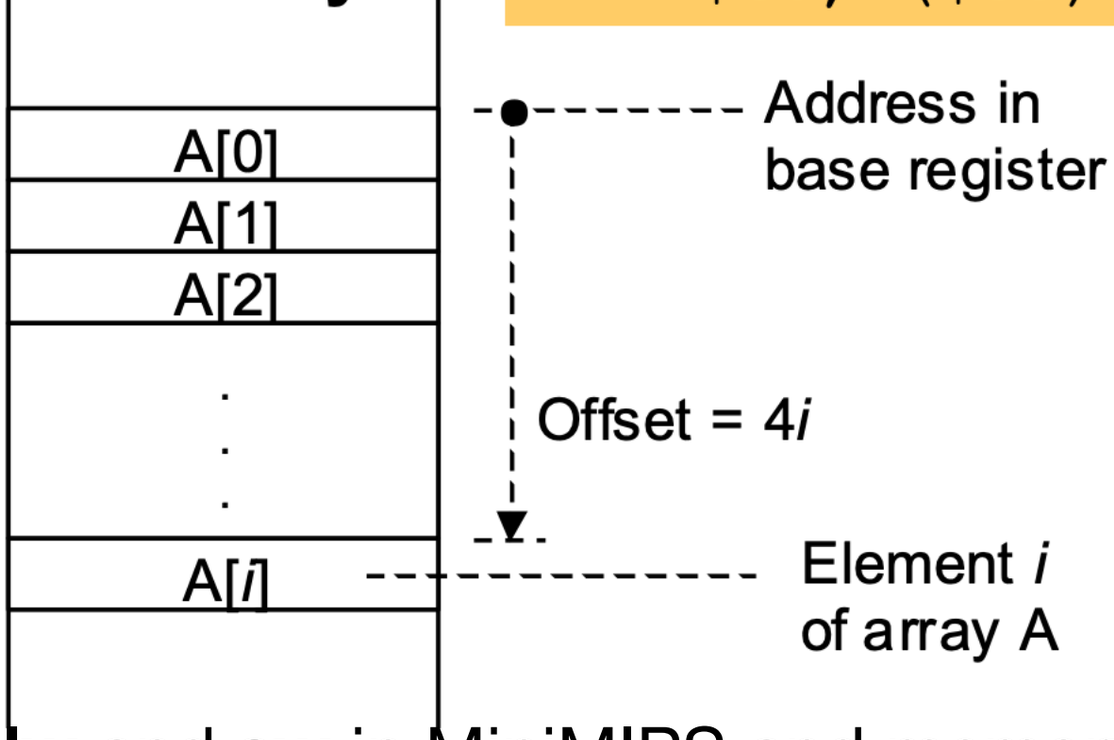
Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	7: lw $s0,b
	0x00400004	0x8c300014	lw $16,0x00000014($1)	
	0x00400008	0x3c011001	lui $1,0x00001001	8: sw $s0,a
	0x0040000c	0xac300000	sw $16,0x00000000($1)	

Lệnh load và store



Memory

```
lw   $t0,40($s3)
lw   $t0,A($s3)
```



Note on base and offset:

The memory address is the sum of (`rs`) and an immediate value. Calling one of these the base and the other the offset is quite arbitrary. It would make perfect sense to interpret the address `A($s3)` as having the base `A` and the offset `($s3)`. However, a 16-bit base confines us to a small portion of memory space.

lw and **sw** in MiniMIPS and memory addressing mechanism allow simple access to the elements of the string over the base address and offset (offset = 4 / i.e., to the i^{th} element (word)).

Mã máy lệnh **load** và **store**

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

```
lw $t0, 32($s3)
```

35	$s3	$t0	32
35	19	8	32
100011	10011	01000	0000 0000 0010 0000

(0x8E680020)

```
sw $s1, 4($t1)
```

43	$t1	$s1	4
43	9	17	4
101011	01001	10001	0000 0000 0000 0100

(0xAD310004)

Các phương pháp truy cập bộ nhớ

Có 4 phương thức truy cập dữ liệu dưới đây:

- Đánh địa chỉ theo nhãn
- Truy cập thanh ghi trực tiếp
- Truy cập thanh ghi gián tiếp
- Truy cập thanh ghi offset

Các phương pháp truy cập bộ nhớ

- Xét đoạn mã giả sau:

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

Mã giả chương trình bậc 2

Đánh địa chỉ theo nhãn

- Một nhãn có thể được định nghĩa cho địa chỉ của một biến.
- Loại dữ liệu này chỉ có thể tồn tại trong đoạn **.data** của chương trình, có nghĩa là dữ liệu này không thể di chuyển hoặc thay đổi kích thước.
- Khi biến được lưu trữ trong phân đoạn **.data** thì có thể được truy cập trực tiếp bằng cách sử dụng nhãn.

Đánh địa chỉ theo nhãn

```
1   #Program 8.1 Quadratic program
2   #Date 2/4/2020
3   #Purpose: Addressing by label
4   .text
5   .globl main
6
7           # Get input value and store it in $s0
8   main:   la $a0, prompt
9           jal PromptInt
10          move $s0, $v0
11
12      # Load constants a, b, and c into registers
13      lw $t5, a
14      lw $t6, b
15      lw $t7, c
16
17      # Calculate the result of y=a*x*x + b*x + c
18      #and store it
19      mul $t0, $s0, $s0
20      mul $t0, $t0, $t5
21      mul $t1, $s0, $t6
22      add $t0, $t0, $t1
23      add $s1, $t0, $t7
```

```
25      # Store the result from $s1 to y
26      sw $s1, y
27
28      # Print output from memory y
29      la $a0, result
30      lw $a1, y
31      jal PrintInt
32      jal PrintNewLine
33      nop
34
35      #Exit program
36      jal Exit
```

a, b và c được tải từ bộ nhớ bằng cách sử dụng toán tử lw với các nhãn.

```
37  .data
38      a: .word 5
39      b: .word 2
40      c: .word 3
41      y: .word 0
42      prompt: .asciiz "Enter a value for x: "
43      result: .asciiz "The result is: "
44
45  .include "utils.asm"
```

Truy cập thanh ghi trực tiếp

Các giá trị được lưu trữ trực tiếp trong các thanh ghi, và do đó bộ nhớ hoàn toàn không được truy cập.

```
1   #Program 8.2 Quadratic program
2   #Date 2/4/2020
3   #Purpose: register direct access
4   .text
5   .globl main
6           # Get input value and store it in $s0
7   main:   la $a0, prompt
8           jal PromptInt
9           move $s0, $v0
10
11          # Load constants a, b, and
12          li $t5, 5
13          li $t6, 2
14          li $t7, 3
15
16          # Calculate the result of
17          # and store it
18          mul $t0, $s0, $s0
19          mul $t0, $t0, $t5
20          mul $t1, $s0, $t6
21          add $t0, $t0, $t1
22          add $s1, $t0, $t7
24          # Print output from memory y
25          la $a0, result
26          move $a1, $s1
27          jal PrintInt
28          jal PrintNewLine
29          nop
30
31          #Exit program
32          jal Exit
33  .data
34      y: .word 0
35      prompt: .asciiz "Enter a value for x: "
36      result: .asciiz "The result is: "
37
38  .include "utils.asm"
```

Truy cập thanh ghi gián tiếp

- Thanh ghi truy cập gián tiếp khác với truy cập trực tiếp ở chỗ thanh ghi không chứa giá trị sử dụng trong phép tính mà chứa địa chỉ bộ nhớ chứa giá trị được sử dụng.

- Ví dụ:

```
.data
    .word 5
    .word 2
    .word 3
y:  .word 0
```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x10010000	0x00000005	0x00000002	0x00000003	0x00000000	0x65746e45

Truy cập thanh ghi gián tiếp

```
1  #Program 8.3 Quadratic program
2  #Date 2/4/2020
3  #Purpose: register indirect access
4  .text
5  .globl main
6
7          # Get input value and store it in $s0
8  main:   la $a0, prompt
9          jal PromptInt
10         move $s0, $v0
11
12         # Load constants a, b, and c into reg
13         lui $t0, 0x1001
14         lw $t5, 0($t0)
15         addi $t0, $t0, 4
16         lw $t6, 0($t0)
17         addi $t0, $t0, 4
18         lw $t7, 0($t0)
19
20         # Calculate the result of y=a*x*x + b
21         # and store it.
22         mul $t0, $s0, $s0
23         mul $t0, $t0, $t5
24         mul $t1, $s0, $t6
25         add $t0, $t0, $t1
26         add $s1, $t0, $t7
```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x10010000	0x00000005	0x00000002	0x00000003	0x00000000	0x65746e45

```
28         # Print output from memory y
29         la $a0, result
30         move $a1, $s1
31         jal PrintInt
32         jal PrintNewLine
33
34         #Exit program
35         jal Exit
36
37 .data
38     .word 5
39     .word 2
40     .word 3
41 y: .word 0
42 prompt: .asciiz "Enter a value for x: "
43 result: .asciiz "The result is: "
44
45 .include "utils.asm"
```

Truy cập thanh ghi qua offset

- Trong lệnh **lw**, giá trị tức thời là khoảng cách từ địa chỉ trong thanh ghi đến giá trị được nạp.

- Trong truy cập thanh ghi gián tiếp, giá trị tức thời này luôn bằng 0 vì thanh ghi chứa địa chỉ thực của giá trị bộ nhớ được nạp vào.

- Trong ví dụ sau, một giá trị offset sẽ được sử dụng để xác định khoảng cách bộ nhớ từ địa chỉ trong thanh ghi đến giá trị được nạp.

Truy cập thanh ghi qua offset

```
1  #Program 8.4 Quadratic program
2  #Date 2/4/2020
3  #Purpose: register offset address
4  .text
5        .globl main
6
7        # Get input value and store
8  main: la $a0, prompt
9        jal PromptInt
10       move $s0, $v0
11
12       # Load constants a, b, and
13       lui $t0, 0x1001
14       lw $t5, 0($t0)
15       lw $t6, 4($t0)
16       lw $t7, 8($t0)
17
18       # Calculate the result of y
19       # and store it
20       mul $t0, $s0, $s0
21       mul $t0, $t0, $t5
22       mul $t1, $s0, $t6
23       add $t0, $t0, $t1
24       add $s1, $t0, $t7
26       # Print output from memory y
27       la $a0, result
28       move $a1, $s1
29       jal PrintInt
30       jal PrintNewLine
31
32       #Exit program
33       jal Exit
34 .data
35    .word 5
36    .word 2
37    .word 3
38 y: .word 0
39 prompt: .asciiz "Enter a value for x: "
40 result: .asciiz "The result is: "
41
42 .include "utils.asm"
```

Con trỏ

- Nếu một thanh ghi có thể chứa địa chỉ của một biến trong bộ nhớ, thì một giá trị bộ nhớ có thể chứa một tham chiếu đến một biến khác tại một vị trí khác trong bộ nhớ.

- Các biến này được gọi là biến con trỏ.

- Chương trình sau đây mô tả việc sử dụng các biến bộ nhớ gián tiếp (con trỏ).

Con trỏ

```
1   #Program 8.5 Quadratic program
2   #Date 2/4/2020
3   #Purpose: memory indirect (pointer) variables
4   .text
5         .globl main
6
7           # Get input value and store it in $s0
8   main:   la $a0, prompt
9           jal PromptInt
10          nop
11          move $s0, $v0
12
13          # Load constants a, b, and c
14          lui $t0, 0x1001
15          lw $t0, 0($t0)
16          lw $t5, 0($t0)
17          lw $t6, 4($t0)
18          lw $t7, 8($t0)
19
20          # Calculate the result of
21          #y=a*x*x + b * x + c and stor
22          mul $t0, $s0, $s0
23          mul $t0, $t0, $t5
24          mul $t1, $s0, $t6
25          add $t0, $t0, $t1
26          add $s1, $t0, $t7
```

```
28
29
30
31
32          jal PrintNewLine
33          nop
34
35          #Exit program
36          jal Exit
37  .data
38      .word constants
39  y: .word 0
40  prompt: .asciiz "Enter a value for x: "
41  result: .asciiz "The result is: "
42  constants:
43          .word 5
44          .word 2
45          .word 3
46  .include "utils.asm"
```

Bộ nhớ ở đầu đoạn **.data** chứa một địa chỉ trỏ đến vị trí lưu các hằng số a, b và c.

```
# Print output from memory y
   la $a0, result
   move $a1, $s1
   jal PrintInt
```

Con trỏ

- Bộ nhớ ở đầu đoạn **.data** chứa một địa chỉ trỏ đến vị trí lưu trữ thực tế của các hằng số a, b và c.
- Các biến này sau đó được truy cập bằng cách tải địa chỉ trong bộ nhớ vào một thanh ghi và sử dụng địa chỉ đó để định vị các hằng số.

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)
0x10010000	0x10010030	0x00000000	0x65746e45	0x20612072	0x756c6176	0x6f662065	0x3a782072
0x10010020	0x65722065	0x746c7573	0x3a736920	0x00000020	0x00000005	0x00000002	0x00000003

Bài tập 8.1

- Bảng sau đây có địa chỉ bộ nhớ ở mỗi hàng và các cột đại diện cho từng loại ranh giới MIPS bao gồm: byte, nửa từ, từ và từ kép.
- Đánh dấu check vào cột nếu địa chỉ của hàng đó thuộc loại ranh giới của cột.

Bài tập 8.1

Address	Boundary Type			
	Byte	Half	Word	Double
0x10010011				
0x10010100				
0x10050108				
0x1005010c				
0x1005010d				
0x1005010e				
0x1005010f				
0x10070104				

Bài tập 8.2

- Tại sao các lệnh "**la label**" luôn cần được dịch thành 2 dòng mã giả?
- Lệnh "**lw label**" là như thế nào?
- Giải thích những điểm giống và khác nhau trong cách chúng được thực hiện trong MARS.

Bài tập 8.3

- Chương trình sau không thể nạp giá trị 8 vào $t0. Trong thực tế, nó tạo ra một ngoại lệ.

- Tại sao?

```
.text
        lui $t0, 1001
        lw $a0, 0($t0)
        li $v0, 1
        syscall

        li $v0, 10
        syscall

.data
        .word 8
```

Bài tập 8.4

- Chuyển đoạn mã giả sau sang hợp ngữ MIPS theo các chế độ địa chỉ và truy cập con trỏ ở trên (5 phương pháp).

- Lưu ý rằng các biến x và y là biến tĩnh và dễ thay đổi, do đó nên được lưu trữ trong bộ nhớ dữ liệu. Khi sử dụng truy cập trực tiếp thanh ghi thì không cần phải lưu trữ các biến trong bộ nhớ.

```
main() {
    static volatile int miles =
        prompt("Enter the number of miles driven: ");
    static volatile int gallons =
        prompt("Enter the number of gallons used: ");
    static volatile int mpg = miles / gallons;
    output("Your mpg = " + mpg);
}
```

Kết thúc tuần 8