Thực hành
Kiến trúc máy tính

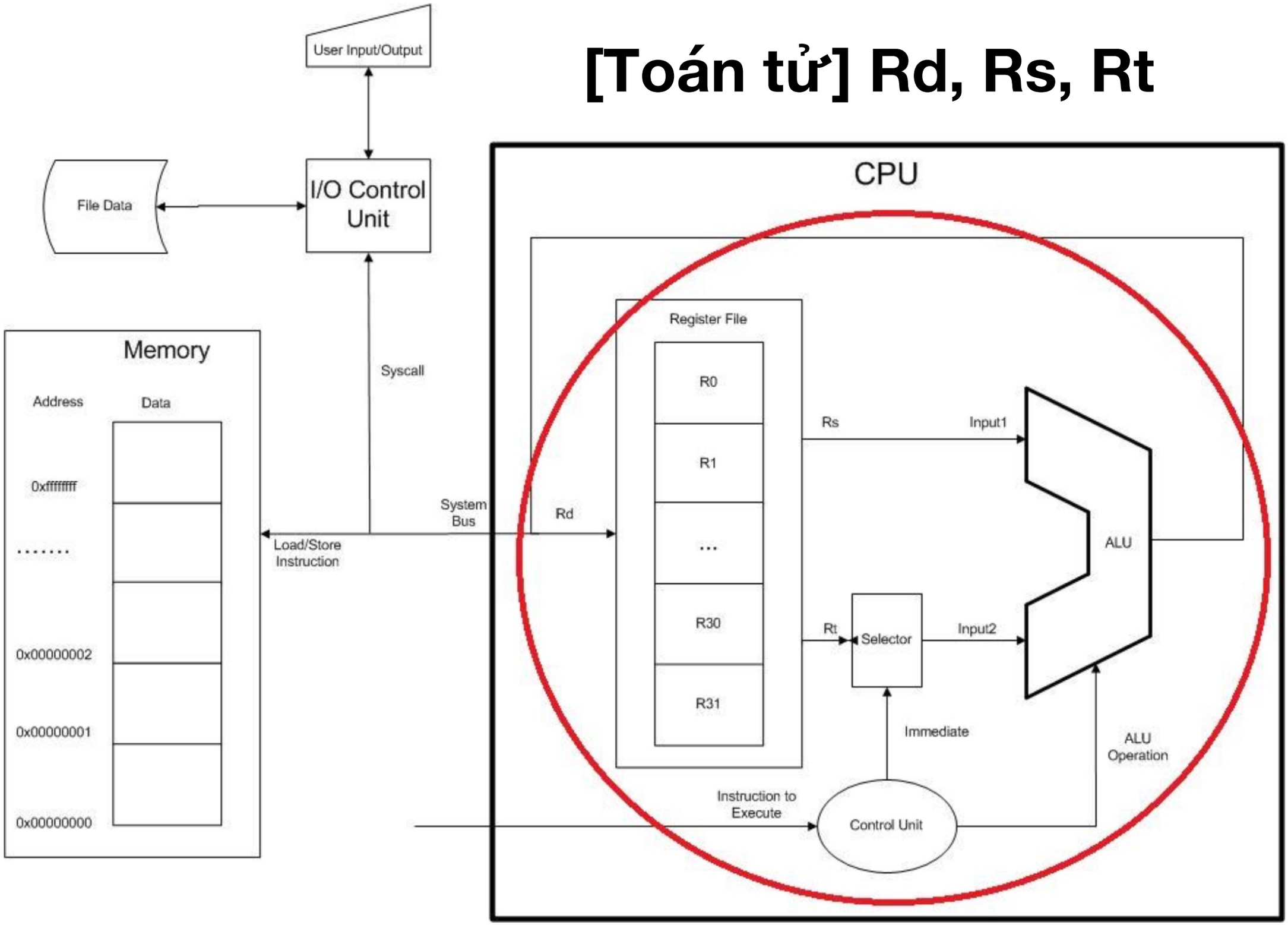
Giảng viên: Nguyễn Thị Thanh Nga
Khoa Kỹ thuật máy tính
Trường CNTT&TT

Tuần 3

- Giới thiệu máy 3-Địa chỉ.
- Sự khác nhau giữa lệnh thực MIPS và giả lệnh.
- Giới thiệu về giả lệnh.
- Toán tử số học MIPS và cách sử dụng
  - Cộng
  - Trừ
  - Nhân
  - Chia

Máy 3-Địa chỉ

[Toán tử] Rd, Rs, Rt



Các 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: **$R_d \leftarrow R_s$ [Toán tử] R_t**

  - 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: **$R_t \leftarrow R_s$ [Toán tử] Giá trị tức thì i**

  - 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 trong hợp ngữ MIPS

- Có 4 toán tử cộng thực trong hợp ngữ MIPS:
  - 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:

  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:

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

  Ý nghĩa: **$R_d \leftarrow R_s$ + 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 $0 \ldots 2^{32}-1$.

- Định dạng và ý nghĩa tương tự như toán tử add:

  Định dạng: **addu R_d, R_s, R_t**

  Ý nghĩa: **Rd ← Rs + 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:

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

Ý nghĩa: **$R_d \leftarrow R_s$ + 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, ý nghĩa và lệnh thực:

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

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

Program 3.1 Ví dụ phép cộng

```
1  # File: Program 3-1.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
```

Program 3.1 Ví dụ phép cộng

Text Segment

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

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

Program 3.1 Ví dụ phép cộng

Text Segment

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

- Cột thứ hai là Basic
- Chuyển tên thanh ghi từ tên gợi nhớ thành số thanh ghi
- Các lệnh giả ngữ được chuyển thành 1 hay nhiều lệnh thực

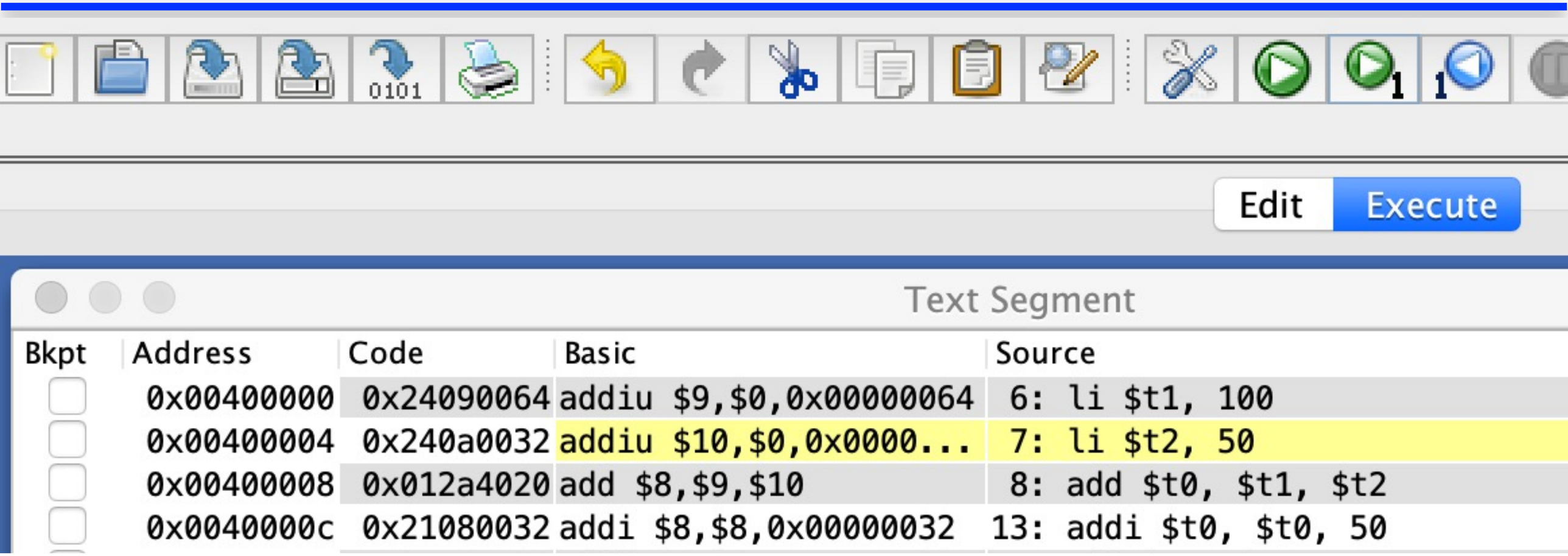
Program 3.1 Ví dụ phép cộng

Text Segment

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

- Dòng đầu tiên của chương trình được bôi màu vàng nghĩa là chương trình đã sẵn sàng được thực thi từ dòng đầu tiên.

Program 3.1 Ví dụ 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)**.

$t1	9	0x00000064

Program 3.1 Ví dụ phép cộng

Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x00400000	0x24090064	addiu $9,$0,0x00000064	6: li $t1, 100
☐	0x00400004	0x240a0032	addiu $10,$0,0x0000...	7: li $t2, 50
☐	0x00400008	0x012a4020	add $8,$9,$10	8: add $t0, $t1, $t2
☐	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

Program 3.1 Ví dụ phép cộng

Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x00400000	0x24090064	addiu $9,$0,0x00000064	6: li $t1, 100
☐	0x00400004	0x240a0032	addiu $10,$0,0x0000...	7: li $t2, 50
☐	0x00400008	0x012a4020	add $8,$9,$10	8: add $t0, $t1, $t2
☐	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 (50_{10})** 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

Program 3.1 Ví dụ phép cộng

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24090064	addiu $9,$0,0x00000064	6: li $t1, 100
	0x00400004	0x240a0032	addiu $10,$0,0x0000...	7: li $t2, 50
	0x00400008	0x012a4020	add $8,$9,$10	8: add $t0, $t1, $t2
	0x0040000c	0x21080032	addi $8,$8,0x00000032	13: addi $t0, $t0, 50
	0x00400010	0x21080032	addi $8,$8,0x00000032	14: add $t0, $t0, 50
	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 $\mathbf{150_{10}}$).

Program 3.1 Ví dụ phép cộng

Text Segment

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

- Tiếp tục cho đến khi dòng 19 được bôi vàng và sẵn sàng để chạy.

- Tại đây, thanh ghi **$t0** có giá trị **0xfa ($250_{10}$)**.

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

Program 3.1 Ví dụ phép cộng

Text Segment

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

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	0xfffffffce
$t1	9	0x00000064

- Thực thi dòng 19

- Giá trị trong thanh ghi **$t0** thay đổi từ **0xfa** thành **0xffffffce** **(-50_{10})**, chứ không phải là **0x96** **(150_{10})** như mong đợi

Các phép toán số học

- Cộng
- Trừ
- Nhân
- Chia

Phép trừ trong hợp ngữ MIPS

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

Phép trừ trong hợp ngữ MIPS

- Phép trừ trong hợp ngữ MIPS tương tự như phép cộng với 1 ngoại lệ.

- Toán tử **sub** và **subu** tương tự như toán tử **add** và **addu**.

- Toán tử **subi** và **subui** không phải là các lệnh thực. Chúng hoạt động như là các lệnh giả.

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:

  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:

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

Ý nghĩa: **$R_d \leftarrow R_s$ - 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…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: **Rd ← Rs - R_t**

Toán tử subiu

- Tương tự như toán tử addi nhưng giá trị tức thời là số ngyên không dấu.

- Định dạng và ý nghĩa:

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

Ý nghĩa: **$R_d \leftarrow R_s$ - 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 trong hợp ngữ MIPS

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

Phép nhân trong hợp ngữ MIPS

- Phép nhân và chia trong hợp ngữ MIPS phức tạp hơn phép cộng và trừ.
- Cần sử dụng hai thanh ghi mới là thanh ghi **hi** và **lo.**
- Kết quả của phép nhân có thể cần gấp đôi số bit đầu vào.
- Thanh ghi **hi** được sử dụng để lưu 32 bit cao của phép nhân.
- Thanh ghi **lo** được sử dụng để lưu 32 bit thấp của phép nhân.

Phép nhân trong hợp ngữ MIPS

- Toán tử **mult**:

mult $t1, $t2 #t1*t2

mflo $t0 #lưu kết quả trong thanh ghi $t0

Giá trị trong thanh ghi **$t2** nhân với giá trị trong thanh ghi **$t1**, và lưu lại kết quả trong thanh ghi **$t0**.

- 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

```
    0011                0011
*   0010            *   0110
---------           ---------
0000 0110           0001 0010
```

Các bit cao = 0

→ Không tràn

Các bit cao
khác 0

→ Tràn

Nhận xét

- Ví dụ: 2*(-3)=-6 và 2*(-8)=-16

```
      0010                    0010
   *  1101                 *  1010
   ------                  ------
   1111  1010              1110 1110
```

4 bit cao là 1111, mở rộng bit dấu của -6

→ Không tràn

4 bit cao là 1110, vì 4 bit cao không phải là 1 nên không phải bit dấu

→ Tràn

Nhận xét

- Ví dụ: 2*(-6)

```
      0010
*     1010
-------------
1111  0010
```

- 4 bit cao là 1111, có vẻ không tràn số?
- Tuy nhiên, 4 bit thấp chỉ ra số dương, vì thế 1111 chỉ ra rang, bit thấp nhất (màu đỏ) là một phần trong kết quả phép nhân và không phải bit dấu. Kết quả này không chỉ ra kết quả tràn số.

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] ← 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:

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:

  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 và ý nghĩa:

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

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

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

mult R_d, 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 và ý nghĩa:

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

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

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

mult R_d, R_s, R_t

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 chia

- Toán tử **div**: 3 định dạng
- Toán tử **rem** (remainder): 2 định dạng

Toán tử **div** định dạng thứ 1

- 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 phần thương lưu trong **lo** và phần dư lưu trong **hi**.

- Định dạng và ý nghĩa:

  Định dạng: **div R_s, R_t**

  Ý nghĩa: **[hi, lo] ← R_s/R_t**

Toán tử **div** định dạng thứ 2

- Định dạng thứ 2 của toán tử div là một giả lệnh.

- Để thực hiện phép chia, lệnh giả này cũng kiểm tra phép chia 0. Phép kiểm tra cụ thể không nằm trong phạm vi bài này vì liên quan đến lệnh **bne** và **break**

- Định dạng và ý nghĩa:

  Định dạng **div R_d , R_s, R_t**

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

  else break

Toán tử **div** định dạng thứ 2

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

break

div R_s, R_t

mflo R_d

Toán tử **div** định dạng thứ 3

- Định dạng thứ 3 là một lệnh giả.
- Định dạng và ý nghĩa:

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

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

addi $\$R_t$, \$zero, Giá trị tức thì

div R_s, R_t

mflo R_d

Toán tử **rem** định dạng thứ 1

- Chỉ có các định dạng mã giả cho lệnh này. Định dạng đầu tiên của 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!=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 2

- 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: **$R_d \leftarrow R_s$ / Giá trị tức thì**

addi $\$R_t$, $zero, Giá trị tức thì

div R_s, R_t

mfhi R_d

Program 3.2 Toán tử chia lấy phần dư, kiểm tra số chẵn lẻ

- Yêu cầu người dùng nhập vào một số nguyên
- In ra kết quả số nguyên đó là chẵn hay lẻ

main

```
{
int x = prompt(“Enter your number: ”);
int y =%2;
printf(“A result of 0 is even, 1 is odd: result=”+y );
}
```

Program 3.2 Toán tử chia lấy phần dư, kiểm tra số chẵn lẻ

```
1  # File: Program 3-2.asm
2  # Author: Charles W. Kann
3  # Purpose: To have a user enter a number,and print 0 if
4  # the number is even, 1 if the number is odd
5
6  .data
7  prompt: .asciiz "Enter your number: "
8  result: .asciiz "A result of 0 is even, 1 is odd: result = "
9
10 .text
11 .globl main      #Declare lable as global to
12                  #enable referencing from other file
13 main:
14    # Get input value
15    addi $v0,$zero,4      # Write Prompt
16    la $a0,prompt
17    syscall
18
19    addi $v0, $zero, 5    # Retrieve input
20    syscall
21    move $s0, $v0
22
23    # Check if odd or even
24    addi $t0,$zero,2      # Store 2 in $t0
25    div $t0,$s0,$t0       # Divide input by 2
26    mfhi $s1              # Save remainder in $s1
27
28    # Print output
29    addi $v0,$zero,4      # Print result string
30    la $a0,result
31    syscall
32
33    addi $v0,$zero,1      # Print result
34    move $a0,$s1
35    syscall
36
37    #Exit program
38    addi $v0,$zero,10
39    syscall
```

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);
}
```

Program 3.3 Giải phương trình

```
1  # File: Program 3-3.asm
2  # Author: Charles Kann
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
```

Program 3.4 Phép chia và độ chính xác của phép chia

- Một điều luôn cần phải nhớ khi thực hiện phép chia số nguyên trong bất kỳ ngôn ngữ nào là kết quả được làm tròn.

- Việc làm tròn kết quả có thể dẫn đến những câu trả lời sai lệch phụ thuộc vào mức độ làm tròn số.

- Ví dụ, đối với phần lớn học sinh lớp 5 phép tính "(10/3) *3 = 10", vì triệt tiêu 3 trên tử và mẫu. Tuy nhiên, trong phép toán số học thì kết quả của phép tính "10/3 = 3", và do đó "(10/3) *3 = 9" (không phải 10).

- Tuy nhiên, nếu đảo thứ tự của phép tính thì có thể thấy "(10*3) / 3 = 10".

Program 3.4 Phép chia và độ chính xác của phép chia

```
1  # File:      Program3-4.asm
2  # Author:    Charles Kann
3  # Purpose:   To show the difference in result if
4  #            ordering of multiplication and division
5  #            are reversed.
6  .data
7        result1: .asciiz "\n(10/3)*3 = "
8        result2: .asciiz "\n(10*3)/3 = "
9
10 .text
11 .globl main
12 main:
13       addi $s0,$zero,10    #Store 10 and 3 in registers $s0 and $s1
14       addi $s1,$zero,3
15
16       div $s2,$s0,$s1      #Write out (10/3) * 3
17       mul $s2,$s2,$s1
18
19       addi $v0,$zero,4     #Print the result 1
20       la $a0,result1
21       syscall
22
23       addi $v0,$zero,1     #Print the result
24       move $a0,$s2
25       syscall
26
27       mul $s2,$s0,$s1      #Write out (10*3)/3
28       div $s2,$s2,$s1
29
30       addi $v0,$zero,4     #Print the result 2
31       la $a0,result2
32       syscall
33
34       addi $v0,$zero,1     #Print the result
35       move $a0,$s2
36       syscall
37
38       addi $v0,$zero,10    #Exit program
39       syscall
```

Bài tập 3.1

Gõ chương trình sau vào công cụ MARS.

```
#Laboratory Exercise 2, Assignment 1
.text
      addi    $s0, $zero, 0x3007 # $s0 = 0 + 0x3007 = 0x3007 ;I-type
      add     $s0, $zero, $0     # $s0 = 0 + 0 = 0             ;R-type
```

Sau đó:

- Sử dụng công cụ gỡ rối, Debug, chạy từng lệnh và dừng lại,
- Ở mỗi lệnh, quan sát cửa sổ Register và chú ý
    - Sự thay đổi giá trị của thanh ghi $s0
    - Sự thay đổi giá trị của thanh ghi $pc
- Ở cửa số Text Segment, hãy so sánh mã máy của các lệnh trên với khuôn dạng lệnh để chứng tỏ các lệnh đó đúng như tập lệnh đã qui định
- Sửa lại lệnh lui như bên dưới. Chuyện gì xảy ra sau đó. Hãy giải thích

```
addi  $s0, $zero, 0x2110003d
```

Bài tập 3.2

Gõ chương trình sau vào công cụ MARS.

```
#Laboratory Exercise 2, Assignment 4
.text
     #  Assign X, Y
      addi  $t1, $zero, 5     # X = $t1 = ?
      addi  $t2, $zero, -1    # Y = $t2 = ?
     #  Expression Z = 2X + Y
     add   $s0, $t1, $t1     # $s0 = $t1 + $t1 = X + X = 2X
     add   $s0, $s0, $t2     # $s0 = $s0 + $t2 = 2X + Y
```

Sau đó:

- Sử dụng công cụ gỡ rối, Debug, chạy từng lệnh và dừng lại,
- Ở mỗi lệnh, quan sát cửa sổ Register và chú ý
    - Sự thay đổi giá trị của các thanh ghi
    - Sau khi kết thúc chương trình, xem kết quả có đúng không?
- Ở cửa số Text Segment, xem các lệnh **addi** và cho biết điểm tương đồng với hợp ngữ và mã máy. Từ đó kiểm nghiệm với khuôn mẫu của kiểu lệnh I
- Ở cửa số Text Segment, chuyển mã máy của lệnh **add** sang hệ 2. Từ đó kiểm nghiệm với khuôn mẫu của kiểu lệnh R.

Bài tập 3.3

Gõ chương trình sau vào công cụ MARS.

```
#Laboratory Exercise 2, Assignment 5
.text
    #  Assign X, Y
    addi  $t1, $zero, 4     # X = $t1 = ?
    addi  $t2, $zero, 5     # Y = $t2 = ?
    #  Expression Z = 3*XY
    mul   $s0, $t1, $t2     # HI-LO = $t1 * $t2 = X * Y  ; $s0 = LO
    mul   $s0, $s0, 3       # $s0 = $s0 * 3 = 3 * X * Y
    #  Z' = Z
    mflo  $s1
```

Sau đó:

- Biên dịch và quan sát các lệnh mã máy trong cửa sổ Text Segment. Giải thích điều bất thường?

- Sử dụng công cụ gỡ rối, Debug, chạy từng lệnh và dừng lại,
- Ở mỗi lệnh, quan sát cửa sổ Register và chú ý
    - Sự thay đổi giá trị của các thanh ghi, đặc biệt là Hi, Lo
    - Sau khi kết thúc chương trình, xem kết quả có đúng không?

Bài tập 3.4

- Sửa lỗi chương trình sau:

Program 1

```
.text
main:
    li $v0, 4
    la $a0, result1
    syscall
    li $v0, 1
    li $a0, 4
    syscall

    li $v0, 4
    la $a0, result2
    syscall
    li $v0, 1
    li $a0, 8
    syscall

    addi $v0, $zero, 10 #Exit program
    syscall
.data
result1: .ascii "\nfirst value = "
result2: .ascii "\nsecond value = "
```

Assigment 3.6

- Sửa lỗi chương trình sau:

Program 2

```
.text
main:
    li $v0, 4
    la $a0, result1
    syscall
    li $v0, 4
    li $a0, 4
    syscall

    li $v0, 4
    la $a0, result2
    syscall
    li $v0, 1
    li $a0, 8
    syscall

    addi $v0, $zero, 10 #Exit program
    syscall
.data
result1: .asciiz "\nfirst value = "
result2: .asciiz "\nsecond value = "
```

Assigment 3.7

- Sửa lỗi chương trình sau:

Program 3

```
.text
main:
    li $v0, 4
    la $a0, result1
    syscall
    li $v0, 4
    li $a0, 4
    syscall

    li $v0, 4
    la $a0, result2
    syscall
    li $v0, 1
    li $a0, 8
    syscall

    addi $v0, $zero, 10 #Exit program
    syscall
result1: .ascii "\nfirst value = "
result2: .ascii "\nsecond value = "
```

Bài tập 3.8

- Viết chương trình cho các biểu thức sau. Người dùng nhập vào các giá trị, chương trình in ra kết quả.

- Chương trình có lời nhắc để người dùng nhập vào các giá trị và sẽ in ra kết quả thích hợp. Kết quả chính xác nhất có thể.

a) $5x + 3y + z$

b) $((5x + 3y + z) / 2) * 3$

c) $x^3+2x^2 +3x+4$

d) $(4x / 3) * y$

Bài tập 3.9

- Định luật khí lý tưởng cho phép tính thể tích của một chất khí với áp suất (P), lượng khí (n) và nhiệt độ (T). Phương trình là:

$$V = nRT / P$$

- Bởi vì chúng ta chỉ có các phép toán số học số nguyên, tất cả các giá trị phải ở dạng số nguyên. Hằng số R là 8.314 và sẽ được viết dưới dạng (8314/1000).

- Viết chương trình tính thể tích khí lý tưởng, yêu cầu người dùng nhập vào các giá trị n, T, và P. Thể tích V sau đó được tính và in kết quả ra màn hình.

- Cần chú ý đến độ chính xác của phép toán.

Kết thúc tuần 3