

# Thực hành Kiến trúc máy tính

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

# Tuần 4

---

- Toán tử logic (Phép toán thao tác bit) trong MIPS
- Cách sử dụng các toán tử logic
- Dịch dữ liệu trong thanh ghi và các phép dịch bit trong MIPS.
- Dịch từ hợp ngữ sang mã máy

# Các toán tử logic

- Các toán tử logic

Input		Output				
A	B	AND	OR	NAND	NOR	XOR
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

- Phép toán thao tác bit (Bitwise operators)  
(MIPS hỗ trợ các phép toán thao tác bit,  
được gọi là toán tử logic): AND/OR/NOT/XOR

# Toán tử and định dạng thứ 1

- Là lệnh thực duy nhất
- Thực hiện phép toán thao tác bit **AND** cho  **$R_s$**  và  **$R_t$** , lưu kết quả vào thanh ghi  **$R_d$**
- Định dạng và ý nghĩa:

Định dạng: **and  $R_d$ ,  $R_s$ ,  $R_t$**

Ý nghĩa:  **$R_d \leftarrow R_s \text{ AND } R_t$**

# Toán tử and định dạng thứ 2

- Lệnh giả, thực hiện phép toán thao tác bit **AND** cho  $R_s$  và một giá trị tức thời, lưu vào thanh ghi  $R_t$ , là cách viết tắt cho toán tử **andi**
- Định dạng và ý nghĩa:

Định dạng: **and  $R_t$ ,  $R_s$ , Giá trị tức thời**

Ý nghĩa:  **$R_t \leftarrow R_s \text{ AND Giá trị tức thời}$**

Thực hiện: **andi  $R_t$ ,  $R_s$ , Giá trị tức thời**

# Toán tử and định dạng thứ 3

- Lệnh giả, thực hiện phép toán thao tác bit **AND** cho  **$R_s$**  và một giá trị tức thời, lưu vào thanh ghi  **$R_s$**
- Định dạng và ý nghĩa:

Định dạng: **and  $R_s$ , Giá trị tức thời**

Ý nghĩa:  **$R_s \leftarrow R_s \text{ AND Giá trị tức thời}$**

Thực hiện: **andi  $R_s$ ,  $R_s$ , Giá trị tức thời**

# Toán tử andi định dạng thứ 1

- Lệnh thực, thực hiện phép toán thao tác bit **AND** cho  $R_s$  và một giá trị tức thời, lưu vào thanh ghi  $R_t$
- Định dạng và ý nghĩa:

Định dạng: **andi  $R_t$ ,  $R_s$ , Giá trị tức thời**

Ý nghĩa:  **$R_t \leftarrow R_s \text{ AND Giá trị tức thời}$**

# Toán tử andi định dạng thứ 2

- Cách ngắn gọn với 1 thanh ghi áp dụng cho andi
- Định dạng và ý nghĩa:

Định dạng: **andi  $R_s$ , Giá trị tức thời**

Ý nghĩa:  **$R_s \leftarrow R_s \text{ AND Giá trị tức thời}$**

Thực hiện: **andi  $R_s$ ,  $R_s$ , Giá trị tức thời**



# Toán tử or định dạng thứ 1

- Lệnh thực, thực hiện phép toán thao tác bit **OR** cho  **$R_s$**  và  **$R_t$** , lưu vào thanh ghi  **$R_d$**
- Định dạng và ý nghĩa:

Định dạng: **or  $R_d, R_s, R_t$**

Ý nghĩa:  **$R_d \leftarrow R_s \text{ OR } R_t$**

# Toán tử or định dạng thứ 2

- Lệnh giả, thực hiện phép toán thao tác bit **OR** cho  **$R_s$**  và một giá trị tức thời, lưu vào thanh ghi  **$R_t$** , là cách viết tắt cho toán tử **ori**
- Định dạng và ý nghĩa:

Định dạng: **or  $R_t$ ,  $R_s$ , Giá trị tức thời**

Ý nghĩa:  **$R_t \leftarrow R_s \text{ OR Giá trị tức thời}$**

Thực hiện: **ori  $R_t$ ,  $R_s$ , Giá trị tức thời**

# Toán tử or định dạng thứ 3

- Lệnh giả, thực hiện phép toán thao tác bit **OR** cho  **$R_s$**  và một giá trị tức thời, lưu vào thanh ghi  **$R_s$**
- Định dạng và ý nghĩa:

Định dạng: **or  $R_s$ , Giá trị tức thời**

Ý nghĩa:  **$R_s \leftarrow R_s \text{ OR Giá trị tức thời}$**

Thực hiện: **ori  $R_s$ ,  $R_s$ , Giá trị tức thời**

# Toán tử ori

- Lệnh thực, thực hiện phép toán thao tác bit **OR** cho  **$R_s$**  và một giá trị tức thời, lưu vào thanh ghi  **$R_t$**
- Định dạng và ý nghĩa:

Định dạng:            **ori  $R_t$ ,  $R_s$ , Giá trị tức thời**

Ý nghĩa:             **$R_t \leftarrow R_s \text{ OR Giá trị tức thời}$**

# Toán tử **ori**

- Có thể sử dụng toán tử **ori** với dạng ngắn gọn (1 thanh ghi).
- Định dạng và ý nghĩa:

Định dạng:            **ori  $R_s$ , Giá trị tức thời**

Ý nghĩa:             **$R_s \leftarrow R_s \text{ OR Giá trị tức thời}$**

Thực hiện:            **ori  $R_s$ ,  $R_s$ , Giá trị tức thời**

# Toán tử xor định dạng thứ 1

- Lệnh thực, thực hiện phép toán thao tác bit **XOR** cho **R<sub>s</sub>** và **R<sub>t</sub>**, lưu vào thanh ghi **R<sub>d</sub>**
- Định dạng và ý nghĩa:

Định dạng:        **xor R<sub>d</sub>, R<sub>s</sub>, R<sub>t</sub>**

Ý nghĩa:        **R<sub>d</sub> ← R<sub>s</sub> XOR R<sub>t</sub>**

# Toán tử xor định dạng thứ 2

- Lệnh giả, thực hiện phép toán thao tác bit **XOR** cho  **$R_s$**  và một giá trị tức thời, lưu vào thanh ghi  **$R_t$** , là cách viết tắt cho toán tử **xori**
- Định dạng và ý nghĩa:

Định dạng: **xor  $R_t$ ,  $R_s$ , Giá trị tức thời**

Ý nghĩa:  **$R_t \leftarrow R_s \text{ XOR Giá trị tức thời}$**

Thực hiện: **xori  $R_t$ ,  $R_s$ , Giá trị tức thời**

# Toán tử xor định dạng thứ 3

- Lệnh giả, thực hiện phép toán thao tác bit **XOR** cho  **$R_s$**  và một giá trị tức thời, lưu vào thanh ghi  **$R_s$**
- Định dạng. và ý nghĩa:

Định dạng:            **xor  $R_s$ , Giá trị tức thời**

Ý nghĩa:             **$R_s \leftarrow R_s \text{ XOR Giá trị tức thời}$**

Thực hiện:            **xori  $R_s$ ,  $R_s$ , Giá trị tức thời**



# Toán tử xori

- Lệnh thực, thực hiện phép toán thao tác bit **XOR** cho  **$R_s$**  và một giá trị tức thời, lưu vào thanh ghi  **$R_t$**
- Định dạng và ý nghĩa như sau:

Định dạng:      **xori  $R_t$ ,  $R_s$ , Giá trị tức thời**

Ý nghĩa:       **$R_t \leftarrow R_s \text{ XOR Giá trị tức thời}$**

# Toán tử xori

- Có thể sử dụng toán tử **xori** với dạng ngắn gọn (1 thanh ghi).
- Định dạng và ý nghĩa:

Định dạng:        **xori  $R_s$ , Giá trị tức thời**

Ý nghĩa:         **$R_s \leftarrow R_s \text{ XOR Giá trị tức thời}$**

Thực hiện:       **xori  $R_s$ ,  $R_s$ , Giá trị tức thời**

# Toán tử not

- Lệnh giả, thực hiện phép toán thao tác bit **NOT** (nghịch đảo từng bit) của thanh ghi **R<sub>t</sub>**, lưu vào thanh ghi **R<sub>s</sub>**
- Định dạng và ý nghĩa:

Định dạng:        **not R<sub>s</sub>, R<sub>t</sub>**

Ý nghĩa:        **R<sub>s</sub> ← NOT(R<sub>t</sub>)**

Thực hiện:        **nor R<sub>s</sub>, R<sub>t</sub>, \$zero**

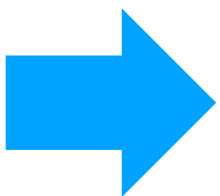
# Sử dụng toán tử logic

---

- Lưu các giá trị tức thời trong thanh ghi
- Chuyển một ký tự từ chữ hoa sang chữ thường
- Toán tử đảo với XOR
- Toán tử dịch bit

# Lưu giá trị tức thời trong thanh ghi

- Lệnh **li  $R_d$ , Giá trị tức thời** được dịch thành  
**addui  $R_d$ , \$zero, Giá trị tức thời**



Phép cộng cần một khoảng thời gian tương đối dài cho thời gian trễ để lưu bit nhớ

- Có thể sử dụng lệnh sau để thay thế:  
**ori  $R_d$ , \$zero, Giá trị tức thời**

# Chuyển một ký tự từ chữ hoa sang chữ thường

---

- Trong bảng mã ASCII, sự khác nhau giữa ký tự hoa và ký tự thường là bao nhiêu?
- Để chuyển từ ký tự hoa sang ký tự thường cần làm như thế nào?
- Nếu ký tự đã là ký tự thường, điều gì sẽ xảy ra?

# Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	\$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

# Program 4.1 Chuyển một ký tự từ chữ hoa sang chữ thường

```
7  .data
8  output1: .asciiz "\nPlease input a character: "
9  output2: .asciiz "\nUppercase to lowercase: result when anding with 0x20: "
10 output3: .asciiz "\nUppercase to lowercase: result when oring with 0x20: "
11
12 .text
13 .globl main
14 main:
15     ori $v0,$zero,4
16     la $a0,output1
17     syscall
18
19     ori $v0,$zero,12
20     syscall
21     move $s0,$v0
22
23     ori $v0,$zero,4
24     la $a0,output2
25     syscall
26
27     or $t0,$s0,$zero
28     addi $a0,$t0,0x20
29     ori $v0,$zero,11
30     syscall
31
32     ori $v0,$zero,4
33     la $a0,output3
34     syscall
35
36     ori $a0,$s0,0x20
37     ori $v0,$zero,11
38     syscall
39
40     ori $v0,$zero,10
41     syscall
```

- Nhập chương trình sau
- Biên dịch và chạy chương trình
- Giải thích kết quả
- Thêm chú thích để giải thích mục đích cho từng khối lệnh



# Toán tử đảo với XOR

- Thao tác bit XOR có thể được sử dụng để đảo một giá trị và khôi phục về giá trị ban đầu.

<b>A</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>
<b>B</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>
<b>A XOR B</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>
<b>(A XOR B) XOR B</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>

# Program 4.2

## Toán tử đảo với XOR

```

6  .data
7  output1: .asciiz "\nPlease input a value: "
8  output2: .asciiz "\nInput value: "
9  output3: .asciiz "\nResult after first XOR: "
10 output4: .asciiz "\nResult after the second XOR: "
11
12 .text
13 .globl main
14 main:
15     ori $v0,$zero,4
16     la $a0,output1
17     syscall
18
19     ori $v0,$zero,5
20     syscall
21     move $s0,$v0
22
23     ori $v0,$zero,4
24     la $a0,output2
25     syscall
26
27     ori $v0,$zero,34
28     move $a0,$s0
29     syscall
30
31     ori $v0,$zero,4
32     la $a0,output3
33     syscall
34
35     xori $s0,$s0,0xffffffff
36     move $a0,$s0
37
38     ori $v0,$zero,34
39     syscall
40
41     ori $v0,$zero,4
42     la $a0,output4
43     syscall
44
45     xori $s0,$s0,0xffffffff
46     move $a0,$s0
47
48     ori $v0,$zero,34
49     syscall
50
51     ori $v0,$zero,10
52     syscall

```

- Nhập chương trình sau
- Biên dịch và chạy chương trình
- Giải thích kết quả
- Thêm chú thích để giải thích mục đích cho từng khối lệnh

# Bài tập 4.1

---

- Hãy viết 1 chương trình:
  - Yêu cầu người dùng nhập vào 2 số nguyên  $a$  và  $b$
  - In ra màn hình 2 số nguyên dưới dạng hexa
  - Thực hiện phép toán  $a \text{ XOR } b$  và in ra kết quả
  - XOR kết quả với  $b$  một lần nữa và in ra kết quả

# Phép dịch bit

- Phép dịch bit cho phép dịch chuyển một hay nhiều bit trong một thanh ghi.
- Có 2 chiều dịch bit: dịch bit trái và dịch bit phải.
  - Phép **dịch bit phải** di chuyển tất cả các bit trong một thanh ghi một số vị trí cụ thể về bên phải.
  - Phép **dịch bit trái** di chuyển tất cả các bit trong một thanh ghi một số vị trí cụ thể về bên trái.

# Phép dịch bit

- Dịch bit trái:
  - ▶ Dịch bit logic: điền bit 0 vào vị trí trống
- Dịch bit phải
  - ▶ Dịch bit logic: điền bit 0 vào vị trí trống
  - ▶ Dịch bit số học: điền bit cao nhất (bit dấu) vào vị trí trống
- Dịch vòng: bit dịch ra được đưa sang phía đối diện của giá trị.

# Toán tử dịch bit logic trái sll

- Dịch bit logic trái: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit sang bên trái, thay thế những bit đã bị dịch bởi bit 0 và lưu giá trị vào  $R_d$ .
- **shamt** không phải giá trị tức thời mà là số lượng bit cần dịch, nằm trong dải từ 0-31.
- Định dạng và ý nghĩa:

Định dạng: **sll  $R_d$ ,  $R_t$ , shamt**

Ý nghĩa:  **$R_d \leftarrow R_t \ll \text{shamt}$**

# Toán tử sllv

- Biến dịch bit logic trái: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit lưu trong thanh ghi  $R_s$  sang bên trái, thay thế những bit đã bị dịch bởi bit 0 và lưu giá trị vào  $R_d$ .
- Giá trị lưu trong  $R_s$  nên nằm trong dải từ 0-31, tuy nhiên lệnh sẽ vẫn được thực hiện với bất kỳ giá trị nào.
- Định dạng và ý nghĩa:

Định dạng: **sll  $R_d, R_t, R_s$**

Ý nghĩa:  **$R_d \leftarrow R_t \ll R_s$**

# Toán tử dịch bit logic phải srl

- Dịch bit logic phải: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit sang bên phải, thay thế những bit đã bị dịch bởi bit 0 và lưu giá trị vào  $R_d$ .
- **shamt** không phải giá trị tức thời mà là số lượng bit cần dịch, nằm trong dải từ 0-31.
- Định dạng và ý nghĩa:

Định dạng: **srl  $R_d$ ,  $R_t$ , shamt**

Ý nghĩa:  **$R_d \leftarrow R_t \gg \text{shamt}$**



# Toán tử srlv

- Biến dịch bit logic phải: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit lưu trong thanh ghi  $R_s$  sang bên phải, thay thế những bit đã bị dịch bởi bit 0 và lưu giá trị vào  $R_d$ .
- Giá trị lưu trong  $R_s$  nên nằm trong dải từ 0-31, tuy nhiên lệnh sẽ vẫn được thực hiện với bất kỳ giá trị nào.
- Định dạng và ý nghĩa:

Định dạng: **srlv**  $R_d, R_t, R_s$

Ý nghĩa:  $R_d \leftarrow R_t \gg R_s$

# Toán tử dịch bit số học phải sra

- Dịch bit số học phải: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit sang bên phải, thay thế những bit đã bị dịch bởi bit dấu và lưu giá trị vào  $R_d$ .

- Định dạng và ý nghĩa:

Định dạng: **sra  $R_d$ ,  $R_t$ , shamt**

Ý nghĩa:  **$R_d \leftarrow R_t \gg \text{shamt}$**

# Toán tử srav

- Biến dịch bit số học phải: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit lưu trong thanh ghi  $R_s$  sang bên phải, thay thế những bit đã bị dịch bởi bit dấu và lưu giá trị vào  $R_d$ .
- Giá trị lưu trong  $R_s$  nên nằm trong dải từ 0-31, tuy nhiên lệnh sẽ vẫn được thực hiện với bất kỳ giá trị nào.
- Định dạng và ý nghĩa:

Định dạng:  $\text{srav } R_d, R_t, R_s$

Ý nghĩa:  $R_d \leftarrow R_t \gg R_s$

# Toán tử dịch vòng trái rol

- Toán tử giả dịch vòng trái: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit sang bên trái, thay thế những bit trống bằng bit đã dịch ra và lưu giá trị vào  $R_d$ .

- Định dạng và ý nghĩa:

Định dạng: **rol  $R_d$ ,  $R_t$ , shamt**

Ý nghĩa:  **$R_d[\text{shamt}..0] \leftarrow R_t[31..31-\text{shamt}+1]$**

**$R_d[31..\text{shamt}] \leftarrow R_t[31-\text{shamt}..0]$**

# Toán tử dịch vòng trái rol

- Toán tử giả dịch vòng trái: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit sang bên trái, thay thế những bit trống bằng bit đã dịch ra và lưu giá trị vào  $R_d$ .
- Thực hiện: **sll \$at, \$R<sub>t</sub>, shamt**  
**srl \$R<sub>d</sub>, \$R<sub>t</sub>, 32-shamt**  
**or \$R<sub>d</sub>, \$R<sub>d</sub>, \$at**

# Toán tử dịch vòng trái rol

Ví dụ:  $R_t$ : 00011010

$\text{rol } R_d, R_t, 3$

	$R_t$	1	1	0	1	0	0	1	0
$\text{sll } \$at, \$R_t, 3$	$\$at$	1	0	0	1	0	0	0	0
$\text{srl } \$R_d, \$R_t, 8-3$	$R_d$	0	0	0	0	0	1	1	0
$\text{or } \$R_d, \$R_d, \$at$	$R_d$	1	0	0	1	0	1	1	0

# Toán tử dịch vòng phải ror

- Toán tử giả dịch vòng phải: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit sang bên phải, thay thế những bit trống bằng bit đã dịch ra và lưu giá trị vào  $R_d$ .

- Định dạng và ý nghĩa như sau:

Định dạng: **ror  $R_d$ ,  $R_t$ , shamt**

Ý nghĩa:  **$R_d[31\text{..shamt}] \leftarrow R_t[31\text{..shamt}]$**

**$R_d[31\text{..}31\text{-shamt}+1] \leftarrow R_t[\text{shamt}-1\text{..}0]$**

# Toán tử dịch vòng phải ror

- Toán tử giả vòng phải: dịch giá trị trong thanh ghi  $R_t$  một khoảng **shamt** bit sang bên phải, thay thế những bit trống bằng bit đã dịch ra và lưu giá trị vào  $R_d$ .
- Thực hiện: **srl \$at, \$R<sub>t</sub>, shamt**  
**sll \$R<sub>d</sub>, \$R<sub>t</sub>, 32-shamt**  
**or \$R<sub>d</sub>, \$R<sub>d</sub>, \$at**



# Toán tử dịch vòng phải ror

Ví dụ:  $R_t$ : 00011010

ror  $R_d$ ,  $R_t$ , 3

	$R_t$	1	1	0	1	0	0	1	0
srl \$at, \$ $R_t$ , 3	\$at	0	0	0	1	1	0	1	0
sll \$ $R_d$ , \$ $R_t$ , 8-3	$R_d$	0	1	0	0	0	0	0	0
or \$ $R_d$ , \$ $R_d$ , \$at	$R_d$	0	1	0	1	1	0	1	0

# Bài tập 4.2

---

- Viết chương trình thực hiện:
  - Yêu cầu người dùng nhập vào một số nguyên
  - Thực hiện các phép dịch bit trái, phải, số học, vòng một số bit giống nhau
  - In kết quả ra màn hình sao cho có thể so sánh được vị trí các bit

# Program 4.3 Toán tử dịch bit

- Nhập chương trình sau
- Biên dịch và chạy chương trình
- Giải thích kết quả
- Thêm chú thích để giải thích mục đích cho từng khối lệnh

```
.data
result1: .ascii "\nshift left logical 4 by 2 bits is "
result2: .ascii "\nshift right logical 16 by 2 bits is "
result3: .ascii "\nshift right arithmetic 34 by 2 bits is "
result4: .ascii "\nshift right arithmetic -34 by 2 bits is "
result5: .ascii "\nrotate right 0xffffffffe1 by 2 bits is "
result6: .ascii "\nrotate left 0xffffffffe1 by 2 bits is "
```

```
.text
.globl main
main:
    addi $t0, $zero, 4
    sll $s0, $t0, 2
    addi $v0, $zero, 4
    la $a0, result1
    syscall

    addi $v0, $zero, 1
    move $a0, $s0
    syscall

    addi $t0, $zero, 16
    srl $s0, $t0, 2
    addi $v0, $zero, 4
    la $a0, result2
    syscall

    addi $v0, $zero, 1
    move $a0, $s0
    syscall

    addi $t0, $zero, 34
    sra $s0, $t0, 2
    addi $v0, $zero, 4
    la $a0, result3
    syscall

    addi $v0, $zero, 1
    move $a0, $s0
    syscall

    addi $t0, $zero, -34
    sra $s0, $t0, 2
    addi $v0, $zero, 4
    la $a0, result4
    syscall

    addi $v0, $zero, 1
    move $a0, $s0
    syscall

    ori $t0, $zero, 0xffffffffe1
    ror $s0, $t0, 2
    li $v0, 4
    la $a0, result6.
    syscall

    li $v0, 34
    move $a0, $s0
    syscall

    ori $t0, $zero, 0xffffffffe1
    rol $s0, $t0, 2
    li $v0, 4
    la $a0, result6
    syscall

    li $v0, 34
    move $a0, $s0
    syscall

    addi $v0, $zero, 10
    syscall
```

# Bài tập 4.2

- Ví dụ

Please input an integer value: -12345

Input value in binary:	111111111111111111111111001111111000111
Result after sll 4 bits:	111111111111111111110011111110001110000
Result after sllv 4 bits:	111111111111111111110011111110001110000
Input value in binary:	111111111111111111111111001111111000111
Result after srl 4 bits:	000011111111111111111111111110011111100
Result after srlv 4 bits:	00001111111111111111111111111110011111100
Input value in binary:	111111111111111111111111001111111000111
Result after sra 4 bits:	11111111111111111111111111111110011111100
Result after srav 4 bits:	11111111111111111111111111111110011111100
Input value in binary:	111111111111111111111111001111111000111
Result after rol 4 bits:	111111111111111111110011111110001111111
Result after ror 4 bits:	01111111111111111111111111111110011111100

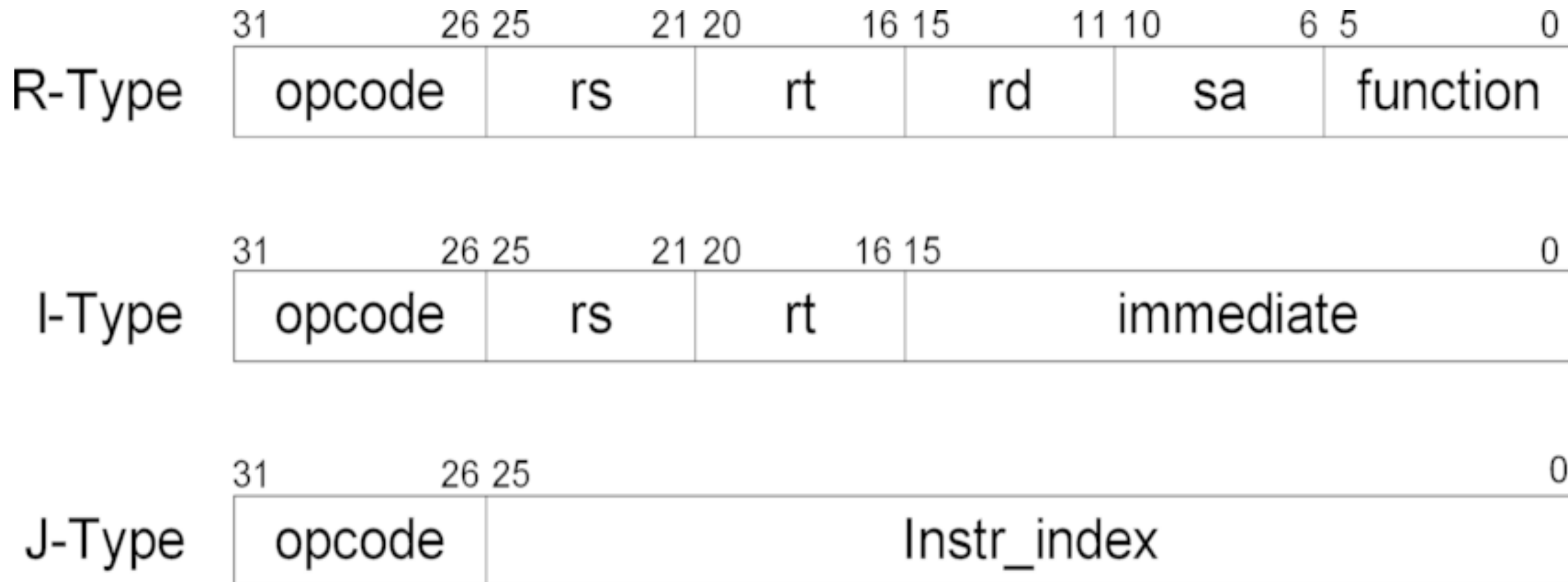
# Dịch mã hợp ngữ sang mã máy

---

- Có 3 cách để định dạng một lệnh trong MIPS:
  - Định dạng R (Thanh ghi): được sử dụng khi các giá trị đầu vào đưa vào ALU đến từ 2 thanh ghi.
  - Định dạng I (tức thì): được sử dụng khi các giá trị đầu vào đưa vào ALU đến từ 1 thanh ghi và 1 giá trị tức thì.
  - Định dạng J (nhảy): dành cho các lệnh nhảy

# Dịch mã hợp ngữ sang mã máy

- Định dạng lệnh



- opcode: mã 6 bit xác định toán tử

# Các định dạng lệnh

Câu lệnh ở ngôn ngữ bậc cao:

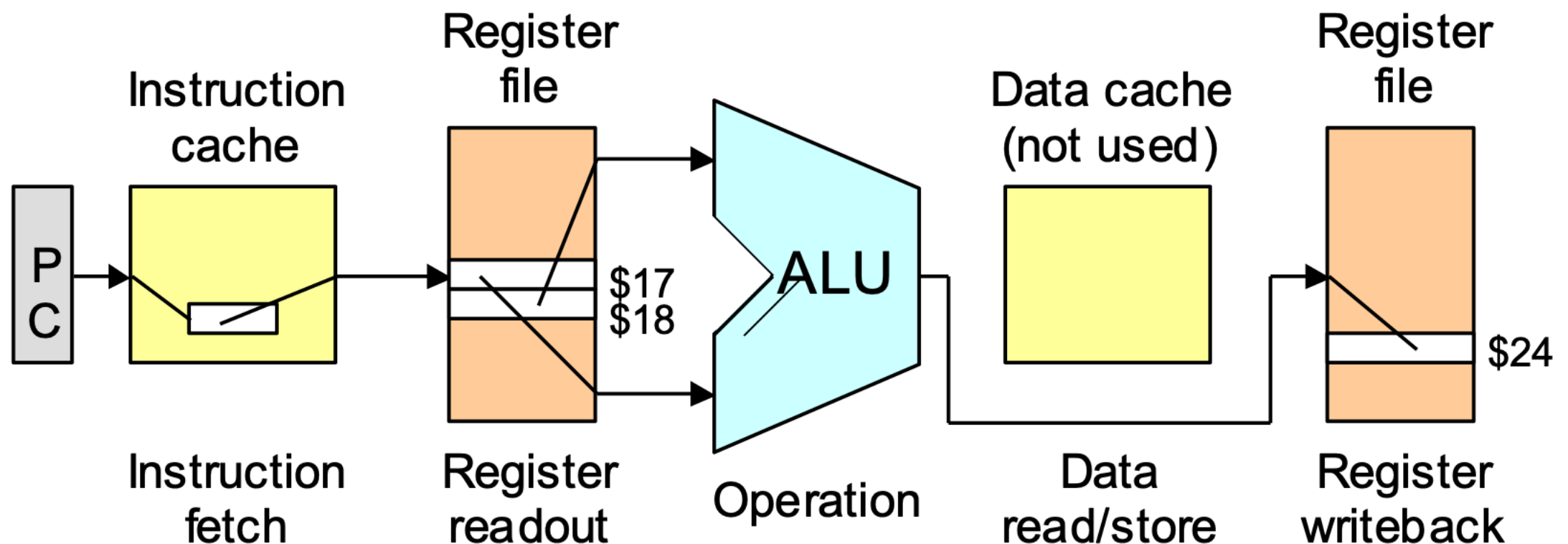
$a = b + c$

Lệnh hợp ngữ:

add \$t8, \$s2, \$s1

Mã máy:

000000	10010	10001	11000	00000	100000
ALU-type instruction	Register 18	Register 17	Register 24	Unused	Addition opcode



# Lệnh R



- $R_s$ : thanh ghi đầu tiên trong lệnh, và luôn là 1 đầu vào của ALU.  $R_s$ ,  $R_t$  và  $R_d$  có chiều dài 5 bit, cho phép đánh địa chỉ của 32 thanh ghi cho mục đích chung.
- $R_t$ : Thanh ghi thứ 2 trong lệnh, là đầu vào thứ 2 của ALU
- $R_d$ : luôn là thanh ghi đích
- (sa) Shift: số bit dịch nếu là toán tử dịch bit, trong các trường hợp khác bằng 0
- Funct: đối với lệnh loại R, được xác định trong toán tử cho ALU



# Lệnh R

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

**add \$t0, \$s1, \$s2**

0	\$s1	\$s2	\$t0	0	add
0	17	18	8	0	32

000000	10001	10010	01000	00000	100000
(0x02324020)					

**sub \$s0, \$t3, \$t5**

0	\$t3	\$t5	\$s0	0	sub
0	11	13	16	0	34

000000	01011	01101	10000	00000	100010
(0x016D8022)					

# Lệnh I

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

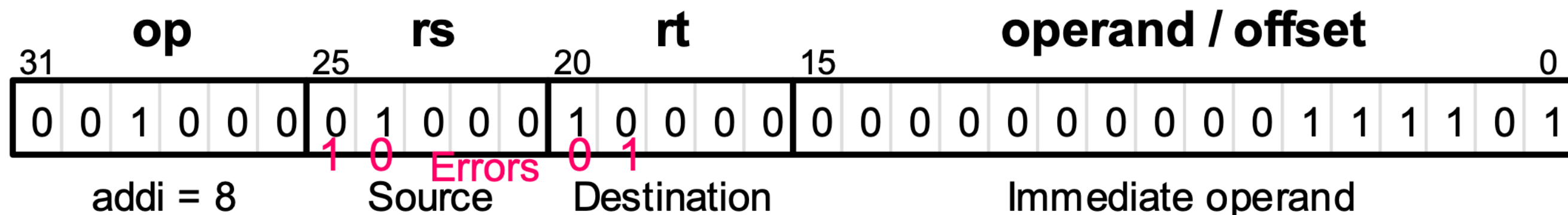
- Được sử dụng với các lệnh logic/số học với các toán tử tức thì và lệnh **load/store**.
- Các tham số lệnh:
  - **op** (operation code - opcode): xác định toán tử
  - **rs**: thanh ghi đầu tiên trong lệnh, luôn là đầu vào cho ALU (thanh ghi nguồn).
  - **rt**: thanh ghi thứ 2 trong lệnh và là thanh ghi đích.
  - **immediate**: giá trị cho lệnh

# Lệnh logic/số học với giá trị tức thì

- Các toán tử trong phạm vi  $[-32768, 32767]$  hoặc  $[0x0000, 0xffff]$  có thể được thiết lập trong trường giá trị tức thì.

```
addi    $t0, $s0, 61      # set $t0 to ($s0)+61
andi    $t0, $s0, 61      # set $t0 to ($s0)^61
ori     $t0, $s0, 61      # set $t0 to ($s0)∨61
xori    $t0, $s0, 0x00ff  # set $t0 to ($s0)⊕ 0x00ff
```

**Đối với các lệnh số học, toán hạng tức thì là số có dấu mở rộng**



# Lệnh logic/số học với giá trị tức thì

- Đối với các toán tử **addi**, **lw**, **sw**, giá trị của thanh ghi cần phải cộng thêm giá trị tức thì:
  - Thanh ghi là 32 bit.
  - Giá trị tức thì là 16 bit, do đó, cần chuyển thành loại 32 bit có dấu mở rộng.

- Ví dụ:

+5 =

0000 0000 0000 0101

16-bit

+5 =

0000 0000 0000 0000 0000 0000 0000 0101

32-bit

-12 =

1111 1111 1111 0100

16-bit

-12 =

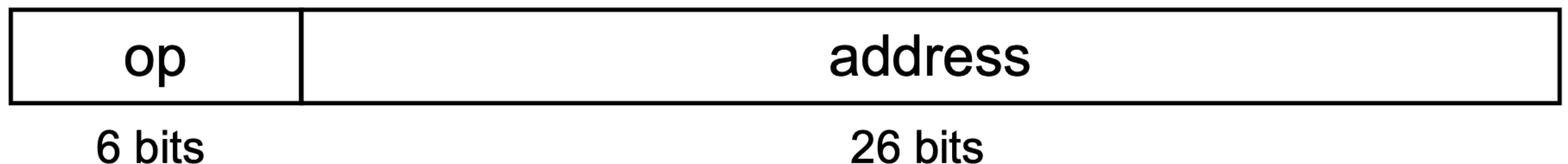
1111 1111 1111 1111 1111 1111 1111 0100

32-bit

# Lệnh J

---

- Toán tử có địa chỉ dài 26-bit
- Được sử dụng cho các lệnh nhảy:
  - **j** (jump): op=000010
  - **jal** (jump and link): op=000011



# Dịch mã hợp ngữ sang mã máy

foration to separate card 2. Fold bottom side (columns 3 and 4) together

## MIPS Reference Data

①



### CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add	R R[rd] = R[rs] + R[rt]	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi	I R[rt] = R[rs] + SignExtImm	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu	I R[rt] = R[rs] + SignExtImm	(2) 9 <sub>hex</sub>
Add Unsigned	addu	R R[rd] = R[rs] + R[rt]	0 / 21 <sub>hex</sub>
And	and	R R[rd] = R[rs] & R[rt]	0 / 24 <sub>hex</sub>
And Immediate	andi	I R[rt] = R[rs] & ZeroExtImm	(3) c <sub>hex</sub>
Branch On Equal	beq	I if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne	I if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4) 5 <sub>hex</sub>
Jump	j	J PC=JumpAddr	(5) 2 <sub>hex</sub>
Jump And Link	jal	J R[31]=PC+8;PC=JumpAddr	(5) 3 <sub>hex</sub>
Jump Register	jr	R PC=R[rs]	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu	I R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu	I R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2) 25 <sub>hex</sub>
Load Linked	ll	I R[rt] = M[R[rs]+SignExtImm]	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui	I R[rt] = {imm, 16'b0}	f <sub>hex</sub>
Load Word	lw	I R[rt] = M[R[rs]+SignExtImm]	(2) 23 <sub>hex</sub>
Nor	nor	R R[rd] = ~ (R[rs]   R[rt])	0 / 27 <sub>hex</sub>
Or	or	R R[rd] = R[rs]   R[rt]	0 / 25 <sub>hex</sub>
Or Immediate	ori	I R[rt] = R[rs]   ZeroExtImm	(3) d <sub>hex</sub>
Set Less Than	slt	R R[rd] = (R[rs] < R[rt]) ? 1 : 0	0 / 2a <sub>hex</sub>

### ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FUNCT (Hex)
Branch On FP True	bclt	FI if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/1/--
Branch On FP False	bclf	FI if(!FPcond)PC=PC+4+BranchAddr	(4) 11/8/0/--
Divide	div	R Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0/--/--/1a
Divide Unsigned	divu	R Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6) 0/--/--/1b
FP Add Single	add.s	FR F[fd] = F[fs] + F[ft]	11/10/--/0
FP Add Double	add.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/--/0
FP Compare Single	c.x.s*	FR FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/--/y
FP Compare Double	c.x.d*	FR FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0	11/11/--/y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)			
FP Divide Single	div.s	FR F[fd] = F[fs] / F[ft]	11/10/--/3
FP Divide Double	div.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/--/3
FP Multiply Single	mul.s	FR F[fd] = F[fs] * F[ft]	11/10/--/2
FP Multiply Double	mul.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}	11/11/--/2
FP Subtract Single	sub.s	FR F[fd]=F[fs] - F[ft]	11/10/--/1
FP Subtract Double	sub.d	FR {F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/--/1
Load FP Single	lwc1	I F[rt]=M[R[rs]+SignExtImm]	(2) 31/--/--/--
Load FP Double	ldc1	I F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/--/--/--
Move From Hi	mfhi	R R[rd] = Hi	0 /--/--/10
Move From Lo	mflo	R R[rd] = Lo	0 /--/--/12
Move From Control	mfc0	R R[rd] = CR[rs]	10 /0/--/0
Multiply	mult	R {Hi,Lo} = R[rs] * R[rt]	0/--/--/18
Multiply Unsigned	multu	R {Hi,Lo} = R[rs] * R[rt]	(6) 0/--/--/19
Shift Right Arith.	sra	R R[rd] = R[rt] >>> shamt	0/--/--/3
Store FP Single	swc1	I M[R[rs]+SignExtImm] = F[rt]	(2) 39/--/--/--
Store FP Double	sdc1	I M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/--/--/--

# Dịch mã hợp ngữ sang mã máy

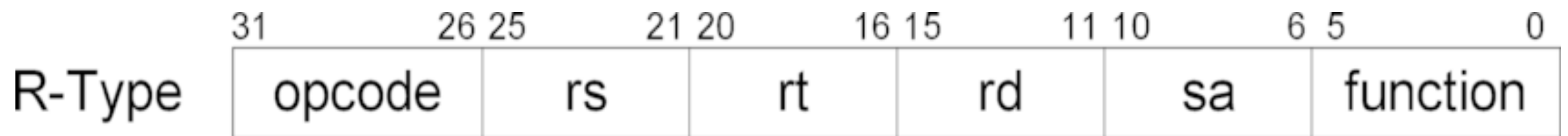
- Assembly to machine code

Text Segment		Machine Code	Original Source Code
Bkpt	Address	Code	Basic
<input type="checkbox"/>	0x00400000	0x012a4020	add \$8,\$9,\$10
<input type="checkbox"/>	0x00400004	0x22300025	addi \$16,\$17,0x00000025

# Mã máy với lệnh add

- Dịch lệnh sau sang mã máy:

**add \$t0, \$t1, \$t2**



- ▶ Lệnh định dạng R
  - ▶  $R_d$ : Thanh ghi \$t0 là \$8 hay 01000
  - ▶  $R_s$ : Thanh ghi \$t1 là \$9 hay 01001
  - ▶  $R_t$ : Thanh ghi \$t2 là \$10 hay 01010
  - ▶ sa (shamt) là 00000
- ▶ Toán hạng/hàm: 0/20  
→ 00 0000/10 0000

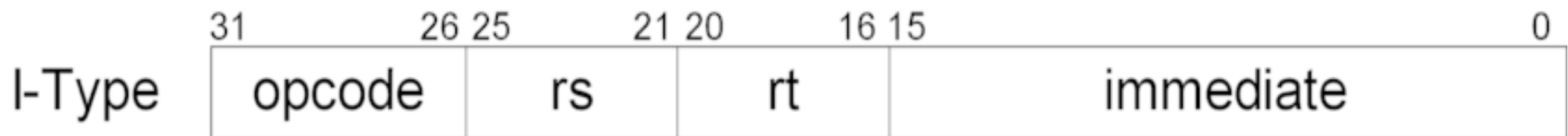
**0000 0010 0011 0010 1000 0000 0010 0010<sub>2</sub> hay 0x02328022**



# Mã máy với lệnh addi

- Dịch lệnh sau sang mã máy:

**addi \$s2, \$t8, 37**



- ▶ Lệnh định dạng I
- ▶ Opcode: 8 → 00 1000
- ▶  $R_t$ : Thanh ghi \$s2 là \$18 hay 10010
- ▶  $R_s$ : Thanh ghi \$t8 là \$24 hay 11000
- ▶ Giá trị tức thì là 37, hay 0x0025

**0010 0011 0001 0010 0000 0000 0010 0101<sub>2</sub> hay 0x23120025**

# Mã máy với lệnh sll

- Dịch lệnh sau sang mã máy:

**sll \$t0, \$t1, 10**

	31	26 25	21 20	16 15	11 10	6 5	0
R-Type	opcode	rs	rt	rd	sa	function	

- ▶ Lệnh định dạng R
- ▶ Opcode/function: 0/00  
→ 00 0000/00 0000
- ▶  $R_d$ : Thanh ghi \$t0 là \$8 hay 01000
- ▶  $R_s$  không được sử dụng
- ▶  $R_t$ : Thanh ghi \$t1 là \$9 hay 01001
- ▶ sa (shamt) là 10 hay 0x01010

**0000 0000 0000 1001 0100 0010 1000 0000<sub>2</sub> hay 0x00094280**

# Bài tập 4.3

- Dịch các lệnh sau sang mã máy:

```
.text
.globl main
main:
    ori $t0, $zero, 15
    ori $t1, $zero, 3
    add $t1, $zero, $t1
    sub $t2, $t0, $t1
    sra $t2, $t2, 2
    mult $t0, $t1
    mflo $a0
    ori $v0, $zero, 1
    syscall
    addi $v0, $zero, 10
    syscall

.data
result: .asciiz "15 * 3 is "
```

# Bài tập 4.4

---

- Dịch các lệnh sau ở dạng mã máy sang lệnh MIPS:

`0x2010000a`

`0x34110005`

`0x012ac022`

`0x00184082`

`0x030f9024`

# Bài tập 4.5

Chương trình sau mô tả cách sử dụng các lệnh logic để lấy thông tin từ thanh ghi.

```
.text
    li    $s0, 0x0563
    andi   $t0, $s0, 0xff
    andi   $t1, $s0, 0x0400
```

- Có thể lấy 1 hoặc nhiều bit tùy theo mặt nạ được sử dụng.
- Đọc và chú thích từng dòng lệnh.

# Bài tập 4.6

Gõ 2 lệnh sau vào MARS, biên dịch và chạy chương trình.

**addiu \$t0, \$zero, 60000**

**ori \$t0, \$zero, 60000**

- Có gì khác nhau giữa hai lệnh này không?
- Kết quả có giống nhau không?
- Có lệnh nào chạy nhanh hơn không?
- Giải thích.

# Bài tập 4.7

- Giả lệnh trong MIPS là các lệnh không được chạy trực tiếp trên MIPS mà phải chuyển sang các lệnh thực của MIPS. Hãy viết lại các giả lệnh sau sử dụng lệnh thực mà bộ xử lý MIPS có thể hiểu được:

a. abs      \$s0, s1  
      s0 <= | \$s1 |

b. move     \$s0, s1  
      s0 <= \$s1

c. not      \$s0  
      s0 <= bit invert (s0)

d. ble      \$s1, s2, L  
      if (s1 <= \$s2)  
          j L

# Assignment 4.1

- Viết một chương trình thực hiện công việc sau:
  - Lấy bit MSB của \$s0
  - Xoá bit LSB của \$s0
  - Thiết lập bit LSB của \$s0 (bit 7->0 thiết lập về 1)
  - Xoá \$s0 (s0=0, phải sử dụng lệnh logic)

s0 = 0x 1 2 3 4 5 6 7 8  
          ↓          ↓  
      MSB      LSB

MSB: Most Significant Byte  
LSB: Least Significant Byte



# Assignment 4.2

---

- Xây dựng một chương trình phát hiện tràn số trong phép cộng, trong đó, khi 2 toán hạng cùng dấu thì tràn số sẽ xảy ra nếu dấu của tổng không cùng dấu với hai toán hạng.

# Assignment 4.3

---

- Viết một chương trình thực hiện phép nhân với bội của 2 (ví dụ 2, 4, 8, 16...).

# Assignment 4.4

---

- Viết một chương trình thực hiện thao tác bit BÙ (NOT) của một số nguyên do người dùng nhập vào.
- Nên sử dụng toán tử XOR để thực hiện phép BÙ NOT, không sử dụng toán tử NOT.
- Chương trình bao gồm các lời nhắc và in kết quả ra màn hình một cách thích hợp.

# Assignment 4.5

---

- Viết chương trình thực hiện phép tính bù 2 của một số nguyên do người dùng nhập vào.
- Chương trình chỉ sử dụng toán tử XOR và ADD.
- Chương trình bao gồm các lời nhắc và in kết quả ra màn hình một cách thích hợp.

# Assignment 4.6

---

- Viết một chương trình đơn giản thực hiện phép toán thao tác bit NAND trong MARS.
- Chương trình bao gồm các lời nhắc và in kết quả ra màn hình một cách thích hợp.

# Assignment 4.7

---

- Thực hiện một chương trình nhân một số nguyên do người dùng nhập vào với 10 chỉ sử dụng toán tử dịch bit và toán tử cộng.
- Kiểm tra độ chính xác bằng toán tử **mult** và **mflo**.
- Chương trình bao gồm các lời nhắc và in kết quả ra màn hình một cách thích hợp.

**Kết thúc tuần 4**