# Thực hành ICT4

TS. Nguyễn Thị Thanh Nga
Bộ môn KTMT
Viện CNTT&TT

# Tuần 4

# Toán tử logic

- Toán tử logic
- Sử dụng toán tử logic

# 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 |

- Toán tử logic
- Toán tử bitwise (MIPS hỗ trợ)
  - AND/OR/NOT/XOR

# Toán tử and

- Lệnh thực, thực hiện 1 bitwise **AND** cho $\mathbf{R_s}$ và $\mathbf{R_t}$, lưu vào thanh ghi $\mathbf{R_d}$

- Định dạng và ý nghĩa như sau:

Định dạng: and $R_d$, $R_s$, $R_t$

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

# Toán tử and

- Lệnh giả, thực hiện 1 bitwise **AND** cho $\mathbf{R_s}$ và một giá trị tức thời, lưu vào thanh ghi $\mathbf{R_t}$ , là cách viết tắt cho toán tử **andi**

- Định dạng, ý nghĩa và thực hiện như sau:

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

  Ý nghĩa: $R_t \leftarrow R_s$ AND Giá trị tức thời

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

# Toán tử and

- Lệnh giả, thực hiện 1 bitwise **AND** cho $\mathbf{R_s}$ và một giá trị tức thời, lưu vào thanh ghi $\mathbf{R_s}$

- Định dạng, ý nghĩa và thực hiện như sau:

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

Ý nghĩa: $R_s \leftarrow R_s$ AND Giá trị tức thời

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

# Toán tử andi

- Lệnh thực, thực hiện 1 bitwise **AND** cho $\mathbf{R_s}$ và một giá trị tức thời, lưu vào thanh ghi $\mathbf{R_t}$

- Định dạng và ý nghĩa như sau:

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

Ý nghĩa: $R_t \leftarrow R_s$ AND Giá trị tức thời

# Toán tử andi

- Cách ngắn gọn với 1 thanh ghi áp dụng cho andi

- Định dạng và ý nghĩa như sau:

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

Ý nghĩa: $R_s \leftarrow R_s$ AND Giá trị tức thời

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

# Toán tử or

- Lệnh thực, thực hiện 1 bitwise **OR** cho $\mathbf{R_s}$ và $\mathbf{R_t}$, lưu vào thanh ghi $\mathbf{R_d}$

- Định dạng và ý nghĩa như sau:

Định dạng: or $R_d$, $R_s$, $R_t$

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

# Toán tử or

- Lệnh giả, thực hiện 1 bitwise **OR** cho $\mathbf{R_s}$ và một giá trị tức thời, lưu vào thanh ghi $\mathbf{R_t}$ , là cách viết tắt cho toán tử **ori**

- Định dạng, ý nghĩa và thực hiện như sau:

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

  Ý nghĩa: $R_t \leftarrow R_s$ OR Giá trị tức thời

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

# Toán tử or

- Lệnh giả, thực hiện 1 bitwise **OR** cho $\mathbf{R_s}$ và một giá trị tức thời, lưu vào thanh ghi $\mathbf{R_s}$

- Định dạng, ý nghĩa và thực hiện như sau:

Định dạng: or $R_s$, Giá trị tức thời

Ý nghĩa: $R_s \leftarrow R_s$ OR Giá trị tức thời

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

# Toán tử ori

- Lệnh thực, thực hiện 1 bitwise **OR** cho $\mathbf{R_s}$ và một giá trị tức thời, lưu vào thanh ghi $\mathbf{R_t}$

- Định dạng và ý nghĩa như sau:

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

Ý nghĩa: $R_t \leftarrow R_s$ OR Giá trị tức thời

# Toán tử ori

- Cách ngắn gọn với 1 thanh ghi áp dụng cho ori

- Định dạng và ý nghĩa như sau:

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

Ý nghĩa: $R_s \leftarrow R_s$ OR Giá trị tức thời

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

# Toán tử xor

- Lệnh thực, thực hiện 1 bitwise **XOR** cho $\mathbf{R_s}$ và $\mathbf{R_t}$, lưu vào thanh ghi $\mathbf{R_d}$

- Định dạng và ý nghĩa như sau:

Định dạng: xor $R_d$, $R_s$, $R_t$

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

# Toán tử xor

- Lệnh giả, thực hiện 1 bitwise **XOR** cho $\mathbf{R_s}$ và một giá trị tức thời, lưu vào thanh ghi $\mathbf{R_t}$ , là cách viết tắt cho toán tử **xori**

- Định dạng, ý nghĩa và thực hiện như sau:

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

  Ý nghĩa: $R_t \leftarrow R_s$ XOR Giá trị tức thời

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

# Toán tử xor

- Lệnh giả, thực hiện 1 bitwise **XOR** cho $\mathbf{R_s}$ và một giá trị tức thời, lưu vào thanh ghi $\mathbf{R_s}$

- Định dạng, ý nghĩa và thực hiện như sau:

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

Ý nghĩa: $R_s \leftarrow R_s$ XOR Giá trị tức thời

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

# Toán tử xori

- Lệnh thực, thực hiện 1 bitwise **XOR** cho $\mathbf{R_s}$ và một giá trị tức thời, lưu vào thanh ghi $\mathbf{R_t}$

- Định dạng và ý nghĩa như sau:

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

Ý nghĩa: $R_t \leftarrow R_s$ XOR Giá trị tức thời

# Toán tử xori

- Cách ngắn gọn với 1 thanh ghi áp dụng cho xori

- Định dạng và ý nghĩa như sau:

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

Ý nghĩa: $R_s \leftarrow R_s$ XOR Giá trị tức thời

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

# Toán tử not

- Lệnh giả, thực hiện 1 bitwise **NOT** (nghịch đảo từng bit) của thanh ghi $\mathbf{R_t}$, lưu vào thanh ghi $\mathbf{R_s}$

- Định dạng và ý nghĩa như sau:

  Định dạng: not $R_s$, $R_t$

  Ý nghĩa: $R_s \leftarrow NOT(R_t)$

  Thực hiện: nor $R_s$, $R_t$, $zero

# Toán tử logic

- Toán tử logic
- Sử dụng toán tử logic

# 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

**Mất một khoảng thời gian do phải lưu bit nhớ**

- Có thể sử dụng:

  ori $R_d$, $zero, Giá trị tức thời

# Chuyển một ký tự từ chữ hoa sang chữ thường

- Nhập chương trình sau
- Chạy và giải thích chương trình

```
.data
        output1: .asciiz "\nValid conversion: "
        output2: .asciiz "\nInvalid conversion, nothing is printed: "

.text
.globl main
main:
        ori $v0, $zero, 4
        la $a0, output1
        syscall

        ori $t0, $zero, 0x41
        addi $a0, $t0, 0x20
        ori $v0, $zero, 11

        ori $v0, $zero, 4
        la $a0, output2
        syscall

        ori $t0, $zero,0x61
        addi $a0, $t0, 0x20
        ori $v0, $zero, 11
        syscall

        ori $v0, $zero, 4
        la $a0, output1
        syscall

        ori $t0, $zero, 0x41
        ori $a0, $t0, 0x20
        ori $v0, $zero, 11
        syscall

        ori $v0, $zero, 4
        la $a0, output1
        syscall

        ori $t0, $zero,0x61
        ori $a0, $t0, 0x20
        ori $v0, $zero, 11
        syscall

        ori $v0, $zero, 10
        syscall
```

# Toán tử đảo với XOR

- Nhập chương trình sau
- Chạy và giải thích chương trình

```
.data
        output1: .asciiz "\nAfter first xor: "
        output2: .asciiz "\nAfter second xor: "

.text
.globl main
main:
        ori $s0, $zero, 0x01234567
        la $a0, output1
        li $v0, 4
        syscall
        xori $s0, $s0, 0xffffffff
        move $a0, $s0
        li $v0, 34
        syscall

        la $a0, output2
        li $v0, 4
        syscall
        xori $s0, $s0, 0xffffffff
        move $a0, $s0
        li $v0, 34
        syscall

        ori $v0, $zero, 10
        syscall
```

# Toán tử 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

# Toán tử 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 $R_d$.

- Định dạng và ý nghĩa như sau:

  Định dạng: sll $R_d$, $R_t$, shamt

  Ý nghĩa: $R_d \leftarrow R_t << shamt$

# 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 $R_d$.

- Định dạng và ý nghĩa như sau:

  Định dạng: sll $R_d$, $R_t$, $R_s$

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

# Toán tử 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 $R_d$.

- Định dạng và ý nghĩa như sau:

  Định dạng: srl $R_d$, $R_t$, shamt

  Ý nghĩa: $R_d \leftarrow R_t >>$ 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 $R_d$.

- Định dạng và ý nghĩa như sau:

  Định dạng: srlv $R_d$, $R_t$, $R_s$

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

# Toán tử 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 như sau:

  Định dạng: sra $R_d$, $R_t$, shamt

  Ý nghĩa: $R_d \leftarrow R_t >>$ 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 $R_d$.

- Định dạng và ý nghĩa như sau:

  Định dạng: srav $R_d$, $R_t$, $R_s$

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

# Toán tử rol

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

- Định dạng và ý nghĩa như sau:

Định dạng: rol $R_d$, $R_t$, shamt

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

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

Thực hiện: srl $at, $Rt, shamt

sll $Rd, $Rt, 32-shamt

or $Rd, $Rd, $at

# Toán tử 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 bang 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: $R_d[31\text{-}shamt..shamt] \leftarrow R_t[31..shamt]$

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

Thực hiện: sll $at, $Rt, shamt

srl $Rd, $Rt, 32-shamt

or $Rd, $Rd, $at

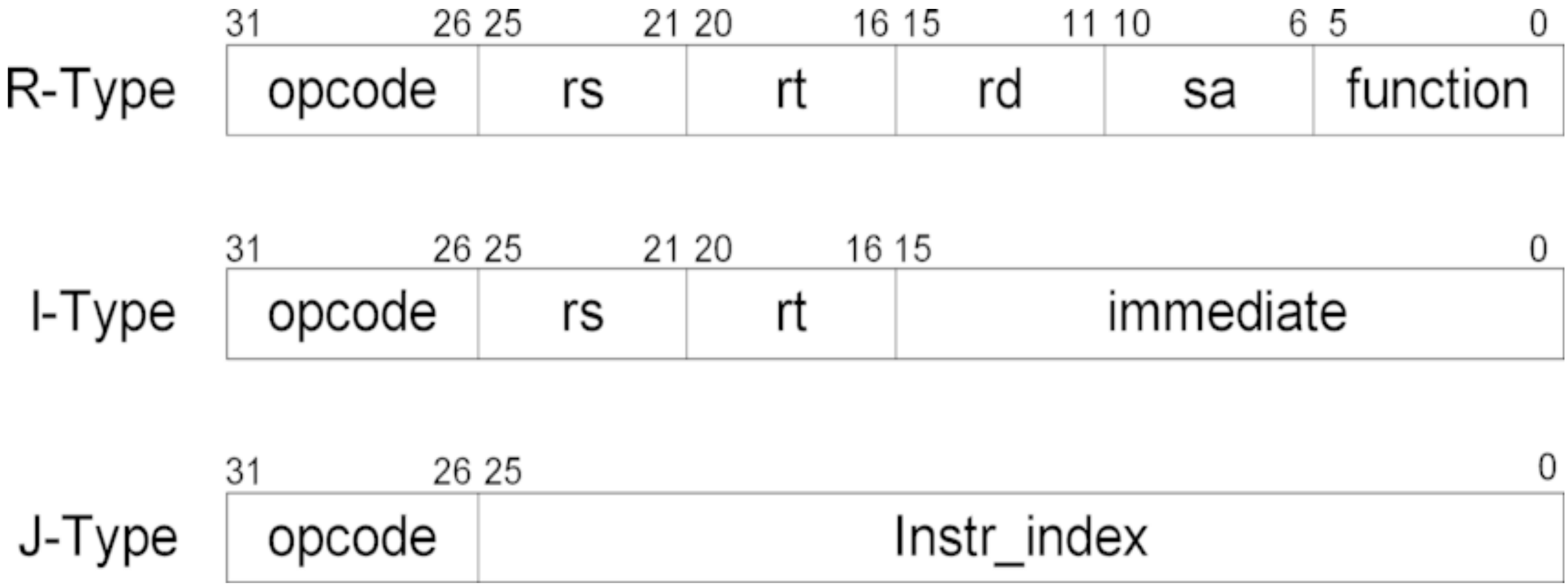
# Toán tử dịch bit

- Nhập chương trình sau
- Chạy và giải thích chương trình

```
.data
        result1: .asciiz "\nshift left logical 4 by 2 bits is "
        result2: .asciiz "\nshift right logical 16 by 2 bits is "
        result3: .asciiz "\nshift right arithmetic 34 by 2 bits is "
        result4: .asciiz "\nshift right arithmetic -34 by 2 bits is "
        result5: .asciiz "\nrotate right 0xfffffe1 by 2 bits is "
        result6: .asciiz "\nrotate left 0xfffffe1 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, 0xfffffe1
        ror $s0, $t0, 2
        li $v0, 4
        la $a0, result6.
        syscall
        li $v0, 34
        move $a0, $s0
        syscall

        ori $t0, $zero, 0xfffffe1
        rol $s0, $t0, 2
        li $v0, 4
        la $a0, result6
        syscall
        li $v0, 34
        move $a0, $s0
        syscall

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

# Dịch mã hợp ngữ sang mã máy

- Định dạng lệnh




- opcode: mã 6 bit xác định toán tử

http://www-inst.eecs.berkeley.edu/~cs61c/resources/MIPS_Green_Sheet.pdf

# Dịch mã hợp ngữ sang mã máy

fold bottom side (columns 3 and 4) together

2. Fold bottom side to separate card

# 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 / $20_{hex}$ |
| Add Immediate | `addi` | I | R[rt] = R[rs] + SignExtImm | (1,2) | $8_{hex}$ |
| Add Imm. Unsigned | `addiu` | I | R[rt] = R[rs] + SignExtImm | (2) | $9_{hex}$ |
| Add Unsigned | `addu` | R | R[rd] = R[rs] + R[rt] | | 0 / $21_{hex}$ |
| And | `and` | R | R[rd] = R[rs] & R[rt] | | 0 / $24_{hex}$ |
| And Immediate | `andi` | I | R[rt] = R[rs] & ZeroExtImm | (3) | $c_{hex}$ |
| Branch On Equal | `beq` | I | if(R[rs]==R[rt]) PC=PC+4+BranchAddr | (4) | $4_{hex}$ |
| Branch On Not Equal | `bne` | I | if(R[rs]!=R[rt]) PC=PC+4+BranchAddr | (4) | $5_{hex}$ |
| Jump | `j` | J | PC=JumpAddr | (5) | $2_{hex}$ |
| Jump And Link | `jal` | J | R[31]=PC+8;PC=JumpAddr | (5) | $3_{hex}$ |
| Jump Register | `jr` | R | PC=R[rs] | | 0 / $08_{hex}$ |
| Load Byte Unsigned | `lbu` | I | R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)} | (2) | $24_{hex}$ |
| Load Halfword Unsigned | `lhu` | I | R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)} | (2) | $25_{hex}$ |
| Load Linked | `ll` | I | R[rt] = M[R[rs]+SignExtImm] | (2,7) | $30_{hex}$ |
| Load Upper Imm. | `lui` | I | R[rt] = {imm, 16'b0} | | $f_{hex}$ |
| Load Word | `lw` | I | R[rt] = M[R[rs]+SignExtImm] | (2) | $23_{hex}$ |
| Nor | `nor` | R | R[rd] = ~ (R[rs] \| R[rt]) | | 0 / $27_{hex}$ |
| Or | `or` | R | R[rd] = R[rs] \| R[rt] | | 0 / $25_{hex}$ |
| Or Immediate | `ori` | I | R[rt] = R[rs] \| ZeroExtImm | (3) | $d_{hex}$ |
| Set Less Than | `slt` | R | R[rd] = (R[rs] < R[rt]) ? 1 : 0 | | 0 / $2a_{hex}$ |

## ARITHMETIC CORE INSTRUCTION SET

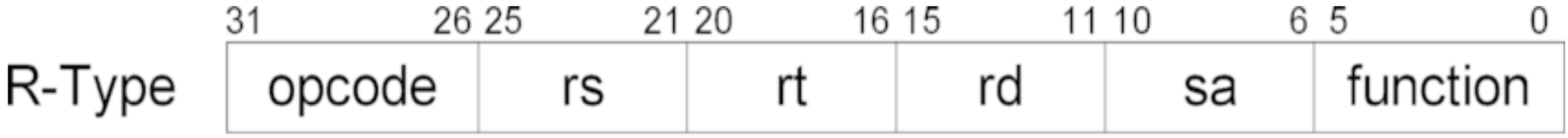
②

| NAME, MNEMONIC | | FOR-MAT | OPERATION | | OPCODE / FMT /FT / FUNCT (Hex) |
|---|---|---|---|---|---|
| Branch On FP True | `bc1t` | FI | if(FPcond)PC=PC+4+BranchAddr | (4) | 11/8/1/-- |
| Branch On FP False | `bc1f` | 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/--/--/-- |

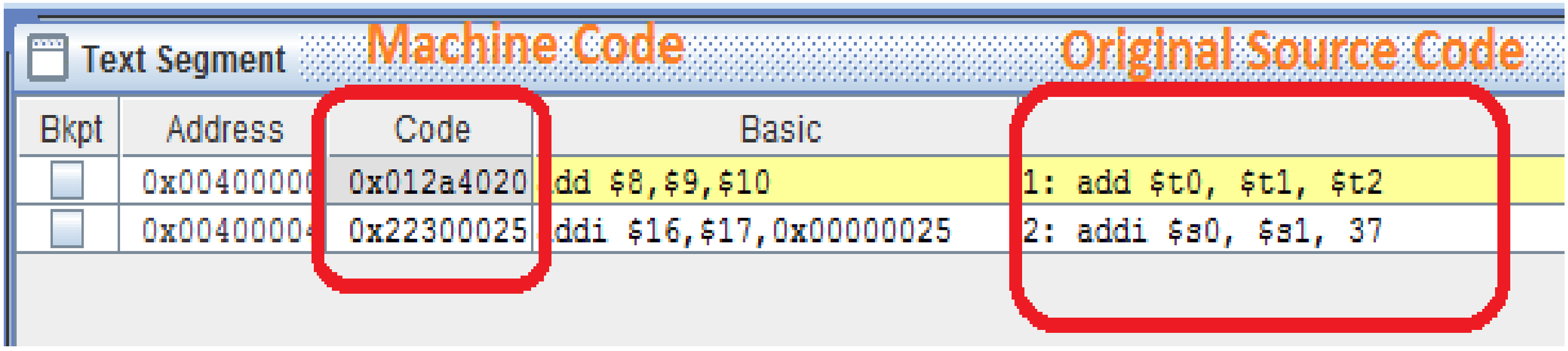
# Lệnh R




- $R_s$: 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) Shamt: 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

# Dịch mã hợp ngữ sang mã máy

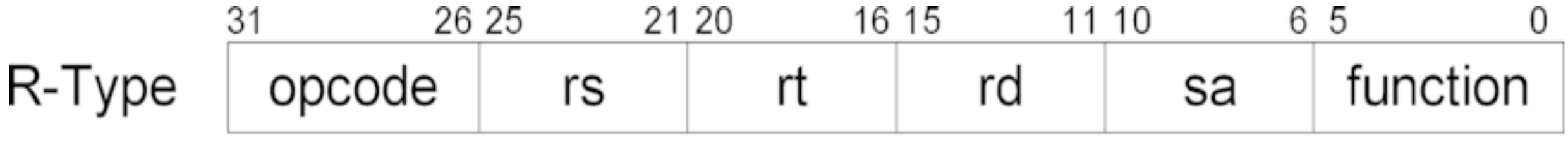
- Kiểm tra từ mã hợp ngữ sang mã máy



# Mã máy với lệnh add

- Dịch lệnh sau sang mã máy:

add $t0, $t1, $t2




- Lệnh định dạng R
- Opcode/function: 0/20

  → 00 0000/10 0000

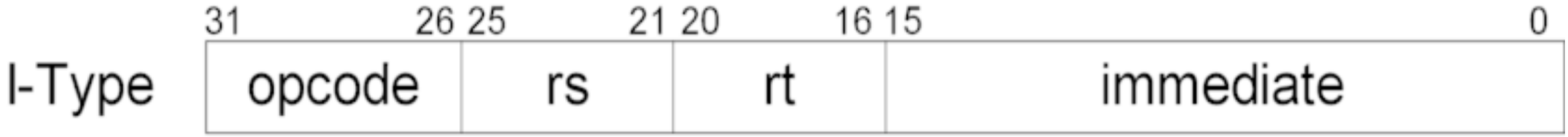
- $R_d$: $t0 là $8 hay 01000
- $R_s$: $t1 là $9 hay 01001
- $R_t$: $t2 là $10 hay 01010
- sa (shamt) là 00000

0000 0010 0011 0010 1000 0000 0010 00102 hay 0x02328022

# Mã máy với lệnh addi

- Dịch lệnh sau sang mã máy:

addi $s2, $t8, 37



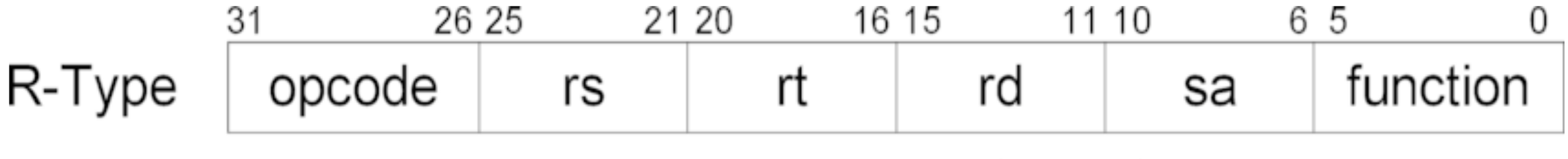

- Lệnh định dạng I
- Opcode: 8 → 00 1000
- $R_t$: $s2 là $18 hay 10010
- $R_s$: $t8 là $24 hay 11000
- Giá trị tức thì là 37, hay 0x0025

0010 0011 0001 0010 0000 0000 0010 01012  hay 0x23120025

# Mã máy với lệnh sll

- Dịch lệnh sau sang mã máy:

sll $t0, $t1, 10




- Lệnh định dạng R
- Opcode/function: 0/00

  → 00 0000/00 0000

- $R_d$: $t0 là $8 hay 01000
- $R_s$ không được sử dụng
- $R_t$: $t1 là $9 hay 01001
- sa (shamt) là 10 hay 0x01010

0000 0000 0000 1001 0100 0010 1000 00002 hay 0x00094280

# Bài tập

- Dịch chương trình hợp ngữ 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

- Dịch mã máy sau sang hợp ngữ MIPS

```
0x2010000a
0x34110005
0x012ac022
0x00184082
0x030f9024
```