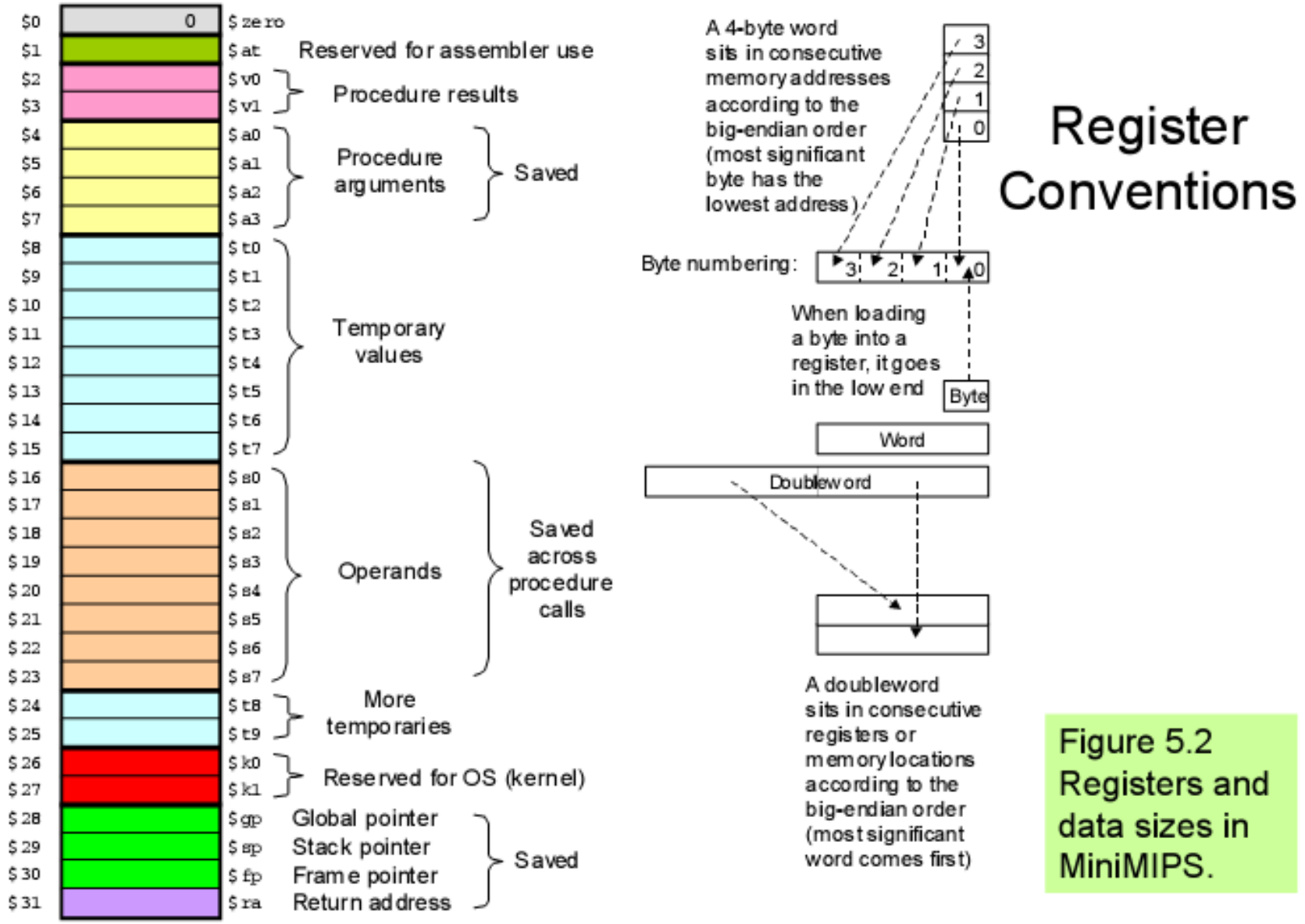
MIPS seminar

Phạm Văn Thuận

Nội dung

- Các thanh ghi của MIPS
- Các khuôn dạng lệnh
- Các chế độ địa chỉ
- Một số lệnh cơ bản
- Khung chương trình hợp ngữ
- Sử dụng trình biên dịch và mô phỏng MIPS2000, MIPS



Các thanh ghi của MIPS

Sử dụng các thanh ghi trong MIPSIT (iregdef.h)

```
#define zero        $0      /* wired zero */
#define AT $at      /* assembler temp */
#define v0  $2      /* return value */
#define v1  $3
#define a0  $4      /* argument registers a0-a3 */
#define a1  $5
#define a2  $6
#define a3  $7
#define t0   $8     /* caller saved  t0-t9 */
…
```

MiniMIPS Instruction Formats

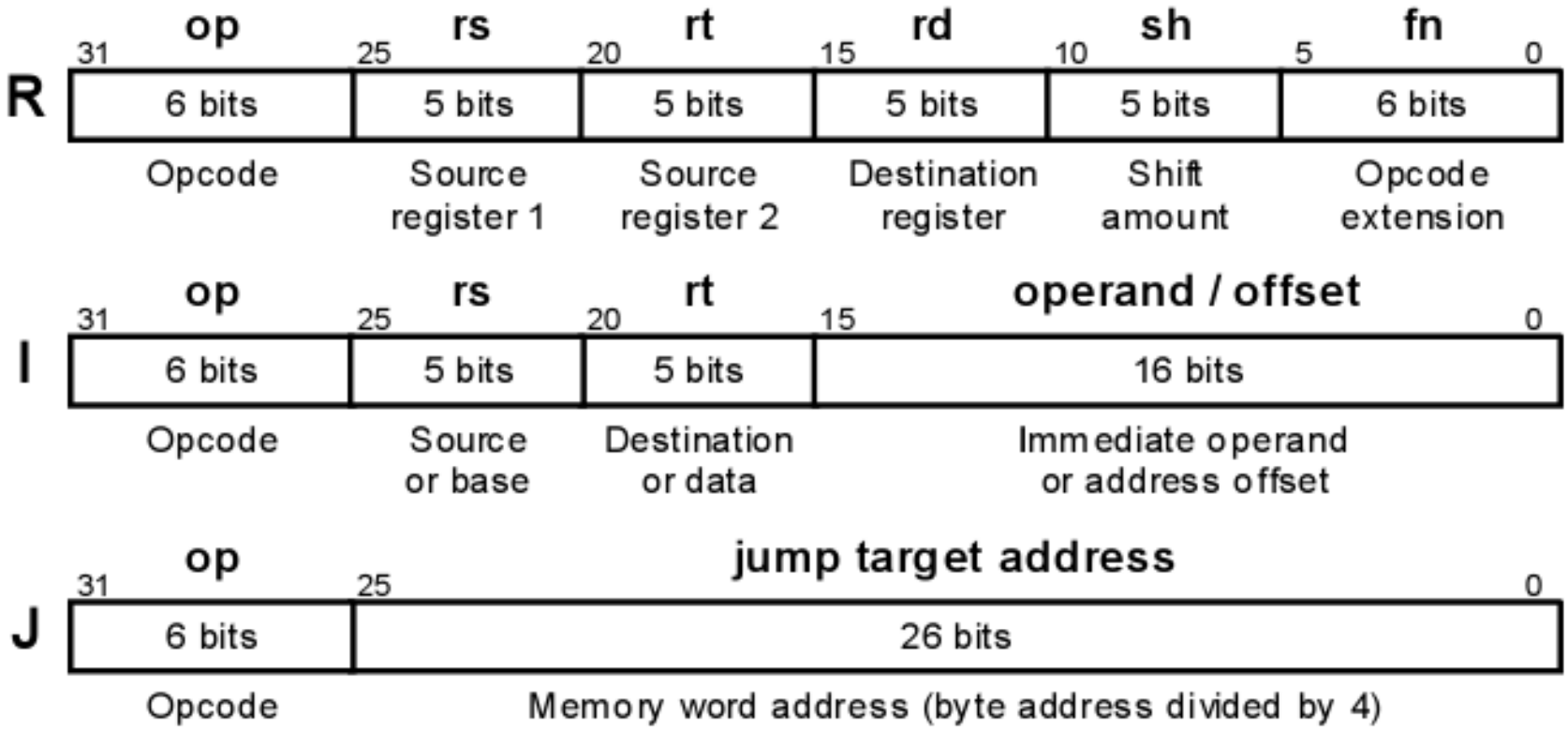
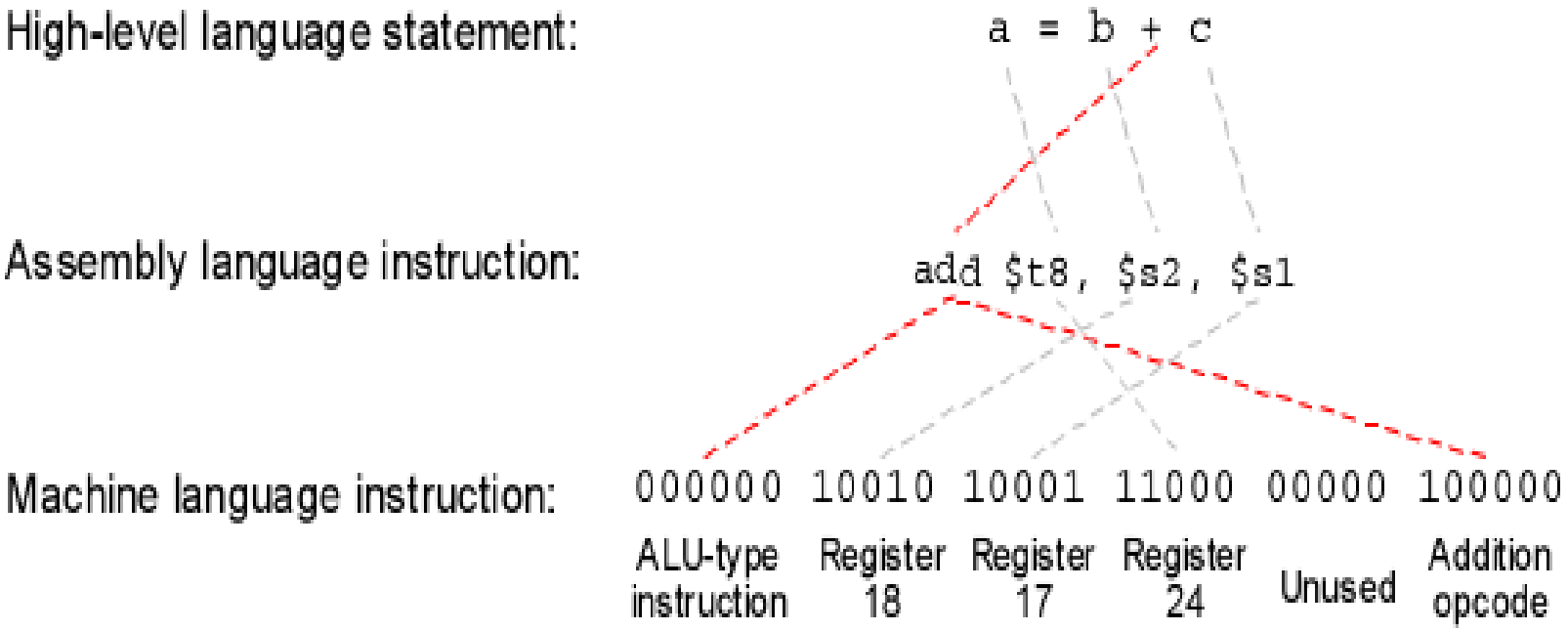


Figure 5.4 MiniMIPS instructions come in only three formats: register (R), immediate (I), and jump (J).

Các khuôn dạng lệnh

Phân tích khuôn dạng lệnh



5.3 Simple Arithmetic/Logic Instructions

Add and subtract already discussed; logical instructions are similar

```
add   $t0,$s0,$s1      # set $t0 to ($s0)+($s1)
sub   $t0,$s0,$s1      # set $t0 to ($s0)-($s1)
and   $t0,$s0,$s1      # set $t0 to ($s0)∧($s1)
or    $t0,$s0,$s1      # set $t0 to ($s0)∨($s1)
xor   $t0,$s0,$s1      # set $t0 to ($s0)⊕($s1)
nor   $t0,$s0,$s1      # set $t0 to (($s0)∨($s1))'
```

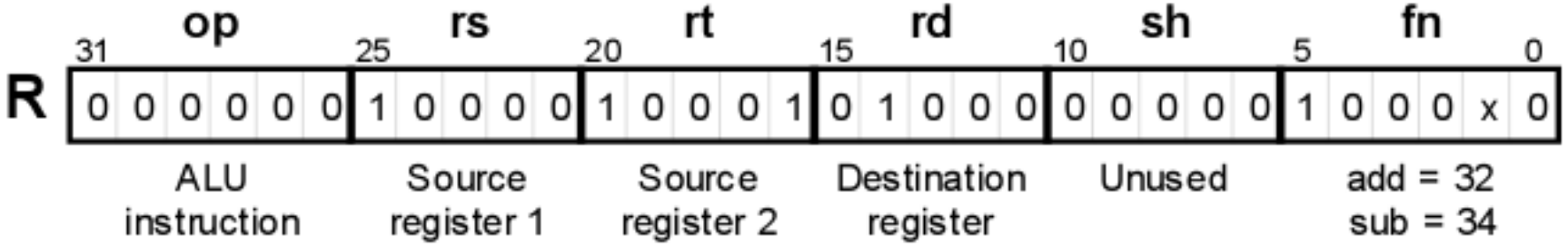


Figure 5.5 The arithmetic instructions `add` and `sub` have a format that is common to all two-operand ALU instructions. For these, the `fn` field specifies the arithmetic/logic operation to be performed.

Arithmetic/Logic with One Immediate Operand

An operand in the range [–32 768, 32 767], or [0x0000, 0xffff], can be specified in the immediate field.

```
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
```

For arithmetic instructions, the immediate operand is sign-extended

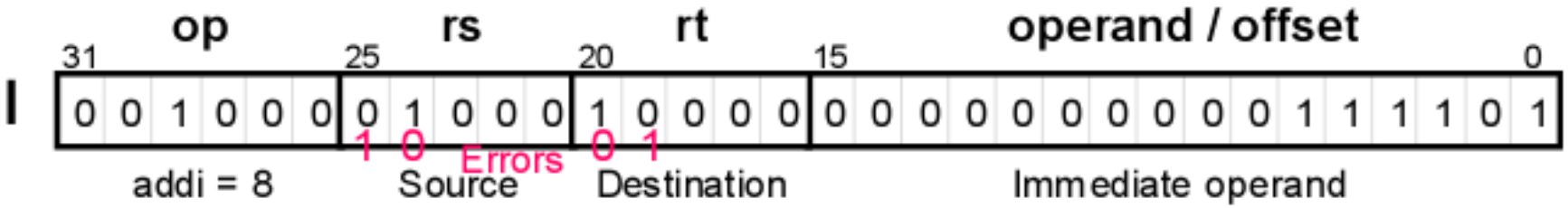


Figure 5.6 Instructions such as `addi` allow us to perform an arithmetic or logic operation for which one operand is a small constant.

5.4 Load and Store Instructions

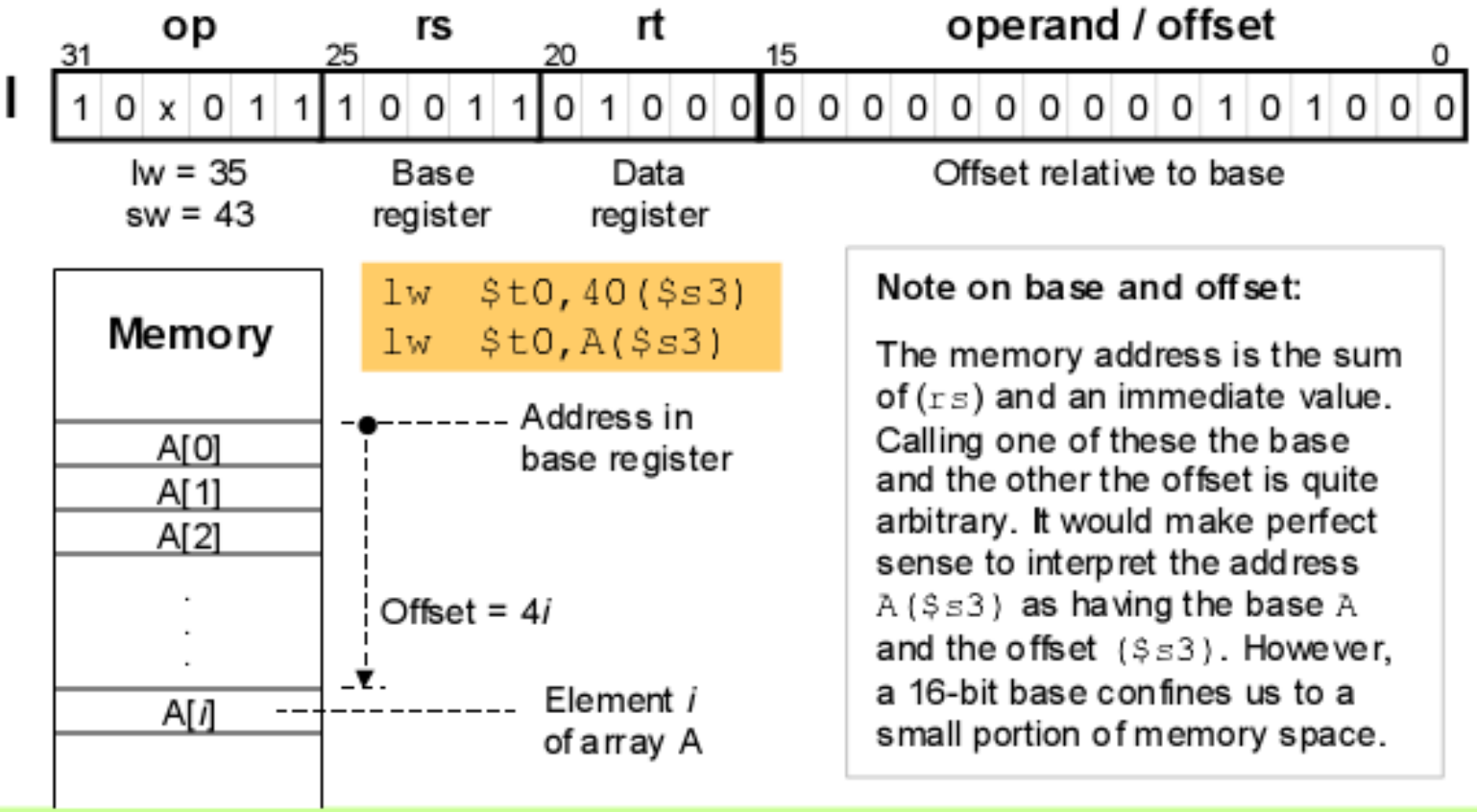


Figure 5.7 MiniMIPS `lw` and `sw` instructions and their memory addressing convention that allows for simple access to array elements via a base address and an offset (offset = 4*i* leads us to the *i*th word).

`lw`, `sw`, and `lui` Instructions

```
lw    $t0,40($s3)      # load mem[40+($s3)] in $t0
sw    $t0,A($s3)       # store ($t0) in mem[A+($s3)]
                       # "($s3)" means "content of $s3"
lui   $s0,61           # The immediate value 61 is
                       # loaded in upper half of $s0
                       # with lower 16b set to 0s
```

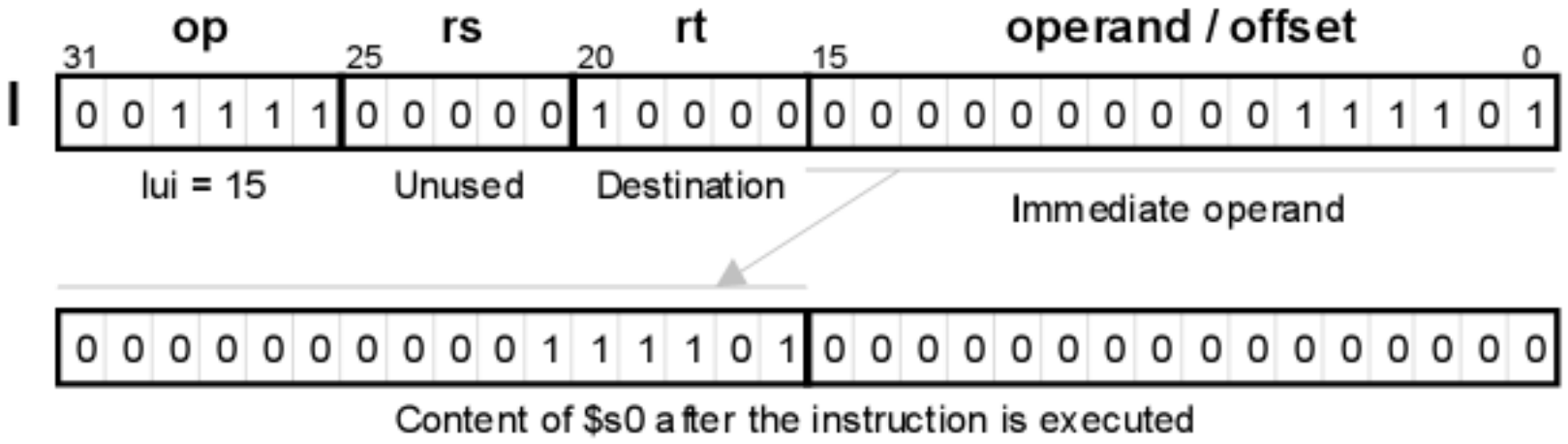


Figure 5.8 The `lui` instruction allows us to load an arbitrary 16-bit value into the upper half of a register while setting its lower half to 0s.

Initializing a Register

Example 5.2

Show how each of these bit patterns can be loaded into `$s0`:

```
0010 0001 0001 0000 0000 0000 0011 1101
1111 1111 1111 1111 1111 1111 1111 1111
```

Solution

The first bit pattern has the hex representation: `0x2110003d`

```
lui  $s0,0x2110      # put the upper half in $s0
ori  $s0,0x003d      # put the lower half in $s0
```

Same can be done, with immediate values changed to `0xffff` for the second bit pattern. But, the following is simpler and faster:

```
nor  $s0,$zero,$zero # because (0 v 0)' = 1
```

5.5 Jump and Branch Instructions

Unconditional jump and jump through register instructions

```
j   verify          # go to mem loc named "verify"
jr  $ra             # go to address that is in $ra;
                    # $ra may hold a return address
```

$ra is the symbolic name for reg. $31 (return address)

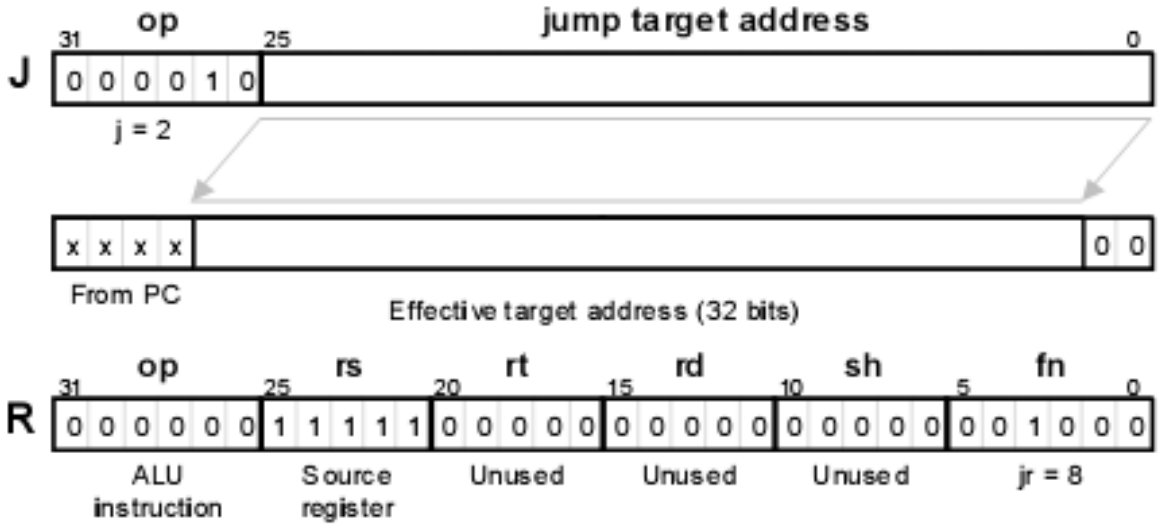


Figure 5.9 The jump instruction `j` of MiniMIPS is a J-type instruction which is shown along with how its effective target address is obtained. The jump register (`jr`) instruction is R-type, with its specified register often being `$ra`.

Conditional Branch Instructions

Conditional branches use PC-relative addressing

```
bltz $s1,L          # branch on ($s1)< 0
beq  $s1,$s2,L      # branch on ($s1)=($s2)
bne  $s1,$s2,L      # branch on ($s1)≠($s2)
```

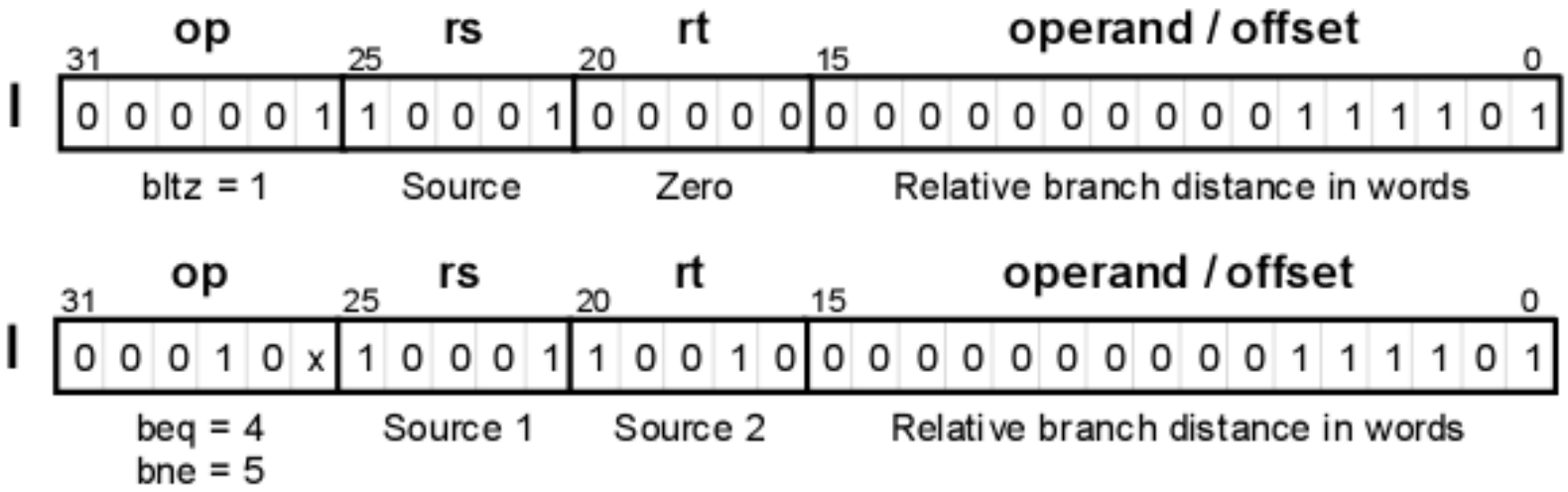


Figure 5.10 (part 1) Conditional branch instructions of MiniMIPS.

Comparison Instructions for Conditional Branching

```
slt   $s1,$s2,$s3     # if ($s2)<($s3), set $s1 to 1
                      # else set $s1 to 0;
                      # often followed by beq/bne
slti  $s1,$s2,61      # if ($s2)<61, set $s1 to 1
                      # else set $s1 to 0
```

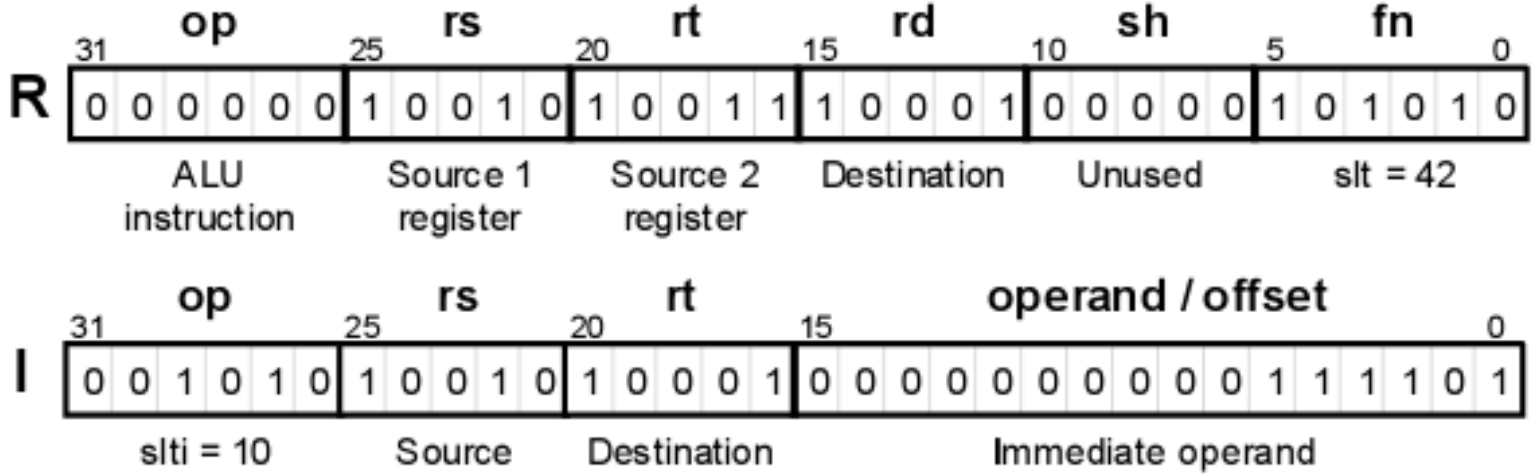


Figure 5.10 (part 2) Comparison instructions of MiniMIPS.

Examples for Conditional Branching

If the branch target is too far to be reachable with a 16-bit offset (rare occurrence), the assembler automatically replaces the branch instruction `beq $s0,$s1,L1` with:

```
      bne  $s1,$s2,L2      # skip jump if (s1)≠(s2)
      j    L1              # goto L1 if (s1)=(s2)
 L2: ...
```

Forming if-then constructs; e.g., `if (i == j) x = x + y`

```
      bne  $s1,$s2,endif   # branch on i≠j
      add  $t1,$t1,$t2     # execute the "then" part
endif: ...
```

If the condition were `(i < j)`, we would change the first line to:

```
      slt  $t0,$s1,$s2     # set $t0 to 1 if i<j
      beq  $t0,$0,endif    # branch if ($t0)=0;
                           # i.e., i not< j or i≥j
```

Compiling if-then-else Statements

Example 5.3

Show a sequence of MiniMIPS instructions corresponding to:

```
if (i<=j) x = x+1; z = 1; else y = y-1; z = 2*z
```

Solution

Similar to the "if-then" statement, but we need instructions for the "else" part and a way of skipping the "else" part after the "then" part.

```
        slt  $t0,$s2,$s1     # j<i? (inverse condition)
        bne  $t0,$zero,else  # if j<i goto else part
        addi $t1,$t1,1       # begin then part: x = x+1
        addi $t3,$zero,1     # z = 1
        j    endif           # skip the else part
else:   addi $t2,$t2,-1      # begin else part: y = y-1
        add  $t3,$t3,$t3     # z = z+z
endif:...
```

5.6 Addressing Modes

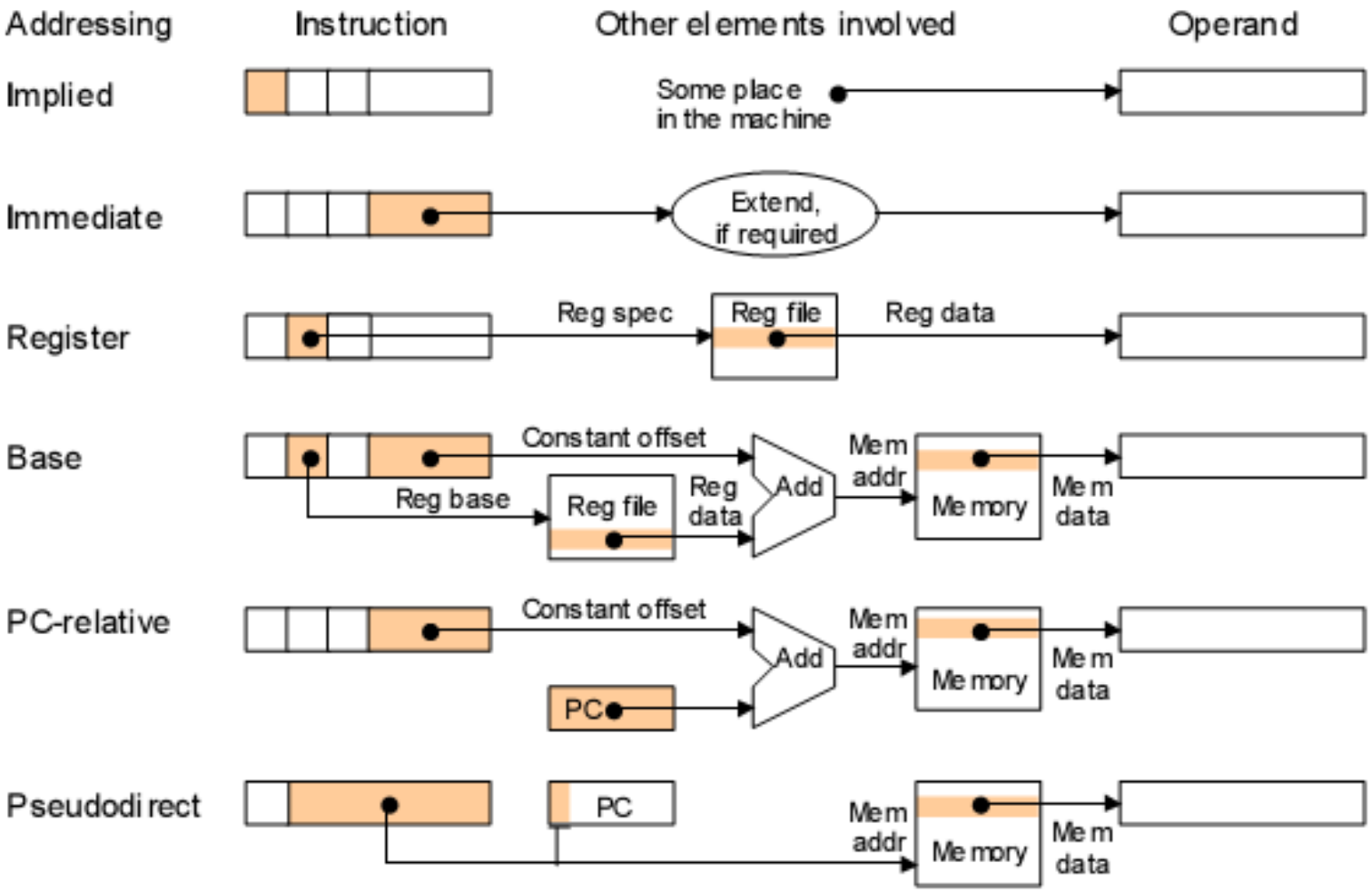


Figure 5.11 Schematic representation of addressing modes in MiniMIPS.

The 20 MiniMIPS Instructions Covered So Far

	Instruction	Usage	op	fn
Copy	Load upper immediate	`lui rt,imm`	15	
Arithmetic	Add	`add rd,rs,rt`	0	32
	Subtract	`sub rd,rs,rt`	0	34
	Set less than	`slt rd,rs,rt`	0	42
	Add immediate	`addi rt,rs,imm`	8	
	Set less than immediate	`slti rd,rs,imm`	10	
Logic	AND	`and rd,rs,rt`	0	36
	OR	`or rd,rs,rt`	0	37
	XOR	`xor rd,rs,rt`	0	38
	NOR	`nor rd,rs,rt`	0	39
	AND immediate	`andi rt,rs,imm`	12	
	OR immediate	`ori rt,rs,imm`	13	
	XOR immediate	`xori rt,rs,imm`	14	
Memory access	Load word	`lw rt,imm(rs)`	35	
	Store word	`sw rt,imm(rs)`	43	
Control transfer	Jump	`j L`	2	
	Jump register	`jr rs`	0	8
	Branch less than 0	`bltz rs,L`	1	
	Branch equal	`beq rs,rt,L`	4	
	Branch not equal	`bne rs,rt,L`	5	

Table 5.1

Khung chương trình hợp ngữ

Giống 8086, chương trình hợp ngữ cho MIPS bao gồm các thành phần

- Định hướng biên dịch
- Lệnh
- Giả lệnh

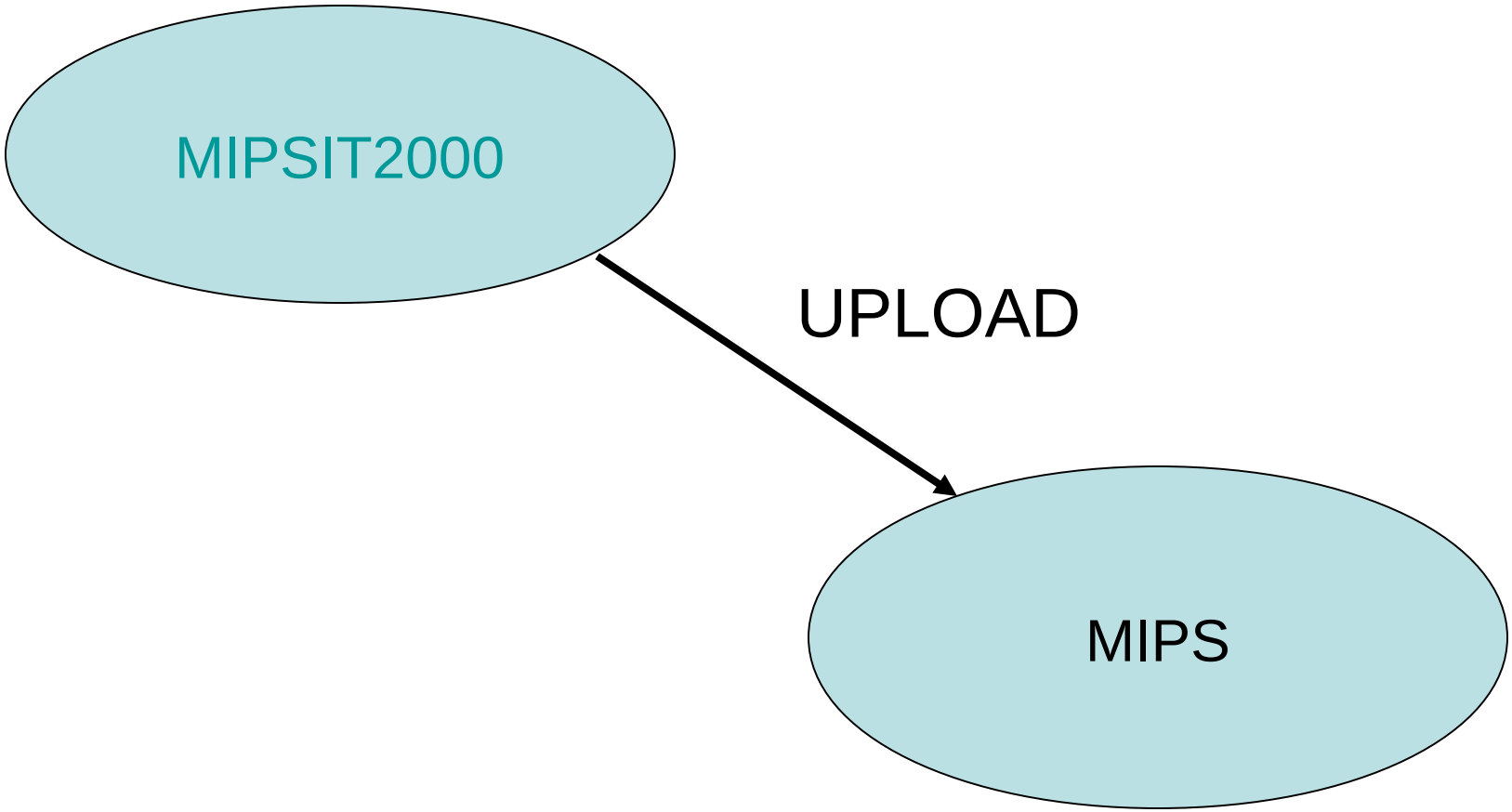
Khung chương trình hợp ngữ

```
#include <iregdef.h>
.data
#Khai báo biến
.text
.globl start
.ent start
start:
#Nội dung chương trình chính
.end start
.ent CTCon
CTCon:
#Nội dung chương trình con
.end CTCon
```

Chương trình ví dụ

```
#include <iregdef.h>
.data
test: .asciiz "Hello World"
.text
.set noreorder
.globl start
.ent start
start:
    la   a0,test        #load the address of test string to a0
    jal  printf         #print test tring to console
.end start
```

Sử dụng MIPSIT & MIPS



ftp://dce.hut.edu.vn/vinhtt/MIPS%20seminar.ppt