IT3280
THỰC HÀNH
KIẾN TRÚC MÁY TÍNH

Course administration

- Course: IT3280 2(0-4-0-4), Thực hành KTMT
- Lecturer: Phạm Ngọc Hưng
  - hungpn@soict.hust.edu.vn
  - B1-802, Department of Computer Engineering
- TA (2023.2):
  - Kiều Thái Thịnh
- Teaching Assistant (2022.2):
  - Đào Xuân Đạt
  - Hoàng Minh Ngọc

Laboratory Exercises

- **Lab 1.** Introduction to MIPS, MIPS Simulation (MARS)
- **Lab 2.** Instruction Set, Basic Instructions, Directives
- **Lab 3.** Load/ Store , Jump & Branch instructions
- **Lab 4.** Arithmetic and Logical operation
- **Lab 5.** Character string with SYSCALL function, and sorting
- **Lab 6.** Array and Pointer
- **Lab 7.** Procedure calls, stack and parameters
- **Lab 8-9.** Mini-Project
- **Lab 10.** Control Peripheral Devices via Simulator
- **Lab 11.** Interrupts & IO programming
- **Lab 12.** Cache Memory
- **Lab 13-15.** Final Project

LAB 1

Introduction to MIPS, MIPS Simulation (MARS)

1. Introduction to MIPS

- **MIPS** (originally an acronym for **Microprocessor without Interlocked Pipeline Stages**)
- MIPS is a RISC (Reduced Instruction Set Computer) instruction set architecture (ISA) developed by **MIPS Technologies** (formerly MIPS Computer Systems, Inc.)
- In 1981, a team led by John L. Hennessy at Stanford University started work on what would become the first MIPS processor.
- Multiple revisions of the MIPS instruction set exist, including MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32, and MIPS64.

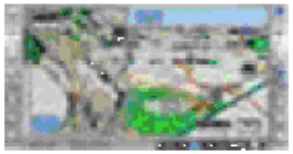
http://en.wikipedia.org/wiki/MIPS_architecture

Applications of MIPS processor

DVD players

Pioneer
DVR-57-H

Kenwood
HDV-810 Car Navigation System



Portable Devices

Canon
EOS 10D Digital

JVC
GR-HD1

Networking

3COM
3102 Business IP Phone

3COM
3106 Cordless Phone

Apple
Airport Extreme WLAN Access Points

Hewlet Packard
Color Laser Jet 2500 Laser Printer

Applications of MIPS processor

Sony Playstation Portable

CPU
Type:MIPS R4000 32bit Core
Clockspeed:333 MHz

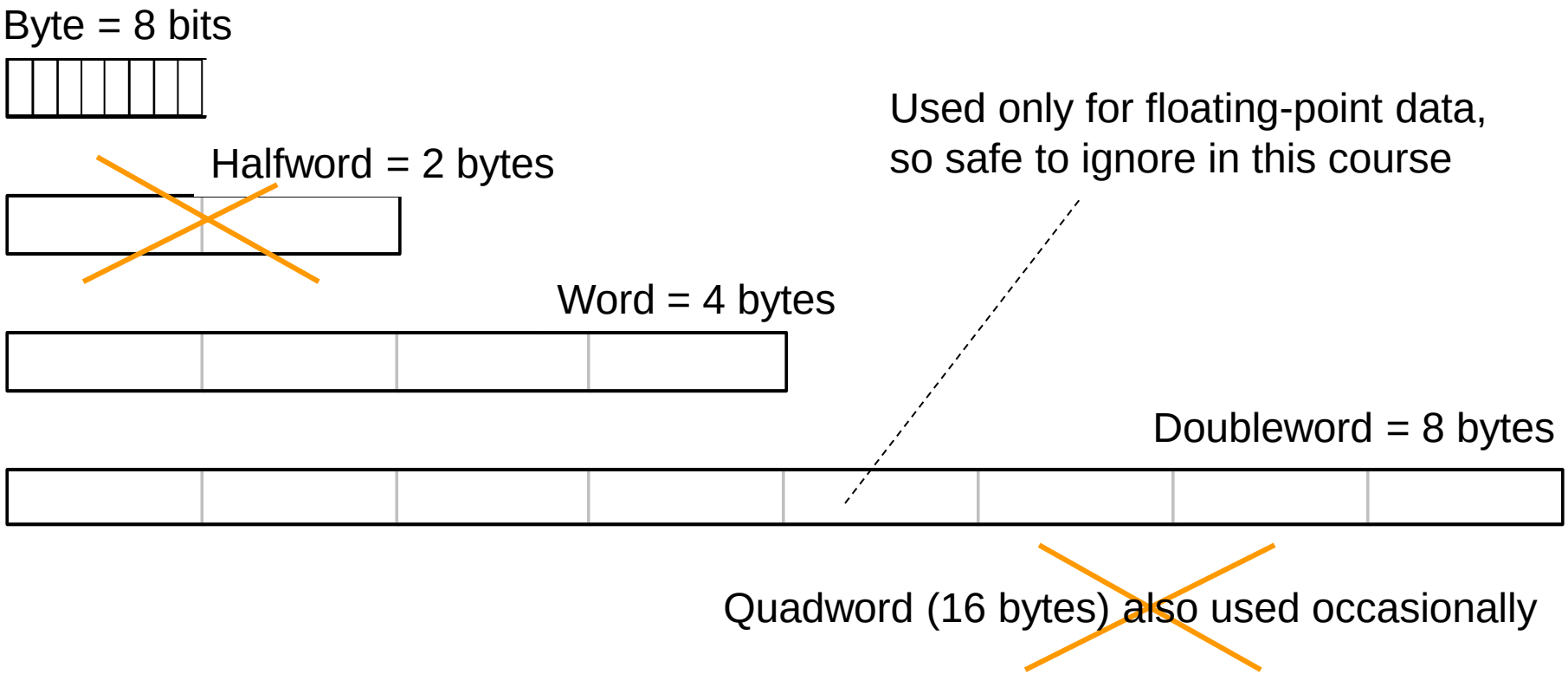
Sony Playstation PSX

CPU
Type:LSI/MIPS R3000A
Architecture:32 Bit
Clockspeed:33,8 MHz

2. MIPS Programming Model

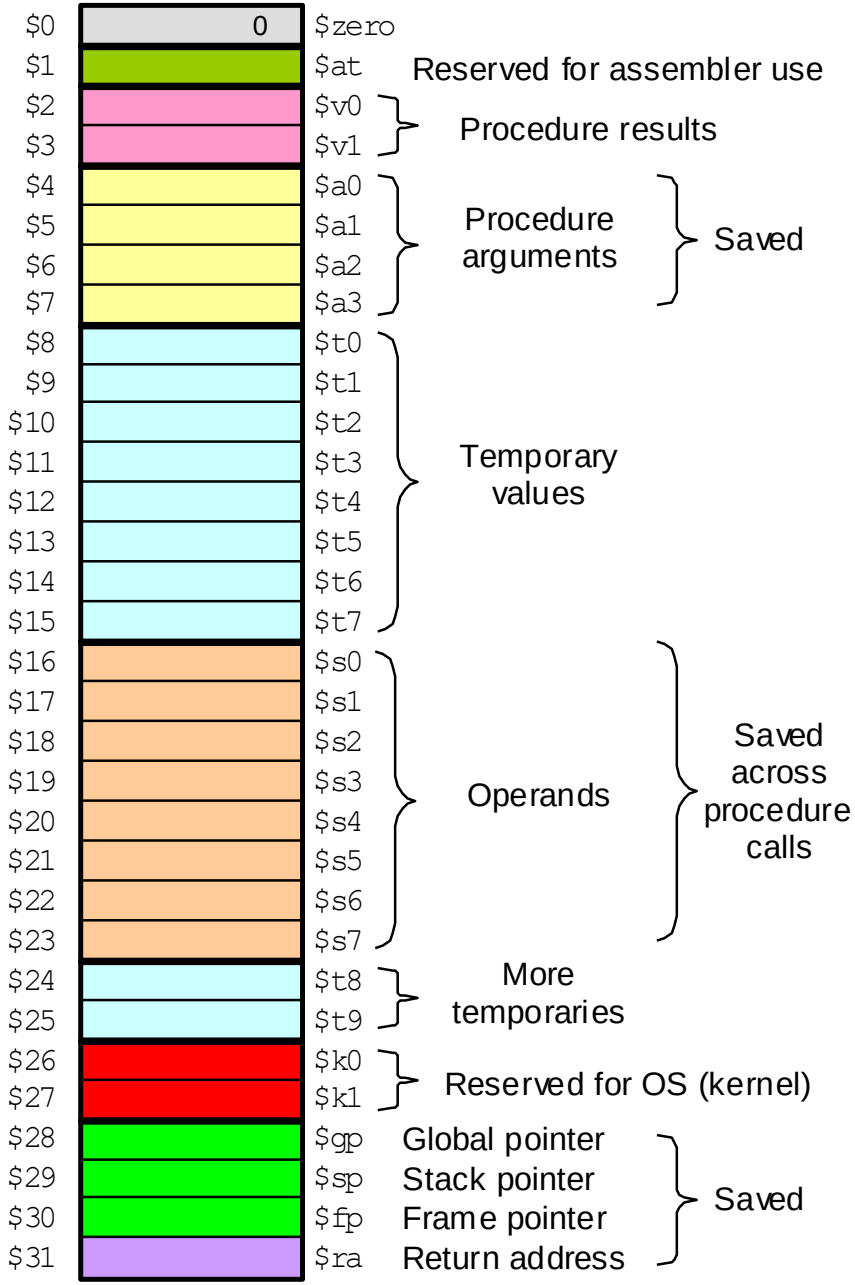
- Data Types
- Registers
- Instruction Formats
- MIPS Instruction

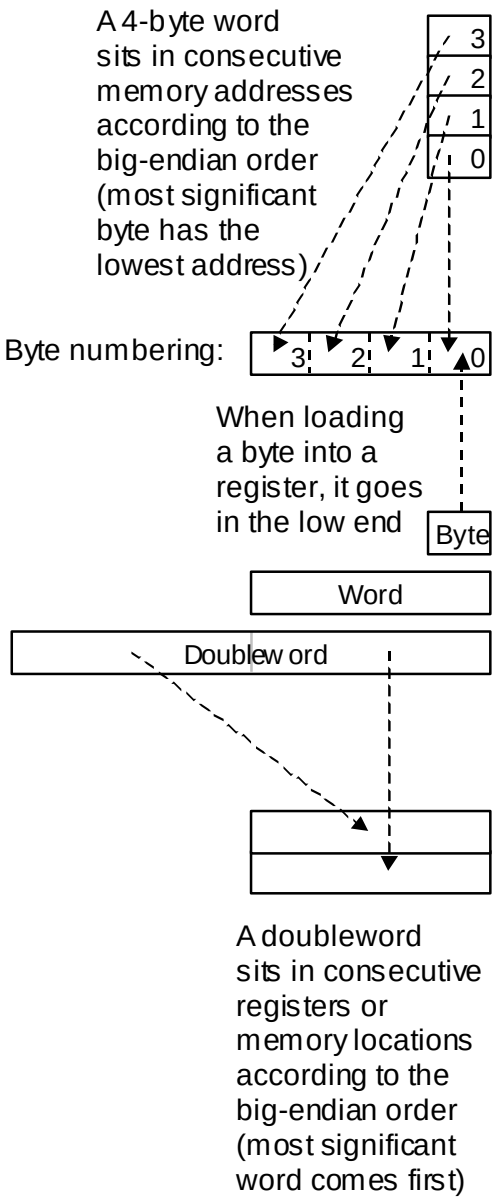
Data Types



MiniMIPS registers hold 32-bit (4-byte) words. Other common data sizes include byte, halfword, and doubleword.

Registers





Registers and data sizes in MiniMIPS.

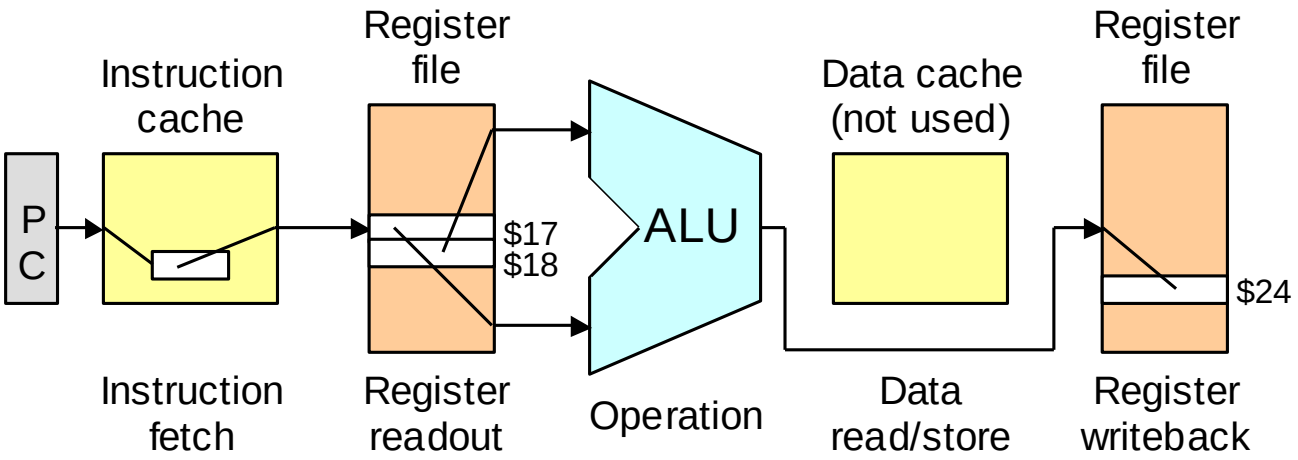
Instruction Formats

High-level language statement: `a = b + c`

Assembly language instruction: `add $t8, $s2, $s1`

Machine language instruction:

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



A typical instruction for MiniMIPS and steps in its execution.

Overview of MIPS instruction set

MIPS assembly language

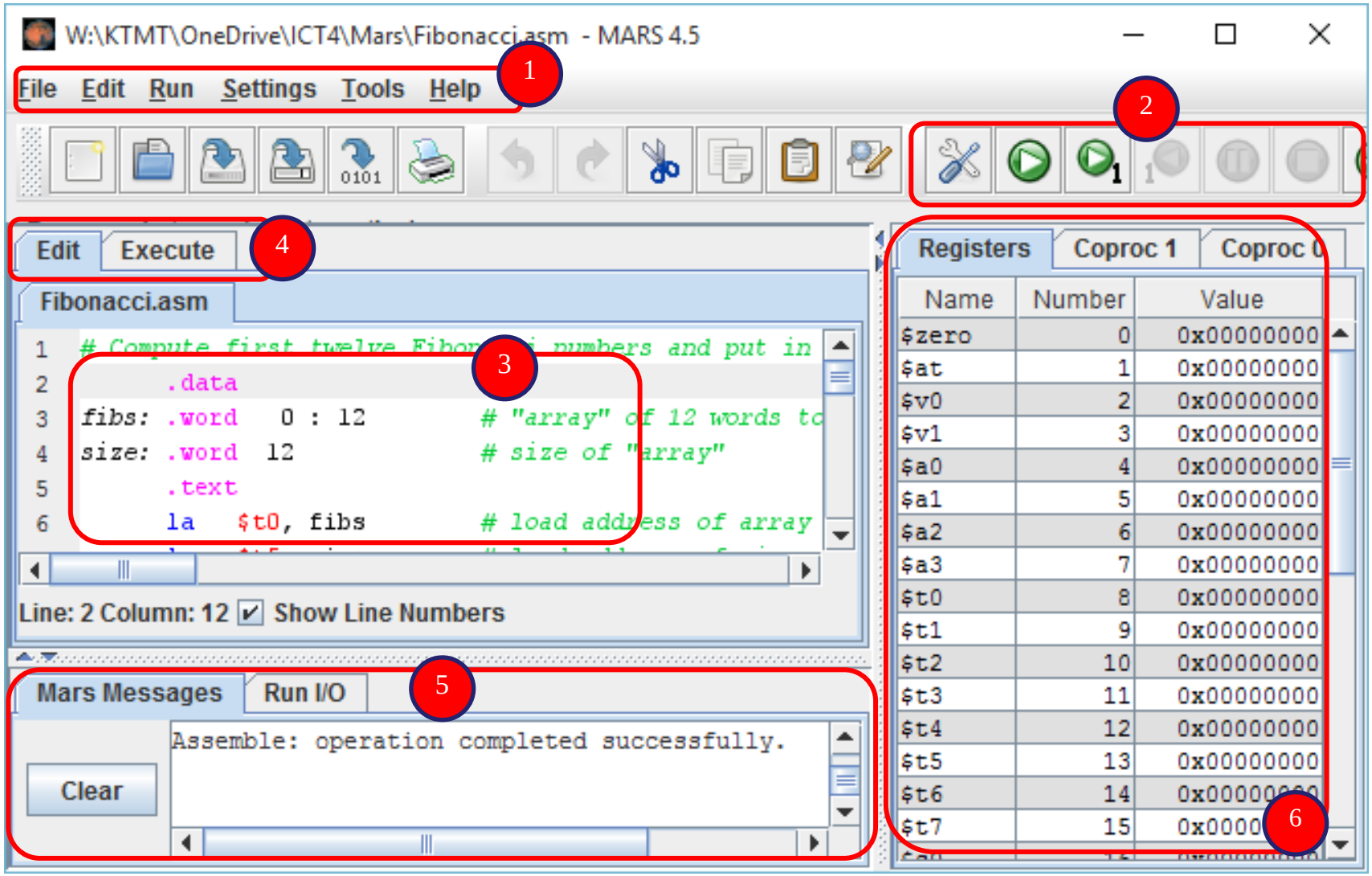
Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

3. MIPS Simulation - Mars

- MARS – MIPS Simulation
- MIPS assembly program

MIPS simulation

- MARS IDE



MIPS assembly program: HelloWorld

```
.data                    # Vung du lieu, chua cac khai bao bien
x:        .word    0x01020304    # bien x, khoi tao gia tri
message:  .asciiz  "Bo mon Ky thuat May tinh"
.text                  # Vung lenh, chua cac lenh hop ngu
   la  $a0, message  #Dua dia chi bien mesage vao thanh ghi a0
   li   $v0, 4             #Gan thanh ghi $v0 = 4
   syscall                 #Goi ham so v0, ham so 4, la ham print

   addi $t1,$zero,2        #Thanh ghi $t1 = 2
   addi $t2,$zero,3        #Thanh ghi $t2 = 3
   add  $t0, $t1, $t2      #Thanh ghi $t0 = $t1 + $t2
```

MIPS assembly program: HelloWorld

Edit | Execute

mips1.asm*

```
.data                          # Vung du lieu, chua cac khai bao bien
x: .word    0x01020304         # bien x, khoi tao gia tri
message:  .asciiz  "Bo mon Ky thuat May tinh"

.text                          # Vung lenh, chua cac lenh hop ngu
   la   $a0, message           #Dua dia chi bien mesage vao thanh ghi a0
   li   $v0, 4                 #Gan thanh ghi v0 = 4
   syscall                     #Goi ham so v0, ham so 4, la ham print

   addi $t1,$zero,2            #Thanh ghi t1 = 2
   addi $t2,$zero,3            #Thanh ghi t2 = 3
   add  $t0, $t1, $t2          #Thanh ghi t- = t1 + t2
```

MIPS assembly program: HelloWorld

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	6: la $a0, message #Dua d...
	0x00400004	0x34240004	ori $4,$1,0x00000004	
	0x00400008	0x24020004	addiu $2,$0,0x00000004	7: li $v0, 4 #Gan t...
	0x0040000c	0x0000000c	syscall	8: syscall #Goi h...
	0x00400010	0x20090002	addi $9,$0,0x00000002	10: addi $t1,$zero,2 #Thanh...
	0x00400014	0x200a0003	addi $10,$0,0x00000003	11: addi $t2,$zero,3 #Thanh...
	0x00400018	0x012a4020	add $8,$9,$10	12: add $t0, $t1, $t2 #Thanh...

Labels

Label	Address
mips1.asm	
x	0x10010000
message	0x10010004

☑ Data ☑ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	. . . .	m o B	K n o	h t y	t a u	y a M	h n i t	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☑ ASCII

MIPS assembly program: HelloWorld

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	6: la $a0, message #Dua d...
	0x00400004	0x34240004	ori $4,$1,0x00000004	
	0x00400008	0x24020004	addiu $2,$0,0x00000004	7: li $v0, 4 #Gan t...
	0x0040000c	0x0000000c	syscall	8: syscall #Goi h...
	0x00400010	0x20090002	addi $9,$0,0x00000002	10: addi $t1,$zero,2 #Thanh...
	0x00400014	0x200a0003	addi $10,$0,0x00000003	11: addi $t2,$zero,3 #Thanh...
	0x00400018	0x012a4020	add $8,$9,$10	12: add $t0, $t1, $t2 #Thanh...

Labels

Label	Address
mips1.asm	
x	0x10010000
message	0x10010004

☑ Data ☑ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	. . . .	m o B	K n o	h t y	t a u	y a M	h n i t	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☑ ASCII

MIPS assembly program: HelloWorld

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	6: la $a0, message #Dua d...
	0x00400004	0x34240004	ori $4,$1,0x00000004	
	0x00400008	0x24020004	addiu $2,$0,0x00000004	7: li $v0, 4 #Gan t...
	0x0040000c	0x0000000c	syscall	8: syscall #Goi h...
	0x00400010	0x20090002	addi $9,$0,0x00000002	10: addi $t1,$zero,2 #Thanh...
	0x00400014	0x200a0003	addi $10,$0,0x00000003	11: addi $t2,$zero,3 #Thanh...
	0x00400018	0x012a4020	add $8,$9,$10	12: add $t0, $t1, $t2 #Thanh...

Labels

Label	Address
mips1.asm	
x	0x10010000
message	0x10010004

☑ Data ☑ Text

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	. . . .	m o B	k n o	h t y	t a u	y a M	h n i t	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☑ ASCII

MIPS assembly program: HelloWorld

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	. . . .	m o B	K n o	h t y	t a u	y a M	h n i t	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

```
.data                          # Vung du lieu, chua cac khai bao bien
x: .word    0x01020304         # bien x, khoi tao gia tri
message:    .asciiz  "Bo mon Ky thuat May tinh"
```

Data Se

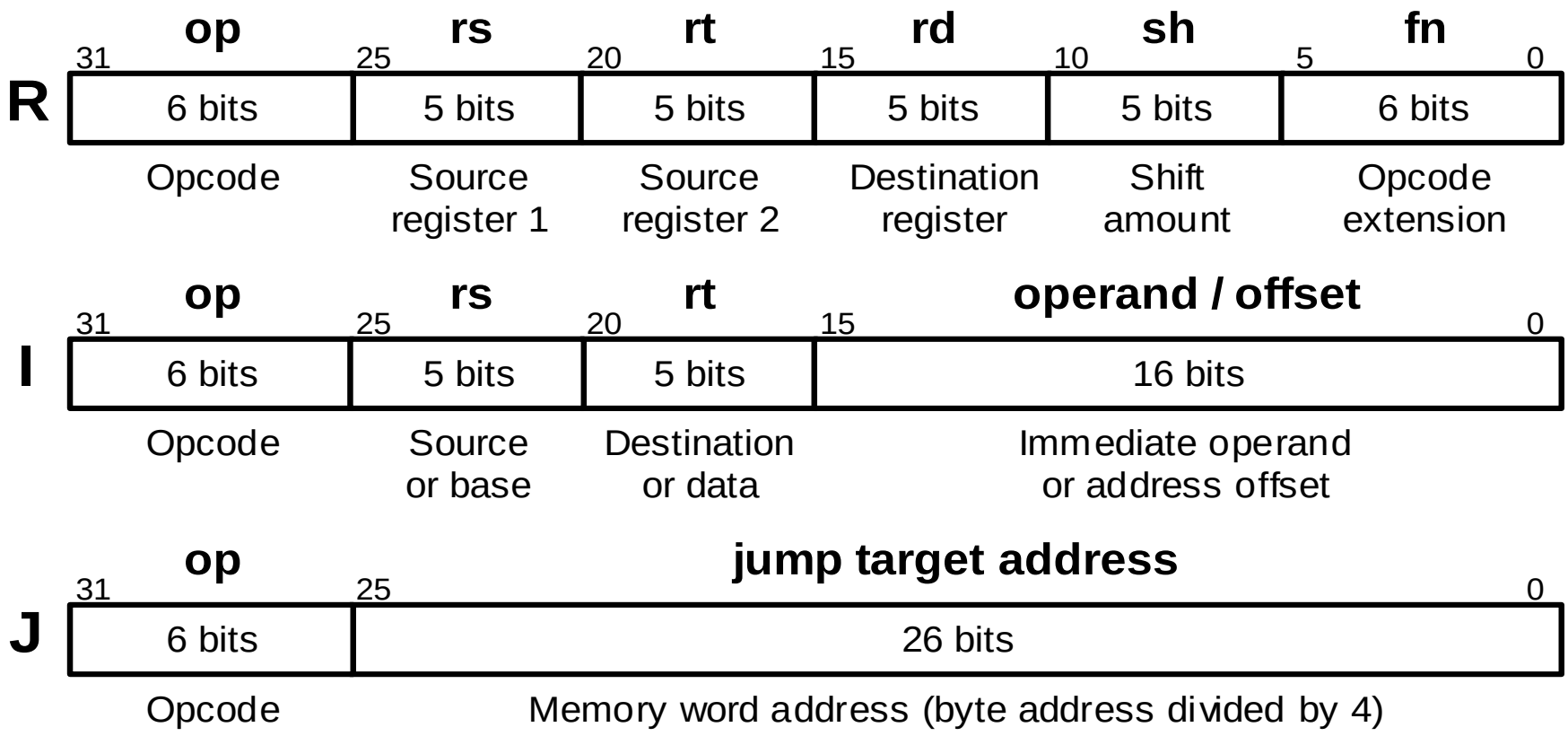
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

LAB 2

Instruction Set, Basic Instructions, Directives

MIPS Instruction Formats



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

Simple Arithmetic/Logic Instructions

Lệnh số học: add, sub, … Lệnh logic: and, or, nor, …

```
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))′
```

	op	rs	rt	rd	sh	fn
	31	25	20	15	10	5 0
R	0 0 0 0 0 0	1 0 0 0 0	1 0 0 0 1	0 1 0 0 0	0 0 0 0 0	1 0 0 0 x 0
	ALU instruction	Source register 1	Source register 2	Destination register	Unused	add = 32 sub = 34

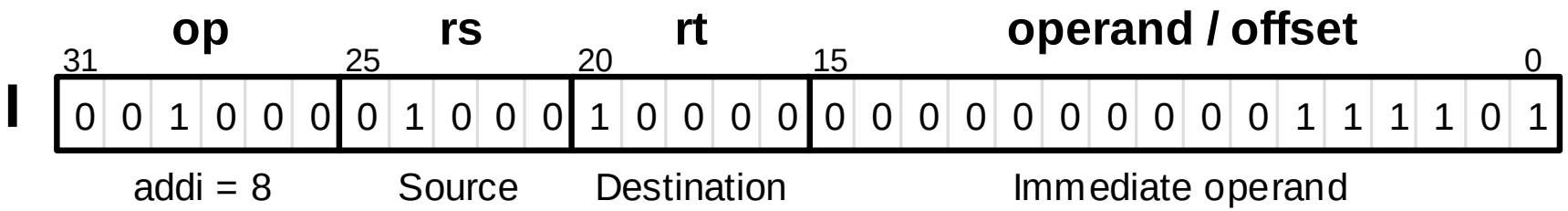
The arithmetic instructions `add` and `sub` have a format that is common to all two-operand ALU instructions. For these, the `fn` field specifies the arithmetic/logic operation to be performed.

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

For arithmetic instructions, the immediate operand is sign-extended



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

Initializing a register

- Khởi tạo giá trị cho thanh ghi

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
```

MARS Help

MARS 4.5 Help

MIPS | MARS | License | Bugs/Comments | Acknowledgements | Instruction Set Song

Operand Key for Example Instructions

`label, target`	any textual label
`$t1, $t2, $t3`	any integer register
`$f2, $f4, $f6`	*even-numbered* floating point register
`$f0, $f1, $f3`	*any* floating point register

Basic Instructions | Extended (pseudo) Instructions | Directives | Syscalls | Exceptions | Macros

abs.d $f2,$f4	Floating point absolute value double precision : Set $f2 to absolute value of $f4, d
abs.s $f0,$f1	Floating point absolute value single precision : Set $f0 to absolute value of $f1, s
add $t1,$t2,$t3	Addition with overflow : set $t1 to ($t2 plus $t3)
add.d $f2,$f4,$f6	Floating point addition double precision : Set $f2 to double-precision floating poin
add.s $f0,$f1,$f3	Floating point addition single precision : Set $f0 to single-precision floating poin
addi $t1,$t2,-100	Addition immediate with overflow : set $t1 to ($t2 plus signed 16-bit immediate)
addiu $t1,$t2,-100	Addition immediate unsigned without overflow : set $t1 to ($t2 plus signed 16-bit im
addu $t1,$t2,$t3	Addition unsigned without overflow : set $t1 to ($t2 plus $t3), no overflow
and $t1,$t2,$t3	Bitwise AND : Set $t1 to bitwise AND of $t2 and $t3
andi $t1,$t2,100	Bitwise AND immediate : Set $t1 to bitwise AND of $t2 and zero-extended 16-bit immed
bc1f 1,label	Branch if specified FP condition flag false (BC1F, not BCLF) : If Coprocessor 1 cond
bc1f label	Branch if FP condition flag 0 false (BC1F, not BCLF) : If Coprocessor 1 condition fl
bc1t 1,label	Branch if specified FP condition flag true (BC1T, not BCLT) : If Coprocessor 1 condi
bc1t label	Branch if FP condition flag 0 true (BC1T, not BCLT) : If Coprocessor 1 condition fla
beq $t1,$t2,label	Branch if equal : Branch to statement at label's address if $t1 and $t2 are equal
bgez $t1,label	Branch if greater than or equal to zero : Branch to statement at label's address if
bgezal $t1,label	Branch if greater then or equal to zero and link : If $t1 is greater than or equal t
bgtz $t1,label	Branch if greater than zero : Branch to statement at label's address if $t1 is great
blez $t1,label	Branch if less than or equal to zero : Branch to statement at label's address if $t1

Close

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

Addressing Mode

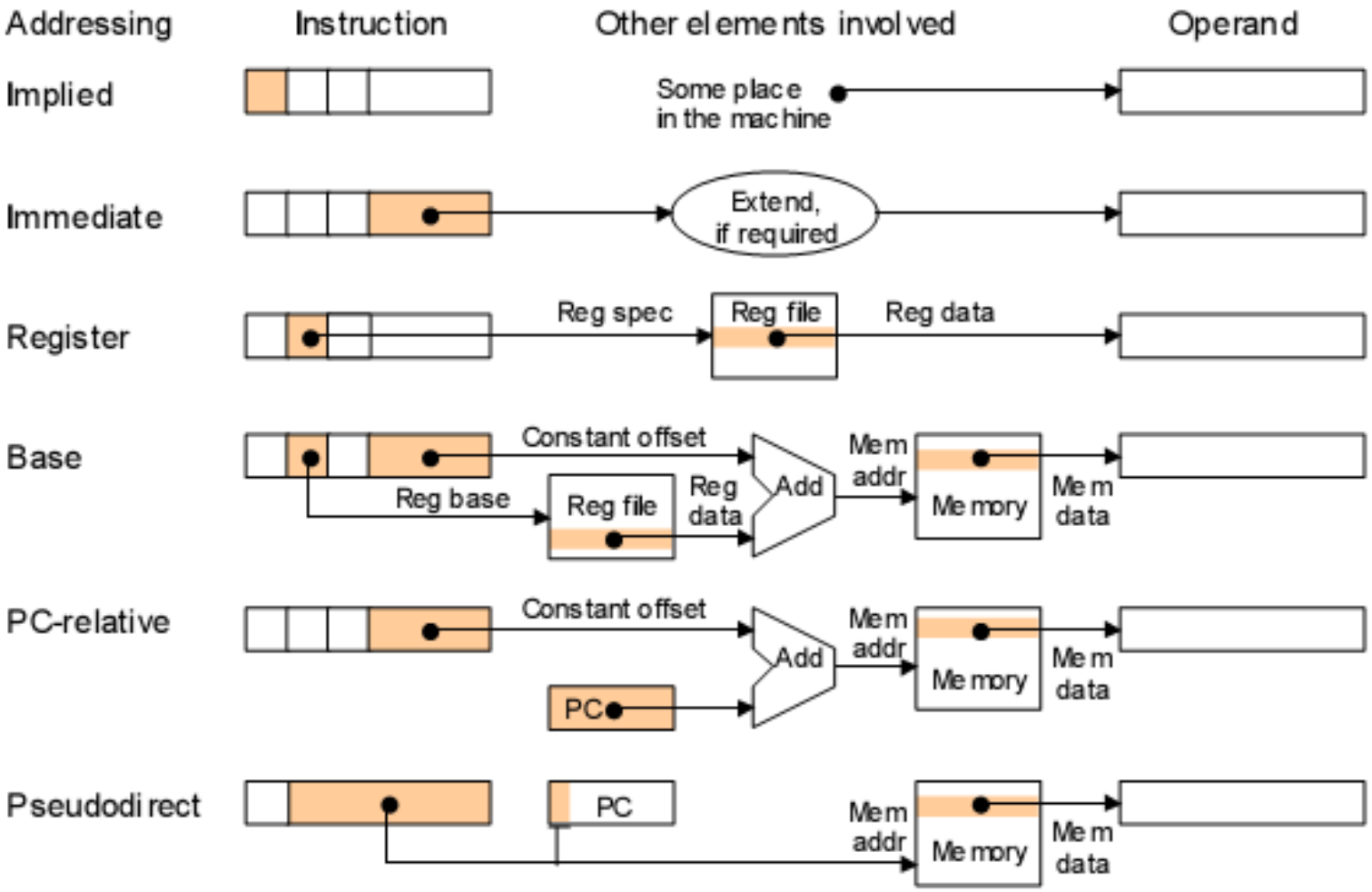
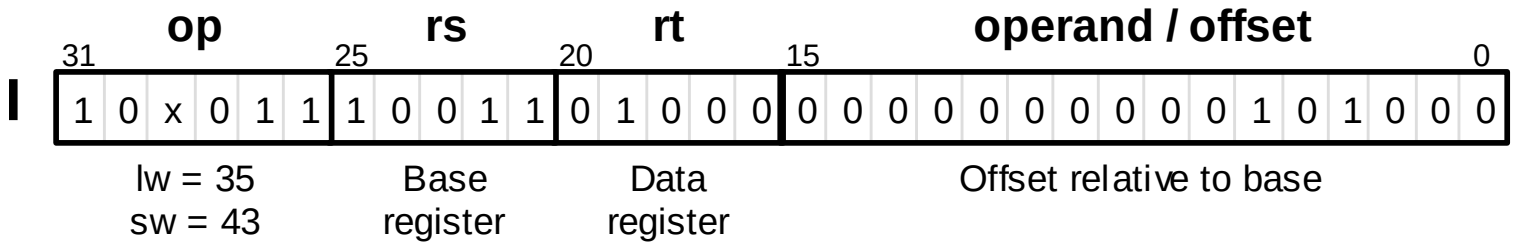


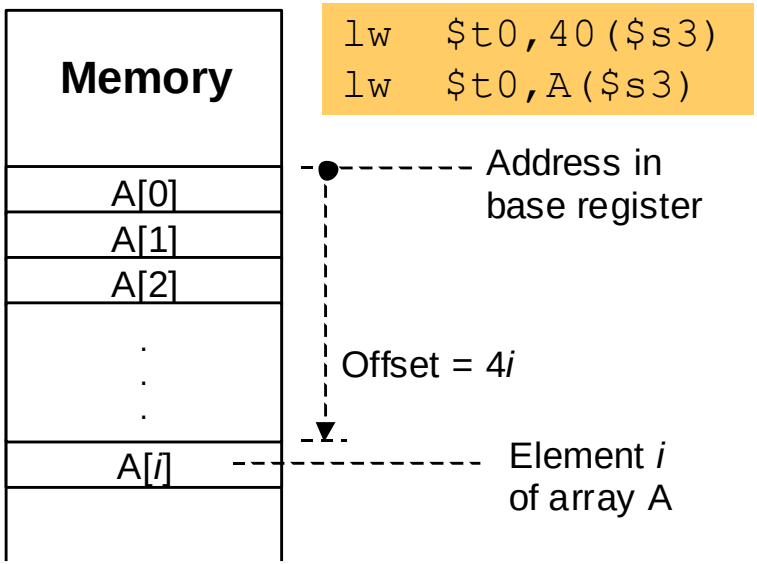
Figure 5.11 Schematic representation of addressing modes in MiniMIPS.

LAB 3

Load/Store, Jump and Branch Instructions

Load and Store Instructions





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.

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
```

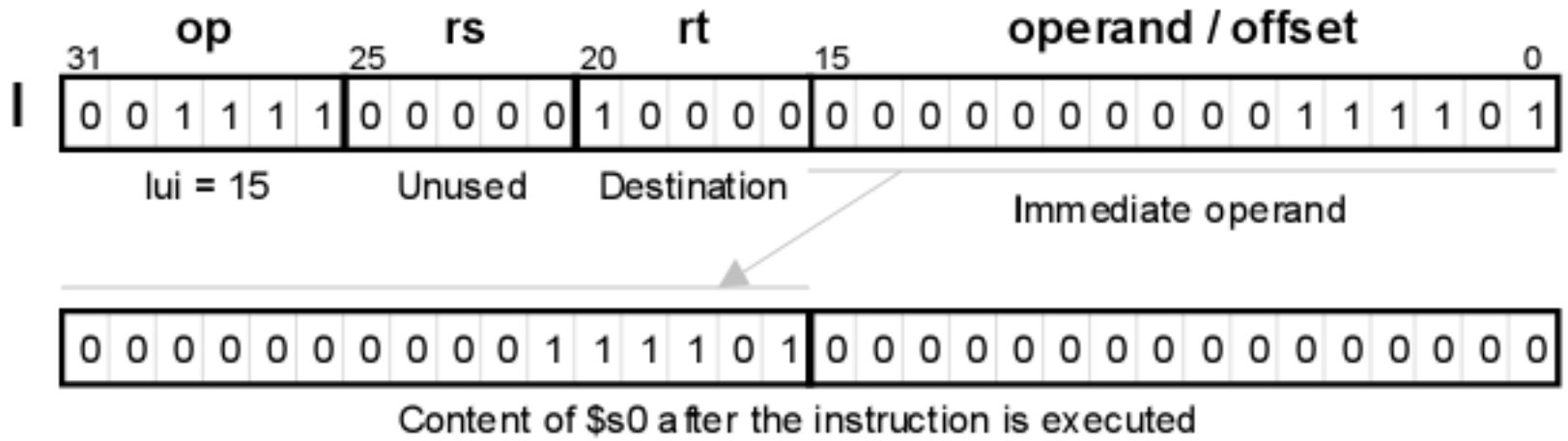


Figure 5.8 The `lui` instruction allows us to load an arbitrary 16-bit value into the upper half of a register while setting its lower half to 0s.

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)

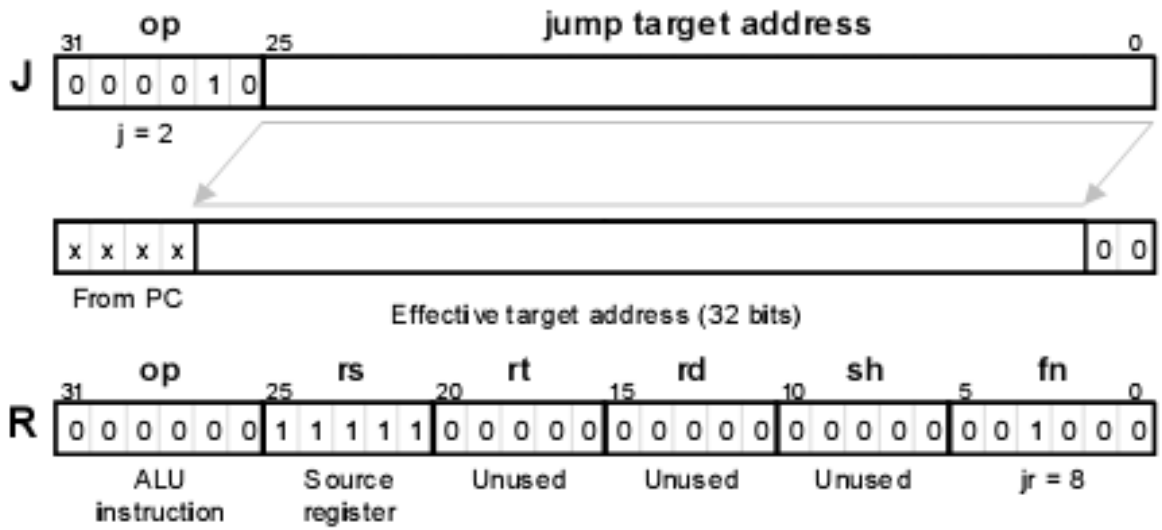


Figure 5.9 The jump instruction `j` of MiniMIPS is a J-type instruction which is shown along with how its effective target address is obtained. The jump register (`jr`) instruction is R-type, with its specified register often being `$ra`.

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)
```

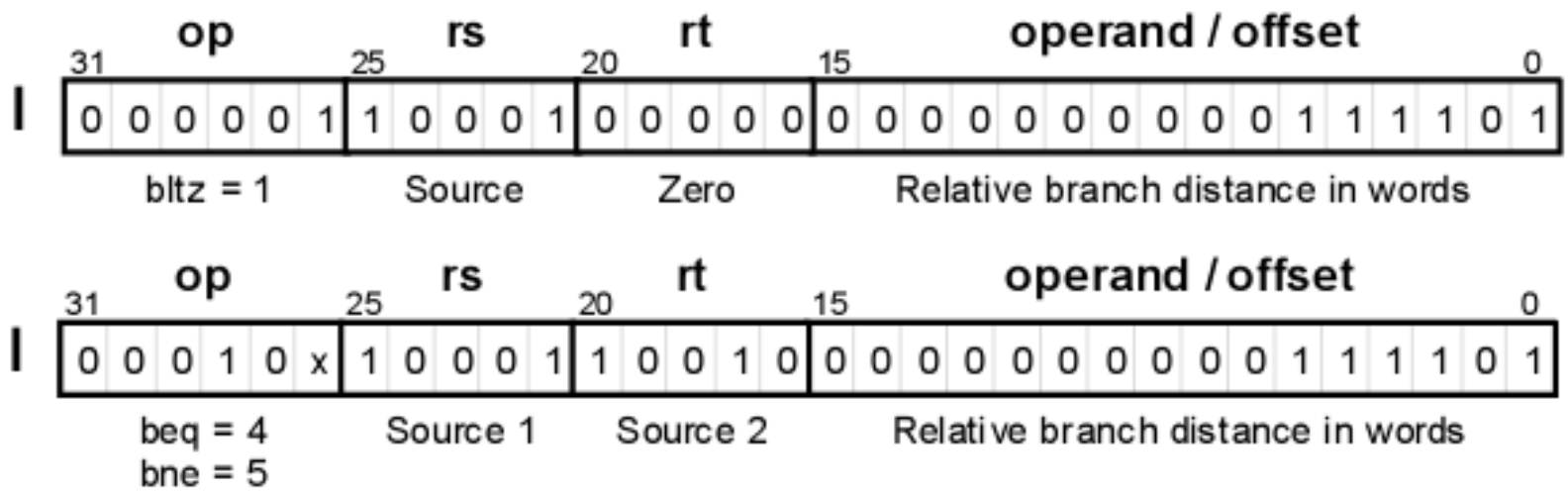


Figure 5.10 (part 1) Conditional branch instructions of MiniMIPS.

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
```

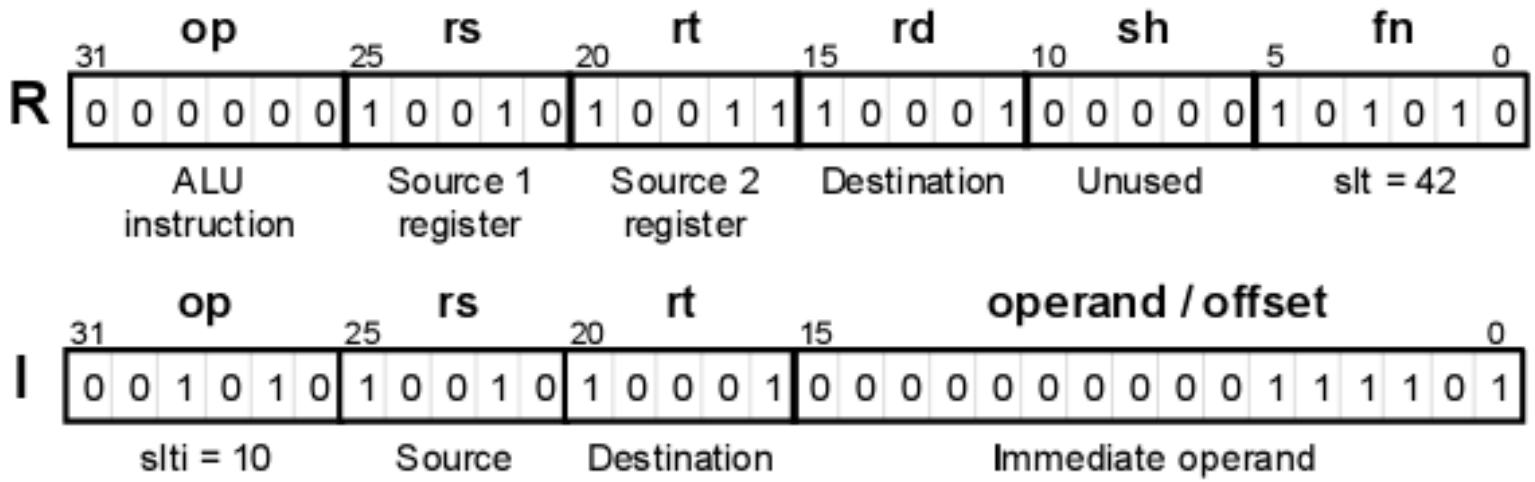


Figure 5.10 (part 2) Comparison instructions of MiniMIPS.

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 $s1,$s2,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:...
```

`while` Statements

Example

The simple while loop: `while (A[i]==k) i=i+1;`
Assuming that: `i, A, k` are stored in `$s1,$s2,$s3`

Solution

```
loop: add   $t1,$s1,$s1  # t1 = 4*i
      add   $t1,$t1,$t1  #
      add   $t1,$t1,$s2  # t1 = A + 4*i
      lw    $t0,0($t1)          # t0 = A[i]
      bne   $t0,$s3,endwhl      #
      addi $s1,$s1,1            #
      j     loop                #
endwhl: ...                     #
```

`switch` Statements

Example

The simple switch

```
switch(test) {
   case 0:
        a=a+1; break;
   case 1:
        a=a-1; break;
   case 2:
        b=2*b; break;
   default:
}
```

Assuming that: `test,a,b` are stored in `$s1,$s2,$s3`

```
        beq     s1,t0,case_0
        beq     s1,t1,case_1
        beq     s1,t2,case_2
        b       default
case_0:
        addi    s2,s2,1         #a=a+1
        b       continue
case_1:
        sub     s2,s2,t1        #a=a-1
        b       continue
case_2:
        add     s3,s3,s3        #b=2*b
        b       continue
default:
continue:
```

Pseudoinstructions

- Pseudoinstructions means “fake instruction”
- Pseudoinstructions do not correspond to real MIPS instructions
- The assembler, that converts assembly language programs to machine code, would then translate pseudoinstructions to real instructions, usually requiring at least one on more instructions.
- Example:

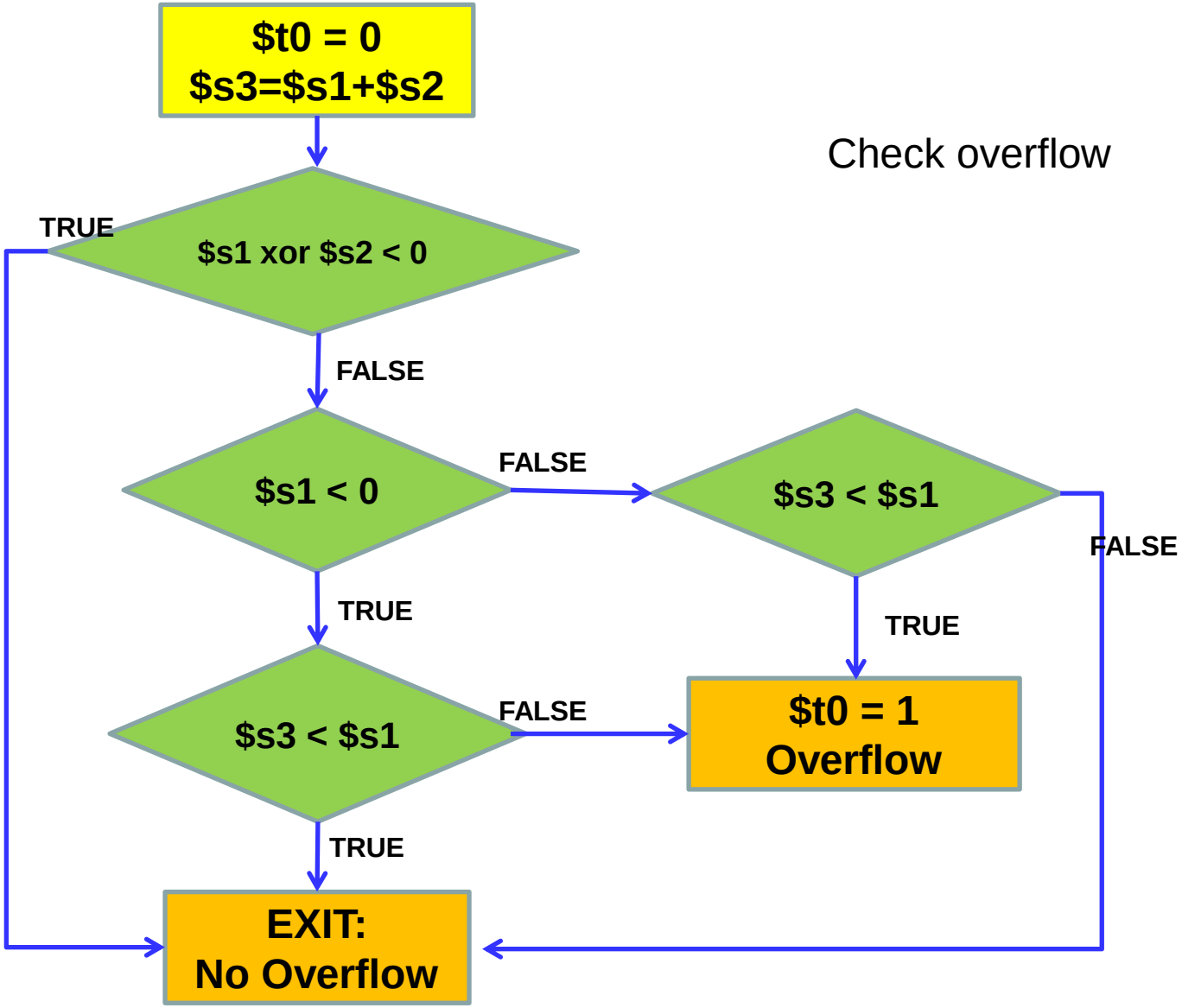
  mov $rt, $rs #Copy contents of register **s** to register **t**, R[t] = R[s]

  => real instruction: **addi $rt, $rs, 0**

LAB 4
Arithmetic and Logical Operations

Lab 4. Arithmetic & Logical Operation

Check overflow



Lab 4. Arithmetic & Logical Operation

- Bit mask in logical operation

```
li   s0,0x0563                    #load test value for these function
```

s0 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 1

```
andi     t0,s0,0xff                    #Extract the LSB of s0
```

MASK 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

t0 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 1

Lab 4. Arithmetic & Logical Operation

- Bit mask in logical operation

```
li   s0,0x0563                    #load test value for these function
```

s0 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 1

```
andi     t1,s0,0x0400                    #Extract  ? of s0
```

MASK 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

t1 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

LAB 5

Character string with SYSCALL function, and sorting

Lab 5. Character string

- strcpy

Địa chỉ xâu y:
a1 = 800203c0

Registers

```
r0/zero=00000000  r1/at  =00000000  r2/v0  =00000000  r3/v1  =0000
r4/a0  =800203d5  r5/a1  =800203c0  r6/a2  =00000000  r7/a3  =0000
r8/t0  =00000000  r9/t1  =800203c6  r10/t2 =00000061  r11/t3 =8002
r12/t4 =00000000  r13/t5 =00000000  r14/t6 =00000000  r15/t7 =0000
r16/s0 =00000006  r17/s1 =00000000  r18/s2 =00000000  r19/s3 =0000
r20/s4 =00000000  r21/s5 =00000000  r22/s6 =00000000  r23/s7 =0000
r24/t8 =00000000  r25/t9 =00000000  r26/k0 =00000000  r27/k1 =0000
r28/gp =00000000  r29/sp =800bbff4  r30/fp =800bc000  r31/ra =8002

pc     =8002005c  hi    =00000000  mdlo   =00000000  conf   =0000
bad va =00000000  status =00400000  cause  =00000000  epc    =0000
```

```
800203AC  03 E0 00 08  _init_sbrk()   ....
800203B0  00 00 00 00                 ....
800203B4  03 E0 00 08  _init_file()   ....
800203B8  00 00 00 00                 ....
800203BC  00 00 00 00                 ....
800203C0  63 6F 70 79  y:             copy
800203C4  20 78 61 75                  xau
800203C8  20 79 20 64                  y d
800203CC  65 6E 20 78                 en x
800203D0  61 75 20 78                 au x
800203D4  00 63 6F 70  x:             .cop
800203D8  79 20 78 61                 y xa
800203DC  00 00 00 00                 ....
800203E0  00 00 00 00                 ....
800203E4  00 00 00 00                 ....
800203E8  00 00 00 00                 ....
800203EC  00 00 00 00                 ....
800203F0  00 00 00 00                 ....
800203F4  00 00 00 00                 ....
800203F8  00 00 00 00                 ....
800203FC  00 00 00 00                 ....
80020400  00 00 00 00                 ....
80020404  00 00 00 00                 ....
80020408  00 00 00 00                 ....
8002040C  00 00 00 00                 ....
80020410  00 00 00 00                 ....
80020414  00 00 00 00                 ....
NOP
```

s0=6, x[6]=y[6]
Ký tự ‘a’

Địa chỉ xâu x:
a0 = 800203d5

```
L1:
add     t1,s0,a1      #address of y[i] in t1
lb      t2,0(t1)      #t2=y[i]
add     t3,s0,a0      #address of x[i] in t3
sb      t2,0(t3)      #x[i]=y[i]
beq     t2,zero,L2    #if y[i]==0, go to L2
nop
addi    s0,s0,1       #i=i+1
j       L1            #go to L1
nop
L2:
```

s0=6, x[6]=y[6]
Ký tự ‘a’

Gõ trực tiếp ngăn nhớ giá trị địa chỉ muốn xem. ví dụ: 800203c0 (xâu y)

LAB 6

Array and Pointer

LAB 7

Procedure calls, stack and parameters

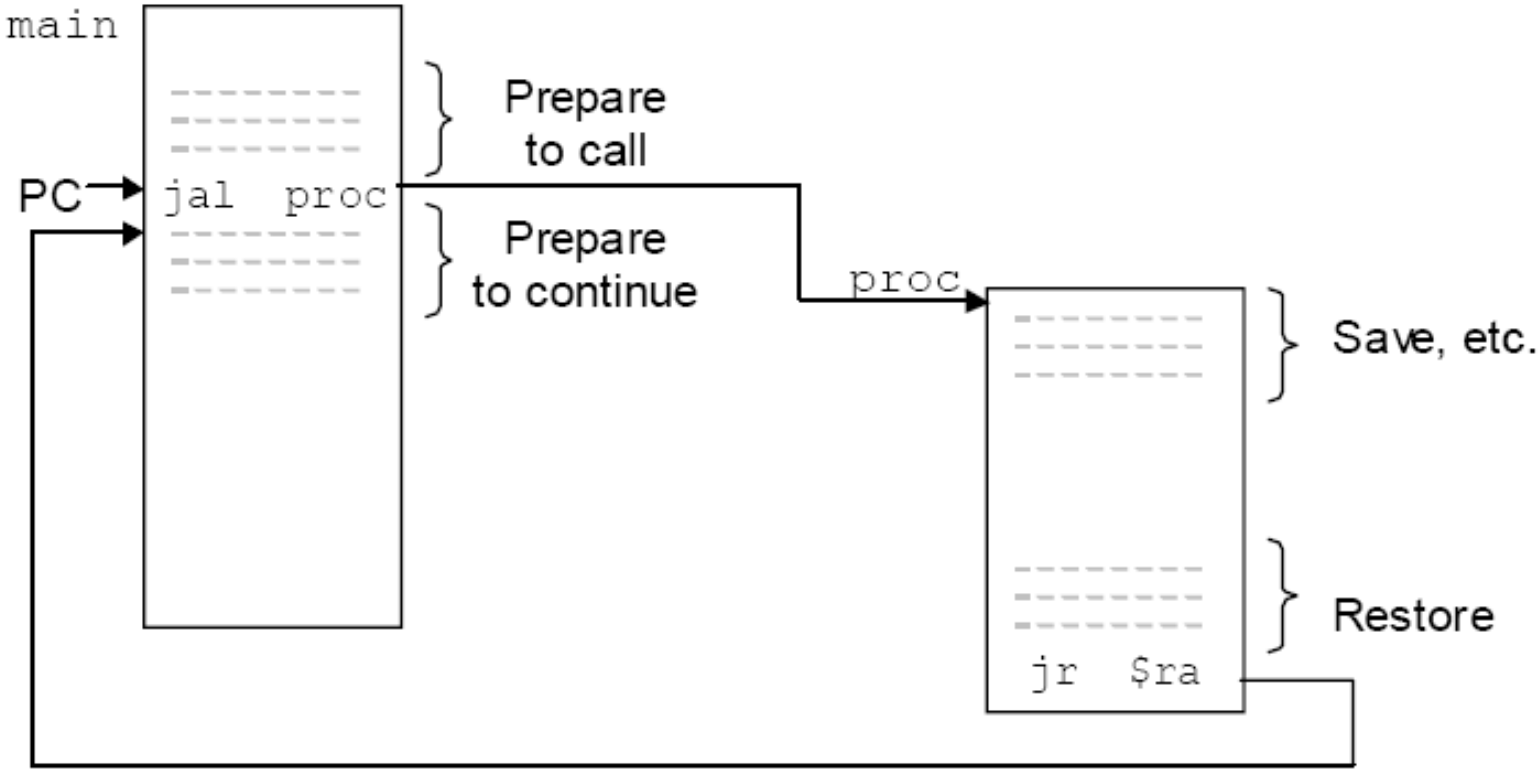
Procedure & Stack

- Procedure call:

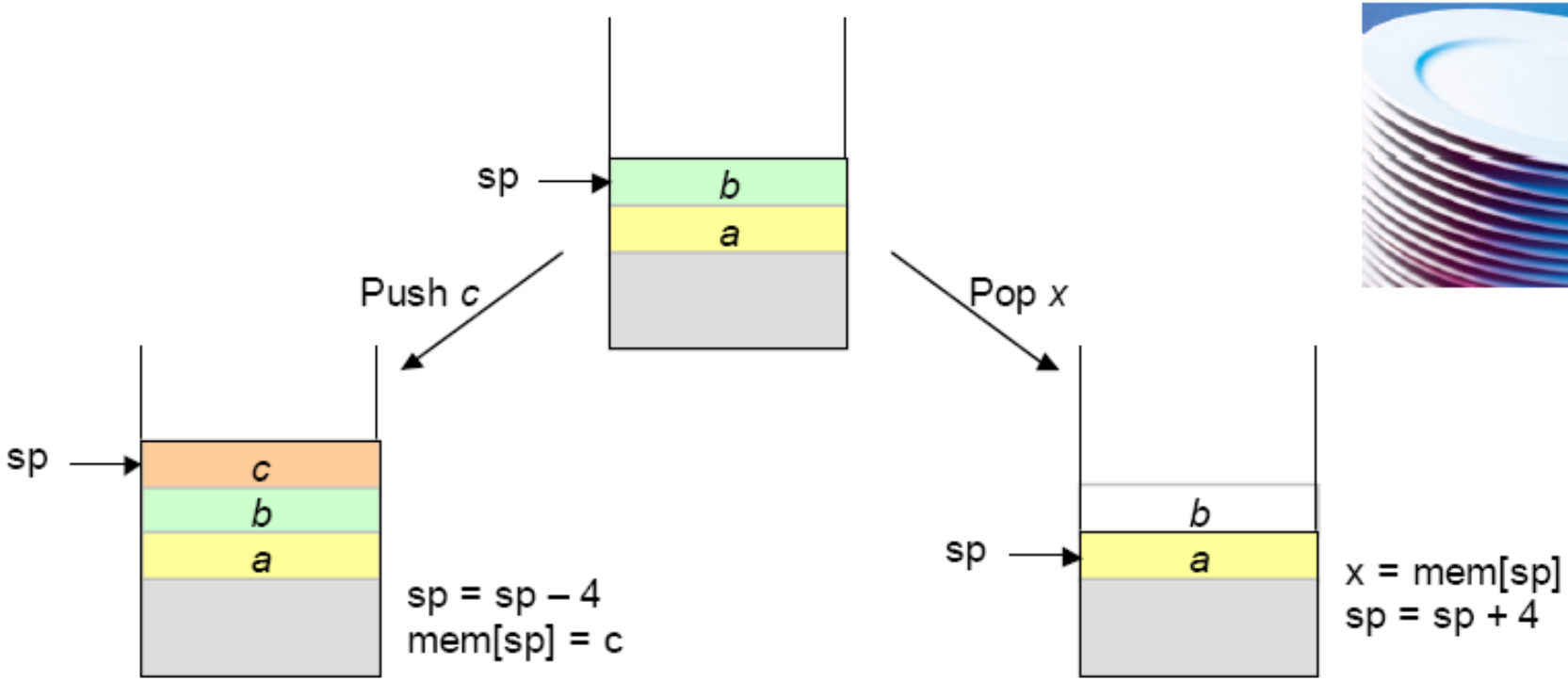
```
jal ( jump and link)
```

- Return to call point

```
jr $ra
```



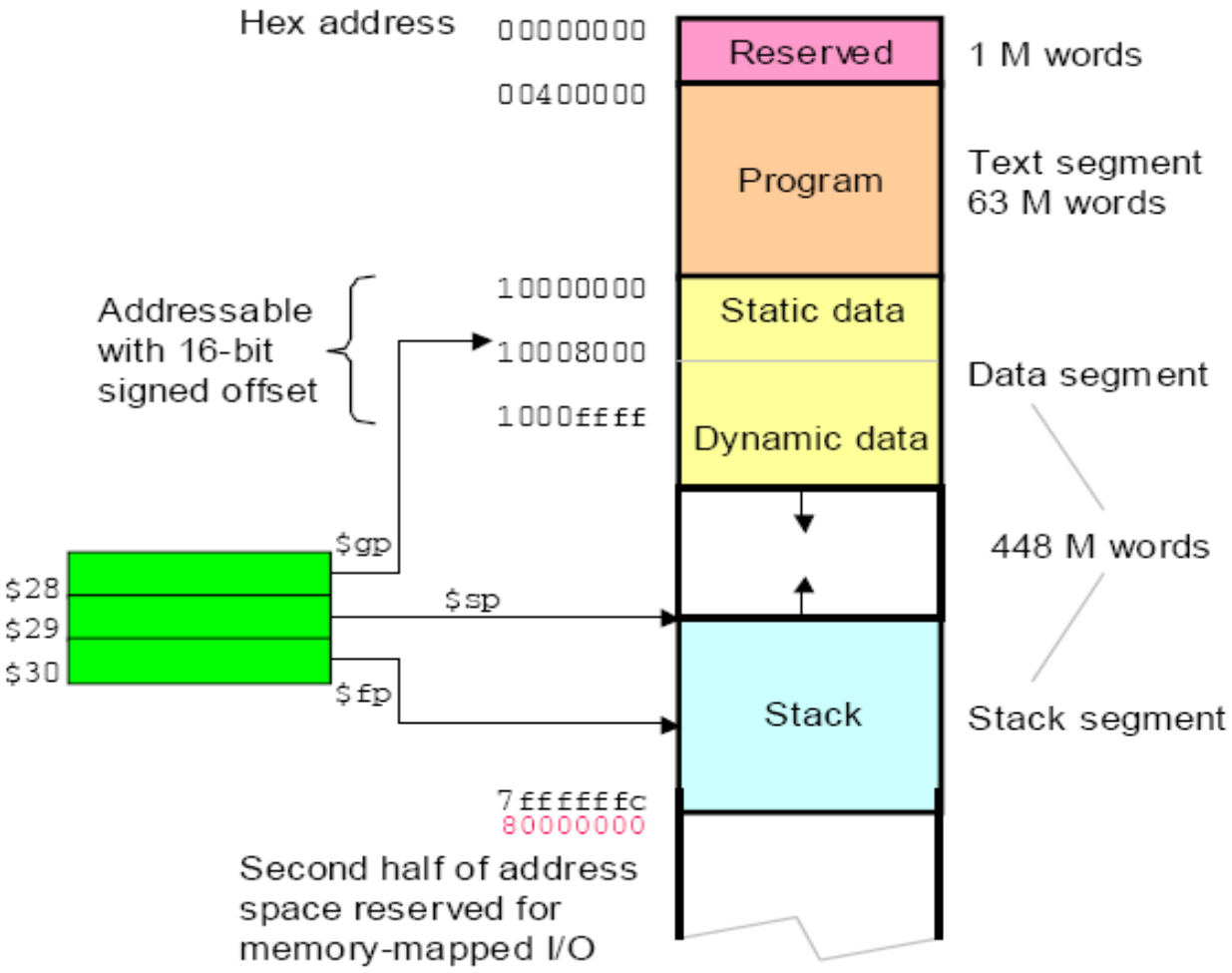
Stack



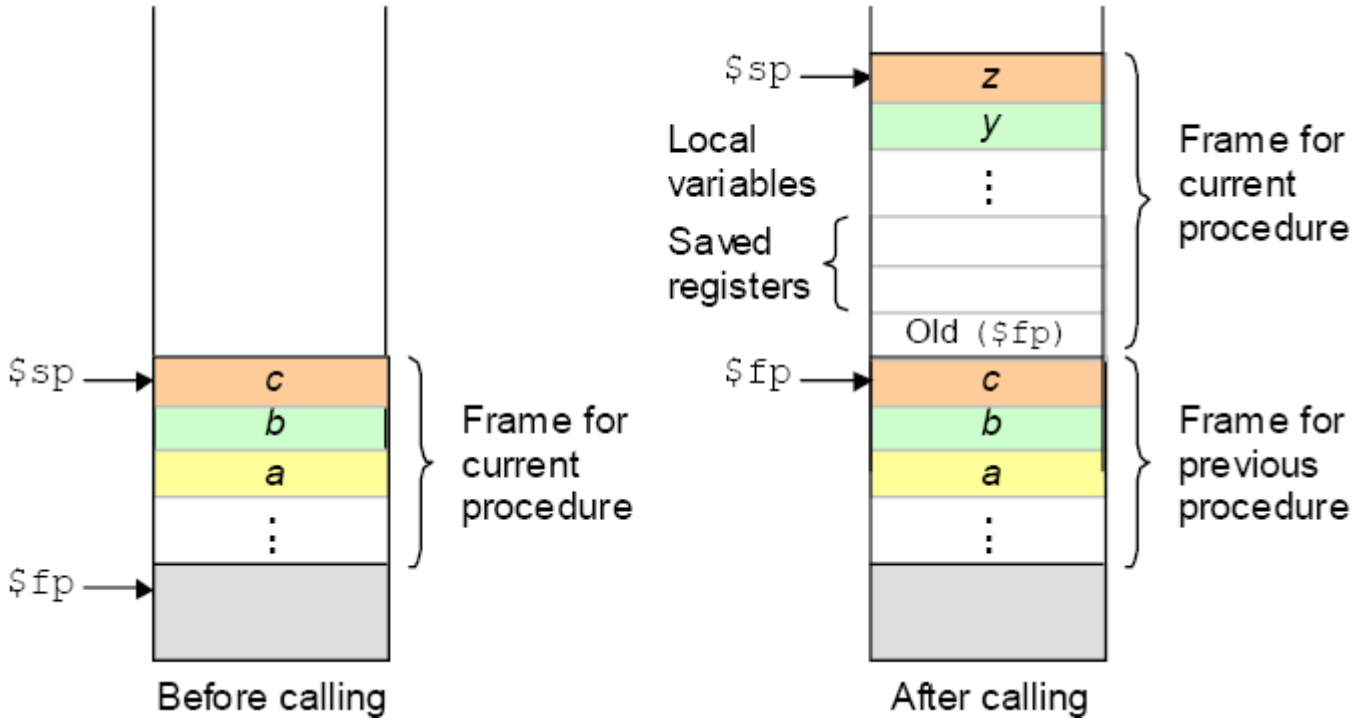
```
push: addi   $sp,$sp,-4
      sw     $t4,0($sp)
```

```
pop: lw     $t5,0($sp)
     addi   $sp,$sp,4
```

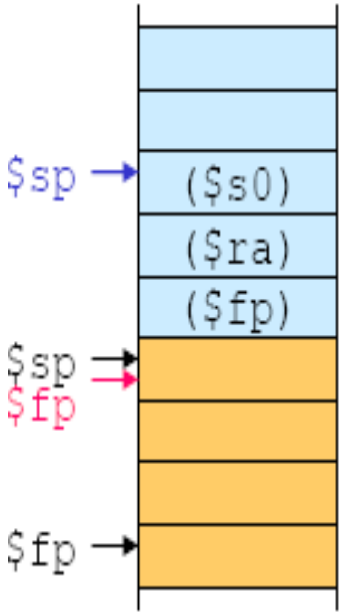
Stack


Hex address
00000000
00400000
10000000
10008000
1000ffff
7fffffffc
80000000
Reserved
Program
Static data
Dynamic data
Stack
1 M words
Text segment
63 M words
Data segment
448 M words
Stack segment
Addressable
with 16-bit
signed offset
$gp
$sp
$fp
$28
$29
$30
Second half of address
space reserved for
memory-mapped I/O

$sp and $fp

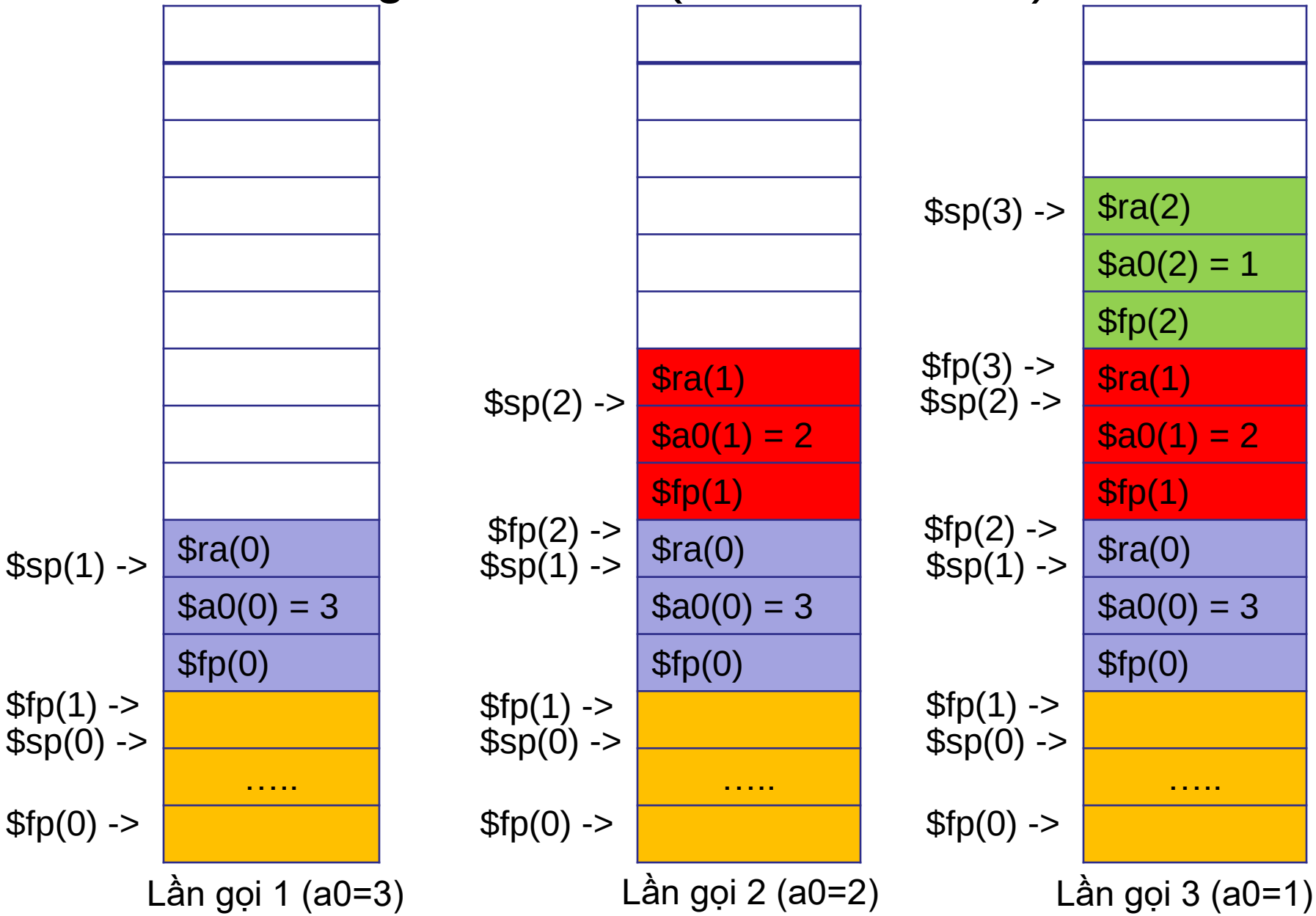

$sp
c
b
a
$fp
Frame for current procedure
Before calling
$sp
z
y
Local variables
Saved registers
Old ($fp)
$fp
c
b
a
Frame for current procedure
Frame for previous procedure
After calling

Example: $sp and $fp




```
proc:  sw   $fp,-4($sp)     # save the old frame pointer
       addi $fp,$sp,0       # save ($sp) into $fp
       addi $sp,$sp,-12     # create 3 spaces on top of stack
       sw   $ra,-8($fp)     # save ($ra) in 2nd stack element
       sw   $s0,-12($fp)    # save ($s0) in top stack element
        .
        .
        .
       lw   $s0,-12($fp)    # put top stack element in $s0
       lw   $ra,-8($fp)     # put 2nd stack element in $ra
       addi $sp,$fp, 0      # restore $sp to original state
       lw   $fp,-4($sp)     # restore $fp to original state
       jr   $ra             # return from procedure
```

Lab 7. Procedure Calls,
Assigment 4. n! (stack with n=3)



LAB 8-9
Mini-Project

LAB 10

Control pheripheral devices via simulators

LAB 11
Interrupts & IO programming

LAB 12
Cache Memory