

IT3280

THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Course administration

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

Lab Exercises

- Week 1-7: Lab exercises 1-7
- Week 8,9: Mini-Projects
- Week 10-12: Lab exercises 10-12
- Week 13,14,15: Final-Projects

Lab Exercises

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

Content

1. Introduction to MIPS
2. MIPS programming model
3. 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

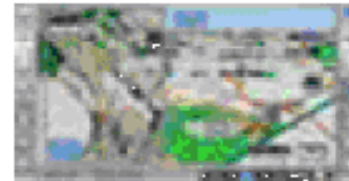
Pioneer

DVR-57-H



Kenwood

HDV-810 Car Navigation System



Networking

3COM

3102 Business IP Phone



3COM

3106 Cordless Phone



Apple

Airport Extreme WLAN Access Points



Applications of MIPS processor

Portable Devices

Canon

EOS 10D Digital

JVC

GR-HD1



Sony Playstation PSX



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

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

Applications of MIPS processor

Residential and Small Office

Samsung

Digital Photo Frame



Sony

Media Server Vaio VGX-X90P



Pioneer

Pure Vision^U Plasma Television 43"

Pure Vision^U Plasma Television 50"



Sony

KDP-51WS550 High Definition TV

KDP-57WS550 High Definition TV

KDP-65WS550 High Definition TV



Hewlett Packard

Color Laser Jet 2500 Laser Printer



2. MIPS Programming Model

- Data Types
- Registers
- Instruction Formats
- MIPS Instruction
- Addressing Mode
- MIPS Assembly program

Data Types

Byte = 8 bits



Halfword = 2 bytes



Used only for floating-point data,
so safe to ignore in this course

Word = 4 bytes



Doubleword = 8 bytes



Quadword (16 bytes) also used occasionally

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

\$0	0	\$zero	
\$1		\$at	Reserved for assembler use
\$2		\$v0	Procedure results
\$3		\$v1	
\$4		\$a0	Procedure arguments
\$5		\$a1	
\$6		\$a2	
\$7		\$a3	
\$8		\$t0	Temporary values
\$9		\$t1	
\$10		\$t2	
\$11		\$t3	
\$12		\$t4	
\$13		\$t5	
\$14		\$t6	
\$15		\$t7	
\$16		\$s0	Operands
\$17		\$s1	
\$18		\$s2	
\$19		\$s3	
\$20		\$s4	
\$21		\$s5	
\$22		\$s6	
\$23		\$s7	
\$24		\$t8	More temporaries
\$25		\$t9	
\$26		\$k0	Reserved for OS (kernel)
\$27		\$k1	
\$28		\$gp	Global pointer Stack pointer Frame pointer Return address
\$29		\$sp	
\$30		\$fp	
\$31		\$ra	

Reserved for assembler use

Procedure results

Procedure arguments

Saved

Temporary values

Operands

Saved across procedure calls

More temporaries

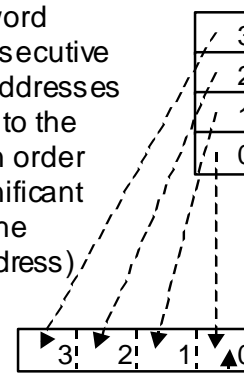
Reserved for OS (kernel)

Global pointer
Stack pointer
Frame pointer
Return address

Saved

A 4-byte word sits in consecutive memory addresses according to the big-endian order (most significant byte has the lowest address)

Byte numbering:



When loading a byte into a register, it goes in the low end

Byte

Word

Double word



A doubleword sits in consecutive registers or memory locations according to the big-endian order (most significant word comes first)

Register Conventions

Figure 5.2
Registers and
data sizes in
MiniMIPS.

Instruction Formats

High-level language statement:

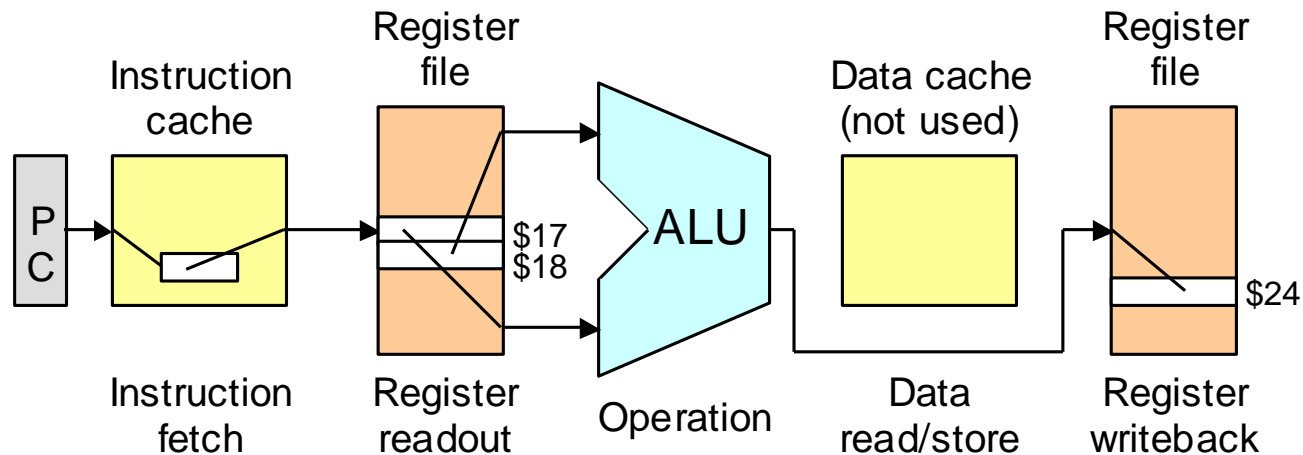
`a = b + c`

Assembly language instruction:

`add $t8, $s2, $s1`

Machine language instruction:

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



A typical instruction for MiniMIPS and steps in its execution.

MiniMIPS Instruction Formats

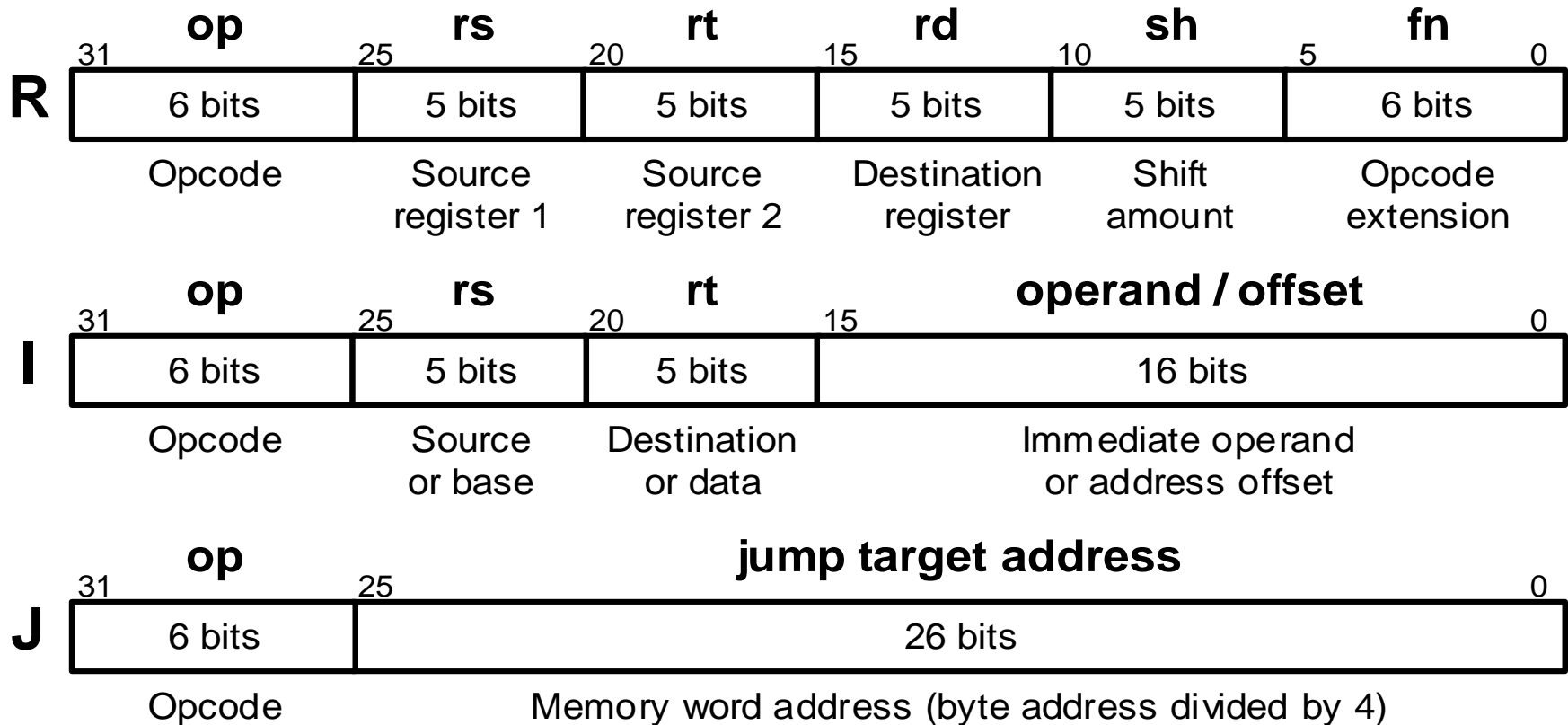


Figure 5.4 MiniMIPS instructions come in only three formats: register (R), immediate (I), and jump (J).

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) \wedge (\$s1)
or	\$t0, \$s0, \$s1	# set \$t0 to (\$s0) \vee (\$s1)
xor	\$t0, \$s0, \$s1	# set \$t0 to (\$s0) \oplus (\$s1)
nor	\$t0, \$s0, \$s1	# set \$t0 to (((\$s0) \vee (\$s1)))'

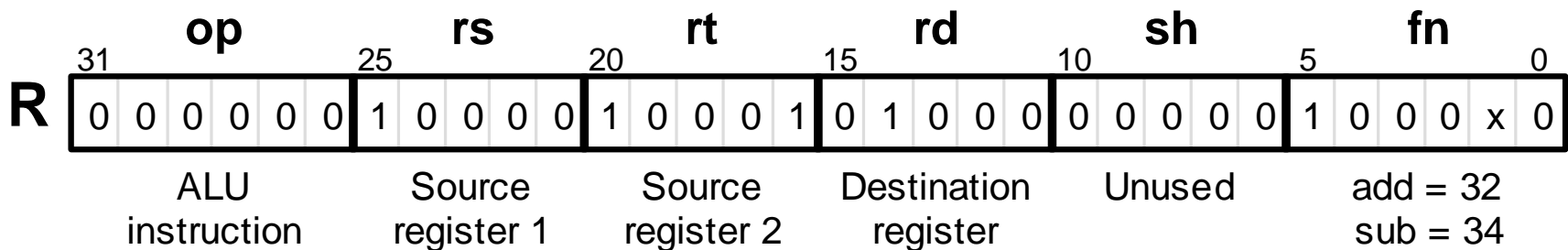


Figure 5.5 The arithmetic instructions `add` and `sub` have a format that is common to all two-operand ALU instructions. For these, the `fn` field specifies the arithmetic/logic operation to be performed.

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)v61
xori    $t0,$s0,0x00ff  # set $t0 to ($s0)⊕ 0x00ff
```

For arithmetic instructions, the immediate operand is sign-extended

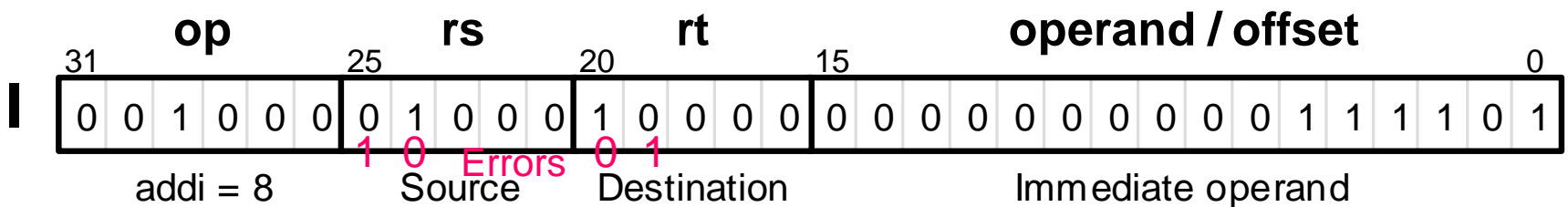
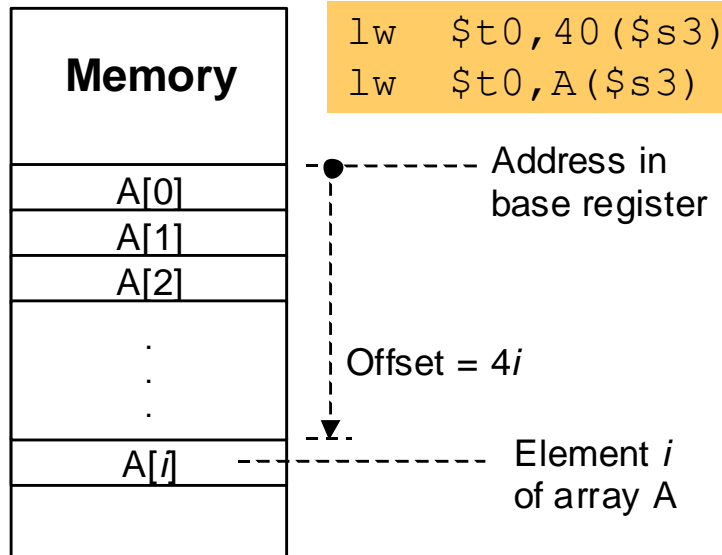
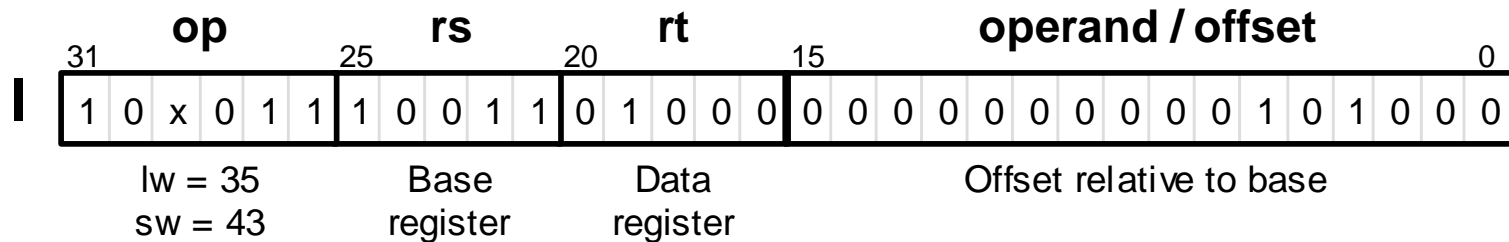


Figure 5.6 Instructions such as `addi` allow us to perform an arithmetic or logic operation for which one operand is a small constant.

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 = $4i$ 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
```

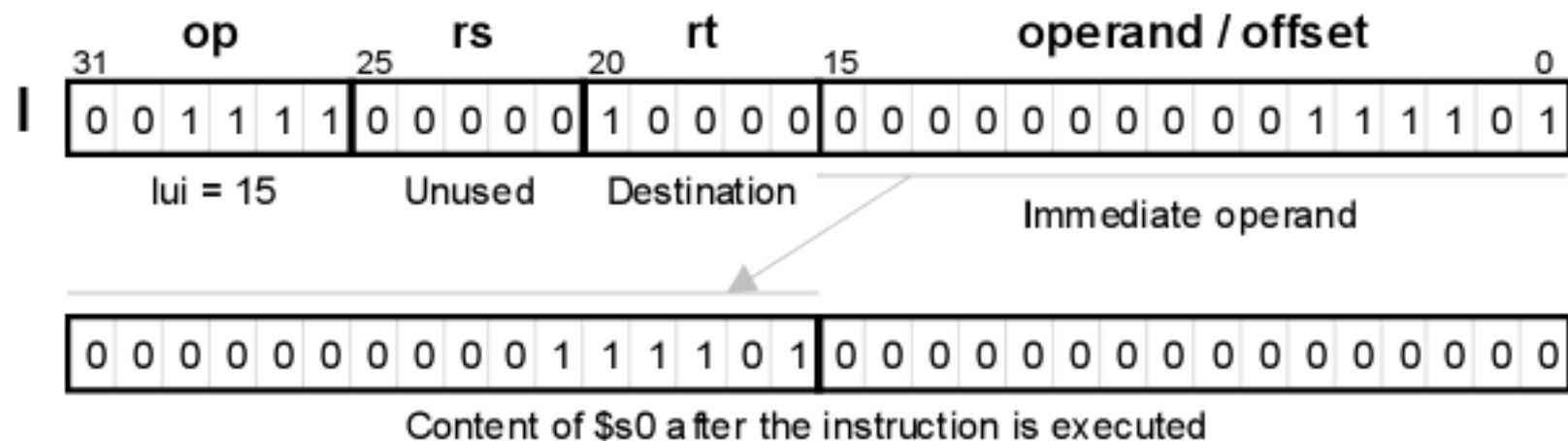


Figure 5.8 The `lui` instruction allows us to load an arbitrary 16-bit value into the upper half of a register while setting its lower half to 0s.

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  $(0 \vee 0)' = 1$ 
```

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)

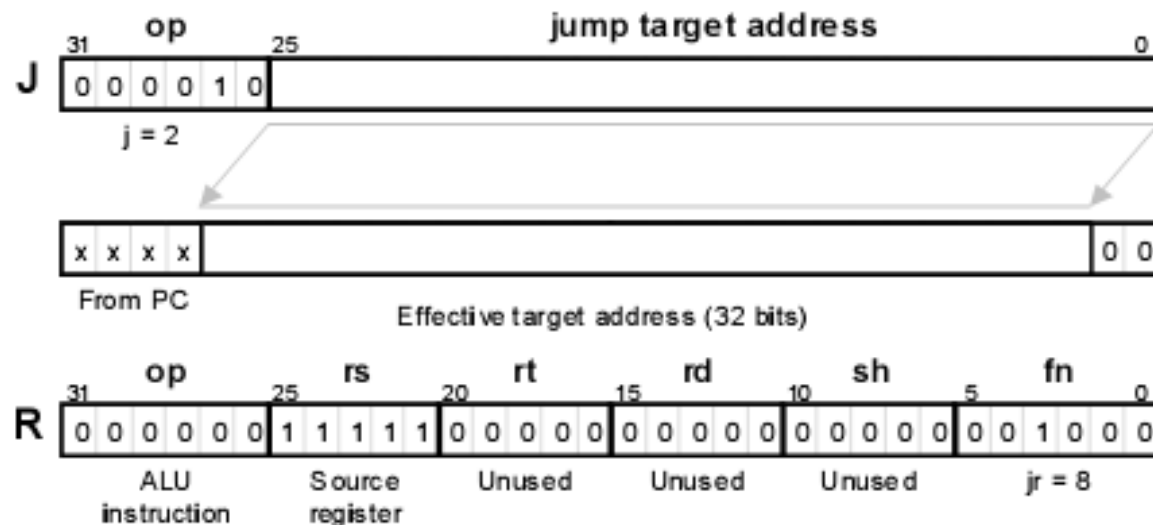


Figure 5.9 The jump instruction j of MiniMIPS is a J-type instruction which is shown along with how its effective target address is obtained. The jump register (jr) instruction is R-type, with its specified register often being $\$ra$.

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)
```

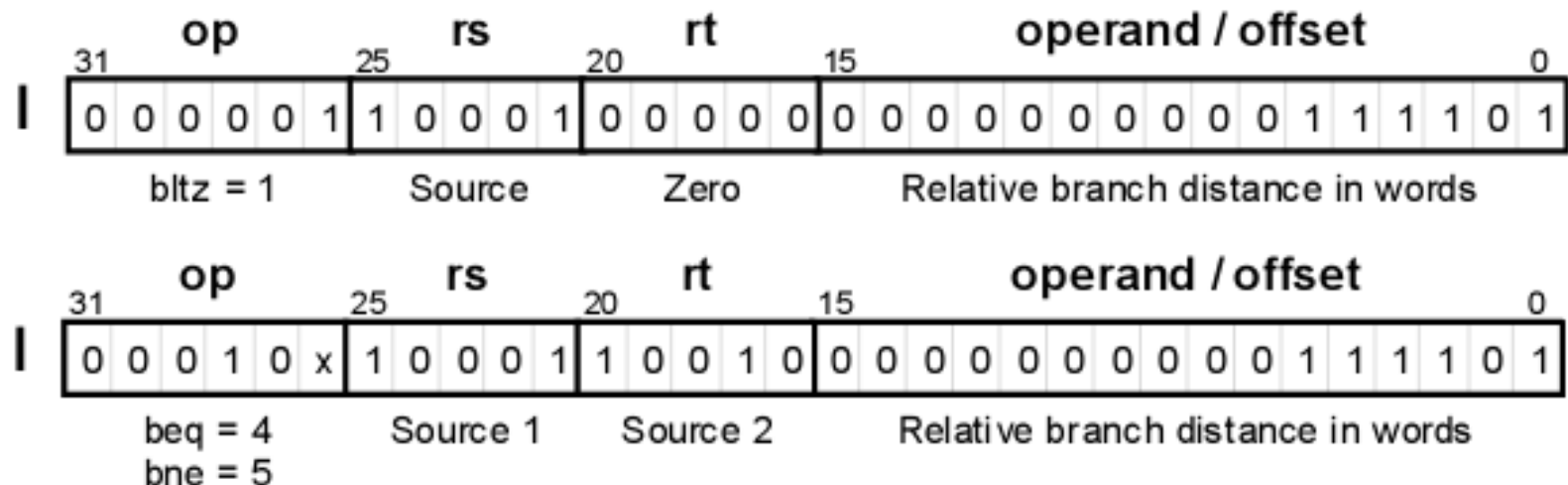


Figure 5.10 (part 1) Conditional branch instructions of MiniMIPS.

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.

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:
```

The 20 MiniMIPS Instructions Covered So Far

Copy

Arithmetic

Logic

Memory access

Control transfer

Table 5.1

Instruction	Usage	op	fn
Load upper immediate	lui rt,imm	15	
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	
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	
Load word	lw rt,imm(rs)	35	
Store word	sw rt,imm(rs)	43	
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	

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 or more instructions.
- Example:
 - **mov \$rt, \$rs** #Copy contents of register **s** to register **t**, $R[t] = R[s]$
=> real instruction: **addi \$rt, \$rs, 0**

Addressing Mode

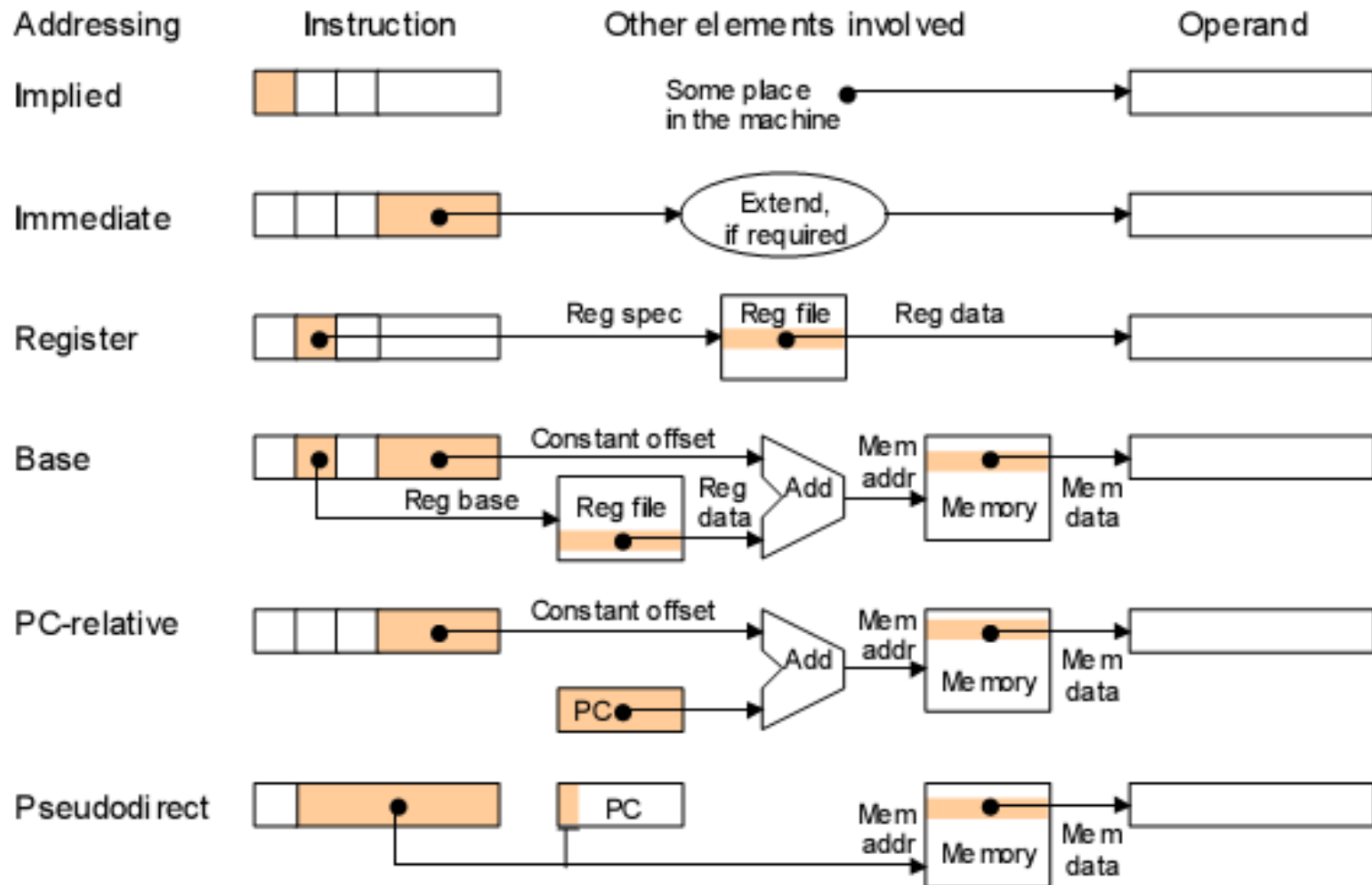


Figure 5.11 Schematic representation of addressing modes in MiniMIPS.

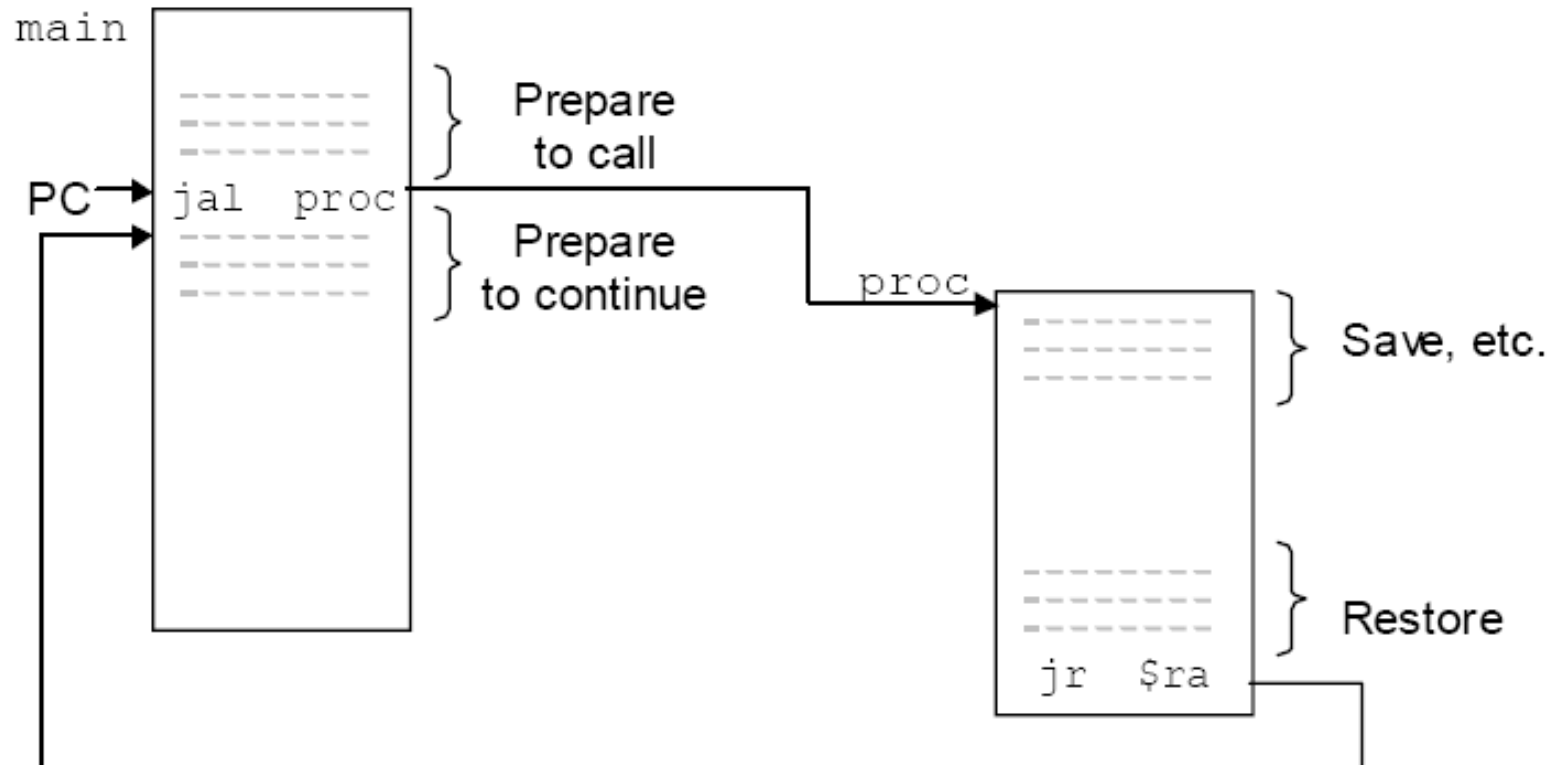
Procedure & Stack

- Procedure call:

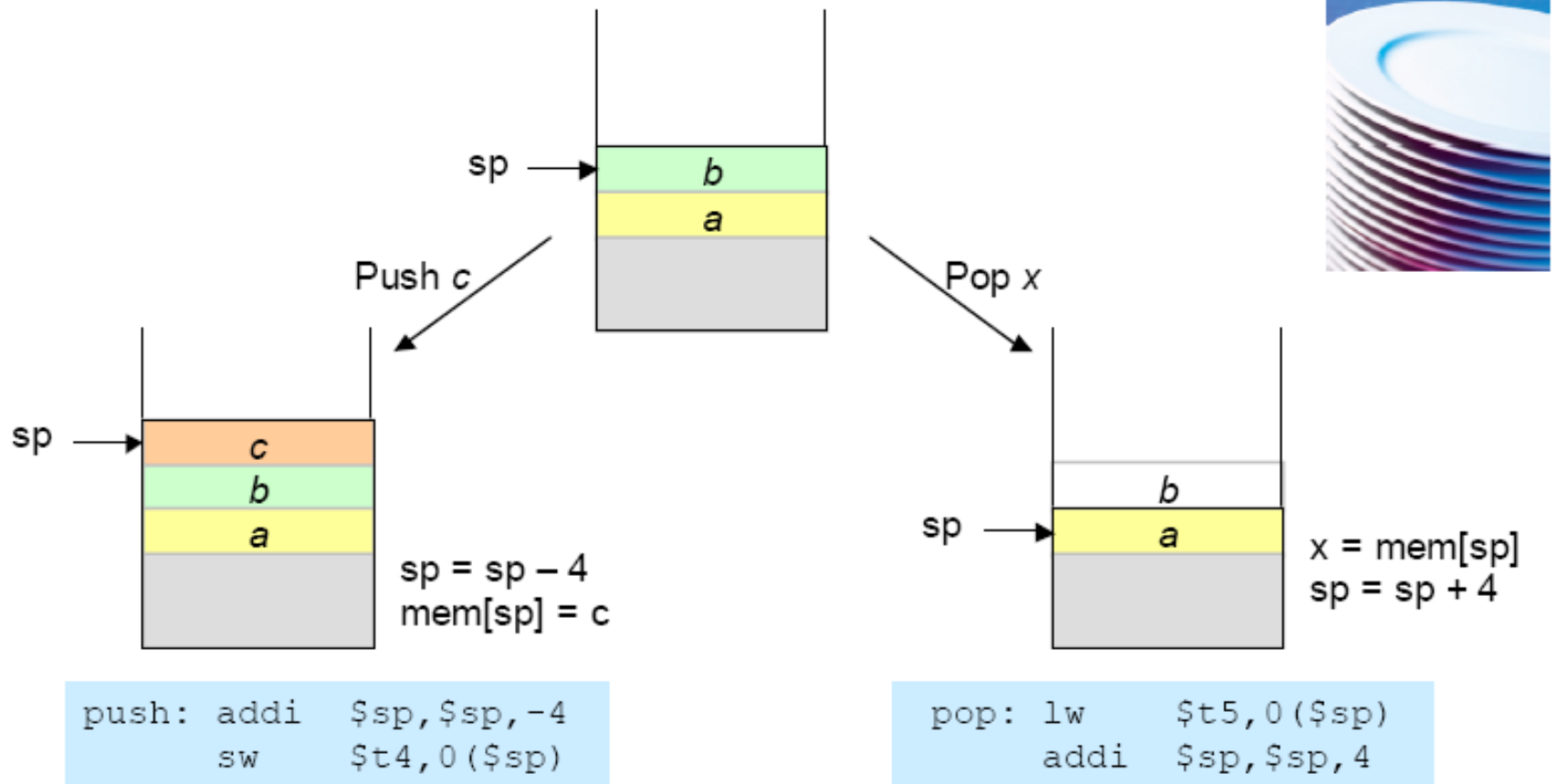
`jal (jump and link)`

- Return to call point

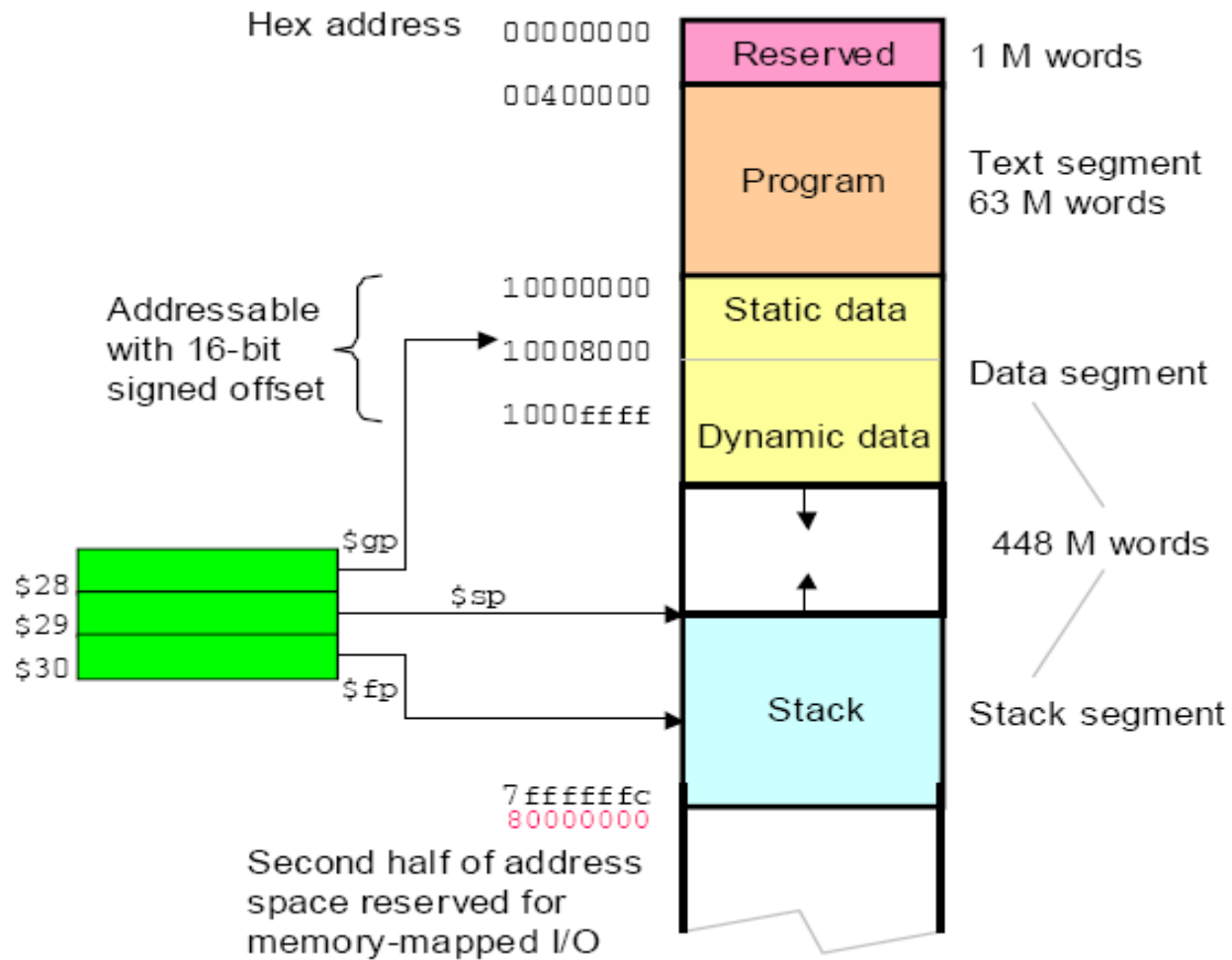
`jr $ra`



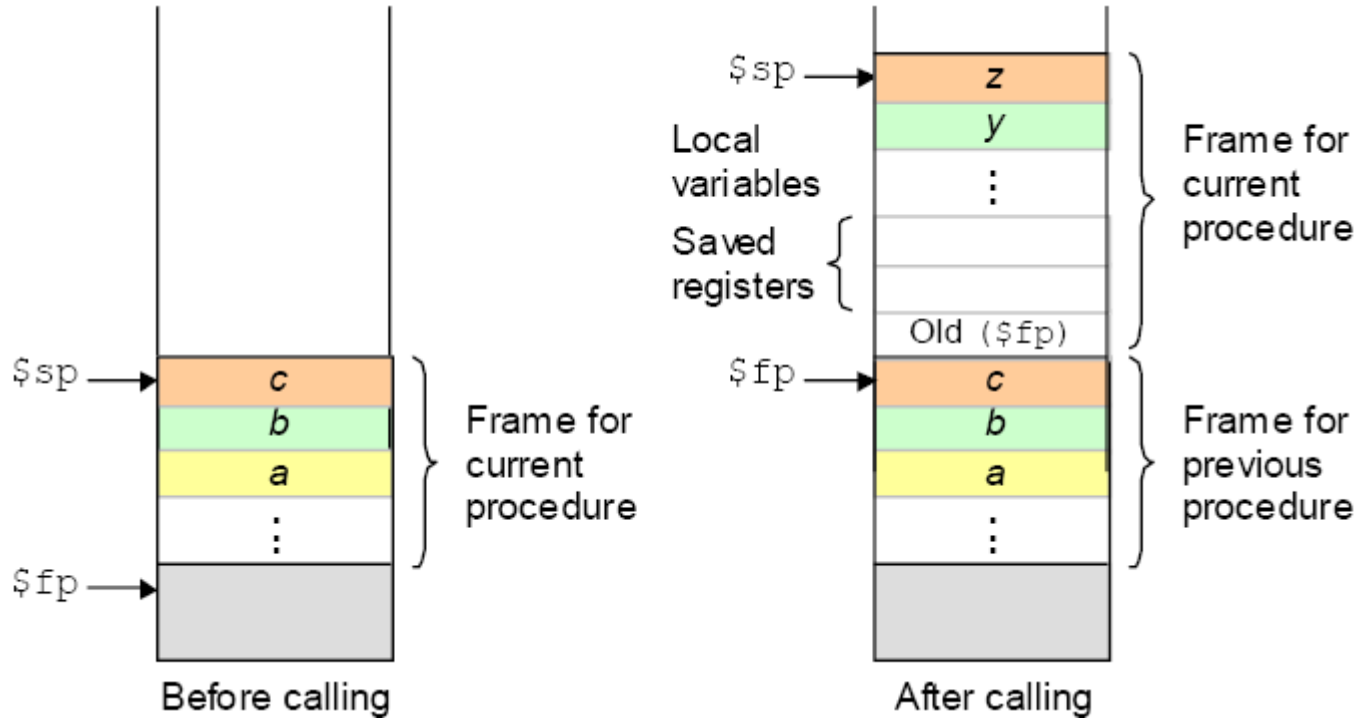
Stack



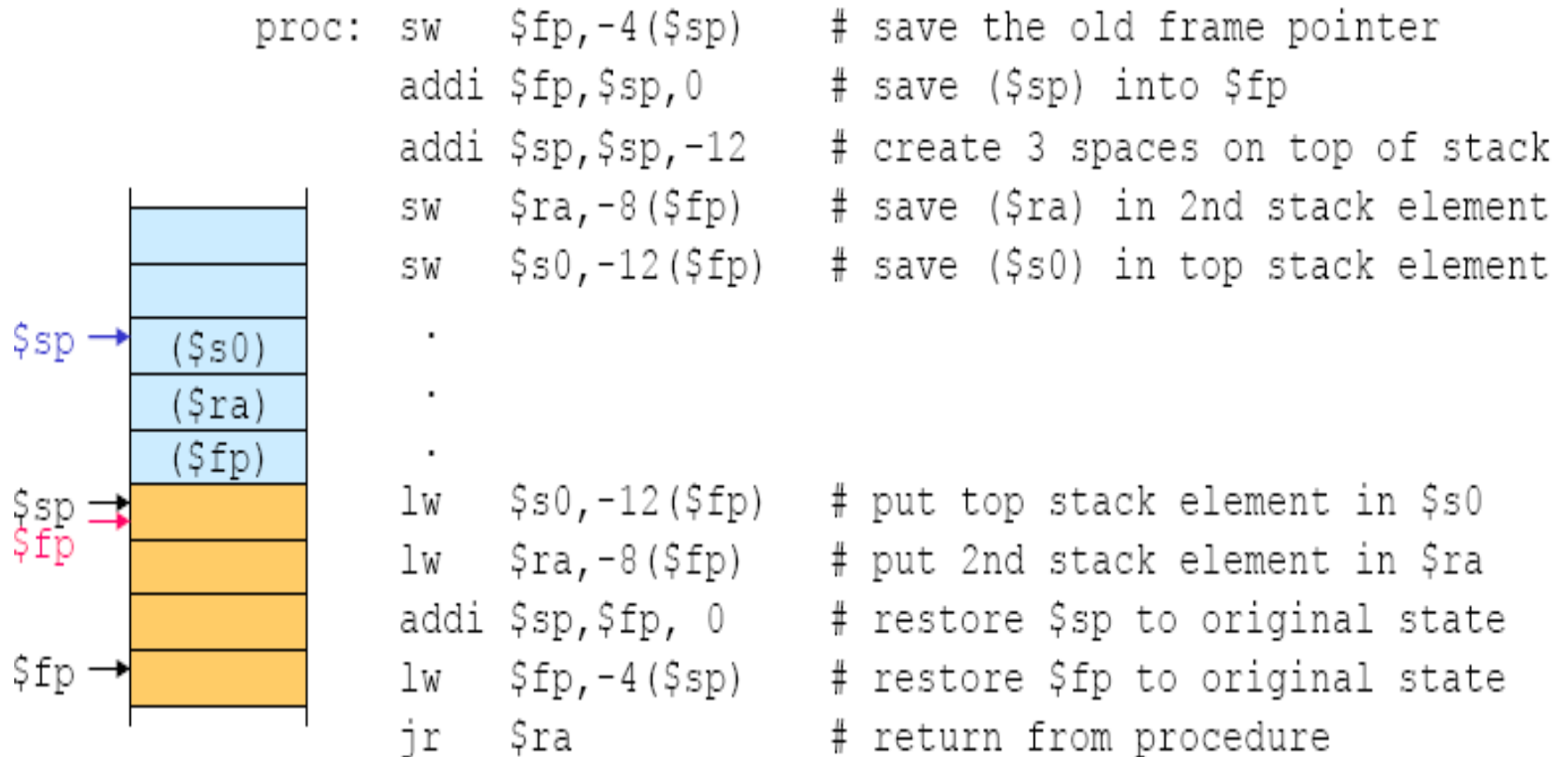
Stack



\$sp and \$fp



Example: \$sp and \$fp

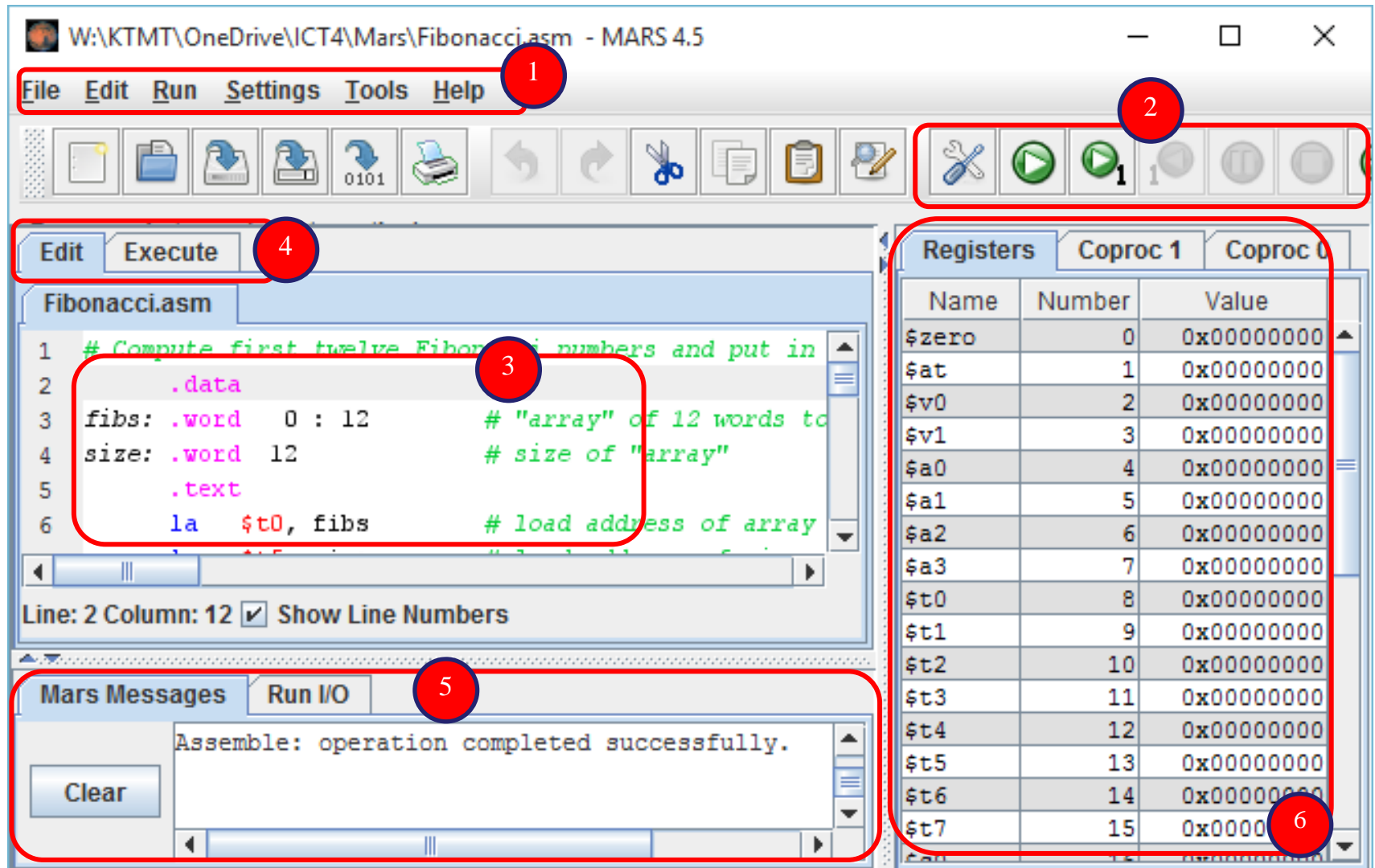


3. MIPS Simulation - Mars

- 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*			
1	.data	# Vung du lieu, chua cac khai bao bien	
2	x: .word	0x01020304	# bien x, khoi tao gia tri
3	message:	.asciiz	"Bo mon Ky thuat May tinh"
4			
5	.text	# Vung lenh, chua cac lenh hop ngu	
6	la	\$a0, message	#Dua dia chi bien mesage vao thanh ghi a0
7	li	\$v0, 4	#Gan thanh ghi v0 = 4
8	syscall	#Goi ham so v0, ham so 4, la ham print	
9			
10	addi	\$t1,\$zero,2	#Thanh ghi t1 = 2
11	addi	\$t2,\$zero,3	#Thanh ghi t2 = 3
12	add	\$t0, \$t1, \$t2	#Thanh ghi t- = t1 + t2

MIPS assembly program: HelloWorld

The screenshot displays a MIPS assembly editor interface with two main panels: 'Text Segment' and 'Data Segment'.

Text Segment: This panel shows the assembly code for the program. It includes a table with columns for Bkpt, Address, Code, Basic, and Source. The code is as follows:

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: A panel on the right shows the labels defined in the program:

Label	Address
mips1.asm	
x	0x10010000
message	0x10010004

Data Segment: This panel shows the memory layout of the program. It includes a table with columns for Address and Value (+0) through Value (+1c). The data is as follows:

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

The bottom of the interface includes navigation buttons (left and right arrows), a dropdown menu showing '0x10010000 (.data)', and checkboxes for 'Hexadecimal Addresses', 'Hexadecimal Values', and 'ASCII'.

MIPS assembly program: HelloWorld

EditExecute

Text Segment

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

The screenshot displays a MIPS assembly editor with two main panels: the Text Segment and the Data Segment. The Text Segment shows assembly code with instructions like `lui $1, 0x00001001` and `ori $4, $1, 0x00000004`. The Data Segment shows memory addresses and their corresponding values, including the string `message` at address `0x10010004`. Red arrows indicate the mapping from assembly instructions to memory locations. Green and purple arrows point to the 'Hexadecimal Addresses', 'Hexadecimal Values', and 'ASCII' checkboxes respectively.

Text Segment

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

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	m o B	E 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

Labels

Label	Address
mips1.asm	
x	0x10010000
message	0x10010004

Options

☒ 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

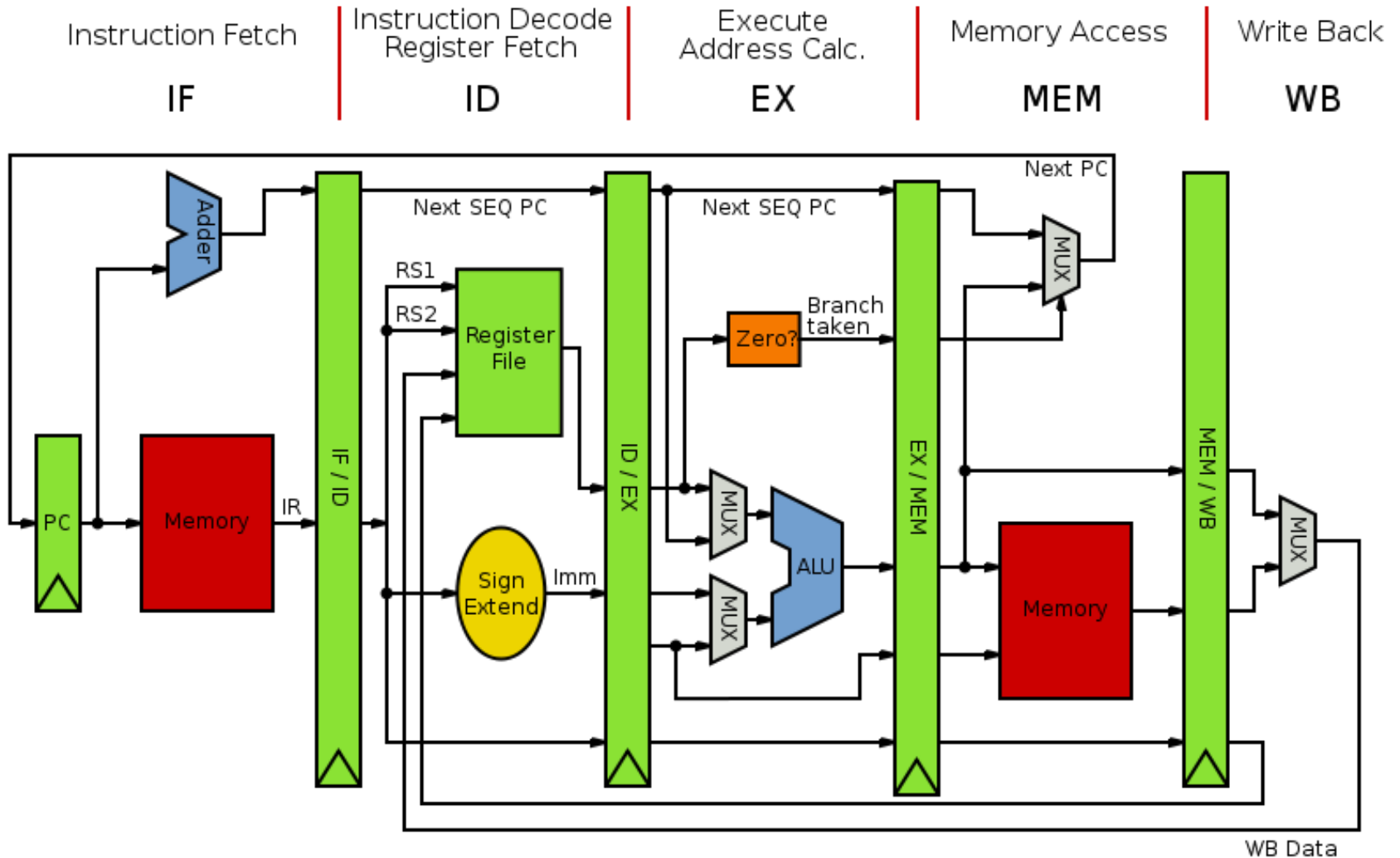
1 .data # Vung du lieu, chua cac khai bao bien
2 x: .word 0x01020304 # bien x, khoi tao gia tri
3 message: .asciiz "Bo mon Ky thuat May tinh"

Data Segment

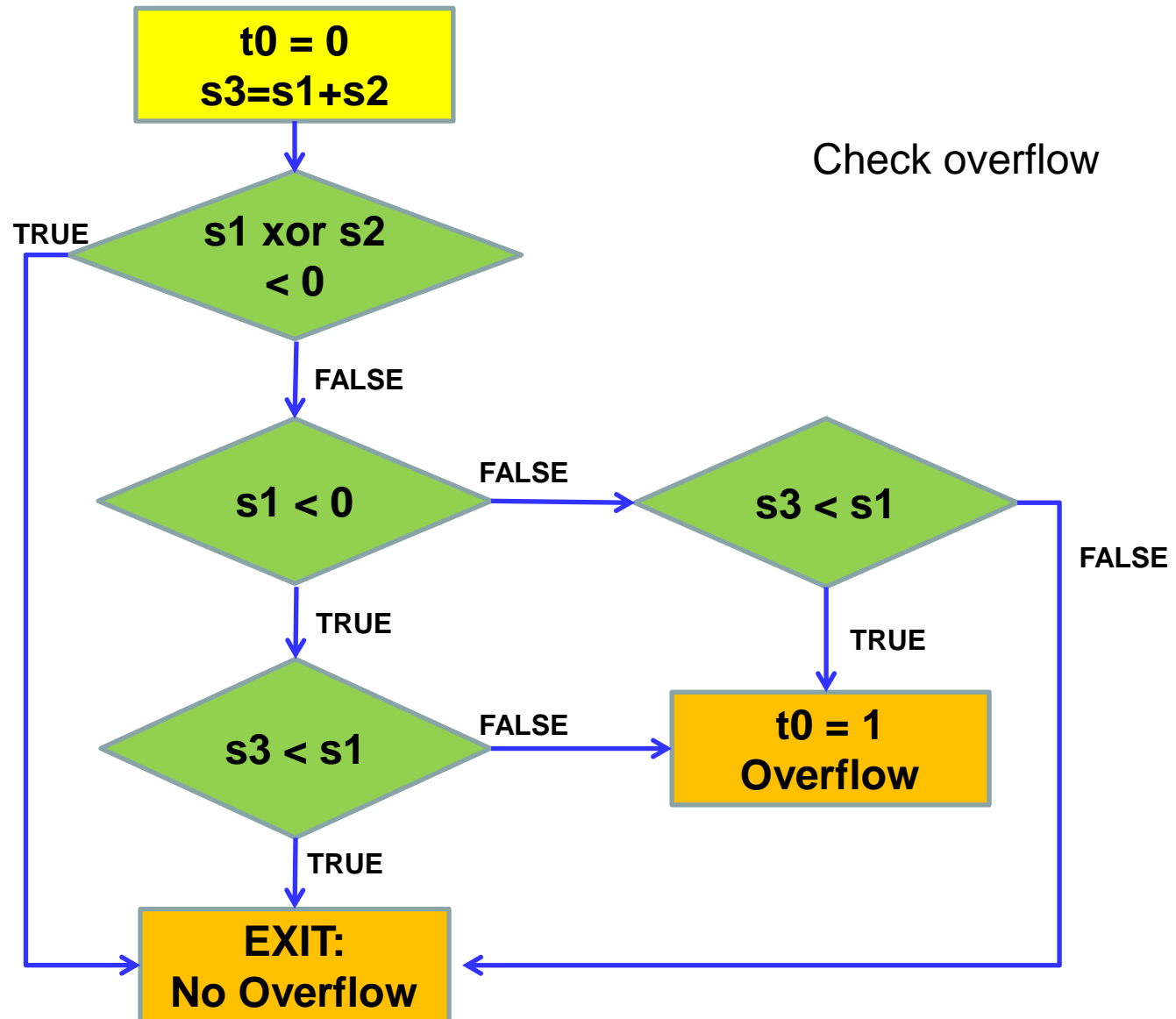
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x01020304	0x6d206f42	0x4b206e6f	0x68742079	0x20746175	0x2079614d	0x686e6974	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040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

Navigation buttons:   **0x10010000 (.data)** ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

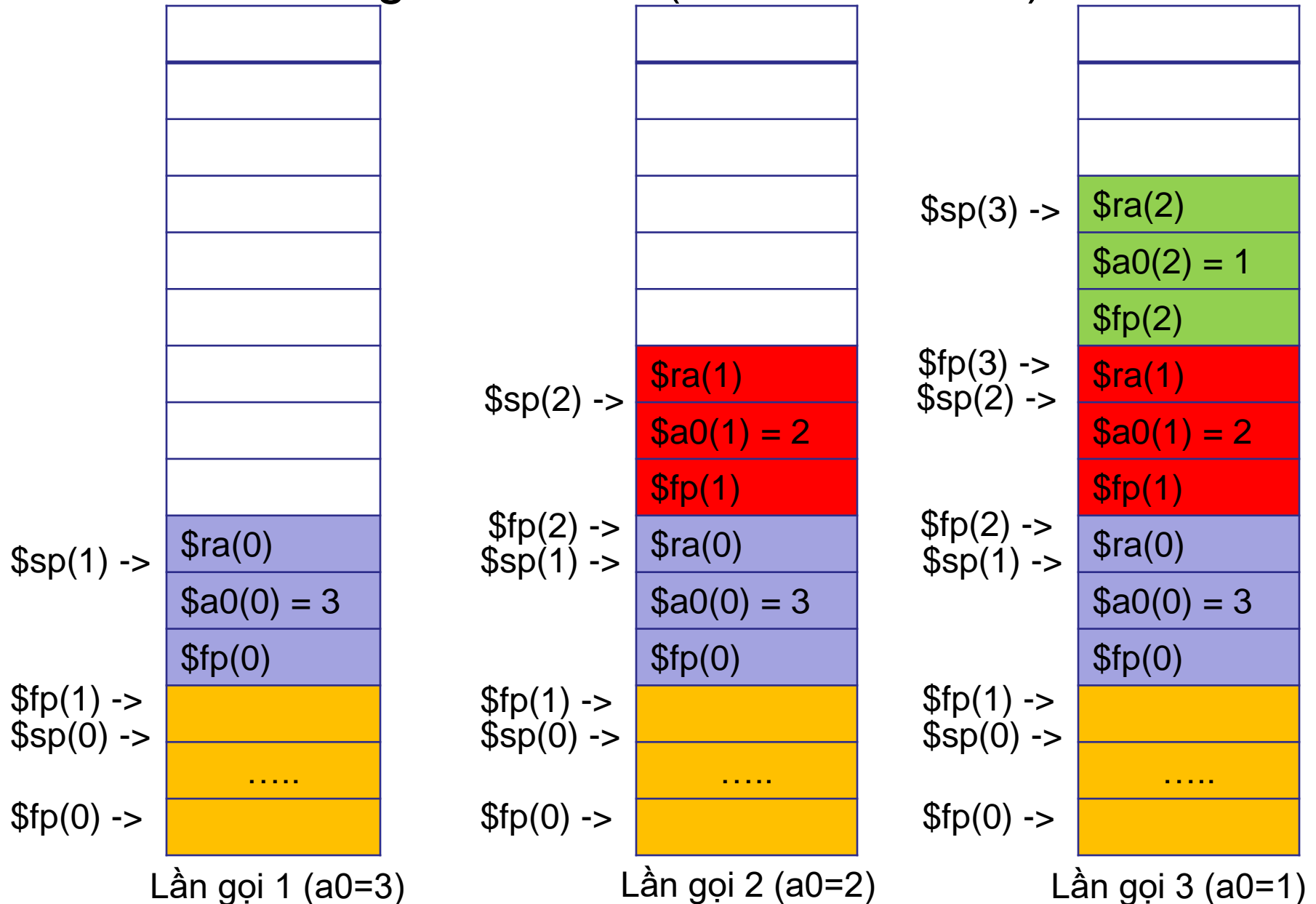
Pipelined MIPS



Lab 4. Arithmetic & Logical Operation



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



Lab 5. Character string

- strcpy

Địa chỉ chuỗi 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
bad va =00000000

s0=6, x[6]=y[6]
Ký tự 'a'

Địa chỉ chuỗi x:
a0 = 800203d5

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	
80020418	00 00 00 00	
8002041C	00 00 00 00	
80020420	00 00 00 00	
80020424	00 00 00 00	
80020428	00 00 00 00	
8002042C	00 00 00 00	
80020430	00 00 00 00	
80020434	00 00 00 00	
80020438	00 00 00 00	
8002043C	00 00 00 00	
80020440	00 00 00 00	
80020444	00 00 00 00	
80020448	00 00 00 00	
8002044C	00 00 00 00	
80020450	00 00 00 00	
80020454	00 00 00 00	
80020458	00 00 00 00	
8002045C	00 00 00 00	
80020460	00 00 00 00	
80020464	00 00 00 00	
80020468	00 00 00 00	
8002046C	00 00 00 00	
80020470	00 00 00 00	
80020474	00 00 00 00	
80020478	00 00 00 00	
8002047C	00 00 00 00	
80020480	00 00 00 00	
80020484	00 00 00 00	
80020488	00 00 00 00	
8002048C	00 00 00 00	
80020490	00 00 00 00	
80020494	00 00 00 00	
80020498	00 00 00 00	
8002049C	00 00 00 00	
800204A0	00 00 00 00	
800204A4	00 00 00 00	
800204A8	00 00 00 00	
800204AC	00 00 00 00	
800204B0	00 00 00 00	
800204B4	00 00 00 00	
800204B8	00 00 00 00	
800204BC	00 00 00 00	
800204C0	00 00 00 00	
800204C4	00 00 00 00	
800204C8	00 00 00 00	
800204CC	00 00 00 00	
800204D0	00 00 00 00	
800204D4	00 00 00 00	
800204D8	00 00 00 00	
800204DC	00 00 00 00	
800204E0	00 00 00 00	
800204E4	00 00 00 00	
800204E8	00 00 00 00	
800204EC	00 00 00 00	
800204F0	00 00 00 00	
800204F4	00 00 00 00	
800204F8	00 00 00 00	
800204FC	00 00 00 00	
80020500	00 00 00 00	
80020504	00 00 00 00	
80020508	00 00 00 00	
8002050C	00 00 00 00	
80020510	00 00 00 00	
80020514	00 00 00 00	
80020518	00 00 00 00	
8002051C	00 00 00 00	
80020520	00 00 00 00	
80020524	00 00 00 00	
80020528	00 00 00 00	
8002052C	00 00 00 00	
80020530	00 00 00 00	
80020534	00 00 00 00	
80020538	00 00 00 00	
8002053C	00 00 00 00	
80020540	00 00 00 00	
80020544	00 00 00 00	
80020548	00 00 00 00	
8002054C	00 00 00 00	
80020550	00 00 00 00	
80020554	00 00 00 00	
80020558	00 00 00 00	
8002055C	00 00 00 00	
80020560	00 00 00 00	
80020564	00 00 00 00	
80020568	00 00 00 00	
8002056C	00 00 00 00	
80020570	00 00 00 00	
80020574	00 00 00 00	
80020578	00 00 00 00	
8002057C	00 00 00 00	
80020580	00 00 00 00	
80020584	00 00 00 00	
80020588	00 00 00 00	
8002058C	00 00 00 00	
80020590	00 00 00 00	
80020594	00 00 00 00	
80020598	00 00 00 00	
8002059C	00 00 00 00	
800205A0	00 00 00 00	
800205A4	00 00 00 00	
800205A8	00 00 00 00	
800205AC	00 00 00 00	
800205B0	00 00 00 00	
800205B4	00 00 00 00	
800205B8	00 00 00 00	
800205BC	00 00 00 00	
800205C0	00 00 00 00	
800205C4	00 00 00 00	
800205C8	00 00 00 00	
800205CC	00 00 00 00	
800205D0	00 00 00 00	
800205D4	00 00 00 00	
800205D8	00 00 00 00	
800205DC	00 00 00 00	
800205E0	00 00 00 00	
800205E4	00 00 00 00	
800205E8	00 00 00 00	
800205EC	00 00 00 00	
800205F0	00 00 00 00	
800205F4	00 00 00 00	
800205F8	00 00 00 00	
800205FC	00 00 00 00	
80020600	00 00 00 00	
80020604	00 00 00 00	
80020608	00 00 00 00	
8002060C	00 00 00 00	
80020610	00 00 00 00	
80020614	00 00 00 00	
80020618	00 00 00 00	
8002061C	00 00 00 00	
80020620	00 00 00 00	
80020624	00 00 00 00	
80020628	00 00 00 00	
8002062C	00 00 00 00	
80020630	00 00 00 00	
80020634	00 00 00 00	
80020638	00 00 00 00	
8002063C	00 00 00 00	
80020640	00 00 00 00	
80020644	00 00 00 00	
80020648	00 00 00 00	
8002064C	00 00 00 00	
80020650	00 00 00 00	
80020654	00 00 00 00	
80020658	00 00 00 00	
8002065C	00 00 00 00	
80020660	00 00 00 00	
80020664	00 00 00 00	
80020668	00 00 00 00	
8002066C	00 00 00 00	
80020670	00 00 00 00	
80020674	00 00 00 00	
80020678	00 00 00 00	
8002067C	00 00 00 00	
80020680	00 00 00 00	
80020684	00 00 00 00	
80020688	00 00 00 00	
8002068C	00 00 00 00	
80020690	00 00 00 00	
80020694	00 00 00 00	
80020698	00 00 00 00	
8002069C	00 00 00 00	
800206A0	00 00 00 00	
800206A4	00 00 00 00	
800206A8	00 00 00 00	
800206AC	00 00 00 00	
800206B0	00 00 00 00	
800206B4	00 00 00 00	
800206B8	00 00 00 00	
800206BC	00 00 00 00	
800206C0	00 00 00 00	
800206C4	00 00 00 00	
800206C8	00 00 00 00	
800206CC	00 00 00 00	
800206D0	00 00 00 00	
800206D4	00 00 00 00	
800206D8	00 00 00 00	
800206DC	00 00 00 00	
800206E0	00 00 00 00	
800206E4	00 00 00 00	
800206E8	00 00 00 00	
800206EC	00 00 00 00	
800206F0	00 00 00 00	
800206F4	00 00 00 00	
800206F8	00 00 00 00	
800206FC	00 00 00 00	
80020700	00 00 00 00	
80020704	00 00 00 00	
80020708	00 00 00 00	
8002070C	00 00 00 00	
80020710	00 00 00 00	
80020714	00 00 00 00	
80020718	00 00 00 00	
8002071C	00 00 00 00	
80020720	00 00 00 00	
80020724	00 00 00 00	
80020728	00 00 00 00	
8002072C	00 00 00 00	
80020730	00 00 00 00	
80020734	00 00 00 00	
80020738	00 00 00 00	
8002073C	00 00 00 00	
80020740	00 00 00 00	
80020744	00 00 00 00	
80020748	00 00 00 00	
8002074C	00 00 00 00	
80020750	00 00 00 00	
80020754	00 00 00 00	
80020758	00 00 00 00	
8002075C	00 00 00 00	
80020760	00 00 00 00	
80020764	00 00 00 00	
80020768	00 00 00 00	
8002076C	00 00 00 00	
80020770	00 00 00 00	
80020774	00 00 00 00	
80020778	00 00 00 00	
8002077C	00 00 00 00	
80020780	00 00 00 00	
80020784	00 00 00 00	
80020788	00 00 00 00	
8002078C	00 00 00 00	
80020790	00 00 00 00	
80020794	00 00 00 00	
80020798	00 00 00 00	
8002079C	00 00 00 00	
800207A0	00 00 00 00	
800207A4	00 00 00 00	
800207A8	00 00 00 00	
800207AC	00 00 00 00	
800207B0	00 00 00 00	
800207B4	00 00 00 00	</