IT3280E
Assembly Language and Computer Architecture Lab

Course Introduction

- Course: IT3280E 2(0-4-0-4)

Assembly Language and Computer Architecture Lab

- Semester 2024.1
- Lecturer: Phạm Ngọc Hưng
  - hungpn@soict.hust.edu.vn
  - B1-802, Department of Computer Engineering
- TA:
  - Nguyễn Đăng Phúc Hưng

- Evaluation:
  - Progress Evaluation (0.3) = Lab exercises (50%) + Mid-term project (50%)
  - Final-Term Evaluation (0.7) = Final Project (50%) + Final Exam (50%)

Laboratory Exercises

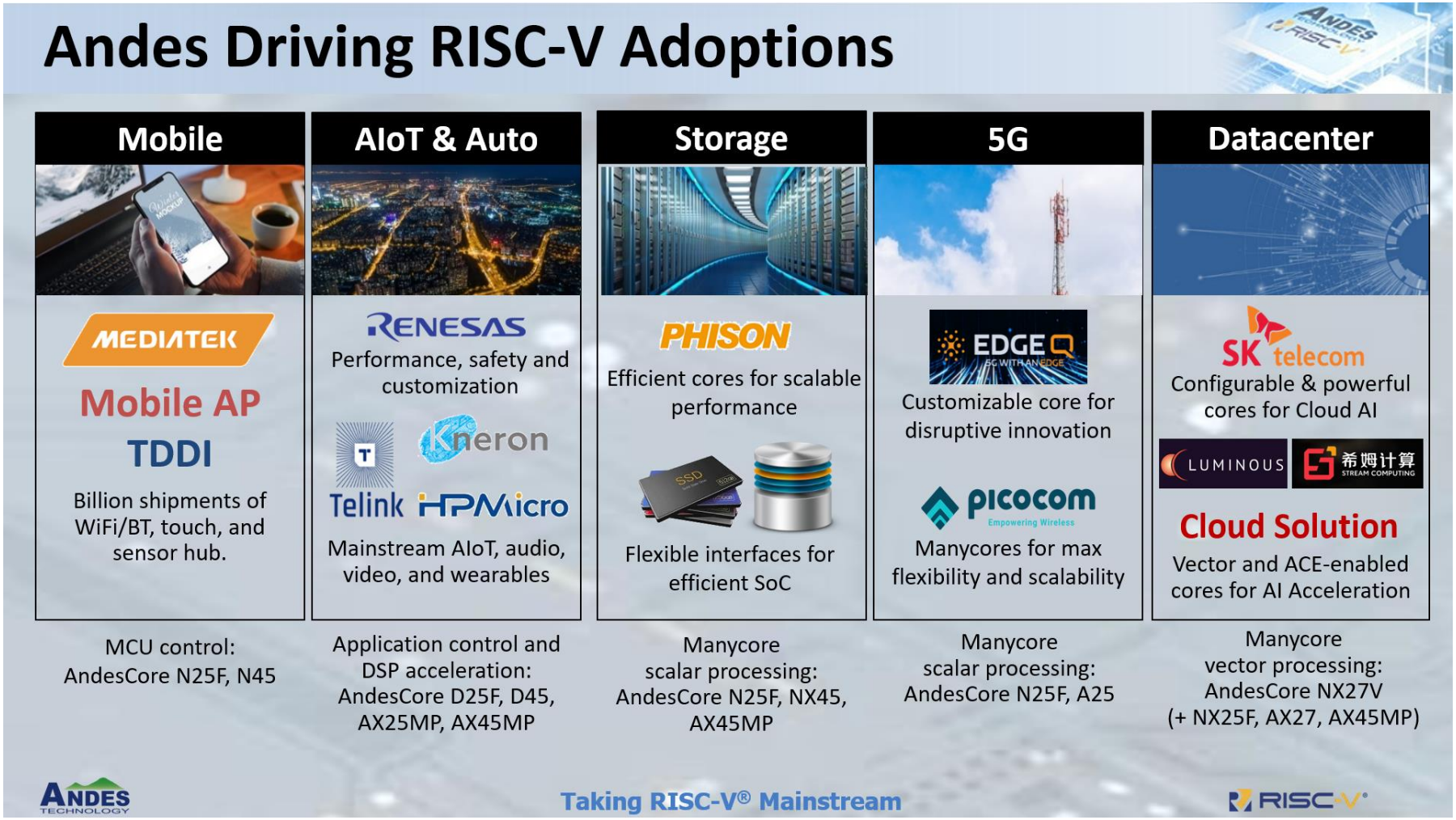
- **Lab 1.** Introduction to RISC-V, RARS
- **Lab 2.** Instruction Set, Basic Instructions, Directives
- **Lab 3.** Jump & Branch instructions
- **Lab 4.** Arithmetic and Logical operations
- **Lab 5.** Character string with ECALL functions
- **Lab 6.** Array and Pointer
- **Lab 7.** Procedure calls, parameters and using stack
- **Lab 8-9.** Mini-Project (
- **Lab 10.** Control Peripheral Devices via Simulator
- **Lab 11.** Interrupts & IO programming
- **Lab 12.** Cache Memory
- **Lab 13.** Assembly programming with ESP32-C3 using wokwi
- **Lab 14.** ESP32-C3 using breadboard
- **Lab 15-16.** Final Project

LAB 1
Introduction to RISC-V and RARS

1. Introduction to RISC-V

- RISC-V is an open standard **instruction set architecture** (ISA) based on established **reduced instruction set computer** (RISC) principles.
- The project began in 2010 at the University of California, Berkeley, transferred to the RISC-V Foundation in 2015, and on to RISC-V International in November 2019.

RISC-V Market (2022)

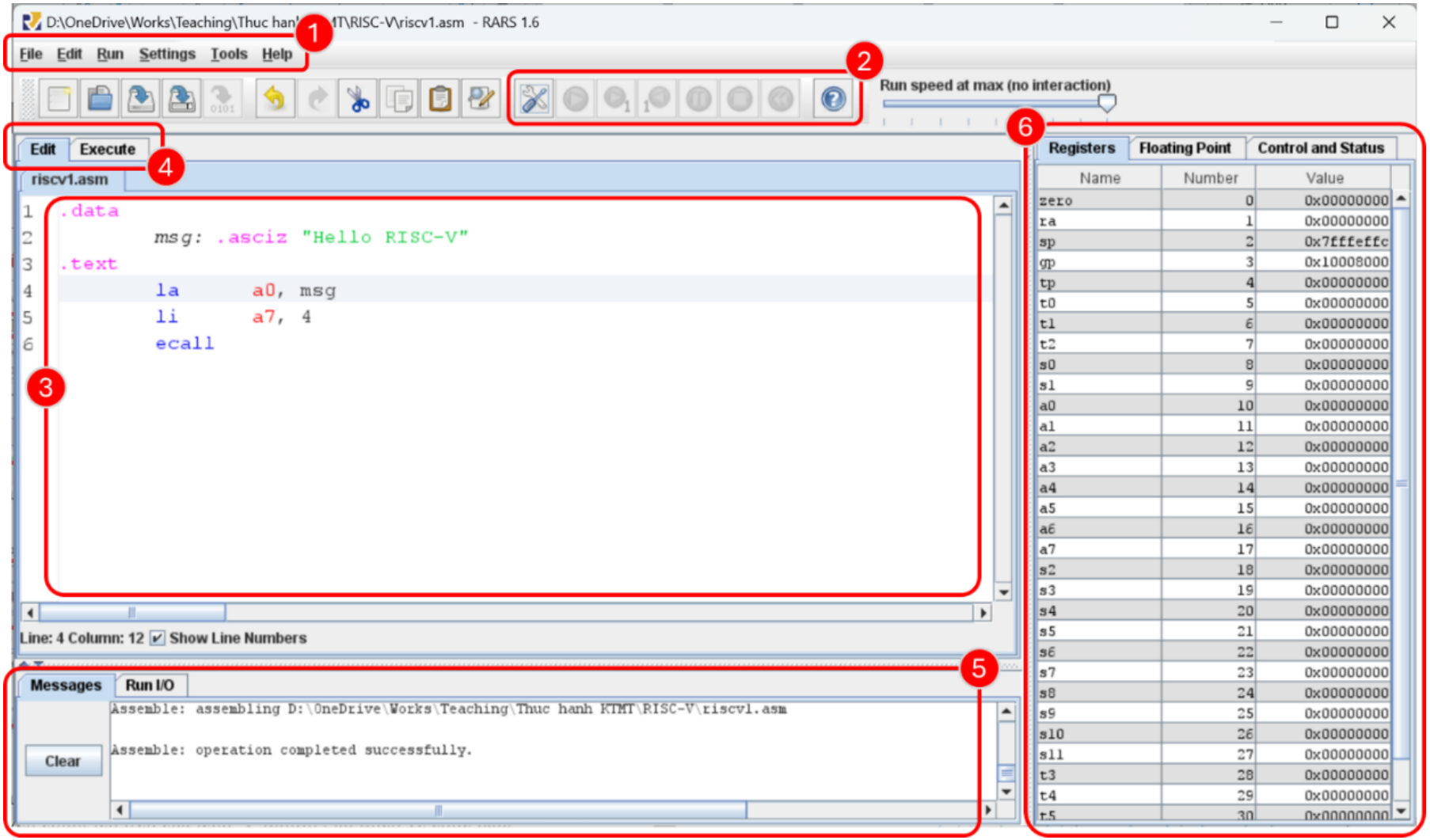


2. RISC-V Simulation – RARS

- RARS – RISC-V Simulation
- RISC-V assembly program

RARS simulation

- RARS IDE



RISC-V assembly program: HelloWorld

```
.data # Data segment (Variable declaration)
x:.word 0x01020304 # word variable x with initialization
msg:  .asciiz  "Hello RISC-V"
.text                # Code segment
   la   a0, msg   #load address of msg to register a0
   li   a7, 4     #Assign a7 is 4
   ecall          #Call system function to print out
string
   addi t1, zero,2 #Assign t1 = 2
   addi t2, zero,3 #Assign t2 = 3
   add  t0, t1, t2 #Assign t0 = t1 + t2
```

RISC-V assembly program: HelloWorld

Edit | Execute

lab1Hello.asm

```
.data    # Data segment (Variable declaration)
x:.word 0x01020304 # word variable x with initialization
msg:  .asciiz  "Hello RISC-V"
.text              # Code segment
   la   a0, msg    #load address of msg to register a0
   li   a7, 4      #Assign a7 is 4
   ecall           #Call system function to print out string
   addi t1, zero,2 #Assign t1 = 2
   addi t2, zero,3 #Assign t2 = 3
   add  t0, t1, t2 #Assign t0 = t1 + t2
```

RISC-V assembly program: HelloWorld

D:\LECTURE\IT3280_ThuchanhKTMT\LearningMaterial2024\lab1\lab1Hello.asm - RARS 1.6

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x0fc10517	auipc x10,0x0000fc10	5: la a0, msg #load address of msg to register a0
	0x00400004	0x00450513	addi x10,x10,4	
	0x00400008	0x00400893	addi x17,x0,4	6: li a7, 4 #Assign a7 is 4
	0x0040000c	0x00000073	ecall	7: ecall #Call system function to print out string
	0x00400010	0x00200313	addi x6,x0,2	8: addi t1, zero,2 #Assign t1 = 2
	0x00400014	0x00300393	addi x7,x0,3	9: addi t2, zero,3 #Assign t2 = 3
	0x00400018	0x007302b3	add x5,x6,x7	10: add t0, t1, t2 #Assign t0 = t1 + t2

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Valu
0x10010000	0x01020304	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0

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

Messages | Run I/O

Clear

```
Warning in D:\LECTURE\IT3280_ThuchanhKTMT\LearningMaterial2024\lab1\lab1Hello.asm line 3 column 7: RARS does not
Assemble: operation completed successfully.
```

Control and Status

Registers | Floating Point

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400000

LAB 2

Instruction Set, Basic Instructions, Directives

LAB 3

Load/Store, Jump and Branch Instructions

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

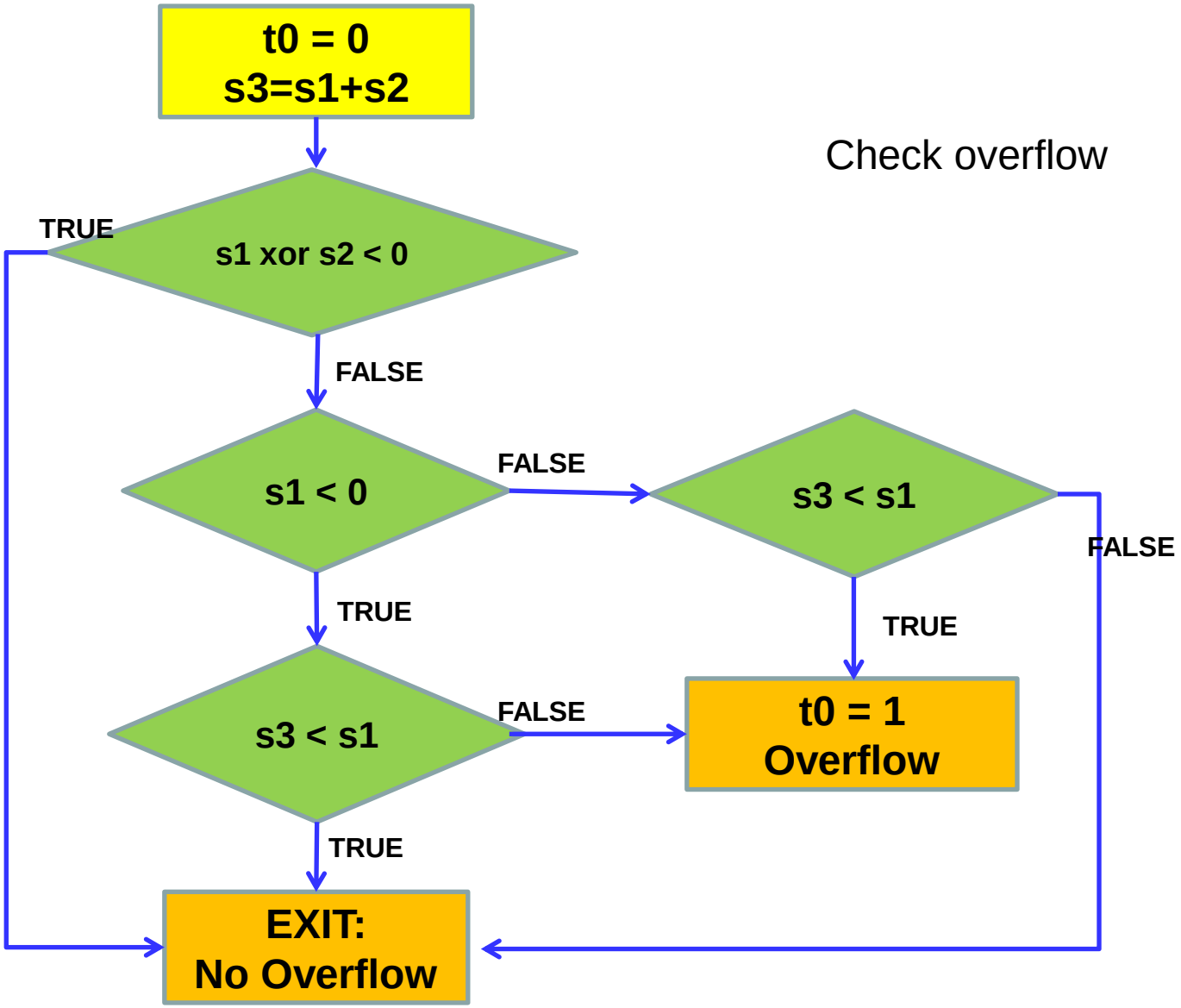
```
switch(test) {
   case 0:
        a=a+1; break;
   case 1:
        a=a-1; break;
   case 2:
        b=2*b; break;
   default:
}
```

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:
```

LAB 4
Arithmetic and Logical Operations

Check overflow



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

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 ECALL function

LAB 6

Array and Pointer

LAB 7

Procedure calls, stack and parameters

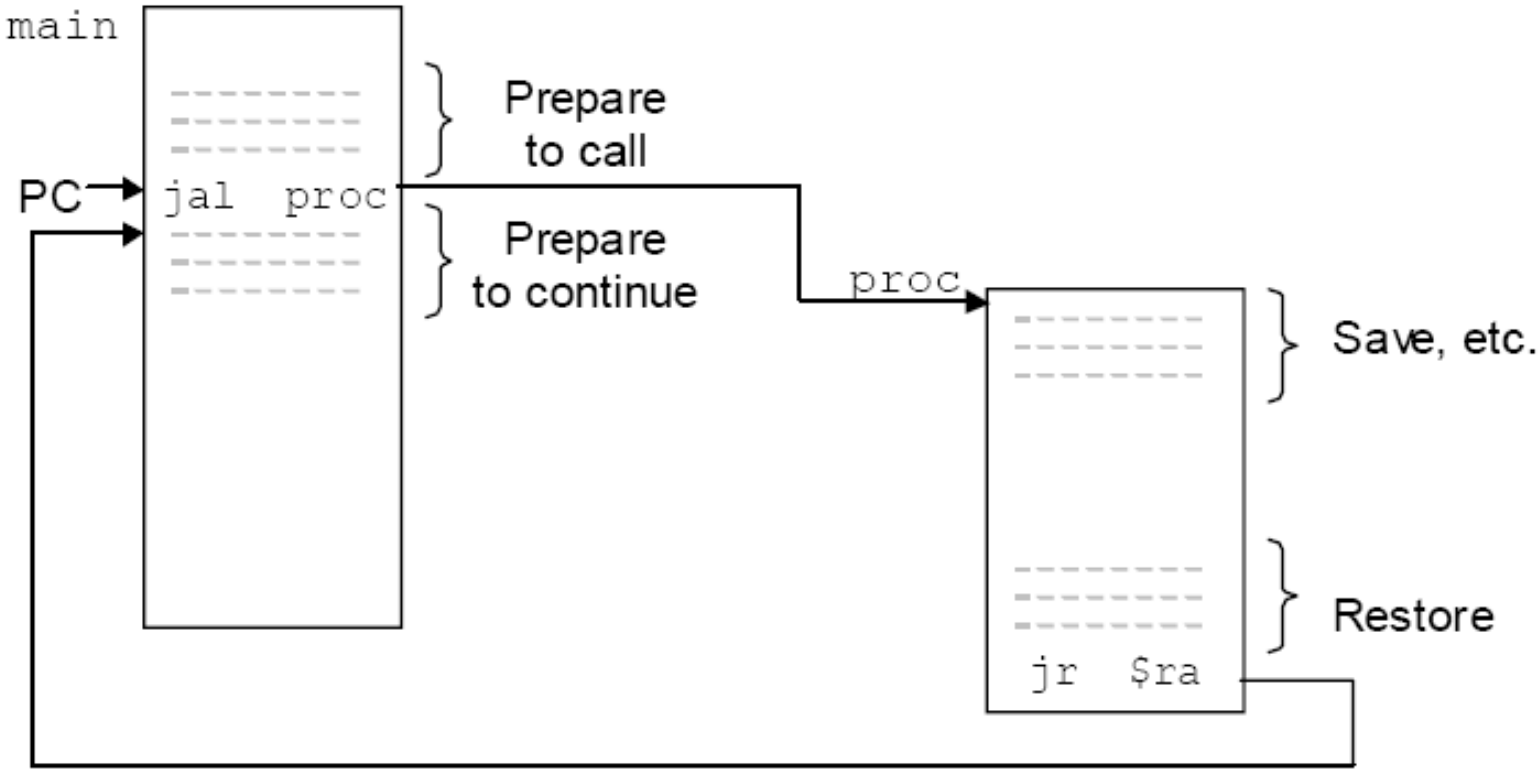
Procedure & Stack

- Procedure call:

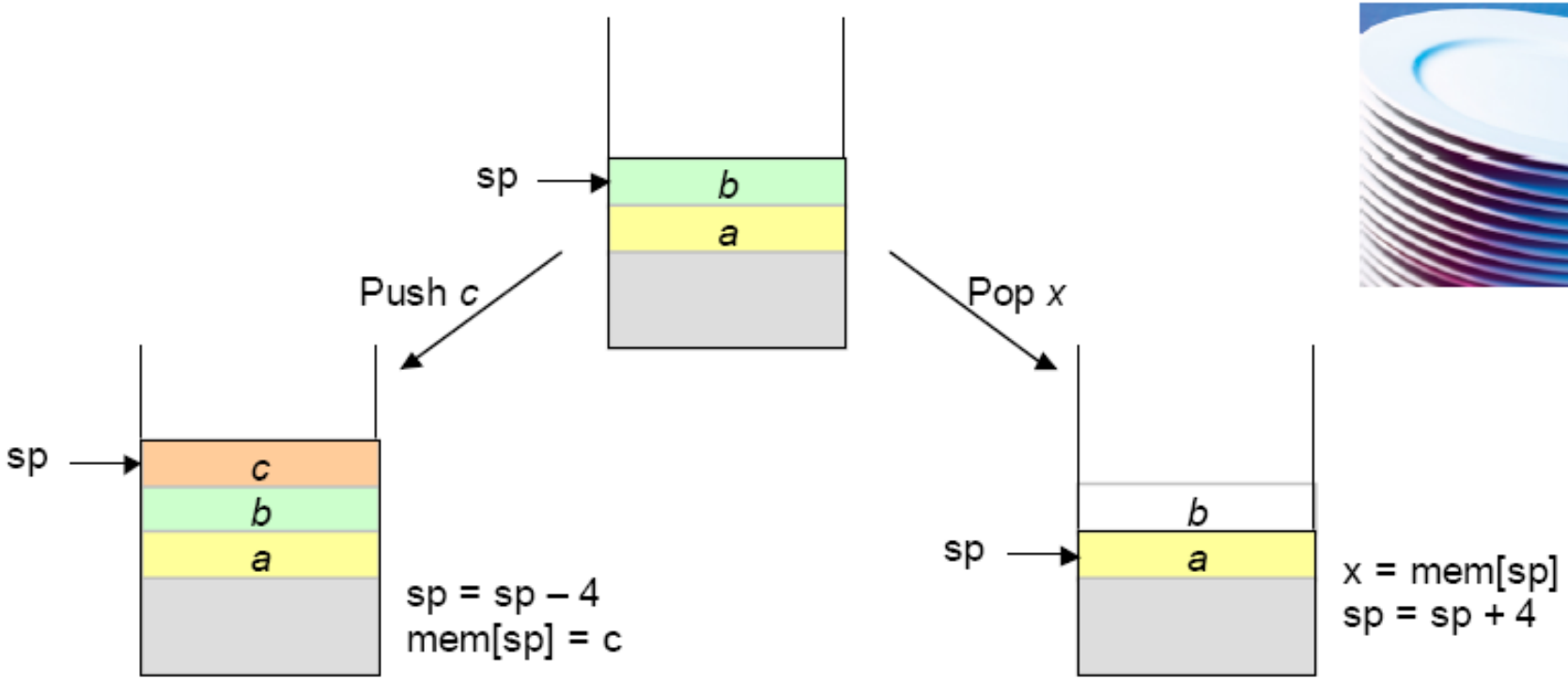
```
jal ( jump and link)
```

- Return to call point

```
jr $ra
```



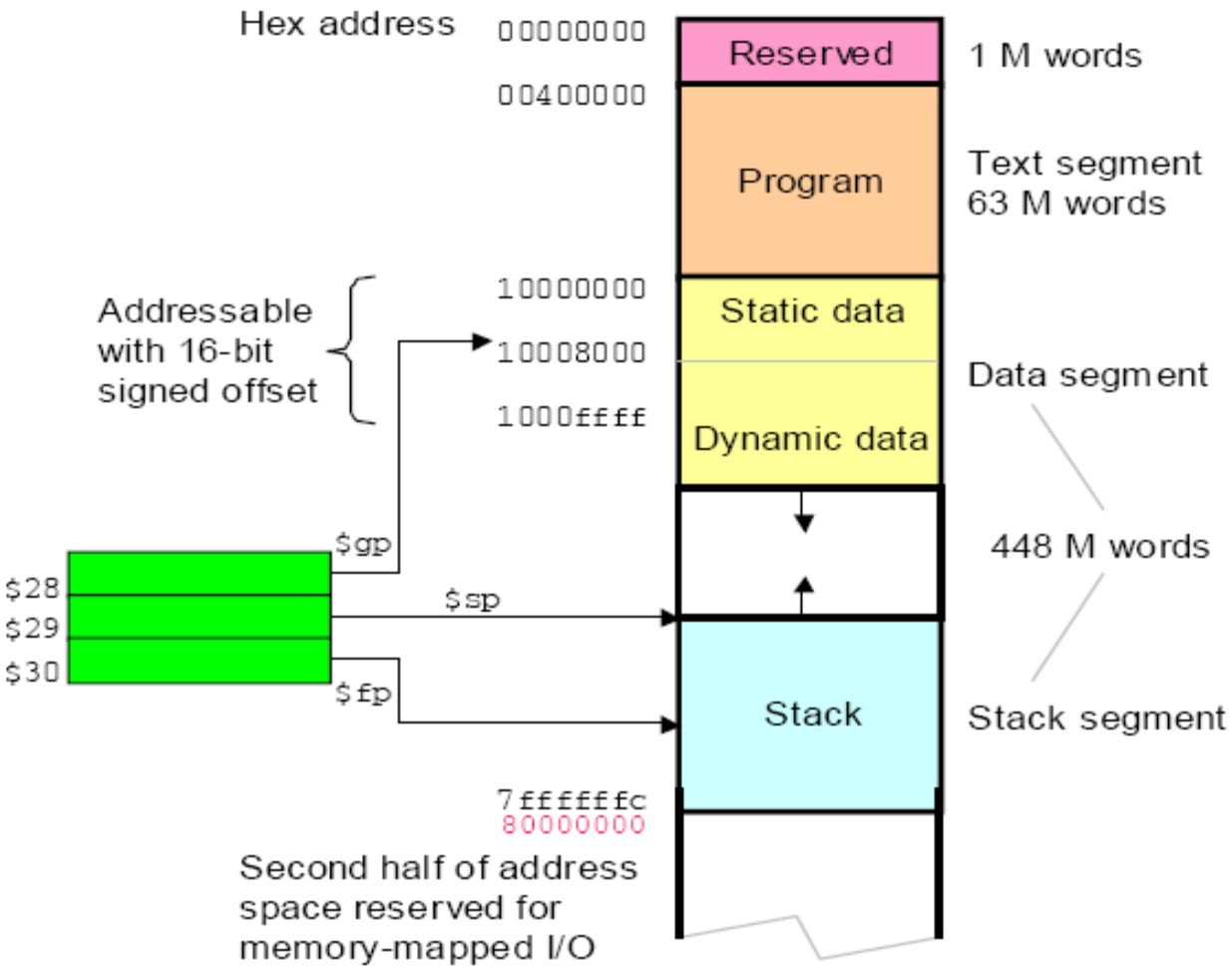
Stack



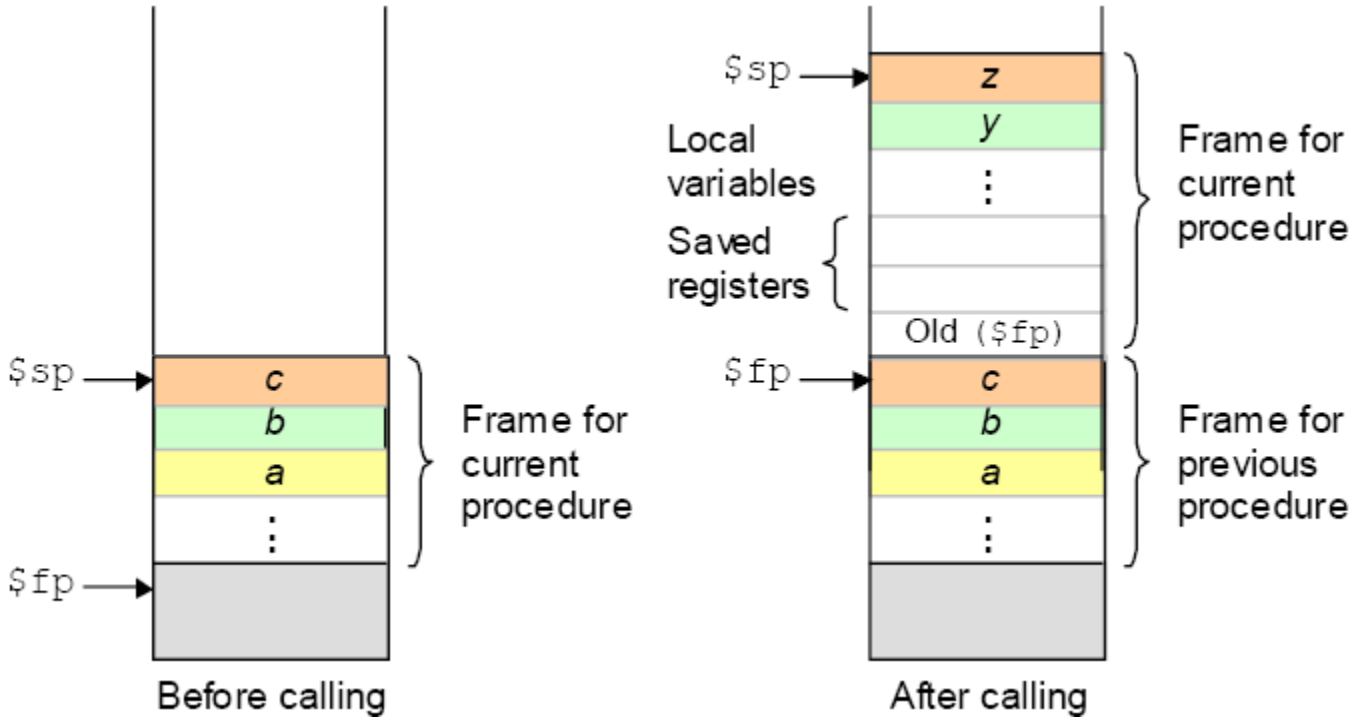
```
push: addi   $sp,$sp,-4
      sw     $t4,0($sp)
```

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

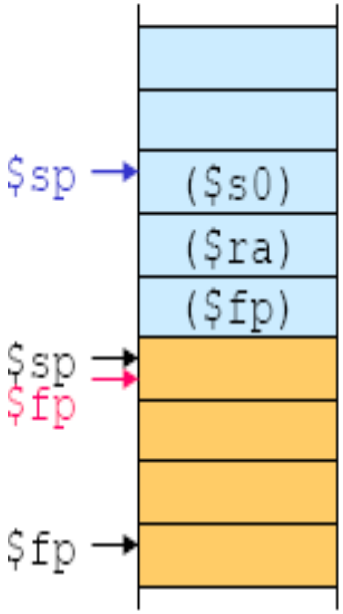
Stack


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

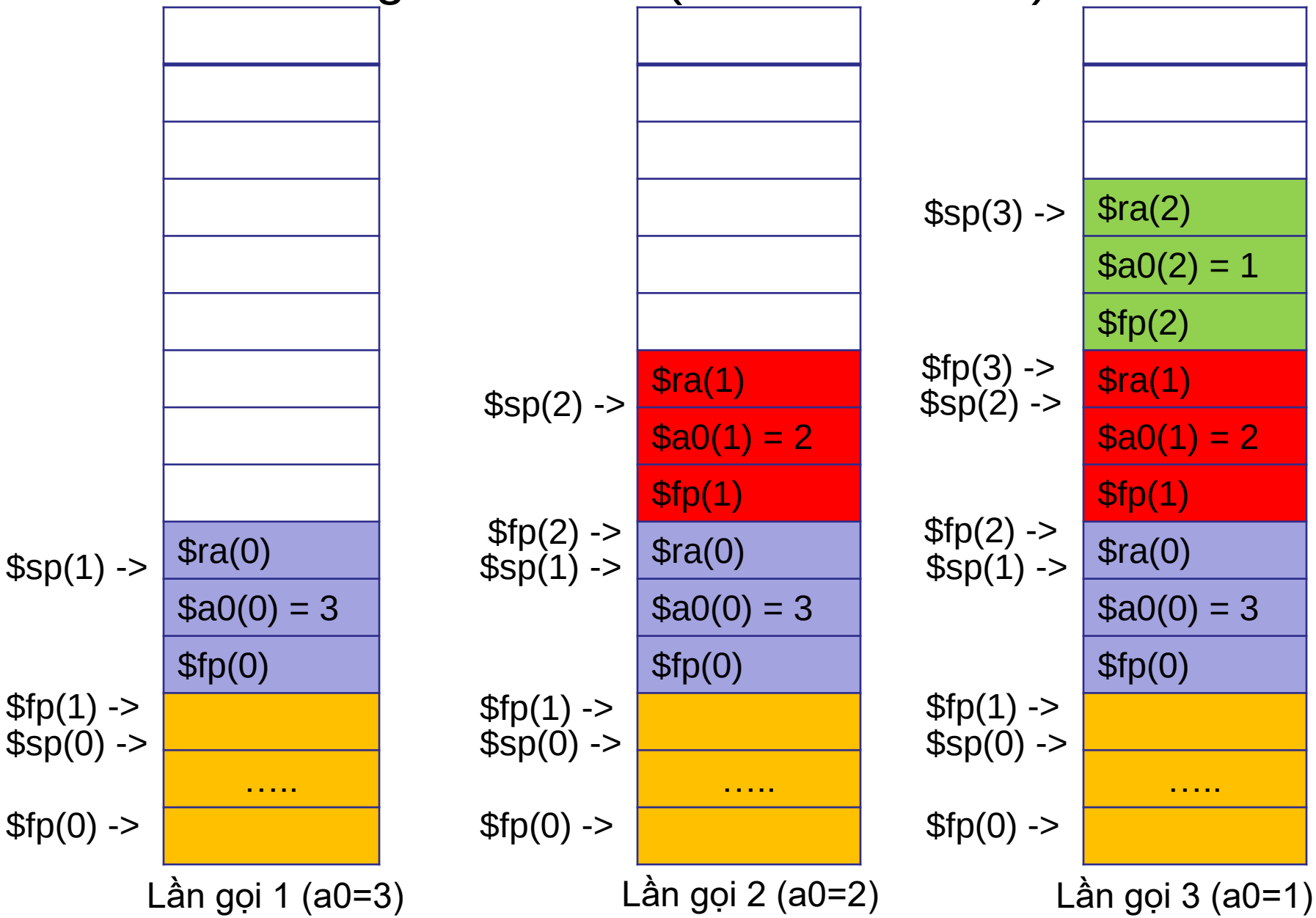

$sp
c
b
a
$fp
Frame for current procedure
Before calling
$sp
z
y
Local variables
Saved registers
Old ($fp)
$fp
c
b
a
Frame for current procedure
Frame for previous procedure
After calling

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
```

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