Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

Computer Architecture

Practical session - Week 4 - Semester 1 - 2020

Notes:

- The main purpose of this week is to practice about the conditional branch and unconditional jump instructions. Students also learn about how MIPS programs control Procedural calls.
- Students compress the assembly files then submit on e-learning.

Question 1. Write a MIPS program with the following steps:

1. Request an integer number from users.
2. If the number is positive, repeat step 1. Otherwise, print sum of all integer numbers that the program has read from users.

Question 2. Implement the following C code by using MIPS code. Assume that b and c are 10 and 5, respectively while input variable is read from keyboard. Print value of a to the terminal.

```
switch (input){
 case 0: a = b + c; break;
 case 1: a = b - c; break;
 case 2: a = c -b; break;
 default: printf{"please input an another integer numbers\n"}; break;
}
```

Question 3. Write a MIPS program with the following requirements:

1. Declare an integer array with 10 synthetic data elements.
2. Read an integer number from users.
3. Find in the array if the integer read from user exists in the array or not. Print the position of the integer number in the array if found; otherwise tell users that the number does not exist in the array.

Question 4. Given the following leaf procedure in ANSI C

```
void swap(int v[], int k)
{
int temp;
temp = v[k]
v[k] = v[k+1];
v[k+1] = temp;
}
```

Assume that the `$a0` register will store the base address of the `v` array while the `$a1` register keeps value `k`. The array `v` consists of 10 elements in integer and is pre-defined in the data section.

1. Write a main program the receive value `k` from user, check the value `k` and call the procedure swap if possible.
2. Watch the `$ra` register before and after the `jal` and `jr` instructions are executed.

Question 5. Given the following factorial MIPS program in a recursive form (as in the slide)

```
fact: addi $sp, $sp, -8      # adjust stack for 2 items
sw   $ra, 4($sp)       # save return address
sw   $a0, 0($sp)       # save argument
slti $t0, $a0, 1       # test for n < 1
beq  $t0, $zero, L1
addi $v0, $zero, 1     # if so, result is 1
addi $sp, $sp, 8       #   pop 2 items from stack
jr   $ra               #   and return
L1:  addi $a0, $a0, -1      # else decrement n
jal  fact              # recursive call
lw   $a0, 0($sp)       # restore original n
lw   $ra, 4($sp)       #   and return address
addi $sp, $sp, 8       # pop 2 items from stack
mul  $v0, $a0, $v0     # multiply to get result
jr   $ra               # and return
```

1. Type the above procedure and write a main program that call the above procedure with different `n`, where `n` is in the `$a0` register. Watch the results
2. When `n` is 2, run the program step by step and watch the execution of instructions as well as the `$ra` register and values store/load to/from the stack.

——————the end——————