REPORT MINI PROJECT

Assembly Language and Computer Architecture Lab

Name: Nguyen Trung Thanh **ID: 20163727**

Name: Ha Minh Duc **ID: 20161067**

Class: ICT - 02 K61

Teacher: Mr.Pham Ngoc Hung

Problem 10: Write a program that gets an integer i from the user and creates the table shown below on the screen (example inputs provided). Subroutines are required for power, square, and hexadecimal (in 32 bit arithemtic, attend to overflowed results).

Hint: Hexadecimal can be done with shifts and masks because the size is 32 bits.

In my problem, I come up with the idea that creating 3 procedures (Square, Power2, Convert_to_hexa). The first step is ask an input from user. The input must be an integer, if the input is invalid (a string, character, float, or nothing was entered), I make a loop for repeating ask user for entering the input. The algorithms I used is simple. We start with the Square procedure, due to its simple I just multiply the input number to itself for getting the result. Next thing is the Power2, I used the recursive case to calculate the power 2 . If input number is 0 return 1, if input number is 1, return 2, else return 2 * Power2(i – 1). The last procedure is convert decimal to hexadecimal that I have to use “rol” (rotate left) in a loop to move every 4 bits to the last for masking with 0xf (1111). If the result <= 9, I will add it to 48 which is the position in ASCII of number, else if the result > 9, I will add it to 55 which in ASCII must be something like (A, B, C, D, F). The final output is the table which show 4 things (I, square(i), power2(i), hexadecimal(i).

*** Detail explanation:**

- First Loop: Ask user to input integer with a dialog. The error will be catch and store in the $a1. If a1 = 0 (valid) we will start calculating the result, else if a1 = -2 which means the input was cancel, else the error is input not valid. In this case, I will jump back to this branch (looping) for asking input from user again

- mult is the instruction I used for calculating the multiple of 2 numbers which may be overflowed. The result will be save to 2 register hi and lo, for the small number I just used mflo to move the result from lo to v0 to display

- convert_dec_to_hex: the function to convert to hexadecimal by receiving the parameter $v0 and save it to register $t2. I begin a loop which will stop when t0 = 0, otherwise it will using “rol” to rotate every 4 bits to the “left”. The purpose is that we convert by every single number (represented by 4 bits) masking with 0xf (1111) using “and” instruction. After masking, we add it to 48 if it <= 9 or add it to 55 in other case. The final result will be save to $t3 then save to constant result for easy use later.

- findPower2: is a recursive procedure, we need to initialize the stack pointer for saving return address and $s0. Because the stack go down, so for initializing we need to subtract it by 8 bits (mean 2 numbers 4 bits). For simple base case, I return v0 = 1 if the parameter a0 = 0, v0 = 2 if the parameter a0 = 1. Otherwise, I return 2 * findPower2(i – 1) by using “jal” findPower2 after subtracting the parameter a0 by 1. Finally, I recovered the return address, saving value, and stack pointer and return to the main with result is saved to v0

- **Meaning of registers used:**

$v0: save the system call code

$a0: store the value I want to print out, or parameter for procedure

$v0: store the result of each procedure

$t2: store temp bits after rotating

$t4: store the result after masking

$t3: save the address of result

$sp: stack pointer for saving value to use recursive procedure

$ra: return address use to back to the outer context

*** Detail explanation of some parts:**

- Loop1: Ask user to input float as dialog, the error code will be saved in $a1 register then we check if $a1 = 0 mean no error occur so branch to endLoop1. If $a1 != 0 mean errors occur so we show the dialog to remind users. We continue the loop until user's input is acceptable.

- mfc1: We use this instruction to move the value of user's input in coprocessor1 ($f0 register). This value is represented as IEEE754 value. If using this, we can deal with decimal numbers which are not integer.

- set_counter: the function to set the counter to $t0 register by adding $a1 and $zero with $a1 is a register which store the number of bits to rotate.

- **Meaning of registers used:**

$a0: keep the message and result after converted

$a1: store the counter

$a2: store number of bits to rotate

$a3: store the number to "mask" with

$t3: save the address of result

$t0: counter using as function

$t4: save the result after "mask"

Problem 8: Write a program to:

- Input the number of students in class.

- Input the name of students in class, mark

- Sort students due to their mark.

*** Preparation:** In this problem, I assume that a class has at most 20 students and a student's name will not exceed 20 letters. A student's name and mark will be stored in 2 different "arrays" with the same index. Therefore, each time a change

occur (like swapping when sorting) in one array, the other must adjust the same. With that in mind, I allocated memory in **.data**:

- nb_student: will store the number of student in the class, this value will be used in all procedure so storing it in .data is convenient.

- name_list: stores the list of names.

- mark_list: stores the list of marks.

- string: 3 purposes:

    + Holds the input string from input dialog.

    + Stores the string that will be printed out to the console.

+ Acts as temporary variable for string swapping.

*** Sub procedure:**

- input_num: get number of student

    + Occupy: v0, a0, a1, a2, a3.

    + Argument: none.

    + Return value: number of student in nb_student.

    + Note: user have to input again if they enter: nothing, not a number, negative number; pressing cancel will quit the program.

- input_name: get student’s name from input dialog

    + Occupy: v0, v1, a0, a1, a2.

    + Argument: none.

    + Return value: student’s name in string.

    + Note: user have to input again if they enter: nothing, more than 20 letters; pressing cancel will quit the program.

- input_mark: get student’s mark from input dialog

    + Occupy: v0, v1, f0, a1.

    + Argument: none.

    + Return value: student’s mark in f0.

+ Note: user have to input again if they enter: nothing, not a floating point number; pressing cancel will quit the program.

- name_cpy: copy a string from a1 to a0

+ Occupy: s0, s1, t1, t2, t3.

+ Argument: destination address at a0, source address at a1.

+ Return value: none.

- print_student: print student's name and mark to console

+ Occupy: v0, a0, f12.

+ Argument: name in string, mark at f12.

+ Return value: none.

* **Main program:** I divided it into 4 step

- Step 1 - input_num: get the number of student in the class and stores it in nb_student.

- Step 2 – get_data: a loop will start from index 0 and end at last index (nb_student – 1), each loop will call 2 procedures input_name and input_mark. After 2 procedures, name will be stored in string and mark will be stored in f0. Next, I save these values to their corresponding index in 2 "arrays" (name_cpy is used to copy name from string to name_list).

- Step 3 – bubble_sort: I perform bubble sort on mark array as follow:

```
i = 0
while i < length – 1:
    i++
    j = -1
    while j < i – 1:
        j++
        if mark[j] > mark[j + 1]:
            swap mark[j], mark[j + 1]
            swap name[j], name[j + 1]
```

Swapping mark is straight forward as I just need to save their value in corresponding index in mark_list. However, swapping name have to performed as follow:

+ Using name_cpy, copy name[j] to string.

+ Using name_cpy, copy name[j + 1] to name[i].

+ Using name_cpy, copy string to name[j + 1].

- Step 4 – print_all: print all students and their mark to the console to prove that they have been sorted by their mark. A loop will start from index 0 and end at last index (nb_student – 1). In each loop, copy name from name_list into string, mark to f12 and call print_student.

*** Register used:**

- v0: system call code

- v1: set to -2 (the status return when user press cancel on input dialog)

- a0: hold address of data loaded from .data, name swapping destination

- a1: name swapping source, status returned from input dialog

- a2: set to 20 (max length of a name) to use in comparison, hold address of loaded data when a0 and a1 are occupied

- a3: hold address of loaded data when a0, a1, a2 are occupied

- t1, t2, t3: used only in name_cpy, copy a single byte from source to dest

- t4, t5: used in the loop of get_data and print_all, hold the offset of current index

- s0: current index of a loop

- s1: last index of a loop

- s3: current index of a loop, used when s0 is occupied

- s4: last index of a loop, used when s1 is occupied

- s5: last index of a loop, used only in j loop of bubble sort

- s6: current index of a loop, used only in j loop of bubble sort

- f0: floating point value get from input dialog

- f12: hold floating point value to be printed out to the console