FIT2085-S1-2021
Interview Prac 1

Guidelines

A. Faithfulness

The main idea for faithfulness is to translate **every line independently of each other**, and to translate it **as given in the high-level code**. That is:

1. Load and store variable values **every** time from memory, rather than reusing from registers used for previous python lines.
2. Ensure each line is translated in its place (for example, if line i += 1 appears as the last line of a while loop, make sure it is the last one before the jump back to the loop).
3. Encode globals as globals, and locals as locals (e.g., if I tell you a variable is assumed to be global, then declare it as such in MIPS). If a global python variable is not initialised to a constant value, initialise it to 0 in MIPS.
4. Encode strings exactly as they are given (i.e., don't add or subtract characters to it).
5. Encode read/prints exactly as they are given (do not hard-code values if the values are read from screen).
6. In the simple case where the expression in an if and elif has no or or and, then if-thens need to be translated as if-thens (one branch, one label). If-then-elses as if-then-elses (one branch, two labels, one jump). If-then-elif-else as such (two branches, three labels, two jumps), etc. In the more complex case with or and/or and , more branches, labels and/or jumps may be necessary to evaluate the expression lazily (see 8.).
7. Encode the >, < conditions in the loops (a > 0, x < 0, etc) exactly as they are given. For condition with >=, <= (like x >= y, a <= b), which only exists in MIPS as a pseudoinstruction (which you are not allowed to use), you must use x < y and a > b and negate the answer.
8. Translate boolean expressions lazily (as python does):
    - For a Cond1 and Cond 2 condition (in if-then or in a loop) you must test first Cond1 and if it fails, go directly to the else. Then test Cond2 and if it fails, also go to the else.
    - For a Cond1 or Cond2 condition you must add a "then" label such that if Cond1 is true you go directly to the then. Then test Cond2 and if false go to the else (otherwise keep executing to the "then").

B. Constraints and assumptions

1. Do not simplify or reorder boolean statements (e.g. in an if) to equivalent (or non-equivalent) statements using boolean logic or arithmetic.
2. Translate any Python list with nnn elements as a MIPS array of size n+1n+1n+1 words, where the first word contains the size nnn (as shown in the lesson videos).

3. You do not have to initialise the values inside a dynamically-allocated array (except for the size of the array, stored at the first position).
4. No need to check for arithmetic overflow unless we ask for it.
5. Do not create helper functions (e.g. to print).
6. Make sure you follow the correct **function calling convention** as described on the MIPS reference sheet.
7. Upon termination, MARS prints an extra new line character, a behaviour which we will disregard for the purpose of this unit. In other words, if Python prints a newline character at the end of the program, so should your MIPS code.

Task 1

Consider the following (quite silly) Python code:

```
first = int(input("Enter first: "))
second = int(input("Enter second: "))

if first > 0 and second >= 0:
    result = second // first
elif first == second or first < second:
    result = first * second
else:
    result = second * 2

print("Result: " + str(result))
```

Faithfully translate the above code into a properly commented MIPS program using the task1.asm file provided.

Task 2

Consider the following Python code, which, given a Python list, computes the minimum element and prints it.

```
size = int(input("Array length: "))
the_list = [None] * size

for i in range(len(the_list)):
    the_list[i] = int(input("Enter num: "))
    if i == 0 or min_item > the_list[i]:
        min_item = the_list[i]

print("The minimum element in this list is " + str(min_item))
```

Faithfully translate the above Python code into properly commented MIPS code using the task2.asm file provided. Note that all variables are global.

Constraints and assumptions

- Assume the size of the list will always be positive.

Task 3

Consider the following Python code, which is a functional version of your Task 2 code.

```
from typing import List


def get_minimum(the_list: List[int]) -> int:
    """Computes the minimum element of a given list of integers.
    :pre: the list has at least one element
    """
    min_item = the_list[0]
    for i in range(1, len(the_list)):
        item = the_list[i]
        if min_item > item:
            min_item = item
    return min_item


def main() -> None:
    my_list = [2, 4, -1]
    print("The minimum element in this list is " + str(get_minimum(my_list)))


main()
```

Faithfully translate the above code (ignoring type hints) into a properly commented MIPS program using the task3.asm file provided.

Constraints and assumptions

- You may reuse code you wrote for Task 2.
- In the MIPS program, the size of the array will be stored both in size and as the first element of the array. Both must hold the correct value, and you can use either.

Task 4

Consider the following Python code, which defines the function bubble_sort(the_list), which sorts the_list in non-decreasing order:

```
from typing import List


def bubble_sort(the_list: List[int]) -> None:
    n = len(the_list)   # assigns the size of the_list to n
    for a in range(n-1):    # loops through number of traversals
        for i in range(n-1):    # loops through each element in the_list
            item = the_list[i]  # item = current element
            item_to_right = the_list[i+1]   # item_to_right = next element
            if item > item_to_right:    # if item > item_to_right, swap them
                the_list[i] = item_to_right  # item_to_right now located at
index i
                the_list[i+1] = item  # item now located at index i+1


def main() -> None:
    """ Calls bubble_sort() and prints each item in the newly sorted list """
    the_list = [4, -2, 6, 7]
    bubble_sort(the_list)

    for i in range(len(the_list)):
        # Print the item without a newline character
        print(the_list[i], end='')
        print(' ', end='')
    print()


main()
```

You are asked to do two subtasks:

- Properly document the task4.py file provided, and add line by line comments to the bubble_sort() function above. Your comments should explain what each line does.
- Faithfully translate the above code (ignoring type hints) into a properly commented MIPS program using the task4.asm file provided.