A1 - Individual

Assessment Overview

Objective

To be able to write MIPS programs and faithfully translate simple Python programs into MIPS, involving decisions, while and for loops, lists (treated as arrays), local variables, and functions.

Submission, Format, and Expectations

Please submit your assignment as a GitHub repository. You will need to clone our sample repository and then edit the files to add your solution code to it.

Important

- Local variables must be stored in the runtime stack.
- Use only MIPS instructions that appear in the MIPS reference sheet.
- Use the function names provided, as the testing relies on those function label names.
- A detailed description of what we mean by "faithfully" is described in the guidelines.

Task 1

The aim of this task is to assess your knowledge in implementing in MIPS different decisions (>=, <=, < and >), complex if-then-else, and in using shift instructions to multiply or divide, as needed.

Task 2

The aim of this task is to assess your knowledge of implementing simple loops in MIPS and accessing arrays. To simplify the MIPS code, we have used `for i in range` rather than `for item in the_list`.

Task 3

The aim of this task is to assess your knowledge of defining and calling functions in MIPS.

Task 4

The aim of this task is to assess your ability to use all your MIPS knowledge (conditions, loops, array access, function definition, etc) together. It's also an introduction to bubble sort, which you will need for Week 4.

Task 5

The aim of this task is to assess your ability to apply what you have learned in MIPS so far into a recursive-style program.

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 registers used for previous python lines.
2. Ensure each line is translated in its place (for example, if the 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 initialized to a constant value, initialize it to 0 in MIPS.
4. Encode strings exactly as they are given (i.e., don't add or subtract characters to them).
5. Encode read/prints exactly as they are given (do not hard-code values if the values are read from the screen).
6. In the simple case where the expression in an if and elif has no `or` or `and`, then if-then need to be translated as if-then (one branch, one label). If-then-else as if-then-else (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 conditions with >=, <= (like x >= y, a <= b), which only exists in MIPS as a pseudo-instruction (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 n elements as a MIPS array of size $n + 1$ words, where the first word contains the size n (as shown in the lesson videos).
3. You do not have to initialize 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 newline character, a behavior 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.

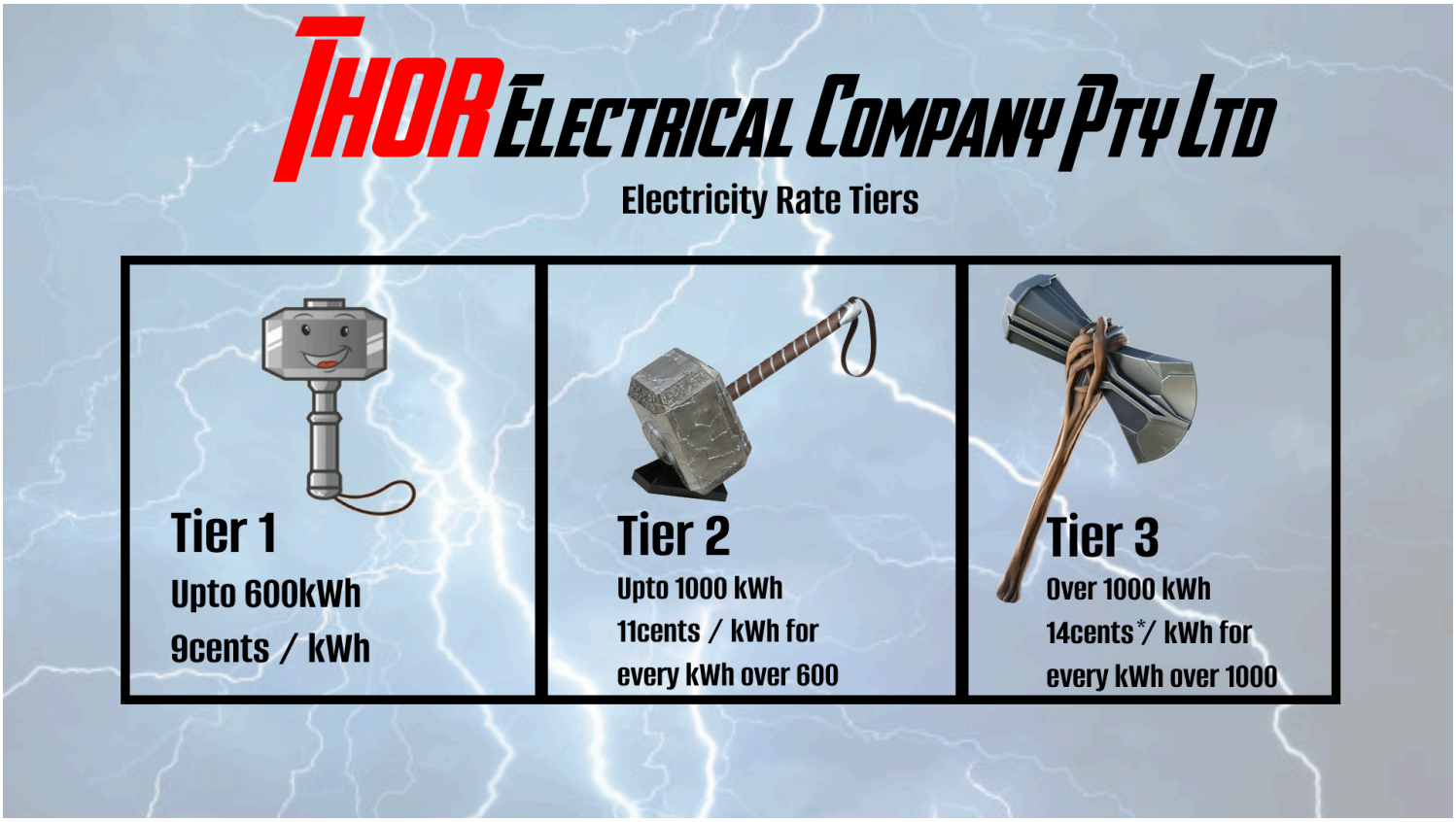
task 1 - thor odinSon

The world hasn't been too kind to the mighty God of Thunder. New Asgard is in peril and all the mighty Thor can do is day drink.

Valkyrie thought of ways to help finance the town and she made Thor start an electricity company to generate some revenue.

We are going to be working with a program that calculates a person's energy bill when they are subscribed to Thor's Electrical company. They use three tiers to charge people, based on their electricity consumption. Furthermore, a standard 10% GST rate is applied to the electrical charge.

The company has the following process to calculate it:



*** Please note that Thor is a philanthropist and even though it costs revenue, he decided to give minors and veterans a 2 cent discount per kWh for Tier 3.**

Same thing in a table:

A sample python program has been given for your reference.

```
"""
This file serves as an example Python code for Task 1 Solo Prac 2
"""
__author__ = "Saksham Nagpal"
__date__ = "18.05.2022"

tier_one_price = 9
tier_two_price = 11
tier_three_price = 14
discount_flag = 0

print("Welcome to the Thor Electrical Company!")
age = int(input("Enter your age: "))
if age <= 18 or age >= 65:
    discount_flag = 1
else:
    discount_flag = 0
```

```
consumption = int(input("Enter your total consumption in kWh: "))
total_cost = 0

if consumption > 1000 and discount_flag == 0:
    total_cost = total_cost + ((consumption-1000)*tier_three_price)
    consumption = 1000
elif consumption > 1000:
    total_cost = total_cost + ((consumption-1000) * (tier_three_price - 2))
    consumption = 1000

if consumption > 600:
    total_cost = total_cost + ((consumption-600)*tier_two_price)
    consumption = 600

total_cost = total_cost + (consumption*tier_one_price)
gst = total_cost // 10

total_bill = total_cost + gst
print(f"Mr Loki Laufeyson, your electricity bill is ${total_bill // 100}.{total_bill % 100}")
```

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

Constraints and assumptions

- Assessment-wide constraints and assumptions.

Helpful stuff

You can assume that the consumption will always be less than 2^{30}

An example of a run is:

```
Welcome to the Thor Electrical Company!
Enter your age: 19
Enter your total consumption in kWh: 200
Mr Loki Laufeyson, your electricity bill is $19.80
```

task 2 - rebuilding the Tower

So it is 2012, NYC has been attacked by Loki and after defeating him (and destroying half of the city in the process), the civil engineers of New York have been paid by Tony to rebuild the Avengers tower. Like all "*good*" engineers, they have to draft a plan on code and they decide to do a primitive build on Python.

For the building material, we are going to use asterisks (with an 'A' on top for good measure). We are extremely passive engineers and the only thing we are going to ask the user is how tall they want the tower.

The only constraint is that the height of the tower cannot be less than 5 since Ant-Man is not the only Avenger who's going to be living in the tower.

```
height = 0
space = " "
```

```
valid_input = 0

while valid_input == 0:
   height = int(input("How tall do you want the tower: "))
   if height >= 5:
       valid_input = 1

for i in range(height):
    for s in range((height+1)*-1, -i):
        print(" ", end="")
    for j in range(i+1):
        if i == 0:
            print("A ", end="")
        else:
            print("* ", end="")
    print()
```

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

Constraints and assumptions

- Assessment-wide constraints and assumptions.

Helpful stuff

An example run is:

```
How tall do you want the tower: 7
        A
       * *
      * * *
     * * * *
    * * * * *
   * * * * * *
  * * * * * * *
```

task 3 - hulk smAaash!

ME HULK!
PUNY HUMANS ATTACK

ME SMASH!

BUT SOME STRONG

THEN ME SAD :(

English Translation:
There is a line of people waiting to fight The Hulk. Their powers vary and if they are weaker or equally as strong as the Hulk, then they get smashed. Otherwise, they make hulk sad. The powers of the challengers are provided in a list and Hulk's power is provided as an integer.

You need to write a function that takes in these two inputs, and does two things:

1. If a challenger is too strong, then print `Hulk Sad :(`

2. If a challenger is weaker or equally as strong as the hulk, then print `Hulk Smash! >:(`. Also, keep a track of how many people Hulk has smashed and return this value from the function.

```
"""
```

```
HULK SMASH!
"""
__author__ = "Saksham Nagpal"
__date__ = "19.05.2022"

from typing import List

def smash_or_sad(the_list: List[int], hulk_power: int) -> int:
    smash_count = 0
    for i in range(len(the_list)):
        if the_list[i] <= hulk_power:
            print("Hulk SMASH! >:(")
            smash_count += 1
        else:
            print("Hulk Sad :(")
    return smash_count


def main() -> None:
    my_list = [10, 14, 16]
    hulk_power = 15
    print(f"Hulk smashed {smash_or_sad(my_list, hulk_power)} people")

main()
```

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

Constraints and assumptions

- Assessment-wide constraints and assumptions.
- **Please ensure that while passing arguments to the function, you should pass 'the_list' as the first argument and 'hulk_power' as the second argument.**

Helpful stuff

You can test your program by clicking "Run", or by running the following command in the **Terminal**:

An example of a run is:

```
Hulk SMASH! >:(
Hulk SMASH! >:(
Hulk Sad :(
Hulk smashed 2 people
```

task 4 - the sacred timeliNe

Oh, Dear! A bunch of Nexus events have happened and a lot of Loki Variants have messed up the sacred timeline!

Events are in disarray and the TVA needs your help to set the order right.

The timekeepers have assigned each event an integer and will send you the list of integers. What you need to do is to implement an insertion sort function to set them in increasing order. Fortunately, Miss Minutes has found an insertion_sort python implementation for your reference to convert into MIPS!

Convert the following code **faithfully** into MIPS

```
from typing import List, TypeVar

# DO NOT add comments to this file
# Use the provided file on the right ->

T = TypeVar('T')

def insertion_sort(the_list: List[T]):
    length = len(the_list)
    for i in range(1, length):
        key = the_list[i]
        j = i-1
        while j >= 0 and key < the_list[j] :
                the_list[j + 1] = the_list[j]
                j -= 1
```

```
        the_list[j + 1] = key

def main() -> None:
    arr = [6, -2, 7, 4, -10]
    insertion_sort(arr)
    for i in range(len(arr)):
        print (arr[i], end=" ")
    print()


main()
```

You are asked to do two subtasks:

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

Constraints and assumptions

- Assessment-wide constraints and assumptions.

Helpful stuff

An example of a run is:

```
$ java -jar Mars4_5.jar task4.asm

MARS 4.5 Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

-10 -2 4 6 7
```

task 5a (1008 & 2085) - doctor strange's muLtivErsE

Thanos is about to destroy the world. You are Dr. Stephen Strange and only have minutes to figure out a solution before he comes to find you and everything is finished. Luckily, you are in possession of the time stone and you can dream walk into multiple realities to find the one where you are sure to defeat Thanos.

A particular set of events should happen in a particular order to defeat Thanos and they all have been given an integer index and placed in a list. Your job is to find all possible combinations of these events occurring so you can visit each of these realities.

A sample python program has been given for your reference where:

- arr - An array of integers
- n - length of `arr`
- r - total number of events to be considered in one reality

Your function should take these arguments `print_combination(arr, n, r)` and print each possible outcome. **Faithfully** translate the following Python code:

```
def print_combination(arr, n, r):

    data = [0] * r

    combination_aux(arr, n, r, 0, data, 0)

def combination_aux(arr, n, r, index, data, i):
```

```
    if (index == r):
        for j in range(r):
            print(data[j], end = " ")
        print()
        return

    if (i >= n):
        return

    data[index] = arr[i]
    combination_aux(arr, n, r, index + 1,
                    data, i + 1)

    combination_aux(arr, n, r, index,
                    data, i + 1)

def main():
    arr = [1, 2, 3, 4, 5]
    r = 3
    n = len(arr)
    print_combination(arr, n, r)

main()
```

You are asked to do two subtasks:

- Properly document the **task5.py** file provided (in the scaffold), and add line-by-line comments to the **combination_aux()** function. Your comments should explain what each line does.
- **Faithfully** translate the above code (ignoring type hints) into a properly commented MIPS program using **task5.asm** file provided.

Constraints and assumptions

- Assessment-wide constraints and assumptions.

Helpful stuff

You can test your program by clicking "Run", or by running the following command in the **Terminal**:

An example of run is:

```
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
```

```
2 4 5
3 4 5
```

task 5b (1054) - darkhoLd-ing the univErsE

Evil Doctor Strange has taken over! He wants to make sure that all universes have an incursion into each other! He is using the darkhold and has gone too deep.

The way we can save our reality (and countless others) is to find a balance between the various universes.

We are going to use the sum of events (*integered*, like the last task) and make sure the sum matches the sum for all universes. We will use the k-partition method to make it easier for ourselves.

In the k–partition problem, we need to partition an array of positive integers into `k` disjoint subsets that all have an equal sum, and they completely cover the set.

Here is a sample python code:

```
# Function to check if all subarrays are filled or not
def checkSum(sumLeft, k):

    r = True
    for i in range(k):
        if sumLeft[i]:
            r = False

    return r


# Helper function for solving `k` partition problem.
```

```
# It returns true if there exist `k` subarrays with the given sum
def subarraySum(S, n, sumLeft, A, k):

    # return true if a subarray is found
    if checkSum(sumLeft, k):
        return True

    # base case: no items left
    if n < 0:
        return False

    result = False

    # consider current item `S[n]` and explore all possibilities
    # using backtracking
    for i in range(k):
        if not result and (sumLeft[i] - S[n]) >= 0:

            # mark the current element subarray
            A[n] = i + 1

            # add the current item to the i'th subarray
            sumLeft[i] = sumLeft[i] - S[n]

            # recur for remaining items
            result = subarraySum(S, n - 1, sumLeft, A, k)

            # backtrack: remove the current item from the i'th subarray
            sumLeft[i] = sumLeft[i] + S[n]

    # return true if we get a solution
    return result


# Function for solving k-partition problem. It prints the subarrays if
# array `S[0…n-1]` can be divided into `k` subarrays with equal sum
def partition(S, k):

    # get the total number of items in `S`
    n = len(S)

    # base case
    if n < k:
        print("k-partition of array S cannot be done")
        return

    # get the sum of all elements in the array
    total = sum(S)
    A = [None] * n

    # create a list of size `k` for each subarray and initialize it
    # by their expected sum, i.e., `sum/k`
    sumLeft = [total // k] * k
```

```
    # return true if the sum is divisible by `k` and array `S` can
    # be divided into `k` subarrays with equal sum
    result = (total % k) == 0 and subarraySum(S, n - 1, sumLeft, A, k)

    if not result:
        print("k-partition of array S cannot be done")
        return

    # print all k-partitions
    for i in range(k):
        print(f"Partition {i} is", [S[j] for j in range(n) if A[j] == i + 1])


if __name__ == '__main__':

    # Input: a of integers
    S = [7, 3, 5, 12, 2, 1, 5, 3, 8, 4, 6, 4]
    k = 3

    partition(S, k)
```

In this example, the event codes are given in the list `S`

Also, we are told that we have to divide these events into `k` universes but the sum of the event codes should be the same.

Therefore, in the given example:

```
Events S = [7, 3, 5, 12, 2, 1, 5, 3, 8, 4, 6, 4]
```

```
Partition 0 = [2, 1, 3, 4, 6, 4] Sum = 20
```

```
Partition 1 = [7, 5, 8] Sum = 20
```

```
Partition 2 = [3, 5, 12] Sum = 20
```

The functions `checkSum` and `subarraySum` have been given for your sanity. Your task is to implement the `partition` function and any related code blocks.

For complex python builtin operations (there are two in partition), ensure that yours MIPS Program treats them like functions, creating stack frames and passing arguments. The exact implementation of these functions is up to you, but you must accurately represent any data being allocated by Python.