MONASH University

FACULTY OF INFORMATION TECHNOLOGY

FIT1008/FIT2085 Introduction to Computer Science

ASSESSMENT 1 Group Practical

Purpose	To enable you to understand the inner workings of a programming language, and understand how a computer analyses and represents information and executes a program by converting high-level code into assembly code.
Your task	You will be given several snippets of high-level code written in Python and you will need to "faithfully" translate the provided Python programs into MIPS. The code snippets will involve arithmetic and conditional operators, loops, lists (treated as arrays), local variables, and function calls.
Value	**20**% of your total marks for the unit
Word Limit	N/A
Due Date	**11:55 pm, 27th of March 2022**
Submission	• Via Moodle Assignment Submission. • Turnitin will be used for similarity checking of all submissions. • Additional details are provided later in this document
Assessment Criteria	Your solution will be assessed using the following criteria: 1. Implementation (50%) a. Correctness b. Quality c. Documentation 2. Peer Evaluation (30%) 3. Interview (20%)
Late Penalties	• 10% deduction per calendar day or part thereof for up to one week • Submissions more than 7 calendar days after the due date will receive a mark of zero (0) and no assessment feedback will be provided.
Support Resources	See Moodle Assessment page
Feedback	Feedback will be provided on student work via: general cohort performance specific student feedback ten working days post submission

Appendix: Changelog

1. **(4 March 2022) - Corrected the type hint of 'target' in Task 5 from str to int**

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

- This assignment is a group assignment.
- Your group should complete the Group Work Agreement before commencing work on this assignment.
- Your group will need to divide the work and submit the assignment together.
- The assignment will require you to submit all MIPS and python files that are detailed in each task.
- The naming convention for each file is as follows:

groupno_task_number.extension

For example - group1_task_1.py OR group1_task_2.asm

Constraints and Assumptions for All Tasks

1. Local variables must be stored in the runtime stack.
2. Use only MIPS instructions that appear in the MIPS reference sheet.
3. Use the function names provided, as the testing relies on those function label names.
4. A detailed description of what we mean by "faithfully" is described later in this document.
5. Do not simplify or reorder boolean statements (e.g. in an if) to equivalent (or non-equivalent) statements using boolean logic or arithmetic.
6. 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).
7. 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).
8. No need to check for arithmetic overflow unless we ask for it.
9. Do not create helper functions (e.g. to print).
10. Make sure you follow the correct function calling convention as described on the MIPS reference sheet.
11. 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.

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):
    a. 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.

    b. 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").

Tasks

Task 1

Aim

The aim of this task is to implement in MIPS different decisions (>=, <=, < and >), complex if-then-elses, and shift instructions to multiply or divide, as needed.

1. Consider the following Python code:

```
number = int(input("Enter the number: "))
first_divisor = int(input("Enter the first divisor: "))
second_divisor = int(input("Enter the second divisor: "))
divisors = 0

if number % first_divisor == 0 and number % second_divisor == 0:
    divisors = 2
elif number % first_divisor == 0 or number % second_divisor == 0:
    divisors = 1
else:
    divisors = 0

print("\nDivisors: " + str(divisors))
```

2. **Faithfully** translate the above code into a **properly commented** MIPS program
3. Compare your output against the provided example test cases
4. Submit the code in a MIPS file.

Constraints and Assumptions

You can assume that both divisors are positive integers (>0)

Example Test Cases (Input and Expected Output)

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter the number: 60
Enter the first divisor: 6
Enter the second divisor: 10
```

```
Divisors: 2
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter the number: 8
Enter the first divisor: 2
Enter the second divisor: 4

Divisors: 2
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter the number: 11
Enter the first divisor: 3
Enter the second divisor: 4

Divisors: 0
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter the number: 44
Enter the first divisor: 3
Enter the second divisor: 11

Divisors: 1
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter the number: 0
Enter the first divisor: 4
Enter the second divisor: 5

Divisors: 2
```

Task 2

Aim

The aim of this task is to implement 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.

1. Consider the following Python code, which, given a Python list, computes the count of multiples of a given number (excluding itself) and prints it:

```
size = int(input("Enter array length: "))
the_list = [None]*size
n = int(input("Enter n: "))
count = 0

for i in range(len(the_list)):
    the_list[i] = int(input("Enter the value: "))
    if the_list[i] % n == 0 and the_list[i] != n:
        count += 1

print("\nThe number of multiples (excluding itself) = " + str(count))
```

2. **Faithfully** translate the above code into a **properly commented** MIPS program and submit the code in a MIPS file.
3. Compare your output against the provided example test cases
4. Submit the code in a MIPS file.

Constraints and Assumptions

- Assume the size of the list will always be positive.
- Assume that n is always a positive integer
- 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, but you must use the appropriate one according to the python code.
- We understand that using this code, 0 will also show as a multiple of any number, but we can assume that 0 is considered as a multiple in our assignment.

Example Test Cases (Input and Expected Output)

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter array length: 1
Enter n: 6
Enter the value: 5

The number of multiples (excluding itself) = 0
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter array length: 2
Enter n: 3
Enter the value: 3
Enter the value: 3

The number of multiples (excluding itself) = 0
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter array length: 3
Enter n: 2
Enter the value: 1
Enter the value: 2
Enter the value: 4

The number of multiples (excluding itself) = 1
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter array length: 4
Enter n: 7
Enter the value: 7
Enter the value: 14
Enter the value: 21
Enter the value: 23

The number of multiples (excluding itself) = 2
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Enter array length: 5
Enter n: 2
Enter the value: 4
Enter the value: 6
Enter the value: 8
Enter the value: 10
Enter the value: 12

The number of multiples (excluding itself) = 5
```

Task 3

Aim

The aim of this task is to define and call functions in MIPS. You can copy and paste the part of the MIPS code you wrote in Task 2 that is useful for this one.

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

```
"""
This file contains python code for calculating the
number of multiples of a given number from a given
list of integers
"""
__author__ = "Saksham Nagpal"
__date__ = "01.08.2021"

from typing import List

def get_multiples(the_list: List[int], n: int) -> int:
    """
    Computes the number of multiples of n
    :pre: n > 0
    :param the_list: The list of integers being passed
    :param n: The number to check for multiples
    :returns: the count of multiples from the list
    """
    count = 0
    for i in range(len(the_list)):
        if the_list[i] % n == 0 and the_list[i] != n:
            count += 1
    return count


def main() -> None:
    my_list = [2, 4, 6]
    n = 3
    print("The number of multiples of " + str(n) + " is: " +
str(get_multiples(my_list, n)))


main()
```

2. **Faithfully** translate the above code into a **properly commented** MIPS program and submit the code in a MIPS file.
3. Compare your output against the provided example test cases
4. Submit the code in a MIPS file.

Constraints and Assumptions

- You may reuse the code for Task 2 but be mindful not to use global variables.
- We understand that using this code, 0 will also show as a multiple of any number, but we can assume that 0 is considered as a multiple in our assignment.
- Please ensure that while passing arguments to the function, you should pass 'the_list' as the first argument and 'n' as the second argument.
- The input in the expected output shows asking for input but its NOT mandatory to build this. You may just build the function that accepts a list and displays the output as calculated

Example Test Cases (Input and Expected Output)

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 1
Enter the value of n: 6
Enter num: 5
The number of multiples of 6 is: 0
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 2
Enter the value of n: 3
Enter num: 3
Enter num: 3
The number of multiples of 3 is: 0
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 3
Enter the value of n: 2
Enter num: 1
Enter num: 2
Enter num: 4
```

```
The number of multiples of 2 is: 1
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 4
Enter the value of n: 5
Enter num: 10
Enter num: 15
Enter num: 20
Enter num: 25
The number of multiples of 5 is: 4
```

Task 4

Aim

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

1. Consider the following Python code, which defines the function `insertion_sort(the_list)`, which sorts `the_list` in non-decreasing order:

```
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:

1. 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.

2. Faithfully translate the above code (ignoring type hints) into a properly commented MIPS program using the task4.asm file provided.

Constraints and Assumptions

- Do not use any global variables
- The input in the expected output shows asking for input but its NOT mandatory to build this. You may just build the function that accepts a list and displays the output as calculated

Example Test Cases (Input and Expected Output)

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 4
Enter num: 1
Enter num: 2
Enter num: 3
Enter num: 4
1 2 3 4
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 4
Enter num: 4
Enter num: 3
Enter num: 2
Enter num: 1
1 2 3 4
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 3
Enter num: 3
Enter num: 3
Enter num: 1
1 3 3
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

```

```
Array length: 2
Enter num: 2
Enter num: 1
1 2
```

Task 5

Aim

The aim of this task is to use all your MIPS knowledge (conditions, loops, array access, function definition, etc) together again. This task requires you to faithfully translate a recursive function from python to MIPS.

1. Consider the following Python code, which defines the function binary_search(the_list, target, low, high), which searches for the target item in the sorted list the_list. The two parameters low and high, should indicate the sub list to search in. Initially this should be set to low = 0 and high = len(the_list) - 1.

```
def binary_search(the_list: list, target: int, low: int, high: int) ->
int:
    if low > high:
        return -1
    else:
        mid = (high + low) // 2

        if the_list[mid] == target:
            return mid
        elif the_list[mid] > target:
            return binary_search(the_list, target, low, mid - 1)
        else:
            return binary_search(the_list, target, mid + 1, high)


def main() -> None:
    arr = [1, 5, 10, 11, 12]
    index = binary_search(arr, 11, 0, len(arr) - 1)
    print(index)


main()
```

You are asked to do two subtasks:

1. Properly document the task5.py file provided (in the scaffold), and add line by line comments to the binary_search() function. Your comments should explain what each line does.

2. Faithfully translate the above code (ignoring type hints) into a properly commented MIPS program using the task5.asm file provided.

Constraints and Assumptions

1. Do not use any global variables

Example Test Cases (Input and Expected Output)

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 5
Enter num: 1
Enter num: 20
Enter num: 30
Enter num: 31
Enter num: 35
Enter target: 1
0
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 5
Enter num: 1
Enter num: 20
Enter num: 30
Enter num: 31
Enter num: 35
Enter target: 25
-1
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 2
Enter num: 1
Enter num: 100
Enter target: 50
-1
```

```
MARS 4.5  Copyright 2003-2014 Pete Sanderson and Kenneth Vollmar

Array length: 5
```

```
Enter num: 1
Enter num: 20
Enter num: 30
Enter num: 31
Enter num: 35
Enter target: 20
1
```