NET103 MARIE Assembly Coursework Feedback 2017-2018

Student: 10072XXX
Group: M18- 1

May 23, 2018

Assessment methodology:

To allow consistency in marking, all submissions have been marked by Frederico Belmonte Klein and reviewed by Dr Maria Papadaki, using the following methodology:

1. The .mas file was initially loaded onto the MARIE code editor and assembled to produce the .mex executable file. The .mex file was then loaded onto the MARIE simulator for further testing. After each calculation, the file was reloaded to enable resetting of variables and register values (variable clean-up is not necessary). If an error occurred, the simulator was reset and the .mex file reloaded so that testing could continue with a clean environment.
2. If present, the routine to calculate the functionality was examined against the following automated tests (see subsections below) using a batch marie-simulator[1] for positive integer division routine (runtime limited to 10E5 operations) and for primality test (runtime limited to 10E6 operations). If an unexpected result was found, such input was tested also on the java Marie Simulator.

If incorrect results were produced, then further testing commenced as to understand how the algorithm works. At the same time, the assembly code .mas file was reviewed to check the student's understanding of the problem, the range of values it can work with, the use of subroutines with JNS and JumpI instructions, as well as the subroutine code itself.

Division

Normal division expected operation should work for the following values can be seen in the table below:

Inputs		Expected output		Your output	
Dividend	Divisor	Quotient	Rest	Quotient	Rest
0	1000	0	0	1	-1000
2	1	2	0	2	0
145	12	12	1	12	1
233	123	1	110	1	110
32767	500	65	267	65	267

Division: error handling

It was not a part of the assignment to divide negative numbers, however, it is expected that your program identifies wrong input and handle them without giving wrong results or worse, entering infinite loops. The expected results and outputs of your algorithm can be seen in the table below:

Inputs		Expected output		Your output	
Dividend	Divisor	Quotient	Rest	Quotient	Rest
0	0	Err.[1]		???.[2]	
1	0	Err.[1]		???.[2]	
-11	1	Err.[1] or -11 rest 0		1	-12
11	-1	Err.[1] or -11 rest 0		???.[2]	
-1	-1	Err.[1] or 1 rest 0		???.[2]	

[1]Err. stands for the standard behaviour for invalid input you implemented in your code. Either ouput a 0 or -1 (a good idea in this case, since negative numbers were not used, so they shouldn't be expected to given as answers) or have the program halt without giving any result. [2]Execution stopped after 10E5 operations.

Prime numbers

The test for primality was done for the numbers 1..7 and the randomly chosen not prime large odd numbers (201 and 649) and large primes (89 and 577) as can be seen in the table below:

[1]Available at `https://github.com/mysablehats/marie-sim`

Input[1]	Expected Output	Your output
1	0	0
2	1	0
3	1	1
4	0	0
5	1	1
6	0	0
7	1	1
89	1	1
201	0	1
577	1	1
649	0	1

[1] List of primes from:
`https://primes.utm.edu/lists/small/10000.txt`

Prime numbers: error handling

The test for primality was done for the numbers on the limit range of the int16 input (-1, 0 and 32767 = 7X31X151) to check for lower and upper bounds and using and the randomly chosen not prime large odd numbers (899 = 29X31 and 4819 = 61X79) and large primes (983 and 4817). For this test the maximum number of operations was increased to 10E6. Expected results and results of the presented algorithm can be seen in table below:

Input[1]	Expected Output	Your output
-1	0 or Err.	0
0	0	0
899	0	1
983	1	1
4817	1 or Err.	1
4819	0 or Err.	1
32767	0 or Err.	1

[1] List of primes from:
`https://primes.utm.edu/lists/small/10000.txt`

Assessment Criteria:

Accuracy of Results

Does the. mas file compile and run on the MarieSim environment? Does the code produce accurate and expected results according to the documentation and the range of values? For a basic functionality, mostly positive small numbers will be considered. For more elaborate work, limit values and whether the functions can handle negative numbers (division: error handling) without outputting incorrect results will be considered.

Functionality and Efficiency

Does the code contain all requested functionality? Can it detect and deal with user errors and unexpected user input (i.e. input of negative numbers, or numbers larger than 100)? Does the algorithm implement efficient computations? The main points will be awarded on this topic if the user has thought of error handling, range checking and overflow handling. Finally, structure and correct use of subroutines (JNS and JUMPI) will be considered and necessary for awarding distinction grades and incorrect syntax (such as INPUT N, OUTPUT N or having an operation in the same line that a routine label is accessed by JNS – that will not ever run since it gets overwritten) were considered as preventing the group to get a distinction mark.

Documentation and commenting (points are additive)

Documentation should be included as comments in the .mas file to specify the intended execution of the code. This is particularly important in the case of errors and incorrect results. Also, the documentation should specify the range of values the programme can work with. Additionally, the documentation should mention what each part of the code (main code or subroutine) is doing.

Results:

Accuracy of Results: 20/40
Functionality and Efficiency: = 17/30
Documentation and commenting: 15/30
Overall Mark: 52/100

Comments:

Your group submitted a version of the division algorithm and testing for primality in MARIE assembly that does compile and runs with appropriate answers for most small numbers, successfully achieving the most of the desired results with the desired level of functionality on the parts that were submitted.

ACCURACY: For small positive numbers, the division works fine, and so does the primality test for numbers 0, 1 and 3 to 7, however division of 0 fails to give the correct result and number 2 is incorrectly labeled as not a prime.
ELABORATE ACCURACY: The algorithm does not deal with unexpected input, not seeming to implement any sort of upper or lower bound check. Additionally, for primality test, large odd numbers seem to be always deemed prime even when it is not the case.

FUNCTIONALITY, RANGE: Your algorithm has no parameter range checking for lower bounds or upper bounds. The structure of your submission was in the moulds as directed by the assignment and all information necessary to correct it was in the .mas file – not needing additional files for explaining your work. JNS and JUMPI are used, to form proper stand-alone subroutines and the subroutine division is used on the implementation of your primality test, which was desirable. The algorithm does not show any functionality to implement efficient computation.

DOCUMENTATION:
Documentation, as comments, is present within the .mas file. The range in which your algorithm work is however **not** commented. But the comments are good, describing each subroutine's general behaviour and helping understand what the algorithm is doing. However the misspelled words such as "Psuedocode" and "Psuedocde" [sic] do not give a good impression about it.

RECOMMENDATIONS:
As a recommendation for increasing marks for future work, we would recommend:

- More systematic error escaping with the same answer for all invalid input: considering your algorithm only deals with positive numbers, you could have chosen a negative number as error code
- Implement upper bound range check
- Implement overflow/underflow check – important if you want to also be able to multiply negative numbers
- Trying to extend the range in which your algorithm gives adequate results
- Writing more general functionality details for subroutines
- Check documentation for spelling errors
- Write the inputs and outputs of each subroutine in your documentation
- Write the range in which your algorithm works in your documentation
- Test your algorithm more before submitting to make sure it gives accurate results within all its range (2 is a prime)
- Consider implementing efficient algorithms. Faster code can be run more times, which eases testing, increases usability and flexibility, as well as facilitates building more complex programs based on your code

OVERALL:
This is overall a pretty good submission, that however has some small mistakes in terms of functionality and lacks the features desired for it to deserve better grade.