Nikita Zdvijkov
EE-2063-01: Comp Org/Microprocess
February 1, 2019

Project 1: MIPS Basics: Math, Conditionals, Loops, I/O

Objective

Assignment: write a MIPS assembly program that finds the sum of the multiples of 3 and/or 5 in the set of natural numbers less than an upper bound with two cases: 10 and 1000.

Introduction

I wrote equivalent Python and MIPS Assembly programs that solve the assigned problem for any user-supplied natural number factors (`FACTOR_1` and `FACTOR_2` Python variables) and upper bound (`MAX`). My programs introduce themselves in their first output to the console:

```
THIS PROGRAM WILL
  1. TAKE THREE NATURAL NUMBERS FOR INPUT,
  2. TEST EACH NUMBER IN THE SET OF {NATURAL NUMBERS LESS THAN THE 1ST INPUT} FOR
DIVISIBILITY BY THE 2ND INPUT - OR THE 3RD - OR BOTH.
  3. OUTPUT THE SUM OF THE NUMBERS THAT PASSED THE TEST.
```

Use the summary of variables (Table 1) below to interpret the flowchart representation of my algorithm (Figure 1) on the following page.

Table 1: Variables Summary

MIPS Register	Python Variable	Meaning
`$s0`	`MAX`	upper bound
`$s1`	`FACTOR_1`	first factor
`$s2`	`FACTOR_2`	second factor
`$s3`	`SUM`	running sum of multiples, outputted at end
`$t0`	`TEST`	# in set to test for divisibility, iterated from 1 to `MAX`

Tools Used

- Python 3.7.2
- MARS (MIPS Assembler and Runtime Simulator) Release 4.5
- This report typeset with LaTeX

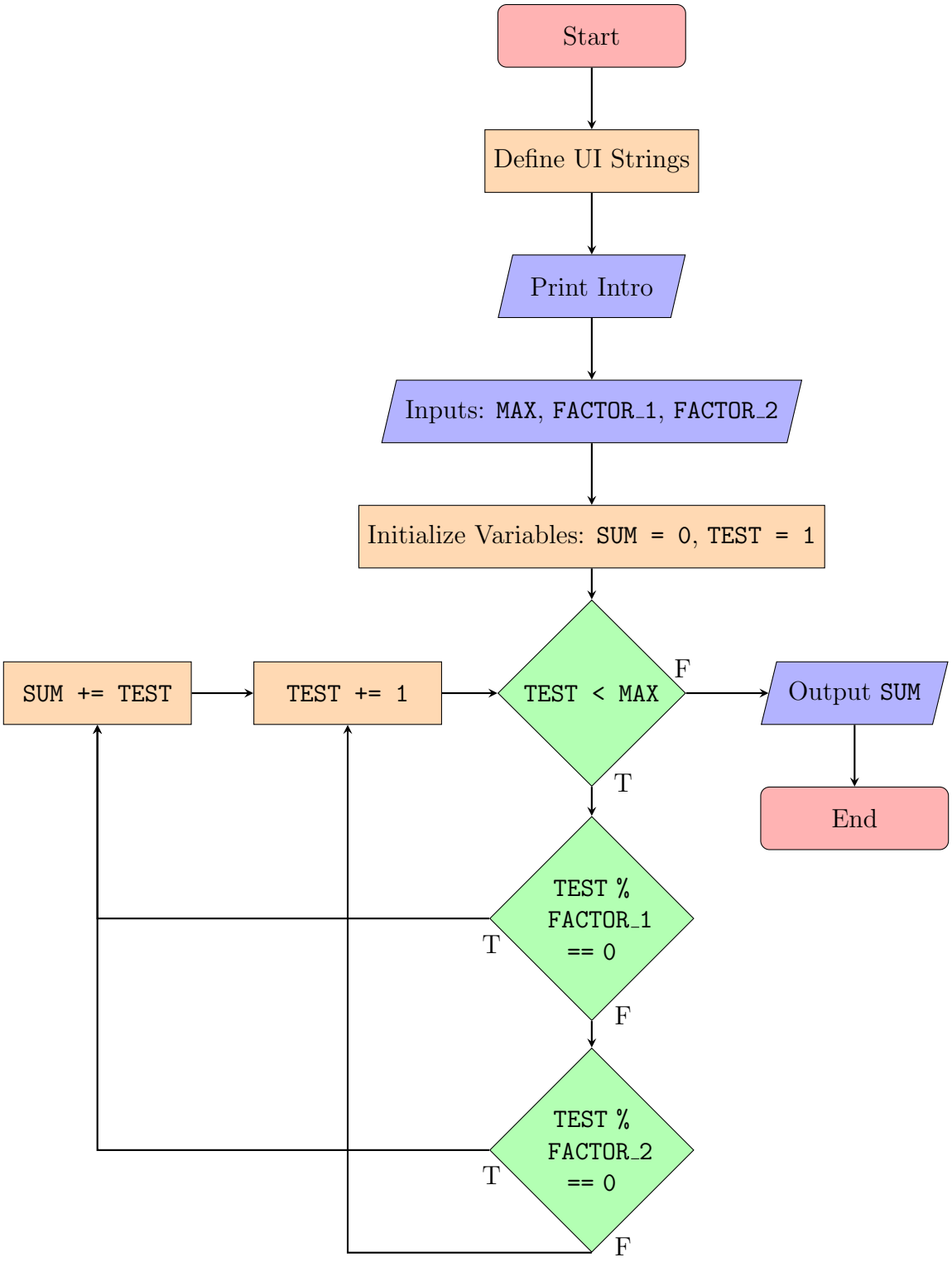

Figure 1: Algorithm Flowchart

Methodology

My Python and assembly scripts (full code in Appendices) are functionally identical, i.e. as far as I could tell, they behave exactly the same when run in their respective consoles. In this section, I will compare the two scripts part-by-part:

— Define User-Interface Strings —

```python
# --- DEFINE UI STRINGS ---

INTRO = "\nTHIS PROGRAM WILL\n  1. TAKE THREE NATURAL NUMBERS FOR INPUT,\n  2. TEST EACH NUMBER IN THE SET OF {NATURAL NUMBERS LESS THAN THE 1ST INPUT} FOR DIVISIBILITY BY THE 2ND INPUT - OR THE 3RD - OR BOTH.\n  3. OUTPUT THE SUM OF THE NUMBERS THAT PASSED THE TEST."

    # prompts for 3 inputs: MAX, FACTOR_1, FACTOR_2
PROMPT_MAX = "\nENTER 1ST INPUT\n(MUST BE A NATURAL NUMBER)\n"
PROMPT_FACTOR_1 = "\nENTER 2ND INPUT\n(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)\n"
PROMPT_FACTOR_2 = "\nENTER 3RD INPUT\n(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)\n"

    # numerical values will be inserted between 5 parts to form final output
OUT_PT_1 = "\nOUTPUT\n(SUM OF MULTIPLES OF "
OUT_PT_2 = " AND "
OUT_PT_3 = " UNDER "
OUT_PT_4 = ")\n"
OUT_PT_5 = "\n"
```

```assembly
# --- DEFINE UI STRINGS ---

.data # entries into memory

        INTRO:              .asciiz "\nTHIS PROGRAM WILL\n  1. TAKE THREE NATURAL NUMBERS FOR INPUT,\n  2. TEST EACH NUMBER IN THE SET OF {NATURAL NUMBERS LESS THAN THE 1ST INPUT} FOR IF IT IS DIVISIBLE BY THE 2ND INPUT - OR THE 3RD - OR BOTH.\n  3. OUTPUT THE SUM OF THE NUMBERS THAT PASSED THE TEST.\n"

    # prompts for 3 inputs: MAX, FACTOR_1, FACTOR_2
        PROMPT_MAX:         .asciiz "\nENTER 1ST INPUT\n(MUST BE A NATURAL NUMBER)\n"
        PROMPT_FACTOR_1:    .asciiz "\nENTER 2ND INPUT\n(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)\n"
        PROMPT_FACTOR_2:    .asciiz "\nENTER 3RD INPUT\n(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)\n"

    # numerical values will be inserted between 5 parts to form final output
        OUT_PT_1:           .asciiz "\nOUTPUT\n(SUM OF MULTIPLES OF "
        OUT_PT_2:           .asciiz " AND "
        OUT_PT_3:           .asciiz " UNDER "
        OUT_PT_4:           .asciiz ")\n"
        OUT_PT_5:           .asciiz "\n"
```

Remarks:

- Assembly has to use "`.asciiz`" to specify text encoding format – not a concern in Python.
- Python has no equivalent to "`.data`".

```
# --- INTRO ---

print(INTRO)
```

```
# --- INTRO ---

.text # from here on: executable

    # output INTRO string
        la $a0, INTRO # load address of INTRO string into syscall argument register
        li $v0, 4 # load print string service code into syscall argument register
        syscall # print to console
        # printing text to console happens according to the above formula
```

Remarks:

- The most basic of tasks, to print a string to the console, can be done in Python with an intuitive one-line command; it takes three instructions in assembly.

— *Inputs* —

```python
# --- INPUTS ---

MAX      = int(input(PROMPT_MAX))
FACTOR_1 = int(input(PROMPT_FACTOR_1))
FACTOR_2 = int(input(PROMPT_FACTOR_2))
```

```
# --- INPUTS ---

    # output PROMPT_MAX string
        la $a0, PROMPT_MAX
        li $v0, 4
        syscall
    # get 1st input: MAX
        li $v0, 5 # load read int service code into syscall argument register
        syscall # get console input
        move $s0, $v0 # move from syscall output register to register associated with MAX variable for future use
        # getting input from console happens according to the above formula

    # output PROMPT_FACTOR_1 string
        la $a0, PROMPT_FACTOR_1
        li $v0, 4
        syscall
    # get 2nd input: FACTOR_1
        li $v0, 5
        syscall
        move $s1, $v0

    # output PROMPT_FACTOR_2 string
        la   $a0, PROMPT_FACTOR_2
        li   $v0, 4
        syscall
    # get 3rd input: FACTOR_2
        li $v0, 5
        syscall
        move $s2, $v0
```

Remarks:

- Each line of Python code in this part gets expanded to six lines of assembly code: it takes MIPS three commands to print the prompt string to the console and another three to accept user input.

— Initialize Variables —

```
# --- INITIALIZE VARS ---

SUM  = 0 # running SUM of multiples of FACTOR_1 and/or FACTOR_2 - initialize to zero
TEST = 1 # number in the set to TEST for divisibility FACTOR_1 and/or FACTOR_2 - initialize to first natural number
```

```
# --- INITIALIZE VARS ---

        li $s3, 0 # SUM register should already be at zero - this is just in case
        li $t0, 1 # value of 1 loaded into register corresponding with var TEST
```

Remarks:

- Clear correspondence between Python and assembly in this part.

— *Loop* —

```
# --- LOOP ---

while TEST < MAX:               # go through the set from 1 --> largest natural number less than MAX
    if TEST % FACTOR_1 == 0:    # if TEST is divisible by FACTOR_1
        SUM += TEST             # then add TEST to running SUM
    elif TEST % FACTOR_2 == 0: # if TEST is divisible by FACTOR_2 but not FACTOR_1
        SUM += TEST             # then add TEST to running SUM
    TEST += 1                   # go to next number in the set
```

```
# --- LOOP ---

while:
        bge $t0, $s0, exit # if TEST is greater than or equal to MAX ...
        # ... then jump to instruction in OUTPUT & FINISH part of program
        # if TEST is less than MAX, then continue on with loop

        div $t0, $s1 # divide TEST by FACTOR_1 and store remainder in hi register
        mfhi $t1 # move remainder from hi register to a useable one: $t1

        beq $t1, $zero, sum # if the remainder is equal to zero, then branch to the sum address
        # ... this means skipping the test for divisibility by FACTOR_2
        div $t0, $s2 # if remainder is not equal to zero the program will end up here ...
        # ... to test for divisibility by FACTOR_2
        mfhi $t1 # again, move remainder to useable register
        bne $t1, $zero, increment # another branch that skips adding TEST to SUM if the remainder is not equal to zero
sum:
        add $s3, $s3, $t0 # SUM += TEST (this is done if TEST is a multiple of FACTOR_1 and/or FACTOR_2
increment:
        addi $t0, $t0, 1 # TEST += 1 (iterate TEST

        j while # jump back to the header of the while loop
```

Remarks:

- Python's simple syntax made it easy to conceptualize and troubleshoot the loop at the heart of this program.

— *Output & Finish* —

```python
# --- OUTPUT & FINISH ---

print(OUT_PT_1, FACTOR_1, OUT_PT_2, FACTOR_2, OUT_PT_3, MAX, OUT_PT_4, SUM, OUT_PT_5, sep = '')
```

```
# --- OUTPUT & FINISH ---

    # output OUT_PT_1 string
exit: # move on to output & finish section if condition for re/entering while loop not met
        la $a0, OUT_PT_1
        li $v0, 4
        syscall
    # output SUM value
        la $a0, ($s1) # load sum into syscall argument
        li $v0, 1 # load print int service code into syscall argument register
        syscall # perform the output
        # the process for outputting a number to console is a little different from outputting a string

    # output OUT_PT_2 string
        la   $a0, OUT_PT_2
        li   $v0, 4
        syscall
    # output SUM value
        la   $a0, ($s2)
        li   $v0, 1
        syscall

    # output OUT string
        la   $a0, OUT_PT_3
        li   $v0, 4
        syscall
    # output SUM value
        la   $a0, ($s0)
        li   $v0, 1
        syscall

    # output OUT string
        la   $a0, OUT_PT_4
        li   $v0, 4
        syscall
    # output SUM value
        la   $a0, ($s3)
        li   $v0, 1
        syscall

    # output OUT string
        la   $a0, OUT_PT_5
        li   $v0, 4
        syscall

    # finish
        li $v0, 10 # load exit service code into syscall argument register
        syscall # exit program
```

Remarks:

- In assembly, instructions for outputting a string and integer are different.
- Assembly requires a syscall to properly exit the program.
- This part exemplifies how much more concise high level languages can be.

Results

```
$ python3 proj1.py

THIS PROGRAM WILL
  1. TAKE THREE NUMBERS FOR INPUT,
  2. TEST EACH NUMBER IN THE SET OF {NATURAL NUMBERS LESS THAN THE 1ST INPUT} FOR IF IT
IS DIVISIBLE BY THE 2ND INPUT - OR THE 3RD - OR BOTH.
  3. OUTPUT THE SUM OF THE NUMBERS THAT PASSED THE TEST.

ENTER 1ST INPUT
(MUST BE A NATURAL NUMBER)
1000

ENTER 2ND INPUT
(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)
3

ENTER 3RD INPUT
(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)
5

OUTPUT
(SUM OF MULTIPLES OF 3 AND 5 UNDER 1000)
233168
```

Figure 2: Python script console end state, `MAX` = 1000, `SUM` = 233168

```
$ python3 proj1.py

THIS PROGRAM WILL
  1. TAKE THREE NATURAL NUMBERS FOR INPUT,
  2. TEST EACH NUMBER IN THE SET OF {NATURAL NUMBERS LESS THAN THE 1ST INPUT} FOR
DIVISIBILITY BY THE 2ND INPUT - OR THE 3RD - OR BOTH.
  3. OUTPUT THE SUM OF THE NUMBERS THAT PASSED THE TEST.

ENTER 1ST INPUT
(MUST BE A NATURAL NUMBER)
10

ENTER 2ND INPUT
(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)
3

ENTER 3RD INPUT
(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)
5

OUTPUT
```

```
(SUM OF MULTIPLES OF 3 AND 5 UNDER 10)
23
```

Figure 3: Python script console end state, MAX = 10, SUM = 23

```
THIS PROGRAM WILL
  1. TAKE THREE NUMBERS FOR INPUT,
  2. TEST EACH NUMBER IN THE SET OF {NATURAL NUMBERS LESS THAN THE 1ST INPUT} FOR IF IT
IS DIVISIBLE BY THE 2ND INPUT - OR THE 3RD - OR BOTH.
  3. OUTPUT THE SUM OF THE NUMBERS THAT PASSED THE TEST.

ENTER 1ST INPUT
(MUST BE A POSITIVE NUMBER)
1000

ENTER 2ND INPUT
(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)
3

ENTER 3RD INPUT
(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)
5

OUTPUT
(SUM OF MULTIPLES OF 3 AND 5 UNDER 1000)
233168

-- program is finished running --
```

Figure 4: Assembly script console end state, MAX = 1000, SUM = 233168

```
THIS PROGRAM WILL
  1. TAKE THREE NATURAL NUMBERS FOR INPUT,
  2. TEST EACH NUMBER IN THE SET OF {NATURAL NUMBERS LESS THAN THE 1ST INPUT} FOR IF IT
IS DIVISIBLE BY THE 2ND INPUT - OR THE 3RD - OR BOTH.
  3. OUTPUT THE SUM OF THE NUMBERS THAT PASSED THE TEST.

ENTER 1ST INPUT
(MUST BE A NATURAL NUMBER)
10

ENTER 2ND INPUT
```

```
(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)
3

ENTER 3RD INPUT
(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)
5

OUTPUT
(SUM OF MULTIPLES OF 3 AND 5 UNDER 10)
23

-- program is finished running --
```

Figure 5: Python script console end state, `MAX` = 10, `SUM` = 23

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x100101ec
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x000003e8
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x000003e8
$s1	17	0x00000003
$s2	18	0x00000005
$s3	19	0x00038ed0
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400120
hi		0x00000000
lo		0x0000014d

Figure 6: MIPS registers end state, `MAX` = 1000, `SUM` = 233168

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x100101ec
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x0000000a
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x0000000a
$s1	17	0x00000003
$s2	18	0x00000005
$s3	19	0x00000017
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400120
hi		0x00000000
lo		0x00000003

Figure 7: MIPS registers end state, `MAX` = 10, `SUM` = 23

Appendix I: Full Python Script

```
# --- DEFINE UI STRINGS ---

INTRO = "\nTHIS PROGRAM WILL\n  1. TAKE THREE NATURAL NUMBERS FOR INPUT,\n  2. TEST EACH NUMBER IN THE SET OF {NATURAL NUMBERS LESS THAN THE 1ST INPUT} FOR DIVISIBILITY BY THE 2ND INPUT - OR THE 3RD - OR BOTH.\n  3. OUTPUT THE SUM OF THE NUMBERS THAT PASSED THE TEST."

    # prompts for 3 inputs: MAX, FACTOR_1, FACTOR_2
PROMPT_MAX = "\nENTER 1ST INPUT\n(MUST BE A NATURAL NUMBER)\n"
PROMPT_FACTOR_1 = "\nENTER 2ND INPUT\n(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)\n"
PROMPT_FACTOR_2 = "\nENTER 3RD INPUT\n(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)\n"

    # numerical values will be inserted between 5 parts to form final output
OUT_PT_1 = "\nOUTPUT\n(SUM OF MULTIPLES OF "
OUT_PT_2 = " AND "
OUT_PT_3 = " UNDER "
OUT_PT_4 = ")\n"
OUT_PT_5 = "\n"

# --- INTRO ---

print(INTRO)

# --- INPUTS ---

MAX      = int(input(PROMPT_MAX))
FACTOR_1 = int(input(PROMPT_FACTOR_1))
FACTOR_2 = int(input(PROMPT_FACTOR_2))

# --- INITIALIZE VARS ---

SUM  = 0 # running SUM of multiples of FACTOR_1 and/or FACTOR_2 - initialize to zero
TEST = 1 # number in the set to TEST for divisibility FACTOR_1 and/or FACTOR_2 - initialize to first natural number

# --- LOOP ---

while TEST < MAX:              # go through the set from 1 --> largest natural number less than MAX
    if TEST % FACTOR_1 == 0:   # if TEST is divisible by FACTOR_1
        SUM += TEST            # then add TEST to running SUM
    elif TEST % FACTOR_2 == 0: # if TEST is divisible by FACTOR_2 but not FACTOR_1
        SUM += TEST            # then add TEST to running SUM
    TEST += 1                  # go to next number in the set

# --- OUTPUT & FINISH ---

print(OUT_PT_1, FACTOR_1, OUT_PT_2, FACTOR_2, OUT_PT_3, MAX, OUT_PT_4, SUM, OUT_PT_5, sep = '')
```

Appendix II: Full MIPS Assembly Script

```
# --- DEFINE UI STRINGS ---

.data # entries into memory

        INTRO:              .asciiz "\nTHIS PROGRAM WILL\n  1. TAKE THREE NATURAL NUMBERS FOR INPUT,\n  2. TEST EACH NUMBER IN THE SET OF {NATURAL NUMBERS LESS THAN THE 1ST INPUT} FOR IF IT IS DIVISIBLE BY THE 2ND INPUT - OR THE 3RD - OR BOTH.\n  3. OUTPUT THE SUM OF THE NUMBERS THAT PASSED THE TEST.\n"

    # prompts for 3 inputs: MAX, FACTOR_1, FACTOR_2
        PROMPT_MAX:         .asciiz "\nENTER 1ST INPUT\n(MUST BE A NATURAL NUMBER)\n"
        PROMPT_FACTOR_1:    .asciiz "\nENTER 2ND INPUT\n(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)\n"
        PROMPT_FACTOR_2:    .asciiz "\nENTER 3RD INPUT\n(MUST BE A NATURAL NUMBER LESS THAN THE 1ST INPUT)\n"

    # numerical values will be inserted between 5 parts to form final output
        OUT_PT_1:           .asciiz "\nOUTPUT\n(SUM OF MULTIPLES OF "
        OUT_PT_2:           .asciiz " AND "
        OUT_PT_3:           .asciiz " UNDER "
        OUT_PT_4:           .asciiz ")\n"
        OUT_PT_5:           .asciiz "\n"

# --- INTRO ---

.text # from here on: executable

    # output INTRO string
        la $a0, INTRO # load address of INTRO string into syscall argument register
        li $v0, 4 # load print string service code into syscall argument register
        syscall # print to console
        # printing text to console happens according to the above formula

# --- INPUTS ---

    # output PROMPT_MAX string
        la $a0, PROMPT_MAX
        li $v0, 4
        syscall
    # get 1st input: MAX
        li $v0, 5 # load read int service code into syscall argument register
        syscall # get console input
        move $s0, $v0 # move from syscall output register to register associated with MAX variable for future use
        # getting input from console happens according to the above formula

    # output PROMPT_FACTOR_1 string
        la $a0, PROMPT_FACTOR_1
        li $v0, 4
        syscall
    # get 2nd input: FACTOR_1
        li $v0, 5
```

```
47          syscall
48          move $s1, $v0
49
50      # output PROMPT_FACTOR_2 string
51          la   $a0, PROMPT_FACTOR_2
52          li   $v0, 4
53          syscall
54      # get 3rd input: FACTOR_2
55          li $v0, 5
56          syscall
57          move $s2, $v0
58
59  # --- INITIALIZE VARS ---
60
61          li $s3, 0 # SUM register should already be at zero - this is just in case
62          li $t0, 1 # value of 1 loaded into register corresponding with var TEST
63
64  # --- LOOP ---
65
66  while:
67          bge $t0, $s0, exit # if TEST is greater than or equal to MAX ...
68          # ... then jump to instruction in OUTPUT & FINISH part of program
69          # if TEST is less than MAX, then continue on with loop
70
71          div $t0, $s1 # divide TEST by FACTOR_1 and store remainder in hi register
72          mfhi $t1 # move remainder from hi register to a useable one: $t1
73
74          beq $t1, $zero, sum # if the remainder is equal to zero, then branch to the sum
    address
75          # ... this means skipping the test for divisibility by FACTOR_2
76          div $t0, $s2 # if remainder is not equal to zero the program will end up here ...
77          # ... to test for divisibility by FACTOR_2
78          mfhi $t1 # again, move remainder to useable register
79          bne $t1, $zero, increment # another branch that skips adding TEST to SUM if the
    remainder is not equal to zero
80  sum:
81          add $s3, $s3, $t0 # SUM += TEST (this is done if TEST is a multiple of FACTOR_1
    and/or FACTOR_2
82  increment:
83          addi $t0, $t0, 1 # TEST += 1 (iterate TEST
84
85          j while # jump back to the header of the while loop
86
87  # --- OUTPUT & FINISH ---
88
89      # output OUT_PT_1 string
90  exit: # move on to output & finish section if condition for re/entering while loop not
    met
91          la $a0, OUT_PT_1
92          li $v0, 4
93          syscall
94      # output SUM value
95          la $a0, ($s1) # load sum into syscall argument
96          li $v0, 1 # load print int service code into syscall argument register
```

```
97          syscall # perform the output
98          # the process for outputting a number to console is a little different from
    outputting a string
99
100     # output OUT_PT_2 string
101         la   $a0, OUT_PT_2
102         li   $v0, 4
103         syscall
104     # output SUM value
105         la   $a0, ($s2)
106         li   $v0, 1
107         syscall
108
109     # output OUT string
110         la   $a0, OUT_PT_3
111         li   $v0, 4
112         syscall
113     # output SUM value
114         la   $a0, ($s0)
115         li   $v0, 1
116         syscall
117
118     # output OUT string
119         la   $a0, OUT_PT_4
120         li   $v0, 4
121         syscall
122     # output SUM value
123         la   $a0, ($s3)
124         li   $v0, 1
125         syscall
126
127     # output OUT string
128         la   $a0, OUT_PT_5
129         li   $v0, 4
130         syscall
131
132     # finish
133         li $v0, 10 # load exit service code into syscall argument register
134         syscall # exit program
```