```
=============
Assignment 2
=============
Student:    Evan Louie
Student ID: 72210099
CSID:       m6d7

------------------------------
Problem 1:
------------------------------

Pseudo-code for rmmovl
Fetch:
    icode:ifun  <- M1[PC]
    rA:rB       <- M1[PC+1]
    valC        <- M4[PC+2]
    valP        <- PC+6
Decode:
    valA        <- R[rA]
    valB        <- R[rB]
Execute:
    valE        <- valC + (valB*ifun)
Memory:
    M4[valE]    <- valA
Write back:
PC update:
    PC          <- valP

Pseudo-code for mrmovl
Fetch:
    icode:ifun  <- M1[PC]
    rA:rB       <- M1[PC+1]
    valC        <- M4[PC+2]
    valP        <- PC+6
Decode:
    valB        <- R[rB]
Execute:
    valE        <- valC + (valB*ifun)
Memory:
    valM        <- M4[valE]
Write back:
    R[rA]       <- valM
PC update:
    PC          <- valP

Comments in the AbstractY86CPU state that Y86 CPU can take multiple arguments to accomplish the same
task in problem 1.
Those arguments being:
    - 4
    - 2
    - 1
    - <no arguments>
So the test is split into 4 different parts to test each of the 4 arguments respectively:
    - testx4
    - testx2
    - testx1
    - textxReg
Everything is the same in each task except that the values read into register/memory differ as well as
constants to adjust for offsets.

Each part of the test loads a value into register %ecx which will be copied into the first 2 slots of
an array usin the now improved rmmovl instruction. After successfuly inserting the values into the
array slots; it will copy the value into a new register using the improved mrmovl instruction. This
```

```
will repeat itself two more times using different positions in memory.

These tests give sufficient coverage of the problem set as it tests all aspects of the newly improved
rmmovl/mrmovl instructions; It inserts different values into memory and then loads these values into
registers. This test is repeated for all possible arguments; verifying that all intended/supported
arguments work and that the functionality of the function is kept in tact as every slot in the array is
overwritten. The test also covers the possiblities of an invalid argument in section testxlnv; When
uncommented, Y86 will fail to load movl.s due to an illegal scale.


Test results for movl.s
-----------------------
NOTE: The best way to run the test is to have break points at each of labels.
Label “testxInvalid”, test invalid arguments for rmmovl and mrmovl.
Uncommenting line 84 or 85 will result an “Assembly error: Illegal scale: must be 1, 2, or 4.”

Pretest:
All values of TheArray should be 7.

Values after running “testx4”:
.pos 0x1000
TheArray:   .long   40
            .long   40
            .long   41
            .long   41
            .long   42
            .long   42
%eax        40
%esi        41
%edi        42

Values after running “testx2”:
.pos 0x1000
TheArray:   .long   20
            .long   20
            .long   21
            .long   21
            .long   22
            .long   22
%eax        20
%esi        21
%edi        22

Values after running “testx1”:
.pos 0x1000
TheArray:   .long   10
            .long   10
            .long   11
            .long   11
            .long   12
            .long   12
%eax        10
%esi        11
%edi        12

Values after running “testxReg”:
.pos 0x1000
TheArray:   .long   0
            .long   0
            .long   1
            .long   1
            .long   2
            .long   2
```

```
%eax        0
%esi        1
%edi        2


------------------------------
Problem 2
------------------------------

Fetch:
    icode:ifun  <- M1[PC]
    rA:rB       <- M1[PC+1]
    valC        <- M4[PC+2]
    valP        <- PC+6
Decode:
    valB        <- R[rB]
Execute:
    valE        <- valB [+, -, *, /, and, or, xor] valC
Memory:
Write back:
    R[rB]       <- valE
PC Update:
    PC          <- valP


Test coverage of iopl.s
-----------------------
iopl.s is comprised of 7 different parts to ensure each arithmatic function operates correctly.
iopl.s provides sufficient test coverage as it tests each arithmatic function twice as well as tests
for function which can be used to mimic i_opl's operations (ala dividing by 0).
Arithmatic operations are checked using i_opl's untouched implemenation and errors are handled to
determine which arithmatic operation was not operating correctly.


Test results for iopl.s
-----------------------
Simply run the test and check %eax value at the end.
If %eax’s value is:
    0 = test passed
    1 = iaddl error
    2 = isubl error
    3 = imul error
    4 = idivl error
    5 = iandl error
    6 = ixorl error
    7 = imodl error


------------------------------
Problem 3
------------------------------

Fetch:
    icode:ifun  <- M1[PC]
    rA:rB       <- M1[PC+1]
    valC        <- M4[PC+2]
    valP        <- PC+6
Decode:
    valB        <- R[rB]
Execute:
    valE        <- valC + (valB*ifun)
Memory:
    valM        <- M4[valE]
```

```
    valE        <- valP
Write back:
    R[rA]       <- valE
PC update:
    PC          <- valM


Test coverage of call.s
-----------------------
call.s provides sufficient test coverage as it checks to see if both the regression as well as newly
implemented function work. By successfully coming to the halt statment shows both functions work
properly.

Test results of call.s
----------------------
Run call.s
if:
    %eax = 7
    %edx = 0x117
    %ebx = 8
    %ecx = 4
    Progam successfully halts
Then:
    Test passed and ran succesfully.
```