Names: Steven Pitts and Skyelar Craver
Course Code: 20636
Course Name: Operating Systems for Engineers
1/10/19 mm/dd/yy form
Lab 01
By submitting this assessment, I hereby declare that I have neither given nor received any unauthorized help or used any unauthorized materials during this assessment, and that I have adhered to any additional policies regarding Academic Honesty set by Wentworth Institute of Technology. Please sign below to acknowledge this or your assessment will not be graded.
Student Signature:

Date:

1/10/19 mm/dd/yy form

Part 1:

```
BEGIN SIMULATION
Instruction Memory
300:1940        301:5941        302:2941        303:0000        304:0000        305:0000        306:0000        307:0000
308:0000        309:0000        30a:0000        30b:0000        30c:0000        30d:0000        30e:0000        30f:0000
Data Memory
940:3   941:2   942:0   943:0   944:0   945:0   946:0   947:0   948:0   949:0   94a:0   94b:0   94c:0   94d:0   94e:0   94f:0
Device 5        2       13      0       0       0       0       0       0       0
Device 6        0       0       0       0       0       0       0       0       0
step: 1
Fetch instruction from memory location 300
AC:     0       PC:     300     IR:     1940
step: 2
Execute the instruction and increment the PC: Load AC from memory location 940
AC:     3       PC:     301     IR:     1940
step: 3
Fetch instruction from memory location 301
AC:     3       PC:     301     IR:     5941
step: 4
Execute the instruction and increment the PC: Add to AC from memory location 941
AC:     5       PC:     302     IR:     5941
step: 5
Fetch instruction from memory location 302
AC:     5       PC:     302     IR:     2941
step: 6
Execute the instruction and increment the PC: Store AC to memory location 941
AC:     5       PC:     303     IR:     2941
Instruction Memory
300:1940        301:5941        302:2941        303:0000        304:0000        305:0000        306:0000        307:0000
308:0000        309:0000        30a:0000        30b:0000        30c:0000        30d:0000        30e:0000        30f:0000
Data Memory
940:3   941:5   942:0   943:0   944:0   945:0   946:0   947:0   948:0   949:0   94a:0   94b:0   94c:0   94d:0   94e:0   94f:0
Device 5        2       13      0       0       0       0       0       0       0
Device 6        0       0       0       0       0       0       0       0       0
END SIMULATION
```

Part 2:

```
static MemoryLine instruction_memory_2[10] =
{
   {0x3005},//Load AC from device 5.
   {0x5940},//Add AC with contents of memory location 940.
   {0x9941},//Divide AC by contents of memory location 941.
   {0x7006},//Store AC to device 6).
   {0x2942},//Store AC to memory location 942.
   {0x3005},//Load AC from device 5.
   {0x8942},//Multiply AC with contents of memory location 942.
   {0x6943},//Subtract contents of memory location 943 from the AC.
   {0x2944},//Store AC to memory location 944.
   {0x7006}//Store AC to device 6.
};
```

```
BEGIN SIMULATION
Instruction Memory
300:3005        301:5940        302:9941        303:7006        304:2942        305:3005        306:8942        307:6943
308:2944        309:7006        30a:0000        30b:0000        30c:0000        30d:0000        30e:0000        30f:0000
Data Memory
940:3   941:5   942:0   943:10  944:0   945:0   946:0   947:0   948:0   949:0   94a:0   94b:0   94c:0   94d:0   94e:0   94f:0
Device 5        2       13      0       0       0       0       0       0       0
Device 6        0       0       0       0       0       0       0       0       0
step: 1
Fetch instruction from memory location 300
AC:     0       PC:     300     IR:     3005
step: 2
Execute the instruction and increment the PC: Load to AC from device 5
AC:     2       PC:     301     IR:     3005
step: 3
Fetch instruction from memory location 301
AC:     2       PC:     301     IR:     5940
step: 4
Execute the instruction and increment the PC: Add to AC from memory location 940
AC:     5       PC:     302     IR:     5940
step: 5
Fetch instruction from memory location 302
AC:     5       PC:     302     IR:     9941
step: 6
Execute the instruction and increment the PC: Divide by AC from memory location 941
AC:     1       PC:     303     IR:     9941
step: 7
Fetch instruction from memory location 303
AC:     1       PC:     303     IR:     7006
step: 8
Execute the instruction and increment the PC: Store AC to device 6
AC:     1       PC:     304     IR:     7006
step: 9
Fetch instruction from memory location 304
AC:     1       PC:     304     IR:     2942
step: 10
Execute the instruction and increment the PC: Store AC to memory location 942
AC:     1       PC:     305     IR:     2942
step: 11
Fetch instruction from memory location 305
AC:     1       PC:     305     IR:     3005
step: 12
Execute the instruction and increment the PC: Load to AC from device 5
AC:     13      PC:     306     IR:     3005
step: 13
Fetch instruction from memory location 306
AC:     13      PC:     306     IR:     8942
step: 14
Execute the instruction and increment the PC: Multiply by AC from memory location 942
AC:     13      PC:     307     IR:     8942
step: 15
Fetch instruction from memory location 307
AC:     13      PC:     307     IR:     6943
step: 16
Execute the instruction and increment the PC: Subtract from AC from memory location 943
AC:     3       PC:     308     IR:     6943
step: 17
Fetch instruction from memory location 308
AC:     3       PC:     308     IR:     2944
step: 18
Execute the instruction and increment the PC: Store AC to memory location 944
AC:     3       PC:     309     IR:     2944
step: 19
Fetch instruction from memory location 309
AC:     3       PC:     309     IR:     7006
step: 20
Execute the instruction and increment the PC: Store AC to device 6
AC:     3       PC:     30a     IR:     7006
Instruction Memory
300:3005        301:5940        302:9941        303:7006        304:2942        305:3005        306:8942        307:6943
308:2944        309:7006        30a:0000        30b:0000        30c:0000        30d:0000        30e:0000        30f:0000
Data Memory
940:3   941:5   942:1   943:10  944:3   945:0   946:0   947:0   948:0   949:0   94a:0   94b:0   94c:0   94d:0   94e:0   94f:0
Device 5        2       13      0       0       0       0       0       0       0
Device 6        1       3       0       0       0       0       0       0       0
END SIMULATION
```

Part 3 + bonus:

```
BEGIN SIMULATION
Instruction Memory
300:3005        301:5940        302:9941        303:7006        304:2942        305:3005        306:8942        307:6943
308:2944        309:7006        30a:0000        30b:0000        30c:0000        30d:0000        30e:0000        30f:0000
Data Memory
940:3   941:5   942:0   943:10  944:0   945:0   946:0   947:0   948:0   949:0   94a:0   94b:0   94c:0   94d:0   94e:0   94f:0
Device 5        2       13      0       0       0       0       0       0       0
Device 6        0       0       0       0       0       0       0       0       0
step: 1
Fetch instruction from memory location 300
AC:     0       PC:     300     IR:     3005
step: 2
Execute the instruction and increment the PC: Load to AC from device 5
AC:     2       PC:     301     IR:     3005
step: 3
Fetch instruction from memory location 301
AC:     2       PC:     301     IR:     5940
step: 4
Execute the instruction and increment the PC: Add to AC from memory location 940
AC:     5       PC:     302     IR:     5940
step: 5
Fetch instruction from memory location 302
AC:     5       PC:     302     IR:     9941
step: 6
Execute the instruction and increment the PC: Divide by AC from memory location 941
AC:     1       PC:     303     IR:     9941
step: 7
Fetch instruction from memory location 303
AC:     1       PC:     303     IR:     4000
step: 8
Execute the instruction and increment the PC: INTERRUPT 0
AC:     1       PC:     304     IR:     4000
step: 9
Fetch instruction from memory location 304
AC:     1       PC:     304     IR:     7006
step: 10
Execute the instruction and increment the PC: Store AC to device 6
AC:     1       PC:     305     IR:     7006
step: 11
Fetch instruction from memory location 305
AC:     1       PC:     305     IR:     2942
step: 12
Execute the instruction and increment the PC: Store AC to memory location 942
AC:     1       PC:     306     IR:     2942
step: 13
Fetch instruction from memory location 306
AC:     1       PC:     306     IR:     4000
step: 14
Execute the instruction and increment the PC: INTERRUPT 0
AC:     1       PC:     307     IR:     4000
step: 15
Fetch instruction from memory location 307
AC:     1       PC:     307     IR:     3005
step: 16
Execute the instruction and increment the PC: Load to AC from device 5
AC:     13      PC:     308     IR:     3005
step: 17
Fetch instruction from memory location 308
AC:     13      PC:     308     IR:     8942
step: 18
Execute the instruction and increment the PC: Multiply by AC from memory location 942
AC:     13      PC:     309     IR:     8942
step: 19
Fetch instruction from memory location 309
AC:     13      PC:     309     IR:     4000
step: 20
Execute the instruction and increment the PC: INTERRUPT 0
AC:     13      PC:     30a     IR:     4000
step: 21
Fetch instruction from memory location 30a
AC:     13      PC:     30a     IR:     6943
step: 22
Execute the instruction and increment the PC: Subtract from AC from memory location 943
AC:     3       PC:     30b     IR:     6943
step: 23
Fetch instruction from memory location 30b
AC:     3       PC:     30b     IR:     2944
step: 24
Execute the instruction and increment the PC: Store AC to memory location 944
AC:     3       PC:     30c     IR:     2944
step: 25
Fetch instruction from memory location 30c
AC:     3       PC:     30c     IR:     7006
step: 26
Execute the instruction and increment the PC: Store AC to device 6
AC:     3       PC:     30d     IR:     7006
Instruction Memory
300:3005        301:5940        302:9941        303:4000        304:7006        305:2942        306:4000        307:3005
308:8942        309:4000        30a:6943        30b:2944        30c:7006        30d:0000        30e:0000        30f:4000
Data Memory
940:3   941:5   942:1   943:10  944:3   945:0   946:0   947:0   948:0   949:0   94a:0   94b:0   94c:0   94d:0   94e:0   94f:0
Device 5        2       13      0       0       0       0       0       0       0
Device 6        1       3       0       0       0       0       0       0       0
END SIMULATION
```

Code:
Processor.h:

```
/*
    Skyelar Craver
    Steven Pitts
    Operating Systems
    Lab 01
    Spring 2018
*/

#pragma once

#include <stdlib.h>
#include <stdio.h>

// static definitions of memory offsets
static const uint16_t INSTRUCTION_ADDRESS_OFFSET = 0x940;
static const uint16_t DATA_ADDRESS_OFFSET = 0x300;

// definitions of valid opcodes for instructions
enum Opcodes
{
    HALT = 0,
    LOAD_ACC = 0b0001,
    STORE_ACC = 0b0010,
    ADD_ACC = 0b0101,
    LOAD_IO = 0b0011,
    SUB_ACC = 0b0110,
    STORE_IO = 0b0111,
    MUL_ACC = 0b1000,
    DIV_ACC = 0b1001,
    INTERRUPT = 0b0100
};

// bitfield packed 16-bit struct to hold strongly typed Instructions
typedef struct Instruction
{
    uint16_t opcode : 4;
    uint16_t address : 12;
} Instruction;

// generic device struct
typedef struct IODevice
{
    int16_t buffer[10];
    uint16_t buffer_index;
} IODevice;

// part 1 program data
// static Instruction instruction_memory[16] =
```

```
// {
//     {0x1,0x940},
//     {0x5,0x941},
//     {0x2,0x941}
// };

// part 2 + bonus program data
static Instruction instruction_memory[16] =
{
    {0x3, 0x005},  // Load AC from device 5.
    {0x5, 0x940},  // Add AC with contents of memory location 940.
    {0x9, 0x941},  // Divide AC by contents of memory location 941.
    {0x7, 0x006},  // Store AC to device 6.
    {0x2, 0x942},  // Store AC to memory location 942.
    {0x3, 0x005},  // Load AC from device 5.
    {0x8, 0x942},  // Multiply AC with contents of memory location 942.
    {0x6, 0x943},  // Subtract contents of memory location 943 from the AC.
    {0x2, 0x944},  // Store AC to memory location 944.
    {0x7, 0x006},  // Store AC to device 6.
};

// part 3 program data
// static Instruction instruction_memory[16] =
// {
//     {0x3, 0x005},  // Load AC from device 5.
//     {0x4, 0x000},  // insterted interupt
//     {0x5, 0x940},  // Add AC with contents of memory location 940.
//     {0x9, 0x941},  // Divide AC by contents of memory location 941.
//     {0x7, 0x006},  // Store AC to device 6.
//     {0x4, 0x000},  // insterted interupt
//     {0x2, 0x942},  // Store AC to memory location 942.
//     {0x3, 0x005},  // Load AC from device 5.
//     {0x4, 0x000},  // insterted interupt
//     {0x8, 0x942},  // Multiply AC with contents of memory location 942.
//     {0x6, 0x943},  // Subtract contents of memory location 943 from the AC.
//     {0x2, 0x944},  // Store AC to memory location 944.
//     {0x7, 0x006},  // Store AC to device 6.
//     {0x4, 0x000},  // insterted interupt
// };

// part 1 data pre-fill
// static uint16_t data_memory[16] =
// {
//     3,
//     2
// };

// part 2+ data pre-fill
static int16_t data_memory[16] =
{
```

```
    3,
    5,
    0,
    10
};

// attached device array (only 5 and 6 used)
static IODevice devices[10];

// prints static array contents
void PrintMemory()
{
    printf("Instruction Memory");
    for (size_t i = 0; i < 16; i++)
        {
            if (!(i % 8)) printf("\n");
            printf("%x:%01x%03x\t", i + 0x300, instruction_memory[i].opcode,
instruction_memory[i].address);
        }
        printf("\nData Memory\n");
        for (size_t i = 0; i < 16; i++)
        {
            printf("%x:%d\t", i + INSTRUCTION_ADDRESS_OFFSET, data_memory[i]);
        }
        printf("\nDevice 5\t");
        for (size_t i = 0; i < 9; i++)
        {
            printf("%d\t", devices[5].buffer[i]);
        }
        printf("\nDevice 6\t");
        for (size_t i = 0; i < 9; i++)
        {
            printf("%d\t", devices[6].buffer[i]);
        }
        printf("\n");
}

// inserts interrupts for bonus, slides array after insertion
void InsertInterrupt(uint16_t index)
{
    Instruction interupt = {0x4,0x000};
    Instruction existing_instruction = instruction_memory[index];
    instruction_memory[index] = interupt;
    for (size_t i = index + 1; i < 15; i++)
    {
        Instruction last_instruction = instruction_memory[i];
        instruction_memory[i] = existing_instruction;
        existing_instruction = last_instruction;
    }
}
```

Processor.c:

```c
/*
    Skyelar Craver
    Steven Pitts
    Operating Systems
    Lab 01
    Spring 2018
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>

#include "processor.h"

int main(int argc, char const *argv[])
{
    // setup block for initializing
    srand(time(NULL)); // seed the PRNG
    uint8_t instruction_index = 0;
    int16_t accumulator = 0;
    uint16_t data_index;
    uint16_t buffer_index;
    uint16_t step_count = 1;

    devices[5].buffer[0] = 2;
    devices[5].buffer[1] = 13;

    // messages to dynamically build printf statements
    const char execute_context_descriptions[][100] =
    {
        "HALT",
        "Load AC from memory location",
        "Store AC to memory location",
        "Load to AC from device",
        "INTERRUPT","Add to AC from memory location",
        "Subtract from AC from memory location",
        "Store AC to device","Multiply by AC from memory location",
        "Divide by AC from memory location"
    };
    // Format with execute context description and memory location
    const char execute_context[] = "Execute the instruction and increment the PC: %s %x\n";
    // Format with memory location
    const char fetch_context[] = "Fetch instruction from memory location %x\n";

    printf("BEGIN SIMULATION\n");
    PrintMemory();

    // main loop
```

```c
    // runs until a HALT (opcode 0)
    while (instruction_memory[instruction_index].opcode)
    {
        Instruction current_instruction = instruction_memory[instruction_index];
        printf("step: %u\n", step_count++);
        printf(fetch_context, instruction_index + 0x300);
        // print register contents
        printf("AC: \t%d\tPC:\t%x\tIR:\t%01x%03x\n",
               accumulator,
               (instruction_index + 0x300),
               current_instruction.opcode,
               current_instruction.address
               );
        instruction_index++;

        printf("step: %u\n", step_count++);
        if (current_instruction.opcode <= 10)
        {
            printf(execute_context,
                   execute_context_descriptions[current_instruction.opcode],
                   current_instruction.address
                   );
        }
        else
        {
            printf(execute_context,execute_context_descriptions[0],
                   current_instruction.address
                   );
            return 0;
        }

        // remove offset
        data_index = current_instruction.address - INSTRUCTION_ADDRESS_OFFSET;
        buffer_index = devices[current_instruction.address].buffer_index;

        // switch on opcode to determine operation
        switch (current_instruction.opcode)
        {
            case LOAD_ACC:
                accumulator = data_memory[data_index];
                break;
            case STORE_ACC:
                data_memory[data_index] = accumulator;
                break;
            case LOAD_IO:
                accumulator = devices[current_instruction.address].buffer[buffer_index];
                devices[current_instruction.address].buffer_index++;
                InsertInterrupt(instruction_index + ((rand() % 2) + 1));
                break;
            case STORE_IO:
```

```
                devices[current_instruction.address].buffer[buffer_index] = accumulator;
                devices[current_instruction.address].buffer_index++;
                InsertInterrupt(instruction_index + ((rand() % 2) + 1));
                break;
            case ADD_ACC:
                accumulator += data_memory[data_index];
                break;
            case SUB_ACC:
                accumulator -= data_memory[data_index];
                break;
            case MUL_ACC:
                accumulator *= data_memory[data_index];
                break;
            case DIV_ACC:
                accumulator /= data_memory[data_index];
                break;
            case INTERRUPT:
                break;
            default:
                printf("Unsupported instruction %01x called\n",
                       current_instruction.opcode
                       );
                break;
        }
        // print register contents
        printf("AC: \t%d\tPC:\t%x\tIR:\t%01x%03x\n",
               accumulator,
               (instruction_index + 0x300),
               current_instruction.opcode,
               current_instruction.address
               );
    }

    PrintMemory();
    printf("END SIMULATION\n");
    return 0;
}
```