```
// Andrea Marquez Tavera
// COP3402
// PM/0 VM

// Header files, all the necessary libraries
#include <stdio.h>
#include <stdlib.h>

typedef struct instructions instructions;// struct for instructions in "text"
portion of the PAS

struct instructions{
    int OP; // Operation code
    int L; // Lexicographical level
    int M; // Modifier
};

// Macros
#define MAX_LINES 150 // max amount of instructions
#define ARRAY_SIZE 512 // max length for PAS array

// Process Address Space Array
int PAS[ARRAY_SIZE];

// Array to keep track of difference from sp and bp for each AR
int stackDiff[ARRAY_SIZE];

// Array to keep track of where the BP is for each AR
int basePtrs[ARRAY_SIZE];

// functions
int base(int BP, int L);// Find base L levels down

int main(int argc, char * argv[])
{
    FILE *inputPTR;// file pointer
    inputPTR = fopen(argv[1], "r");// opens file from command line and allows
reading

    instructions IR;// struct for Instruction Register

    if(inputPTR == NULL)// in case file doesn't exist, terminate program
    {
        printf("Failed to open file\n");
        return -1;
    }
    // Initial process address space values are all zero
    for(int i = 0; i < ARRAY_SIZE; i++)
    {
        PAS[i] = 0;
    }

    int BP = 0;// creates BP variable

    int index = 0;//will keep track of how many values have been scanned from IR

    // while loop to read ELF file
    while(!feof(inputPTR))// as long as there still lines left to read, scan
    {
```

```
        fscanf(inputPTR, "%d %d %d", &PAS[index], &PAS[index + 1], &PAS[index +
2]);
        index += 3;
        BP = index;// default BP Value
    }

    // DEFAULT SP AND PC VALUES
    int SP = BP - 1;
    int PC = 0;
    int ARNum = 0;// number of AR's
    int ARFlag = 0;// Flag to make sure there are AR's

    printf("\t\tPC\tBP\tSP\tstack\n");
    printf("Initial values: %d\t%d\t%d\n\n", PC, BP, SP);

    int haltFlag = 1;
    while (haltFlag != 0)// as long as the program is not done, keep looping
    {
        // initialize OP, L, and M per instruction line
        IR.OP = PAS[PC];
        IR.L = PAS[PC + 1];
        IR.M = PAS[PC + 2];
        PC += 3;// increment PC by 3 since there is 3 values per instruction, op,
l, and m

        // Execute
        if (IR.OP == 1) { // LIT 0, M
            SP = SP + 1;
            PAS[SP] = IR.M;
            printf("  LIT %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
            ARFlag = 1;
        } else if (IR.OP == 2) { // OPR 0, #
            if (IR.M == 0) { // RTN
                SP = BP - 1;
                BP = PAS[SP + 2];
                PC = PAS[SP + 3];
                printf("  RTN %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
                stackDiff[ARNum] = 0;
                basePtrs[ARNum] = 0;
                ARNum--;
            } else if (IR.M == 1) { // ADD
                PAS[SP -1] = PAS[SP - 1] + PAS[SP];
                SP = SP - 1;
                printf("  ADD %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
            } else if (IR.M == 2) { // SUB
                PAS[SP - 1] = PAS[SP - 1] - PAS[SP];
                SP = SP - 1;
                printf("  SUB %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
            } else if (IR.M == 3) { // MUL
                PAS[SP - 1] = PAS[SP - 1] * PAS[SP];
                SP = SP - 1;
                printf("  MUL %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
            } else if (IR.M == 4) { // DIV
                PAS[SP - 1] = PAS[SP - 1] / PAS[SP];
                SP = SP - 1;
                printf("  DIV %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
            } else if (IR.M == 5) { // EQL
                PAS[SP - 1] = PAS[SP - 1] == PAS[SP];
                SP = SP -1;
```

```
            printf("  EQL %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
        } else if (IR.M == 6) { // NEQ
            PAS[SP - 1] = PAS[SP - 1] != PAS[SP];
            SP = SP - 1;
            printf("  NEQ %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
        } else if (IR.M == 7) { // LSS
            PAS[SP - 1] = PAS[SP - 1] < PAS[SP];
            SP = SP - 1;
            printf("  LSS %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
        } else if (IR.M == 8) { // LEQ
            PAS[SP - 1] = PAS[SP - 1] <= PAS[SP];
            SP = SP - 1;
            printf("  LEQ %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
        } else if (IR.M == 9) { // GTR
            PAS[SP - 1] = PAS[SP - 1] > PAS[SP];
            SP = SP - 1;
            printf("  GTR %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
        } else if (IR.M == 10) { // GEQ
            PAS[SP - 1] = PAS[SP - 1] >= PAS[SP];
            SP = SP - 1;
            printf("  GEQ %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
        }

    } else if (IR.OP == 3) { // LOD L, M
        SP = SP + 1;
        PAS[SP] = PAS[base(BP, IR.L) + IR.M];
        printf("  LOD %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);

    } else if (IR.OP == 4) { // STO L, M
        PAS[base(BP, IR.L) + IR.M] = PAS[SP];
        SP = SP - 1;
        printf("  STO %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);

    } else if (IR.OP == 5) { // CAL L, M
        PAS[SP + 1] = base(BP, IR.L); // Static Link (SL)
        PAS[SP + 2] = BP; // Dynamic Link (DL)
        PAS[SP + 3] = PC; // Return Address (RA)

        BP = SP + 1;
        PC = IR.M;

        printf("  CAL %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);

        ARFlag = 1;
        if(ARFlag == 1)
            ARNum++;

    } else if (IR.OP == 6) { // INC 0, M
        SP = SP + IR.M;
        printf("  INC %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
        ARFlag = 1;

    } else if (IR.OP == 7) { // JMP 0, M
        PC = IR.M;
        printf("  JMP %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);

    } else if (IR.OP == 8) { // JPC 0, M
        if (PAS[SP] == 0)
        {
```

```
                PC = IR.M;
            }
            SP = SP - 1;
            printf("  JPC %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);

        } else if (IR.OP == 9) { // SYS 0, #
            if (IR.M == 1)
            {
                printf("Output result is: %d\n", PAS[SP]);
                SP = SP - 1;
                printf("  SYS %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
            } else if (IR.M == 2) {
                printf("Please Enter an Integer: ");
                SP = SP + 1;
                scanf("%d", &PAS[SP]);
                printf("  SYS %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
                ARFlag = 1;
            } else if(IR.M == 3) {
                printf("  SYS %d\t%d\t%d\t%d\t%d\t", IR.L, IR.M, PC, BP, SP);
                haltFlag = 0;
            }
        }

        //update arrays
        basePtrs[ARNum] = BP;
        stackDiff[ARNum] = SP - BP;

        // print stack
        if(ARFlag == 1) {
            stackDiff[ARNum] = SP - BP + 1;
            basePtrs[ARNum] = BP;

            for(int i = 0;i < (ARNum + 1); i++)
            {
                for(int j = 0; j < stackDiff[i]; j++)
                {
                    if(i != 0 && j == 0)
                    {
                        printf("| ");// print before new AR (solved extra | issue)
                    }
                    printf("%d ", PAS[basePtrs[i] + j]);
                }
            }
        }

        printf("\n");// new line in between each buffer
    }
    fclose(inputPTR);// close file
    return 0;// exit program
}

int base(int BP, int L)
{
    int arb = BP; // arb = activation record base
    while ( L > 0)     //find base L levels down
    {
        arb = PAS[arb];
        L--;
    }
```

```
    return arb;
}
```