Project 3

Summary

Build upon Project 2 and implement a virtual machine which can execute the test program outlined below.

Submit

A zip file in Canvas.

- Your zip file should be named <YourName>_p3.zip, where <YourName> is replaced by your name.
- Your zip file should contain your makefile, assembly file, include files, and source code.

Test Program

Convert the following program to assembly; do not try to rewrite it:

```
const int SIZE = 7;
int cnt;
int tenth;
char c[SIZE];
int data;
int flag;
int opdv;

/*
  Convert char j to an integer if possible.
  If the flag is not set, use the sign indicator and
  the tenths indicator to compute the actual
  value of j.  Add the value to the accumulator opdv.
*/

void opd(char s, int k, char j) {
  int t = 0; // Local var
  if (j == '0') // Convert
    t = 0;
  else if (j == '1')
    t = 1;
  else if (j == '2')
    t = 2;
  else if (j == '3')
    t = 3;
  else if (j == '4')
    t = 4;
  else if (j == '5')
    t = 5;
  else if (j == '6')
    t = 6;
  else if (j == '7')
    t = 7;
  else if (j == '8')
    t = 8;
  else if (j == '9')
    t = 9;
  else {
    printf("%c is not a number\n", j);
```

```
    flag = 1;
  }

  if (!flag) {
    if (s == '+')
      t *= k;
    else
      t *= -k;
    opdv += t;
  }
}


/*
  Discard keyboard input until a newline 'n' is encountered.
*/

void flush() {
  data = 0;
  c[0] = getchar();
  while (c[0] != '\n')
    c[0] = getchar();
}


/*
  Read one char at a time from the keyboard after a newline 'n'
  has been entered. If there is room, place the char in the array c.
  Otherwise, indicate the number is too big and flush the keyboard input.
*/

void getdata() {
  if (cnt < SIZE) { // Get data if there is room
    c[cnt] = getchar();  // Your TRP 4 should work like getchar()
    cnt++;
  }
  else {
    printf("Number too big\n");
    flush():
  }
}


/*
  Reset c to all 0s.
  Assign values to data, opdv, cnt and flag.
*/

void reset(int w, int x, int y, int z) {
  int k;  // Local var
  for (k = 0; k < SIZE; k++)
    c[k] = 0;
  data = w;
  opdv = x;
  cnt = y;
  flag = z;
}
```

```
/*
  Get input from the keyboard until the symbol '@' is encountered.
  Convert the char data input from the keyboard to an integer.
  Account for the sign of the number.
  If no sign is used, always assume the number is positive.
*/

void main() {
  reset(1, 0, 0, 0); // Reset globals
  getdata();
  while (c[0] != '@') { // Check for stop symbol '@'
    if (c[0] == '+' || c[0] == '-') { // Determine sign
      getdata(); // Get most significant byte
    }
    else { // Default sign is '+'
      c[1] = c[0]; // Make room for the sign
      c[0] = '+';
      cnt++;
    }
    while(data) { // Loop while there is data to process
      if (c[cnt - 1] == '\n') { // Process data now
        data = 0;
        tenth = 1;
        cnt = cnt - 2;
        while (!flag && cnt != 0) { // Compute a number
          opd(c[0], tenth, c[cnt]);
          cnt--;
          tenth *= 10;
        }
        if (!flag) // Good number entered
          printf("Operand is %d\n", opdv);
      }
      else
        getdata(); // Get next byte of data
    }
    reset(1, 0, 0, 0); // Reset globals
    getdata(); // Get data
  }
}
```

Grading

The project is worth 100 points. Systematic deductions are taken off the top before individual project deductions are made. Systematic deductions are like penalties for doing something a senior shouldn't do because they are just too fundamental and simple.

Systematic Deductions

Late	-100
Naming executable or asm file incorrectly	-15
Sending an un-executable project	-30
Doubles with each offense	
VM hangs when finished	-15
VM crashes during execution	-50 or greater
Debug Output	-15
Failure to send all the elements in your zip	-15
Not following instructions, which forces the project to be regraded (e.g., altering instructions)	-20

Project Deductions

Each major element	-20 max

Hints

- Procedure calls
  - Procedure call at any location
  - Each call is matched with a return
  - Procedure call can be nested
  - Procedure call is recursive
- Using registers for procedure calls
  - Any location is okay
  - Match with a return is okay
  - Nesting is limited
  - Recursion is not applicable
- Using a stack
  - Any location
  - Match with a return
  - Recursion is okay
- Just implement the given program in assembly. Do not try to rewrite it.
- Implement TRP 4 to work just like the getchar() in C. You don't have to implement the getchar() in assembly.
  - The '\n' typed at the keyboard is seen by the program.
  - A <return> must be typed before the input is read.
- Only registers can be used to pass parameters and store the return address
  - A limited amount of nested function calls can be made.
  - Recursive calls can't be made
- Create an activation record on the run-time stack for each function call.
  - This is a great solution, but more difficult to implement.
  - ***This is the best option for project 4.***
- Mixed solution
  - Put the return address on the run-time stack, but pass arguments using registers.
- Add registers SL, SP, FP, and SB
  - You should already be using R0-R15
  - You should already be using PC (== 16)
  - Add four new registers: SL == 17, SB == 18, SP == 19, and FP == 20

- Your ASM ***must*** GET/SET the PC properly
  - Only use ***MOV***, to **GET** the PC and copy it into another register (to calculate a return address). For example:

```
MOV R1, PC
ADD R1, sizeof_instruction * instruction_count
```

  - Only use ***Jump/Branch*** instructions to **SET** the PC

```
BNZ ENDWHILE
```

  - ***DO NOT use LDA*** (Load Effective Address) to ***calculate and set a new PC***. *Only use LDA to load a pointer to a memory address into a register!* For example:

```
LDA R1, ARR
```