Project 2

Summary

Implement a virtual machine, building upon project 1. This vm should execute the test program outlined below and have implemented many of the instructions listed in the Instructions to Implement document. ***Bytecode must exactly match the instructor's*** unless you aren't implementing byte addressability. Don't forget to write the PC in the first four bytes of the bytecode. This implies that reserve space for the PC that your bytecode offset needs to start at 4, not 0. Please see the updated example_assembly_text.asm file and the updated example_bytecode.bin file (under Files | Projects in Canvas). Storing the PC at the beginning of the bytecode allows you, with minor changes to the main function, to assemble the code once and debug it often.

Submit

A zip file in Canvas.

- Your zip file should be named <YourName>_p2.zip, where <YourName> is replaced by your name.
- Your zip file should contain your makefile, assembly file, include files, and source code.

Specifications

Implement all of the following machine instructions. See the Instructions to Implement document for more details.

- Jump Instructions
- Move Instructions
- Compare Instructions
- Register indirect Addressing

Devise a test to ensure that all these instructions have been implemented correctly.

Details

Test Program

Take the following C-like program and rewrite it your assembly language. Make sure you implement your code such that ARR is accessed as an array. Your code must loop SIZE number of times. There must be variables named ARR and SIZE in your assembly program.

Part 1

```
int SIZE = 10;
int ARR[] = { 10, 2, 3, 4, 15, -6, 7, 8, 9, 10 };
int i = 0;
int sum = 0;
int temp;
int result;

while (i < SIZE) {
      sum += ARR[i];
      result = ARR[i] % 2; // Do this using simple algebra
      if (result == 0)
            printf("%d is even\n", ARR[i]);
      else
            printf("%d is odd\n", ARR[i]);
}
```

```
printf("Sum is %d\n", sum);
printf("");
```

Part 2

Note: Only do this if you have byte addressable memory. If your project does not support byte addressable memory do not attempt Part 2 just skip it. If you implement Part 2 but don't have byte addressable memory, I will conclude you are trying to deceive me and give you a failing grade (30% or lower) for Project 2.

Place DAGS into continuous memory (on an integer boundary if it matters to your VM. It has never mattered to anyone up until now). Access DAGS as an integer and store the integer value for "DAGS" into a memory location labeled GADS.

```
DAGS  .BYT 'D'
      .BYT 'A'
      .BYT 'G'
      .BYT 'S'
GADS  .INT -99   ; Will end up with value of DAGS
```

Using the memory location labeled GADS, swap the 'D' (1st byte) and the 'G' (3rd byte) in memory. Note memory will contain 'G', 'A', 'D', 'S'.

Write assembly code to do the following comparison between the bytes of DAGS and bytes of GADS. This can be implemented as either a for loop or a while loop.

```
char rel;
for (i = 0; i < 4; i++) {
      if (DAGS[i] < GADS[i]) rel = '<'
      else if (DAGS[i] > GADS[i]) rel = '>'
      else rel = '=';
      printf("%c %c %c--", DAGS[i], rel, GADS[i]);
}
printf("")
```

Next, in assembly, access DAGS as an integer and GADS as an integer. Subtract GADS from DAGS (DAGS - GADS) and print the result as follows.

```
print("%d - %d = %d\n", DAGS_value, GADS_value, value);
```

Note: If your integer values don't match that of other students (e.g., it will be positive), check if your programming language environment is using Big Endian vs. Little Endian. Everyone should be using Little Endian (e.g., I've only seen this as a problem with Java byte array or if you create your own integer format.)

Program Output

Print the integer in the array followed by "is even" if it is even (e.g., ### is even).
Print the integer in the array followed by "is odd" if it is odd (e.g, ### is odd).
Print "Sum is " followed by the sum of the array (e.g., Sum is ###).
Print a blank line.
Print out the result of comparing the bytes of DAGS and GADS.(e.g., A = A--)
Print a blank line.
Print the result of DAGS – GADS. (e.g., ### - ### = ###)

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 Sr. 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
Falsely showing byte addressability	-70 or greater
Missing byte addressability	-35

Hints

- You don't need to define a modulus operator '%', just figure out a way using the operations you have to compute if a number is even or odd. There is a simple solution.
- You don't need to define a printf function. Just figure out a way to print the desired output on the screen. Don't use strings. They don't exist for our purposes.
- You do not need to define an increment operator '++', just figure out a way to use the operations you have to add 1 to an existing variable.
- Use register indirect addressing to access the arrays
- You will need register indirect addressing EA = (R)
    - Load and store commands only (see below)
      ```
      LDR R1 R2
      STR R1 R2
      Also convert the byte versions of load and store
      LDB R1 R3
      STB R5 R4
      ```
- The arrays ARR, DAGS, and GADS must have only one label associated with its data elements.
- LDA is an important instruction to you.
  ```
  LDA R1 ARR
  LDR R5 R1
  ```

Converting Control Structures to Assembly

If Statement

```
if (a == b)  { … }  X

LDR R3, A
LDR R4, B
CMP R3, R4 ; Notice the semantics for CMP
BNZ R3, SKIPIF
      |
Statements of the if
      |
SKIPIF: X
```

If-Else Statement

```
if (a == b)  { … }
else { … } X

LDR R3, A ; R3 gets A
LDR R4, B ; R4 gets B
CMP R3, R4
BNZ R3, ELSE
      |
Statements of the if
      |
JMP ENDIF
ELSE:
      |
Statements of the else
      |
ENDIF: X
```

While Statement

```
while (a == b)  { …  }  X

LDR R3, A
LDR R4, B
WHILE:
CMP R3, R4
BNZ R3, ENDWHILE
      |
Statements of the while
      |
JMP WHILE
ENDWHILE: X
```