Lab1 Report

Liron Cohen 207481268
Yuval Mor 209011543

Warm Up Questions

1. asm_cmd (LD, 0, 0, 5, 10)
2. asm_cmd (LD, 0, 0, 5, -512)
3. asm_cmd (SUB, 4, 3, 4, 0)
4. asm_cmd (JLT, 0, 3, 1, X) where X is the desired immediate.
5. In order to load a 32-bit constant into a register we can:
    a. Divide the constant into two parts of 16 bits each. X will denote the number that represents the low 16 bits and Y will denote the number that represents the high 16 bits.
    b. Use the ADD operation to load the low 16 bits (X) of the constant to a register. The command will be asm_cmd (ADD, 2, 0, 1, X)
    c. Use the LHI operation to load the high 16 bits (Y) of the constant to another register. The command will be asm_cmd (LHI, 3, 0, 1, Y)
    d. use the ADD operation to add the results together. The command will be asm_cmd (ADD, 4, 2, 3, 0)
6. In order to implement subroutine calls we need to jump between instructions and remember the PC counter before the call in order to come back to it after executing the instructions in the routine and keep executing the main program.
   For that, we can use the R7 register and when a jump into a subroutine is taken for PC X, we need to write X+1 into the R7 register.
   When the execution of the subroutine is finished, we can read X+1 from R7 (we can't use this register for any other purposes in the execution of the program) and execute the next instructions in the main programs.
   For nested subroutine, this implementation won’t work because register R7 will be ran over every subroutine. In order to support nested subroutine, we can use a stack of PCs instead of one register.

Example program

1. The program computes the cumulative sum of values from memory.
   The program stores the sum of all previous values (inclusive) in the memory.
2. The input is stored at mem [15] to mem [22].
3. The output is stored at mem [16] to mem [22].
4.

```
asm_cmd(ADD, 2, 1, 0, 15);// 0: R2 = 15
asm_cmd(ADD, 3, 1, 0, 1); // 1: R3 = 1
asm_cmd(ADD, 4, 1, 0, 8); // 2: R4 = 8
asm_cmd(JEQ, 0, 3, 4, 11); // 3: if R3 = R4, jump to 11
asm_cmd(LD,  5, 0, 2, 0); // 4: R5 = mem[R2 value]
asm_cmd(ADD, 2, 2, 1, 1); // 5: R2 = R2 + 1
asm_cmd(LD,  6, 0, 2, 0); // 6: R6 = mem[R2 value]
asm_cmd(ADD, 6, 6, 5, 0); // 7: R6 = R6 + R5
asm_cmd(ST,  0, 6, 2, 0); // 8: mem[R2 value] = R6
asm_cmd(ADD, 3, 3, 1, 1); // 9: R3 = R3 + 1
asm_cmd(JEQ, 0, 0, 0, 3); // 10: jump to 3
asm_cmd(HLT, 0, 0, 0, 0); // 11: halt
```

5. In every iteration, we can add the next number to the cumulative sum we calculated until now (without reading two values for every iteration)

```
asm_cmd(ADD, 2, 1, 0, 15);// 0: R2 = 15
asm_cmd(ADD, 3, 1, 0, 1); // 1: R3 = 1
asm_cmd(ADD, 4, 1, 0, 8); // 2: R4 = 8
asm_cmd(JEQ, 0, 3, 4, 12); // 3: if R3 = R4, jump to 11
asm_cmd(ADD, 5, 0, 6, 0); // 4: R5 = R6
asm_cmd(LD,  6, 0, 2, 0); // 5: R6 = mem[R2 value]
asm_cmd(ADD, 6, 6, 5, 0); // 6: R6 = R6 + R5
asm_cmd(ST,  0, 6, 2, 0); // 7: mem[R2 value] = R6
asm_cmd(ADD, 2, 2, 1, 1); // 8: R2 = R2 + 1
asm_cmd(ADD, 3, 3, 1, 1); // 9: R3 = R3 + 1
asm_cmd(JEQ, 0, 0, 0, 3); // 10: jump to 3
asm_cmd(HLT, 0, 0, 0, 0); // 11: halt
```

ISS simulator Testing #1

The assembly code of the problem is attached below.

The program divides the inputs into three flows - same sign numbers, different signs with $|A| > |B|$, and different signs with $|B| > |A|$. We used this division instead of four cases (for four combinations of signs) for time efficiency.

Other files are included in the zip file.

```
asm_cmd(LD, 2, 0, 1, 1000); // 0: R2 = MEM[1000] = A
asm_cmd(LD, 3, 0, 1, 1001); // 1: R3 = MEM[1001] = B
asm_cmd(ADD, 4, 0, 1, 1); // 2: R4 = 1
asm_cmd(ADD, 5, 0, 1, 31); // 3: R5 = 31
asm_cmd(LSF, 4, 4, 5, 0); // 4: R4 = 1 (0**31);
asm_cmd(SUB, 5, 4, 1, 1); // 5: R5 = 0 (1**31);
asm_cmd(AND, 6, 2, 4, 0); // 6: R6 = R4&R2 = A SIGN
asm_cmd(AND, 4, 3, 4, 0); // 7: R4 = R4&R3 = B SIGN
asm_cmd(AND, 2, 2, 5, 0); // 8: R2 = R5&R2 = A MAGNITUDE
asm_cmd(AND, 3, 3, 5, 0); // 9: R3 = R5&R3 = B MAGNITUDE
asm_cmd(JEQ, 0, 6, 4, 18); // 10: JUMP TO 18 IF R6 = R4 SAME SIGN
asm_cmd(JLT, 0, 2, 3, 15); // 11: JUMP TO 15 IF |A| < |B|
asm_cmd(SUB, 5, 2, 3, 0); // 12: R5 = |A| - |B|
asm_cmd(OR, 5, 5, 6, 0); // 13: R5 = A + B (WITH SIGN)
asm_cmd(JEQ, 0, 0, 0, 20); // 14: JUMP TO STORE
asm_cmd(SUB, 5, 3, 2, 0); // 15: R5 = |B| - |A|
asm_cmd(OR, 5, 5, 4, 0); // 16: R5 = A + B (WITH SIGN)
asm_cmd(JEQ, 0, 0, 0, 20); // 17: JUMP TO STORE
asm_cmd(ADD, 5, 2, 3, 0); // 18: R5 = |A| + |B|
asm_cmd(OR, 5, 5, 6, 0); // 19: R5 = A + B (WITH SIGN)
asm_cmd(ST, 0, 5, 1, 1002); // 20: STORE RESULT
asm_cmd(HLT, 0, 0, 0, 0); // 21: HALT
```

ISS simulator Testing #2

The assembly code of the problem is attached below.

We used the most time-efficient flow we could think of; given the very limited number of registers we could use.

Other files are included in the zip file.

```
asm_cmd(LD, 2, 0, 1, 1000); // 0: R2 = MEM[1000] = INPUT
asm_cmd(ADD, 3, 0, 0, 0); // 1: R3 = 0 = SQRT
asm_cmd(JEQ, 0, 0, 0, 5); // 2: JUMP TO MUL START
asm_cmd(JLT, 0, 2, 5, 11); // 3: JUMP TO END IF INPUT < MUL_RES // CHECK MUL_RES
asm_cmd(ADD, 3, 3, 1, 1); // 4: R3++
asm_cmd(ADD, 4, 0, 0, 0); // 5: R4 = 0 = COUNTER // MUL START
asm_cmd(ADD, 5, 0, 0, 0); // 6: R5 = 0 = MUL_RES
asm_cmd(JLE, 0, 3, 4, 3); // 7: IF SQRT <= COUNTER JUMP TO CHECK MUL_RES // COUNTER CONDITION
asm_cmd(ADD, 5, 5, 3, 0); // 8: R5 = R5 + SQRT
asm_cmd(ADD, 4, 4, 1, 1); // 9: R4++
asm_cmd(JEQ, 0, 0, 0, 7); // 10: JUMP TO COUNTER CONDITION // MUL END
asm_cmd(SUB, 3, 3, 1, 1); // 11: R3-- // END
asm_cmd(ST, 0, 3, 1, 1001); // 12: STORE RESULT
asm_cmd(HLT, 0, 0, 0, 0); // 13: HALT
```

ISS simulator implementation

Other files are included in the zip file.