Rony Kositsky – 205817893
Ofir Guthman – 205577018

Lab 01: ASIC Hierarchical Verification: High-Level Simulator

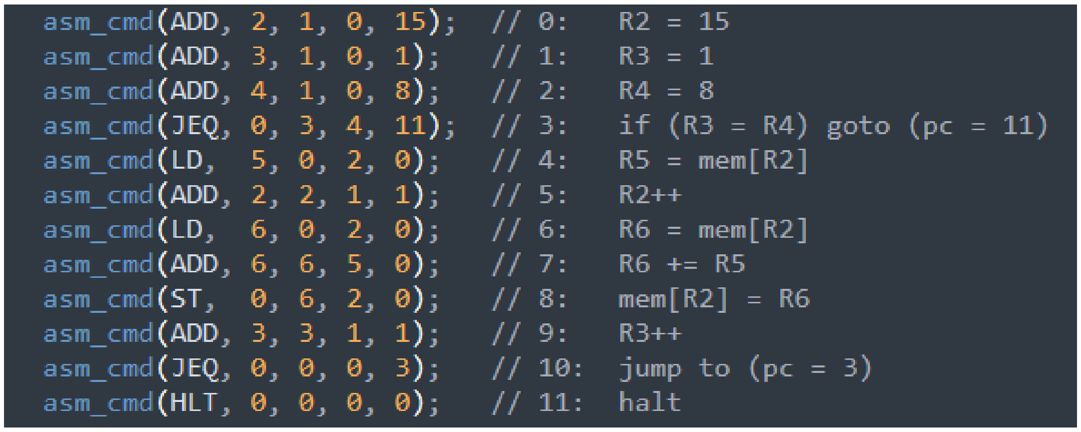
Assignments

1. **Warm-up questions:**
   1.1. Writing the command which will load the constant 10 into register 5:
   $asm\_cmd(ADD, 5, 0, 1, 10)$
   1.2. Writing the command to load the constant (-512) into register 5:
   $asm\_cmd(SUB, 5, 0, 1, 512)$
   1.3. Writing the command to calculate R3 – R4, and save it into R4:
   $asm\_cmd(SUB, 4, 3, 4, 0)$
   1.4. Writing a command that will jump to immediate only if this immediate is greater than R3:
   $asm\_cmd(JLT, 0, 3, 1, value)$
   1.5. Explaining how to load a 32 bit constant into a register:
   We need to take the constant and dissemble into 16bit MSB and LSB like the following:
   $C_{MSB} = (C \,\&\, 0xFFFF0000) \gg 16, \qquad C_{LSB} = (C \,\&\, 0x0000FFFF)$
   Now, we can use the two constants as follows (let take Reg[2] for example):
   $asm\_cmd(LHI, 2, 0, 0, C_{MSB}) \qquad // \, Reg[2][31{:}16] = C_{MSB}$
   $asm\_cmd(ADD, 2, 0, 1, C_{LSB}) \qquad // \, Reg[2] = Reg[2] + C_{LSB}$
   1.6. Explain how subroutine calls can be implemented on this processor, refer to nested subroutine call handling:
   In order to use a subroutine, we can define one register that its purpose is to save the PC from the caller (we can randomly chose, so let take Reg[7]). Now, we want that when subroutine is start, the subroutine will save the data located on Reg[7] into the memory, so if the subroutine has a nested subroutine we will not forget the return address. At the end of each subroutine, we need to restore the value from the memory to Reg[7] and then jump to the returning value.

2. **Example program:**
   2.1. What the assembly program does?
   The program calculate the sum of two follows number in the memory and store the value at the higher index. The program start the calculation from index 15 until 22. Therefore, we will get the sum of all of stored numbers (from mem[15] until mem[22]) at the memory in index 22.
   2.2. Where are the inputs stored?
   The inputs are stored at the memory from index 15 until index 22.
   2.3. Where are the outputs stored?
   The outputs are stored at the memory from index 16 until index 22.

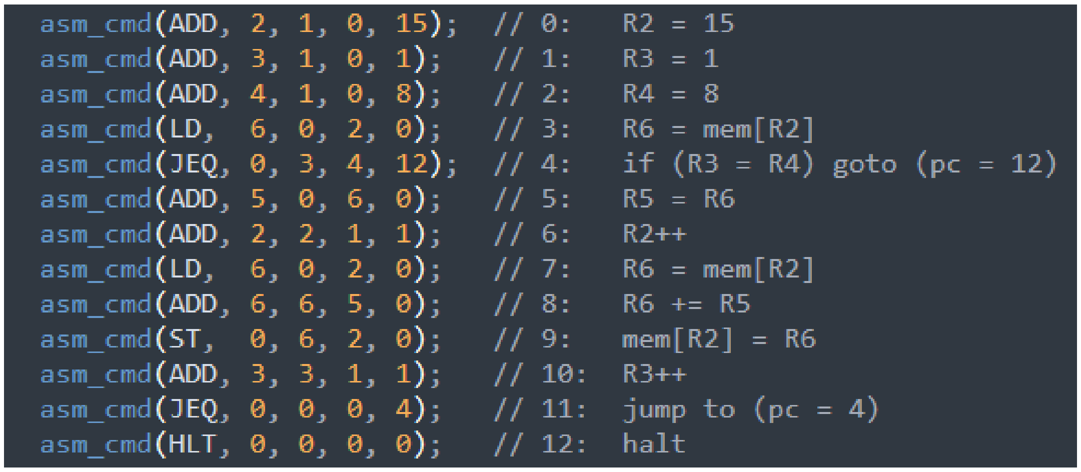
Rony Kositsky – 205817893
Ofir Guthman – 205577018

2.4. Assembly program contains pseudo code comments:



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

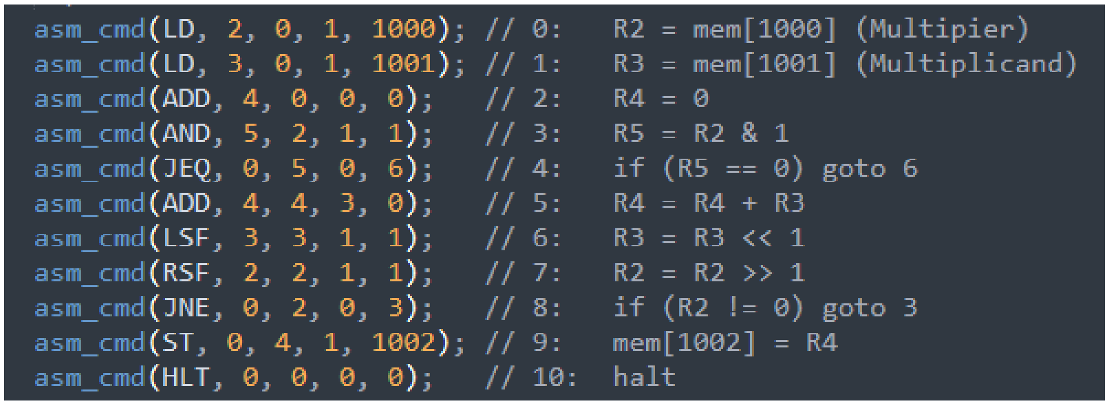
2.5. Rewriting the example program to having fewer references to memory:



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

3. ISS simulator Testing #1

3.1. Writing the assembly code for the multiplication program:

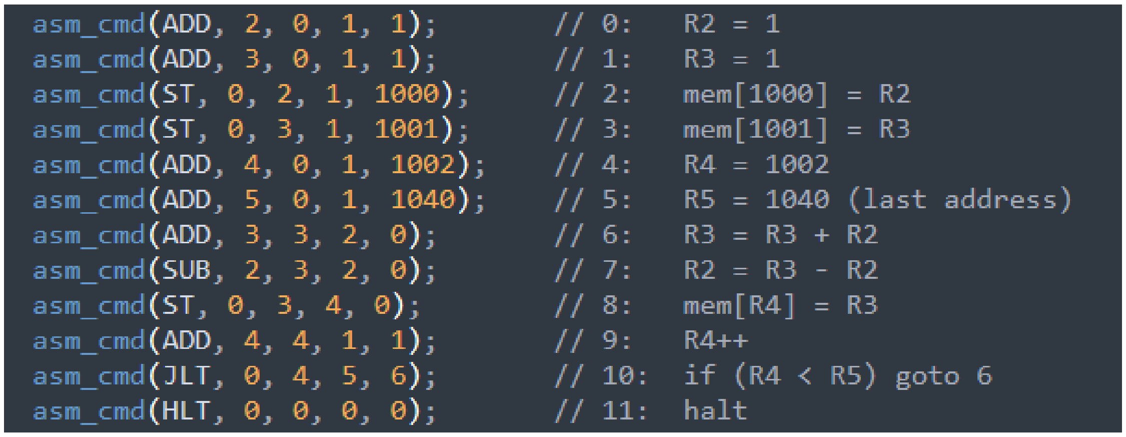


```
asm_cmd(LD, 2, 0, 1, 1000); // 0:   R2 = mem[1000] (Multipier)
asm_cmd(LD, 3, 0, 1, 1001); // 1:   R3 = mem[1001] (Multiplicand)
asm_cmd(ADD, 4, 0, 0, 0);   // 2:   R4 = 0
asm_cmd(AND, 5, 2, 1, 1);   // 3:   R5 = R2 & 1
asm_cmd(JEQ, 0, 5, 0, 6);   // 4:   if (R5 == 0) goto 6
asm_cmd(ADD, 4, 4, 3, 0);   // 5:   R4 = R4 + R3
asm_cmd(LSF, 3, 3, 1, 1);   // 6:   R3 = R3 << 1
asm_cmd(RSF, 2, 2, 1, 1);   // 7:   R2 = R2 >> 1
asm_cmd(JNE, 0, 2, 0, 3);   // 8:   if (R2 != 0) goto 3
asm_cmd(ST, 0, 4, 1, 1002); // 9:   mem[1002] = R4
asm_cmd(HLT, 0, 0, 0, 0);   // 10:  halt
```

Rony Kositsky – 205817893
Ofir Guthman – 205577018

4. ISS simulator Testing #2

4.1. Writing the assembly code for the Fibonacci sequence program:



```
asm_cmd(ADD, 2, 0, 1, 1);        // 0:   R2 = 1
asm_cmd(ADD, 3, 0, 1, 1);        // 1:   R3 = 1
asm_cmd(ST, 0, 2, 1, 1000);      // 2:   mem[1000] = R2
asm_cmd(ST, 0, 3, 1, 1001);      // 3:   mem[1001] = R3
asm_cmd(ADD, 4, 0, 1, 1002);     // 4:   R4 = 1002
asm_cmd(ADD, 5, 0, 1, 1040);     // 5:   R5 = 1040 (last address)
asm_cmd(ADD, 3, 3, 2, 0);        // 6:   R3 = R3 + R2
asm_cmd(SUB, 2, 3, 2, 0);        // 7:   R2 = R3 - R2
asm_cmd(ST, 0, 3, 4, 0);         // 8:   mem[R4] = R3
asm_cmd(ADD, 4, 4, 1, 1);        // 9:   R4++
asm_cmd(JLT, 0, 4, 5, 6);        // 10:  if (R4 < R5) goto 6
asm_cmd(HLT, 0, 0, 0, 0);        // 11:  halt
```

4.3. What would happen if we ran this program for the first 100 elements?
As we can see from the sram_out file, the last calculate number is 0x06197ecb (102334155). Therefore, after add few additional elements (above 40) we will get an overflow on the result register.