Rony Kositsky – 205817893
Ofir Guthman – 205577018

Lab 01: ASIC Hierarchical Verification: Low-Level Simulator

Assignments

Question 1

1. $r[2] = 12,\ r[3] = 45,\ r[4] = 57,\ r[5] = -12$
2. We try to use the new value, which $r[3]$ received. We should use the old value; therefore, this is illegal scenario.

Question 2

1. 7 clock cycles for the first instruction to step the IDLE state, and for each another state we will get 6 instructions. In total, $6n + 1$.
2. Each state has its own unique design purpose. Therefore, accessing memory is possible only at EXEC0 state.
3. Advantage – Simple, so it is easy to implement.
   Disadvantage – More clock cycles compare to pipeline.

Question 6

We added 2 new opcodes, CPY (25) which preforming the background copy, and ASK (26) which used by the program to poll the copy status.
The main function who does the process called DMA function, which copies at the background ("in parallel") the lines while the memory not in use by the simulator (by the global int mem_availablity). The function gets the global int mem_availablity, which is equal to 1 if we know that the memory is available for the next 2 clock cycles, and 0 otherwise. The DMA also contain 4 states which he can be in only one state at a given time (IDLE/WAIT/READ/WRITE).

Rony Kositsky – 205817893
Ofir Guthman – 205577018

Question 7

Assembly code:

```
asm_cmd(ADD, 3, 1, 0, 50);      // 0: R3 = 50 (src)
asm_cmd(ADD, 4, 1, 0, 60);      // 1: R4 = 60 (dst)
asm_cmd(ADD, 5, 1, 0, 50);      // 2: R5 = 50 (copy length)

// execute dma copy command
asm_cmd(CPY, 4, 3, 5, 0);       // 3: CPY length of R[5] values from R[3] (src) to R[4] (dst)

// run memory operation while copy
asm_cmd(ADD, 2, 1, 0, 50);      // 4: R2 = 50
asm_cmd(ADD, 5, 1, 0, 50);      // 5: R5 = 50 (length)
asm_cmd(ADD, 3, 0, 0, 0);       // 6: R3 = 0
asm_cmd(LD,  4, 0, 2, 0);       // 7: R4 = MEM[R2]
asm_cmd(ADD, 3, 3, 4, 0);       // 8: R3+= R4
asm_cmd(ADD, 2, 2, 1, 1);       // 9: R2++
asm_cmd(JLT, 0, 2, 5, 7);       // 10: if R2 < R5 jump to line 7

// check if DMA done copy, if not wait
asm_cmd(ASK, 2, 0, 0, 0);       // 11: R[2] = #copy_transactions_remaining
asm_cmd(JNE, 2, 0, 0, 11);      // 12: if (R[2] != 0) jump to 11

// check if the copy succeed
asm_cmd(ADD, 2, 1, 0, 1);       // 13: R2 = 1 (copy passed)
asm_cmd(ADD, 3, 1, 0, 50);      // 14: R3 = 50
asm_cmd(ADD, 4, 1, 0, 60);      // 15: R4 = 60
asm_cmd(ADD, 5, 1, 0, 100);     // 16: R5 = 100 (length + R3)
asm_cmd(LD, 6, 0, 3, 0);        // 17: R6 = MEM[R3]
asm_cmd(LD, 7, 0, 4, 0);        // 18: R7 = MEM[R4]
asm_cmd(JEQ, 0, 6, 7, 22);      // 19: if (R[6] == R[7]) goto 22
asm_cmd(ADD, 2, 0, 0, 0);       // 20: R2 = 0 (copy failed)
asm_cmd(JEQ, 0, 0, 0, 25);      // 21: goto 25 (halt)
asm_cmd(ADD, 3, 3, 1, 1);       // 22: R3++
asm_cmd(ADD, 4, 4, 1, 1);       // 23: R4++
asm_cmd(JLT, 0, 3, 5, 16);      // 24: if R3 < R5 jump to line 16
asm_cmd(HLT, 0, 0, 0, 0);       // 25: HALT
```