Onn Rengingad 304845951
Bar Ben Ari 204034284
Barak Levy 311431894

Lab 2 - Advanced Computer Architecture

1. Understanding registers simulation (Question 1):

1. The values: r[2] = 12, r[3] = 45, r[4] = 9, r[5] = -12.

2. The code is wrong because the "old" value (spro->r[1]) should never be written and the "new" value (sprn->r[3]) should never be read from.

2. Understanding micro-architecture (Question 2):

1. All instructions takes six clock cycles except for the first one that takes seven clock cycles (move from state 0 to state 1). so it would take 6n + 1 clock cycles to execute n instructions.
2. A pipelined microarchitecture that can support memory access every clock cycle.
3. Advantages:
    a. No structure or data hazards.
    b. Easy to implement (simple).

    Disadvantages:
    a. Throughput is lower, cannot run instructions in parallel.

3. Low level simulator + testing (Questions 3-5):

1. The files attached within the folder.

4. Background DMA copy (Question 6):

We created a DMA state machine capable of copying a block of memory in the background while the main operation regime continues execution. The DMA was implemented using a multithreaded protocol in C, to be able to process both, the main operation and memory copy operation, in parallel. The DMA state machine is comprised of the following states:

1) DMA_IDLE - this state informs that the DMA is silent, before being ordered to perform the first memory block copy (If present). When a COPY instruction is received the DMA transfers to DMA_WAIT state.
2) DMA_WAIT - this state informs that the DMA is waiting to resume its memory-copy operation that was given earlier, or before further COPY instructions are received. In this state, further COPY operations will be ignored, if a COPY operation is already in-progress. The DMA transfers to DMA_START when its thread is prioritized.
3) DMA_START - this state informs that the operating system has prioritized the DMA thread over the main thread, and during this state the DMA performs the copy of the memory block until it is stopped again by the operating system or until it is done copying. Then the DMA returns to DMA_WAIT.

On HLT instruction, the DMA transfers to DMA_IDLE.

Lastly, the instructions we programmed to utilize the DMA operation are:

1) COPY - this instruction receives a source address (src0 reg), a destination address(dest reg) and the number of memory cells in the block we'd like to copy(src1 reg), and it initiates the memory block copy operation.
2) POLL - this instruction receives a destination address (dst reg) to which the DMA inserts the number of cells remaining to be copied.

Sp.c contains the DMA functionality.

Onn Rengingad 304845951
Bar Ben Ari 204034284
Barak Levy 311431894

5. **DMA testing (Question 7):**

1. The DMA testing files attached within the folder.
2. The assembly code that validates the DMA operation is as follows:

```
//Setting registers for COPY operation
asm_cmd(ADD, 4, 1, 0, 30);   // 0: R4 = 30 -COPY source address
asm_cmd(ADD, 5, 1, 0, 1000); // 1: R5 = 1000 -COPY destination address
asm_cmd(ADD, 6, 1, 0, 1000); // 2: R6 = 1000 -COPY operation number of cells taken

//Initiate copy regime in a different thread
asm_cmd(COPY, 5, 4, 6, 0);   // 3: COPY - DMA copy operation is initiated:
// It will copy 1000 cells from address


//Begin an Accumulative-negative-sum computation in the main thread
//Variables initiation
asm_cmd(ADD, 2, 1, 0, 50);   // 4: R2 = 50
asm_cmd(ADD, 3, 1, 0, 200);  // 5: R3 = 200
asm_cmd(ADD, 4, 0, 0, 10000);// 6: R4 = 0

//Load values and compute
asm_cmd(LD, 5, 0, 2, 0);     // 7: R5 = Mem[R2]
asm_cmd(SUB, 4, 4, 5, 0);    // 8: R4 -= R5
asm_cmd(ADD, 2, 2, 1, 1);    // 9: R2++
asm_cmd(JLT, 0, 2, 3, 7);    // 10: if R2<R3 jump to line 7


//Polling the DMA copy operation after the main Accumulative-negative-sum computation operation had finished
asm_cmd(POLL, 7, 0, 0, 0);   // 11: POLL - R[7] recieves the number of remaining cells to be copied by the DMA
asm_cmd(JNE, 7, 0, 0, 18);   // 12: if R[7]!=0 return to line 17 until COPY operation is done

//Following the termination of the DMA and Sum calculation operations, we compare the copied block of addresses
//Setting registers for comparison operation
asm_cmd(ADD, 4, 1, 0, 30);   // 13: R4 = 30 -COPY source address
asm_cmd(ADD, 5, 1, 0, 1000); // 14: R5 = 1000 -COPY destination address
asm_cmd(ADD, 6, 1, 0, 1000); // 15: R6 = 1000 -COPY operation total number of cells

asm_cmd(LD, 2, 0, 4, 0);     // 16: R2 = Mem[R4]
asm_cmd(LD, 3, 0, 5, 0);     // 17: R3 = Mem[R5]
asm_cmd(JNE, 0, 2, 3, 25);   // 18: if R2!=R3 (meanining Mem[R4]!=Mem[R5]) than DMA copy operation failed! jumping to pc=25
asm_cmd(ADD, 4, 4, 1, 1);    // 19: R4++ (Advancing source address)
asm_cmd(ADD, 5, 5, 1, 1);    // 20: R5++ (Advancing destination address)
asm_cmd(SUB, 6, 6, 1, 1);    // 21: R6-- (Decresing the number of cells remanining to validate)
asm_cmd(JNE, 0, 6, 0, 16);   // 22: if R6!=0 than there are still copied cell that need to be validated. jumping to pc=16 to proceed validating

asm_cmd(ADD, 2, 0, 1, 1);    // 23: R2=1 -> DMA Copy & Fib operation were successful
asm_cmd(JIN, 0, 1, 0, 26);   // 24: Jumping to HALT (Indirect jump to the end)
asm_cmd(ADD, 2, 0, 0, 0);    // 25: R2=0 -> DMA Copy Failed
asm_cmd(HLT, 0, 0, 0, 0);    // 26: HALT....
```