Itai shchorry 305584690
Oded Navon 303066674

Architecture Lab – Lab 1

Question 1

1. LD R2 R1 R0 5
2. LD R2 R1 R0 -5
3. SUB R2 R3 R2 0
4. We'll notice we can perform all the different kinds of inequations using given operations:
    a. $b > a \leftrightarrow a < b \leftrightarrow JLT\ R0\ a\ b\ imm$
    b. $b \geq a \leftrightarrow a \leq b \leftrightarrow JLE\ R0\ a\ b\ imm$

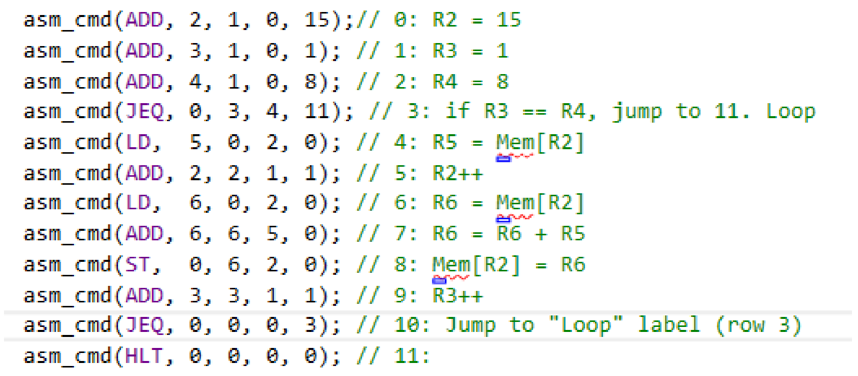
    All other operations are implemented directly.
5. we'll perform the load within 3 steps:
    a. use LHI command to load first 16 required bits into the 16 MSB's of the register. These bits will end up being the 16 LSB's of the number, so we will load the relevant bits/
    b. use RSF command to shift the 16 bits inserted into the register into the 16 lower bits. The immediate given for the shift operation will be 16.
    c. Use LHI command again to load the 16 MSB's of the constant.
6. We can implement subroutine calls on this processor in the following way – for each subroutine we want to perform from a given program:
    a. Use a flow control command in the given program to jump to relevant sub-routine's PC. If subroutine is unconditioned, we'll use the JIN operation, loading the sub-routine's address to some register for this matter. **The address from which the jump was made will be saved to R7.**
    b. Perform the subroutine
    c. At the end of the subroutine, use the unconditioned JIN operation with R7 to return to the program we jumped from

    In case we want to use a nested call – since we know another jump will overwrite our original program's address, which currently resides inside R7, we are obliged to save it to some other free register (denote as Ra). So, the handling of the inner sub-routine will be – save R7 value to some other register, jump and perform inner subroutine, return from it using R7, and at the end of the outer subroutine, use Ra to jump back to original program.
    if the depth of the call-stack is deeper, perform the described method recursively.

Question 2

1. Given a program name, it translates a specific program into machine code. The outcome of this program is that for $i \in [15,22]$, Mem[i] represents $\Sigma_{j=15}^{i} Mem[j]$
2. The input to the program is stored in the memory, at addresses 15-22 (including). Another input to the program is the address of the output file.
3. The output is printed to the file located at the address given at the start of the program execution.
4. The program with comments:

Itai shchorry 305584690
Oded Navon 303066674

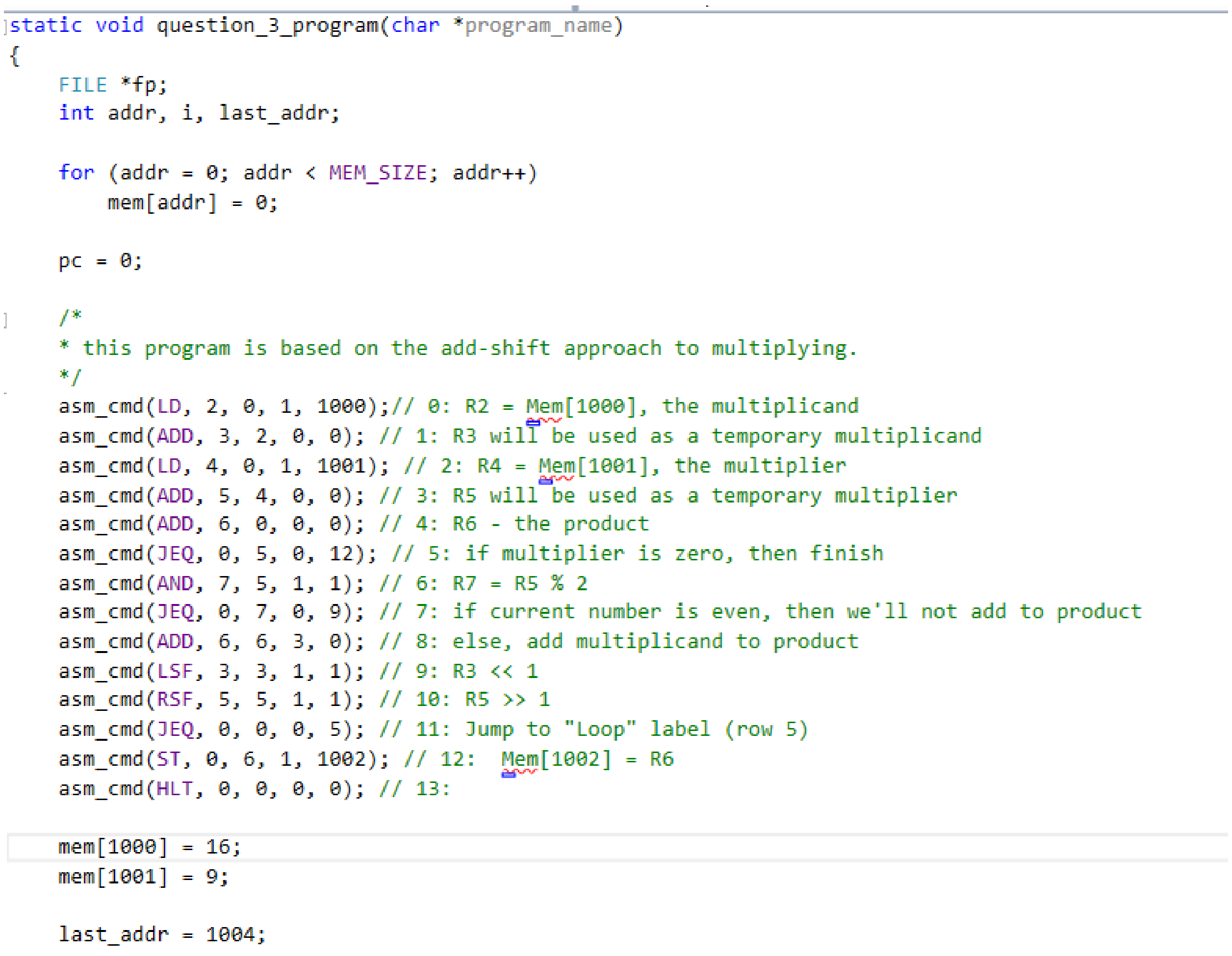


```
asm_cmd(ADD, 2, 1, 0, 15);// 0: R2 = 15
asm_cmd(ADD, 3, 1, 0, 1); // 1: R3 = 1
asm_cmd(ADD, 4, 1, 0, 8); // 2: R4 = 8
asm_cmd(JEQ, 0, 3, 4, 11); // 3: if R3 == R4, jump to 11. Loop
asm_cmd(LD,  5, 0, 2, 0); // 4: R5 = Mem[R2]
asm_cmd(ADD, 2, 2, 1, 1); // 5: R2++
asm_cmd(LD,  6, 0, 2, 0); // 6: R6 = Mem[R2]
asm_cmd(ADD, 6, 6, 5, 0); // 7: R6 = 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 "Loop" label (row 3)
asm_cmd(HLT, 0, 0, 0, 0); // 11:
```

5. One way to cut down significantly the number of memory accesses is to use R7 as a mediator between row 8 and row 5 – add another row (i.e between 8 and 9) which saves the value of R6 to R7 (ADD R7 R6 R0 0), and then at row 4 we can just feed R7's value to R5, instead of loading it from memory (ADD R5 R7 R0 0). This means we make 1 load per each location in memory that we're using, which is optimal from memory accesses point of view.

Question 3

The code for creating the assembly:



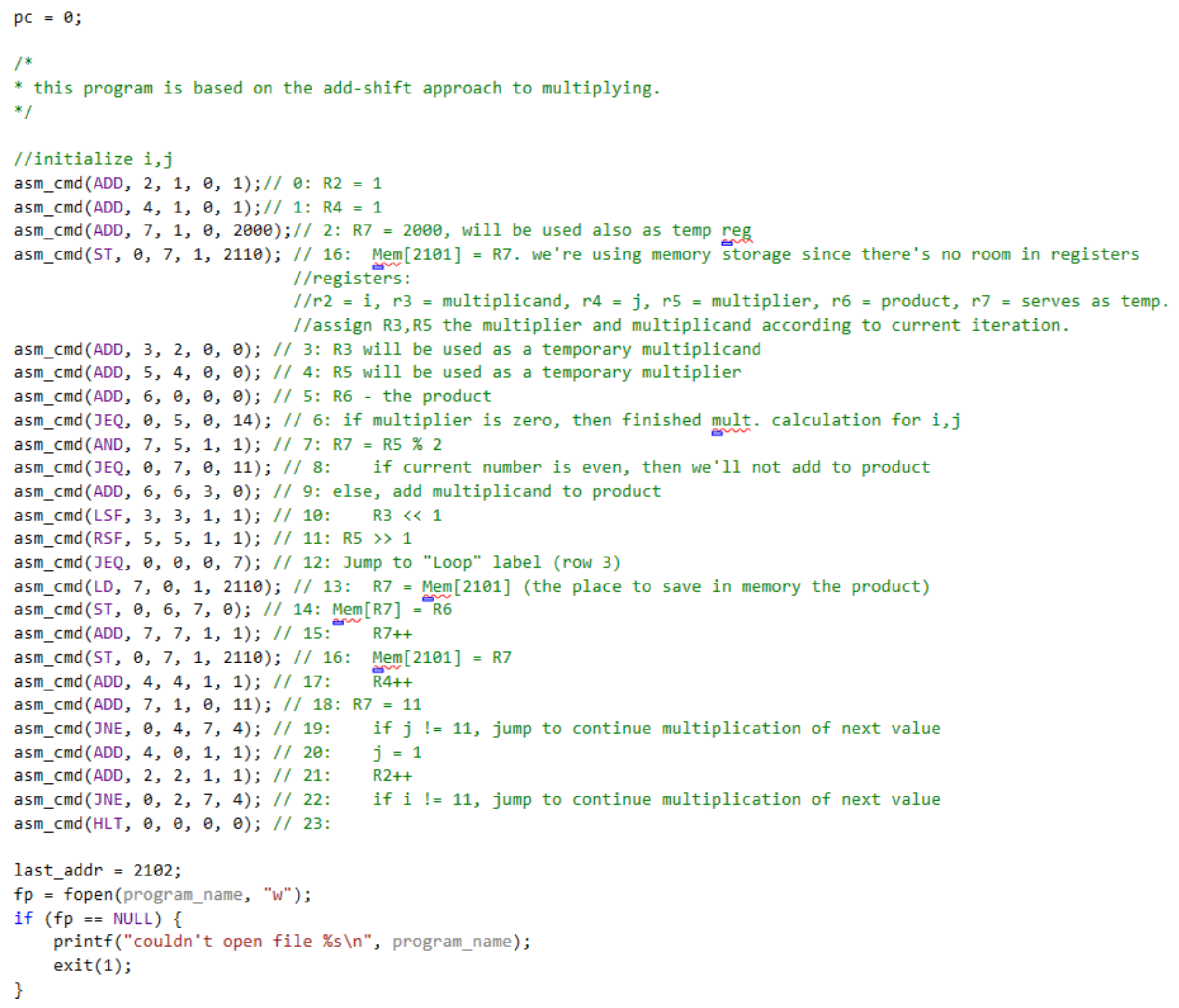
```
static void question_3_program(char *program_name)
{
    FILE *fp;
    int addr, i, last_addr;

    for (addr = 0; addr < MEM_SIZE; addr++)
        mem[addr] = 0;

    pc = 0;

    /*
    * this program is based on the add-shift approach to multiplying.
    */
    asm_cmd(LD, 2, 0, 1, 1000);// 0: R2 = Mem[1000], the multiplicand
    asm_cmd(ADD, 3, 2, 0, 0); // 1: R3 will be used as a temporary multiplicand
    asm_cmd(LD, 4, 0, 1, 1001); // 2: R4 = Mem[1001], the multiplier
    asm_cmd(ADD, 5, 4, 0, 0); // 3: R5 will be used as a temporary multiplier
    asm_cmd(ADD, 6, 0, 0, 0); // 4: R6 - the product
    asm_cmd(JEQ, 0, 5, 0, 12); // 5: if multiplier is zero, then finish
    asm_cmd(AND, 7, 5, 1, 1); // 6: R7 = R5 % 2
    asm_cmd(JEQ, 0, 7, 0, 9); // 7: if current number is even, then we'll not add to product
    asm_cmd(ADD, 6, 6, 3, 0); // 8: else, add multiplicand to product
    asm_cmd(LSF, 3, 3, 1, 1); // 9: R3 << 1
    asm_cmd(RSF, 5, 5, 1, 1); // 10: R5 >> 1
    asm_cmd(JEQ, 0, 0, 0, 5); // 11: Jump to "Loop" label (row 5)
    asm_cmd(ST, 0, 6, 1, 1002); // 12:  Mem[1002] = R6
    asm_cmd(HLT, 0, 0, 0, 0); // 13:

    mem[1000] = 16;
    mem[1001] = 9;

    last_addr = 1004;
```

Itai shchorry 305584690
Oded Navon 303066674

Trace and sram_out files are attached.

<u>Question 4</u>

The code for creating the assembly:



```
pc = 0;

/*
* this program is based on the add-shift approach to multiplying.
*/

//initialize i,j
asm_cmd(ADD, 2, 1, 0, 1);// 0: R2 = 1
asm_cmd(ADD, 4, 1, 0, 1);// 1: R4 = 1
asm_cmd(ADD, 7, 1, 0, 2000);// 2: R7 = 2000, will be used also as temp reg
asm_cmd(ST, 0, 7, 1, 2110); // 16:  Mem[2101] = R7. we're using memory storage since there's no room in registers
                            //registers:
                            //r2 = i, r3 = multiplicand, r4 = j, r5 = multiplier, r6 = product, r7 = serves as temp.
                            //assign R3,R5 the multiplier and multiplicand according to current iteration.
asm_cmd(ADD, 3, 2, 0, 0); // 3: R3 will be used as a temporary multiplicand
asm_cmd(ADD, 5, 4, 0, 0); // 4: R5 will be used as a temporary multiplier
asm_cmd(ADD, 6, 0, 0, 0); // 5: R6 - the product
asm_cmd(JEQ, 0, 5, 0, 14); // 6: if multiplier is zero, then finished mult. calculation for i,j
asm_cmd(AND, 7, 5, 1, 1); // 7: R7 = R5 % 2
asm_cmd(JEQ, 0, 7, 0, 11); // 8:    if current number is even, then we'll not add to product
asm_cmd(ADD, 6, 6, 3, 0); // 9: else, add multiplicand to product
asm_cmd(LSF, 3, 3, 1, 1); // 10:    R3 << 1
asm_cmd(RSF, 5, 5, 1, 1); // 11: R5 >> 1
asm_cmd(JEQ, 0, 0, 0, 7); // 12: Jump to "Loop" label (row 3)
asm_cmd(LD, 7, 0, 1, 2110); // 13:  R7 = Mem[2101] (the place to save in memory the product)
asm_cmd(ST, 0, 6, 7, 0); // 14: Mem[R7] = R6
asm_cmd(ADD, 7, 7, 1, 1); // 15:    R7++
asm_cmd(ST, 0, 7, 1, 2110); // 16:  Mem[2101] = R7
asm_cmd(ADD, 4, 4, 1, 1); // 17:    R4++
asm_cmd(ADD, 7, 1, 0, 11); // 18: R7 = 11
asm_cmd(JNE, 0, 4, 7, 4); // 19:    if j != 11, jump to continue multiplication of next value
asm_cmd(ADD, 4, 0, 1, 1); // 20:    j = 1
asm_cmd(ADD, 2, 2, 1, 1); // 21:    R2++
asm_cmd(JNE, 0, 2, 7, 4); // 22:    if i != 11, jump to continue multiplication of next value
asm_cmd(HLT, 0, 0, 0, 0); // 23:

last_addr = 2102;
fp = fopen(program_name, "w");
if (fp == NULL) {
    printf("couldn't open file %s\n", program_name);
    exit(1);
}
```

Trace and sram_out files are attached.

<u>Question 5</u>

Attached code