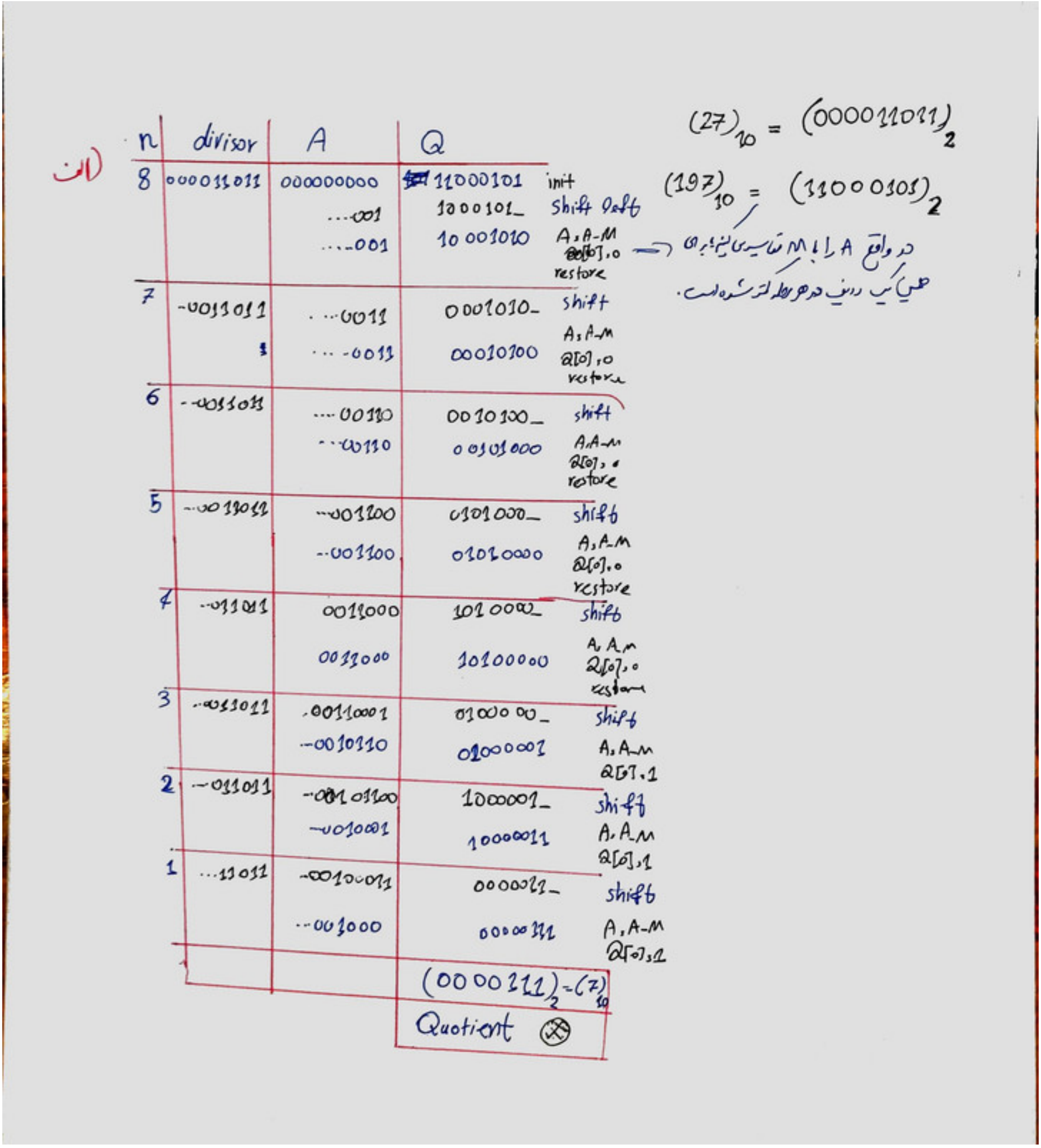
Computer Architecture, HW2

Name	Student number
Kourosh Alinaghi	810101476
Arshia Motaghian	810101503

Part A



Part B

C code

We used insertion sort in this code. It basically find the `i` th minimum everytime and puts it in `arr[i]`. Here's the code:

```
int main() {
    int arr[20];

    for (int i = 0; i < 20; i++) {
        for (int j = i + 1; j < 20; j++) {
            if (arr[i] > arr[j]) {
                int tmp = arr[j];
                arr[j] = arr[i];
                arr[i] = tmp;
            }
        }
    }

    return 0;
}
```

Variables linking table

This table shows in which register each variable is stored:

Variable value	Register
`&a[0]`	`s0`
`i`	`s1`
`j`	`s2`
`&a[i]`	`t1`
`&a[j]`	`t2`
`20`	`t3`
`a[i]`	`t4`
`a[j]`	`t5`

RISC-V assembly code

```
addi t3, zero, 20
```

```
addi s1, zero, 0
addi t1, s0, 0

L1:
    bge s1, t3, END_L1

    addi s2, s1, 1

    addi t2, t1, 4
    jal zero, L2

    L2:
        bge s2, t3, L1Add

        lw t4, 0(t1)
        lw t5, 0(t2)
        bge t5, t4, L2Add

        sw t4, 0(t2)
        sw t5, 0(t1)

    L2Add:
        addi s2, s2, 1
        addi t2, t2, 4
        jal zero, L2

    L1Add:
        addi s1, s1, 1
        addi t1, t1, 4
        jal zero, L1

END_L1:
```

Part C (Bonus)

In this part we need to initalize our array. We used this code to do so. At the end of the execution of this code, our array would be `[20, 19, 18, ..., 2, 1]`.

```
addi sp, sp, -120
addi s0, sp, 0

addi t1, zero, 20
sw t1, 0(s0)
addi t1, zero, 19
sw t1, 4(s0)
addi t1, zero, 18
sw t1, 8(s0)
addi t1, zero, 17
sw t1, 12(s0)
addi t1, zero, 16
sw t1, 16(s0)
```

```
addi t1, zero, 15
sw t1, 20(s0)
addi t1, zero, 14
sw t1, 24(s0)
addi t1, zero, 13
sw t1, 28(s0)
addi t1, zero, 12
sw t1, 32(s0)
addi t1, zero, 11
sw t1, 36(s0)
addi t1, zero, 10
sw t1, 40(s0)
addi t1, zero, 9
sw t1, 44(s0)
addi t1, zero, 8
sw t1, 48(s0)
addi t1, zero, 7
sw t1, 52(s0)
addi t1, zero, 6
sw t1, 56(s0)
addi t1, zero, 5
sw t1, 60(s0)
addi t1, zero, 4
sw t1, 64(s0)
addi t1, zero, 3
sw t1, 68(s0)
addi t1, zero, 2
sw t1, 72(s0)
addi t1, zero, 1
sw t1, 76(s0)
addi t1, zero, 0
sw t1, 80(s0)
```

Memory before sorting the array:

HEX DEC BINARY

0x000005e0:	000 000 000 000
0x000005dc:	000 000 000 000
0x000005d8:	000 000 000 000
0x000005d4:	000 000 000 001
0x000005d0:	000 000 000 002
0x000005cc:	000 000 000 003
0x000005c8:	000 000 000 004
0x000005c4:	000 000 000 005
0x000005c0:	000 000 000 006
0x000005bc:	000 000 000 007
0x000005b8:	000 000 000 008
0x000005b4:	000 000 000 009
0x000005b0:	000 000 000 010
0x000005ac:	000 000 000 011
0x000005a8:	000 000 000 012
0x000005a4:	000 000 000 013
0x000005a0:	000 000 000 014
0x0000059c:	000 000 000 015
0x00000598:	000 000 000 016
0x00000594:	000 000 000 017
0x00000590:	000 000 000 018
0x0000058c:	000 000 000 019
0x00000588:	000 000 000 020
	FREE SPACE
0x00000584:	000 000 000 000
0x00000580:	000 000 000 000

Memory after sorting the array:

HEX DEC BINARY

0x000005e0:	000 000 000 000
0x000005dc:	000 000 000 000
0x000005d8:	000 000 000 000
0x000005d4:	000 000 000 020
0x000005d0:	000 000 000 019
0x000005cc:	000 000 000 018
0x000005c8:	000 000 000 017
0x000005c4:	000 000 000 016
0x000005c0:	000 000 000 015
0x000005bc:	000 000 000 014
0x000005b8:	000 000 000 013
0x000005b4:	000 000 000 012
0x000005b0:	000 000 000 011
0x000005ac:	000 000 000 010
0x000005a8:	000 000 000 009
0x000005a4:	000 000 000 008
0x000005a0:	000 000 000 007
0x0000059c:	000 000 000 006
0x00000598:	000 000 000 005
0x00000594:	000 000 000 004
0x00000590:	000 000 000 003
0x0000058c:	000 000 000 002
0x00000588:	000 000 000 001
	FREE SPACE
0x00000584:	000 000 000 000
0x00000580:	000 000 000 000

As you can see, the array is sort in ascending order. (It's from down to up)

Final code

This is the final code that is tested on RSIC-V Simulator:

```
main:
    addi sp, sp, -120
    addi s0, sp, 0

    addi t1, zero, 20
    sw t1, 0(s0)
    addi t1, zero, 19
    sw t1, 4(s0)
    addi t1, zero, 18
    sw t1, 8(s0)
    addi t1, zero, 17
    sw t1, 12(s0)
    addi t1, zero, 16
    sw t1, 16(s0)
    addi t1, zero, 15
    sw t1, 20(s0)
    addi t1, zero, 14
    sw t1, 24(s0)
    addi t1, zero, 13
    sw t1, 28(s0)
    addi t1, zero, 12
    sw t1, 32(s0)
    addi t1, zero, 11
    sw t1, 36(s0)
    addi t1, zero, 10
    sw t1, 40(s0)
    addi t1, zero, 9
    sw t1, 44(s0)
    addi t1, zero, 8
    sw t1, 48(s0)
    addi t1, zero, 7
    sw t1, 52(s0)
    addi t1, zero, 6
    sw t1, 56(s0)
    addi t1, zero, 5
    sw t1, 60(s0)
    addi t1, zero, 4
    sw t1, 64(s0)
    addi t1, zero, 3
    sw t1, 68(s0)
    addi t1, zero, 2
    sw t1, 72(s0)
    addi t1, zero, 1
    sw t1, 76(s0)

    addi t3, zero, 20

    addi s1, zero, 0
    addi t1, s0, 0
```

```
L1:
    bge s1, t3, END_L1

    addi s2, s1, 1

    addi t2, t1, 4
    jal zero, L2

    L2:
        bge s2, t3, L1Add

        lw t4, 0(t1)
        lw t5, 0(t2)
        bge t5, t4, L2Add

        sw t4, 0(t2)
        sw t5, 0(t1)

    L2Add:
        addi s2, s2, 1
        addi t2, t2, 4
        jal zero, L2

    L1Add:
        addi s1, s1, 1
        addi t1, t1, 4
        jal zero, L1

END_L1:
```