# CSE3666 HW2 solutions

## 1. Array.

### a)

We only need to add one instruction to the code in the slide. See the red instruction in part b).

There are 8 instructions in the loop. The loop is executed for 100 times. There are two instructions before the loop. So the total number of executed instructions is 802.

### b)

```
for (i = 0; i < 100; i += 2) {
      B[i] = A[i] + 4;
      B[i+1] = A[i+1] + 4;
}

      addi  s4, x0, 100
      addi  s1, x0, 0   # we are sure s1 < s4 after initialization
loop:
      slli  t0, s1, 2   # t0 = i * 4
      add   t2, t0, s2  # compute addr of A[i]
      add   t3, t0, s3  # compute addr of B[i]

      lw    t1, 0(t2)   # A[i]
      addi  t1, t1, 4   # the new instruction for +4
      sw    t1, 0(t3)

      lw    t4, 4(t2)   # A[i+1]
      addi  t4, t4, 4
      sw    t4, 4(t3)

      lw    t5, 8(t2)   # A[i+2]
      addi  t5, t5, 4
      sw    t5, 8(t3)

      lw    t6, 12(t2)  # A[i+3]
      addi  t6, t6, 4
      sw    t6, 12(t3)

      addi  s1, s1, 4
      blt   s1, s4, loop # 11 instructions in the loop
```

Each iteration has 17 instructions. The total number of instructions executed is 2 + 17 * 25 = 427.

## 2. Two-dimensional array.

An example implementation is shown below. Pay attention to the nested loops and the calculation of T[i][j]'s address.

| Register | s9 | t0 | t1 | s0 | s1 |
|---|---|---|---|---|---|
| Variable/value | address of T | i | j | 16 | 8 |

```
for (i = 0; i < 16; i += 1)
    for (j = 0; j < 8; j += 1)
        T[i][j] = 256 * i + j;
```

```
# We are sure the initial value of i and j are less than 16 and 8, respectively
# So we do not jump to condition test before the loops
# Instructions that depend only on i can be placed outside of the inner loop
# Here, we just convert the loops from C to RISC-V literally.

        addi   s0, x0, 16
        addi   s1, x0, 8

        addi   t0, x0, 0           # i = 0

        # beq x0, x0 CONDI         # this is not needed here
LOOPI:
        addi  t1, x0, 0            # j = 0

LOOPJ:
        slli  t5, t0, 8            # t5 = 256 * i
        add   t6, t5, t1           # t6 = 256 * i + j

        slli  t2, t0, 3            # 8 * i. Number of words before T[i][0]
        addi  t3, t2, t1           # 8 * i + j. Number of words before T[i][j]
        slli  t3, t3, 2            # offset of T[i][j]
        add   t3, t3, s9           # address of T[i][j]
        sw    t6, 0(t3)

        addi  t1, t1, 1            # j = j + 1
        blt   t1, s1, LOOPJ        # continue the inner loop if j < 8

        addi  t0, t0, 1            # i = i + 1
        blt   t0, s0, LOOPI        # continue the outer loop if i < 16
```

## 3. Encoding.

Steps: Find bits in each field, and then write the machine code.

```
==========
013964B3 or s1, s2, s3 (or x9,x18,x19)
00000001001110010110010010110011
R-type
opcode:   0110011
rd:       01001
funct3:   110
rs1:      10010
rs2:      10011
funct7:   0000000
==========
01039313 slli t1, t2, 16 (slli x6,x7,0x10)
00000001000000111001001100010011
I-type shift
shamt:    10000
imm_i:    000000010000
opcode:   0010011
rd:       00110
funct3:   001
rs1:      00111
rs2:      10000
funct7:   0000000
==========
FFF0C093 xori x1, x1, -1 (xori x1,x1,0xffffffff)
11111111111100001100000010010011
I-type
imm_i:    111111111111
opcode:   0010011
rd:       00001
funct3:   100
rs1:      00001
rs2:      11111
funct7:   1111111
==========
F9C1A103 lw x2, -100(x3) (lw x2,0xffffff9c(x3))
11111001110000011010000100000011
I-type
imm_i:    111110011100
opcode:   0000011
rd:       00010
funct3:   010
rs1:      00011
rs2:      11100
funct7:   1111100
```

## 4. Decoding.

Steps: Find the bits in each field. Check opcode, identify the instruction format, find out the immediate if necessary, and then write the instruction.

```
==========
FEACA823 sw x10,-16(x25)
11111110101011001010100000100011
opcode:   0100011
rd:       10000
funct3:   010
rs1:      11001
rs2:      01010
funct7:   1111111
S-type
imm_s:    111111110000
==========
04020713 addi x14,x4,64
00000100000000100000011100010011
opcode:   0010011
rd:       01110
funct3:   000
rs1:      00100
rs2:      00000
funct7:   0000010
I-type
imm_i:    000001000000
==========
00557BB3 and x23,x10,x5
00000000010101010111101110110011
opcode:   0110011
rd:       10111
funct3:   111
rs1:      01010
rs2:      00101
funct7:   0000000
R-type
==========
414FDF13 srai x30,x31,20
01000001010011111101111100010011
opcode:   0010011
rd:       11110
funct3:   101
rs1:      11111
rs2:      10100
funct7:   0100000
I-type shift
imm_i:    010000010100
shamt:    10100
```