1. **Why** does the MIC-1' microinstruction word have only **4 bits** to control which register is connected to the **B bus**?

2. The opcode for the IJVM "SWAP" instruction is hex 5F. **What address** is the "**swap1**" microinstruction stored at in the MIC-1's **control store**?

3. **Why** does the MIC-1 microinstruction word require **9 bits** for the "**Next Address**" field?

4. Circle the part of the MAL instructions below which control the indicated part of the microinstruction word:

| Circle the part of each statement below… | …that controls this part of the microinstruction word |
|---|---|
| PC = PC + 1; rd; goto main1 | ALU Control Signals |
| PC = PC + 1; rd; goto main1 | Next Address Field |
| PC = PC + 1; rd; goto main1 | Memory control signals |
| PC = PC + 1; rd; goto main1 | B bus |
| PC = PC + 1; rd; goto main1 | C bus |

5. **If** the MIC-1 microprogram issues a **memory read** (by including the "**rd**" keyword in a MAL statement) during clock cycle 1, **during what clock cycle** can the value that has been read be **used**?

6. Identify which of the following MAL statements are invalid, and why:

    **SP = H - LV; rd; if (Z) goto T else goto F**

    **MAR = SP = OPC = SP – 1; wr**

    **MPC = MPC + 1; fetch; goto main**

    **MDR = H – 1; wr**

    **MAR = LV + MDR; rd**

7. **When is it necessary** for the microprogram to change the value in the **TOS** register?

8. The microcode for the POP IJVM instruction looks like this:

    ```
    pop1    MAR = SP = SP – 1;  rd
    pop2
    pop3    TOS = MDR;  goto Main1
    ```

    **Why is it necessary** to have a "pop2" microinstruction that does **nothing**?

9. The **wide1** microinstruction is:

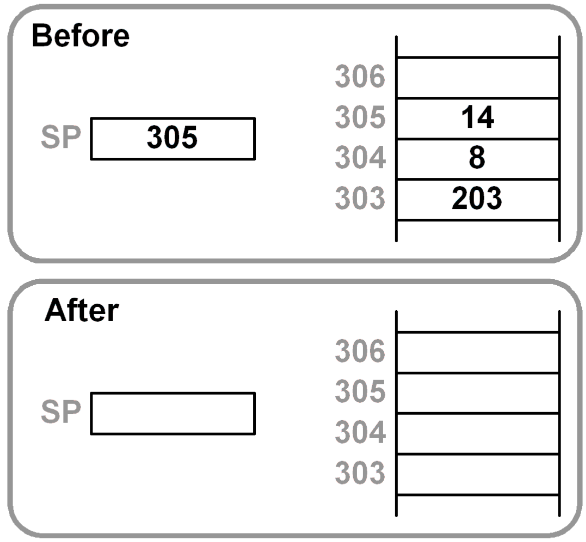
    ```
    wide1    PC = PC + 1;  fetch;  goto (MBR OR 0x100)
    ```

    **It ORs the opcode** of the next IJVM instruction **with hex 100** in order to find the **next microinstruction address**.

    The **opcode** of the IJVM "**ISTORE**" instruction is hex **36**. **At what control store address** is the "**wide_istore1**" microinstruction located?

10. The "**before**" diagram at right shows the **stack** before the following set of microinstructions is executed. **Fill in the "after" diagram** to show **how the stack has changed** after the all of the microinstructions finish executing.

**x1 SP = MAR = SP – 1; rd**
**x2 H = TOS**
**x3 H = H + TOS**
**x4 TOS = MDR = MDR + H; wr**




11. **Write the MIC-1 microcode** to implement an IJVM "**ISUBR**" (Subtract Reversed) instruction. This instruction **works just like "ISUB"**, except that the **order of the operands is reversed**. With the "before" stack shown at right, an "ISUB" instruction would subtract 14 from 8 and give a result of -6. The "ISUBR" instruction that you should write subtracts 8 from 14 and gives a result of 6 as shown in the "after" example at right.