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

   Only 1 register at a time is allowed to be connected to the B bus, so the 4 bits can specify which single register is to be connected.

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**?

   At control store address hex 5F

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

   Because the control store holds 512 microinstruction words, and 9 bits are needed to choose one of 512 addresses ($2^9 = 512$)

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**?

   During clock cycle 3

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

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

Invalid – can't do "H - LV" (ALU function B – A is OK, not A – B)

MAR = SP = OPC = SP – 1; wr

Valid – ALU result can go to any number of registers via C bus

MPC = MPC + 1; fetch; goto main

Invalid – MPC is not on the data path and can't be loaded from the ALU result

MDR = H – 1; wr

Invalid – can't do "H – 1" (ALU function B – 1 is OK, not A – 1)

MAR = LV + MDR; rd

Invalid – LV and MDR can't both be sent to the ALU at the same time

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

Whenever the there is a new "top of stack" value. This includes:

- **When a new value is put onto the top of the stack**
- **When a value is removed from the stack, leaving another value behind that becomes the new "top of stack"**

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**?

Because the value read from memory by "pop1" can't be used in the very next clock cycle.

9. The **wide1** microinstruction is:

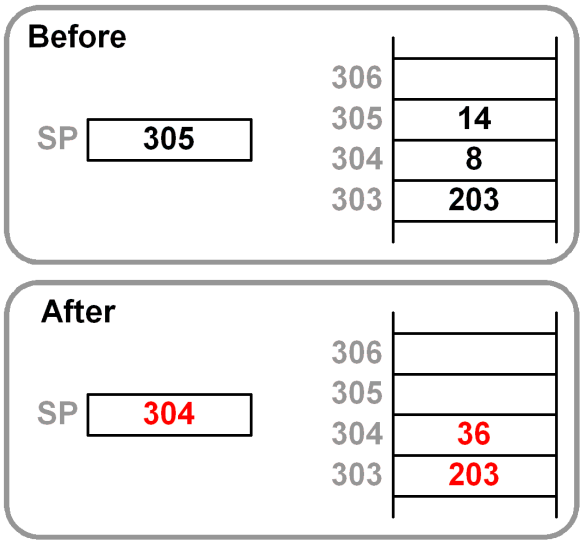
```
wide1   PC = PC + 1; fetch; goto (MBR OR 0x100)
```

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?

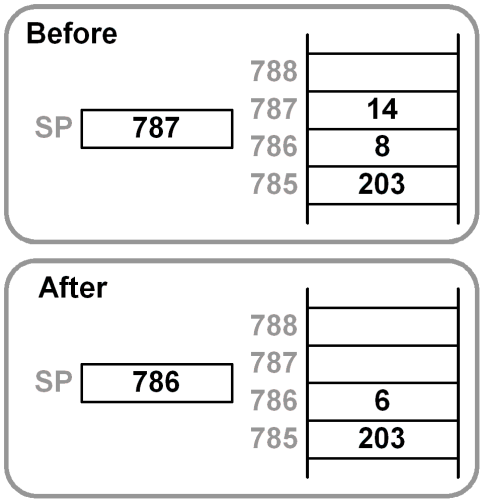
at control store address hex "100" OR "36" = hex 136

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.




```
isubr1    MAR = SP = SP – 1; rd
isubr2
isubr3    H = MDR
isubr4    MDR = TOS = TOS – H; wr; goto main1
```

Note that this is very similar to the microcode for the regular "ISUB" instruction, except that we have to do the subtraction the other way around. The 2nd word on the stack is read into MDR, and we have to subtract it from TOS. But we can't move TOS to H and then use "H – MDR", so we have to put the MDR value into the H register to subtract it (the ALU allows "xxx minus H" but not "H minus xxx"). This means we need an extra clock cycle while we wait for the 2nd word on the stack to be read into MDR before we can move it to H.