Today:

Briefly go over the problems in Homework #2:

- Sometimes, a very simple experiment can answer a lot of questions. **Just try it.**
- Modifying code on the fly – question 5
- Addressing mode – question 6

Question 1,2,3

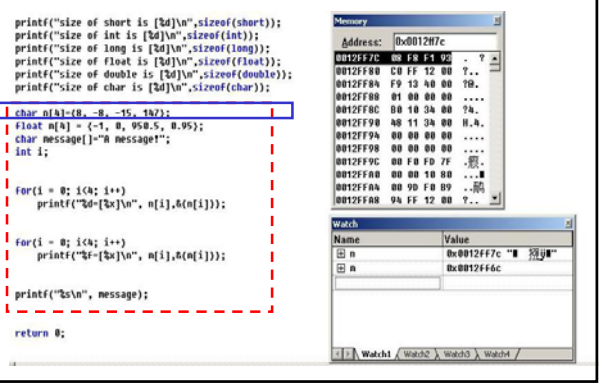
```
char n[4]={8, -8, -15, 147};
float m[4] = {-1, 0, 950.5, 0.95};
char message[]="A message!";
int i;

for(i = 0; i<4; i++)
        printf("%d=[%x]\n", n[i],&(n[i]));

for(i = 0; i<4; i++)
        printf("%f=[%x]\n", m[i],&(m[i]));

printf("%s\n", message);
```

Breakpoint here for question 1

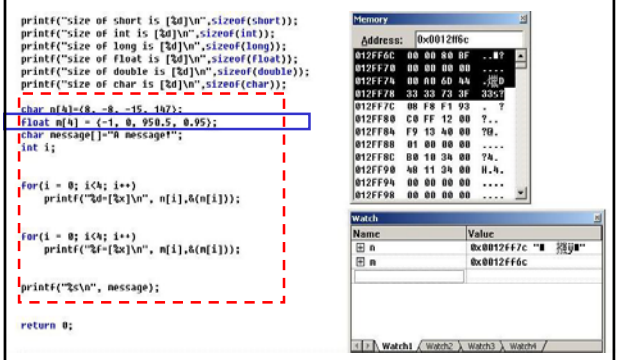
Breakpoint here for question 2

Breakpoint here for question 3

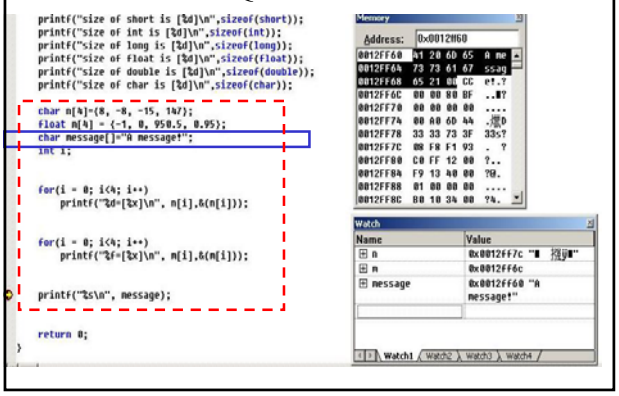
Question 1



Question 2



Question 3



Question 4

- We only have ADD instruction, so we need to invert an integer when we do subtraction.
  - X-Y=X+(-Y)
- How to invert an integer in SSI-0?
  - Only use ADD and XOR
  - !! (-Y) = (Y XOR 0Xffffffff) ADD 1
    - Why?

```
0x00   XOR 0x0d, 0x0d, 0x0d          # x = 0
0x01   ADD 0x0d, 0x0d, 0x0e          # x = A[0]
0x02   XOR 0x0f, 0x0f,  0x0b         # A[1] = A[1] XOR 0xffffffff
0x03   ADD 0x0f, 0x0f,  0x0c         # A[1] += 1
0x04   ADD 0x0d, 0x0d, 0x0f          # x += A[1]
0x05   ADD 0x0d, 0x0d, 0x10          # x += A[2]
0x06   XOR 0x11, 0x11,  0x0b         # A[3] = A[4] XOR 0xffffffff
0x07   ADD 0x11, 0x11,  0x0c         # A[3] += 1
0x08   ADD 0x0d, 0x0d, 0x11          # x += A[3]
0x09   ADD 0x0d, 0x0d,  0x12         # x += A[4]
0x0a   STOP
0x0b   .word 0xffffffff              #constant  0xffffffff
0x0c   .word 1                       #constant  1
0x0d   .word 0                       # this is 'x'
0x0e   .word 0xffffffff              # this is A[0]
0x0f   .word 0x00000000              # A[1]
0x10   .word 0x00000001              # A[2]
0x11   .word 0x00000002              # A[3]
0x12   .word 0x00000004              # A[4]
```

Question 5

- How to implement a loop in SSI-0?
- How to address a memory whose address is changing in the running of the program?
  - Using a variable (memory word) to store the address.
    - We only have direct addressing mode, "BZ target c"
    - We don't have indirect addressing modes. No way to do "BZ target Mem[c]"

```
0x00   XOR 0x08, 0x08, 0x08          # x = 0
0x01   BZ   0x05, 0x11               # check if null
0x02   ADD 0x08, 0x08,  0x07         # x += 1
0x03   ADD 0x01, 0x01, 0x07          # Modify instruction
0x04   BZ   0x01, 0x07               # go back to 0x01
0x05   STOP                          # Stop
0x06   .word 0                       # constant 0
0x07   .word 1                       #constant  1
0x08   .word 0                       # this is 'x'
0x09   ......                        # we don't care
0x0a   ......                        # we don't care
0x0b   ......                        # we don't care
0x0c   ......                        # we don't care
0x0d   ......                        # we don't care
0x0e   ......                        # we don't care
0x0f   ......                        # we don't care
0x10   ......                        # we don't care
0x11   XXXXX                         # String starts here
```

- Memory is just memory. CPU has no idea of the meaning of the memory until it is fetched.
- Every word goes into IR is deemed as instruction.
- In SSI-0, we can modify/create code on the fly.
- In modern systems, it is not possible because of the separation of code and data. The instruction segment is protected by OS.
- However sometimes, other form of code modifications are helpful.
  - java bytecode modification for security

Problem 6

- In SSI-2, the only addressing method you can use is indirect register addressing.
  - LOAD rd, rt => GPR[rd] = Mem[GPR[rt]]
- No immediate addressing mode is allowed. Everything(content/address) must go through registers.
- "Load a, b" can load element of vector into GPR[a], but how do we put the address of the vector into b?
  - in start position, PC=0, and GPRs have random contents
- We also need some constants. How do we put them into GPRs?

```
Add     0, 0, 1              #  0X00000001
XOR     0, 0, 0              #  clear GPR[0]
LOAD    1, 0                 #  GPR[1] now is 1
XOR     2, 2, 2              #  clear GPR[2]
ADD     2, 2, 1              #  GPR[2] += GPR[1]
ADD     2, 2, 1              #  GPR[2] += GPR[1]
                                Now GPR[2] is 2
......
AddressofVec1:  .word address(Vec1)     # address of Vec1
AddressofVec2:  .word address(Vec2)     # address of Vec2
N:              .word N                 # the length of of the vectors
```

At this point, you can (in theory) construct any constant you want, including AddressofVec1, AddressofVec2, and N

And then you can load the element of each vector into your general purpose register.

(use AddressofVec2+N-1 to get the address of the last element of vector2)

And then you can do everything need for this program.(iteration, subtraction, etc.)