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

```c
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);
```

Question 1

```c
print("size of short is \[\$\]", sizeof(short));
print("size of long is \[\$\]", sizeof(long));
print("size of double is \[\$\]", sizeof(double));
print("size of char is \[\$\]", sizeof(char));
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);
```

Question 2

```c
print("size of short is \[\$\]", sizeof(short));
print("size of long is \[\$\]", sizeof(long));
print("size of double is \[\$\]", sizeof(double));
print("size of char is \[\$\]", sizeof(char));
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);
```

Question 3

```c
print("size of short is \[\$\]", sizeof(short));
print("size of long is \[\$\]", sizeof(long));
print("size of double is \[\$\]", sizeof(double));
print("size of char is \[\$\]", sizeof(char));
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);
```

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?
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]”

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 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.)