Objectives:

- Fetch and Execute; Load and Store, Executing Java bytecodes
- Algorithms underlie programs
- Von Neumann architecture

1. Computer Science Terminology – did your neighbor do the readings?

A _____ is a memory location (or locations) that has been given a name.

A _____ is a kind of control structure

A _____ is a named sequence of instructions

Four components of programming cycle:

Why does Java use a virtual machine? What are the advantages? What is a virtual machine?

What is a compile error?

2. A simple machine language:

ZERO_REG DESTR – puts a zero in the specified register.

ADD SRCR1 + SRCR2 -> DESTR – add two registers together and write the result in the destination register.

ADD SRCR1 + CONSTANT -> DESTR – add a register to a constant and write the result in the destination register.

SUB SRCR1 - SRCR2 -> DESTR – subtract one register from another and write the result in the destination register.

SUB SRCR1 - CONSTANT -> DESTR – subtract a constant from a register and write the result in the destination register.

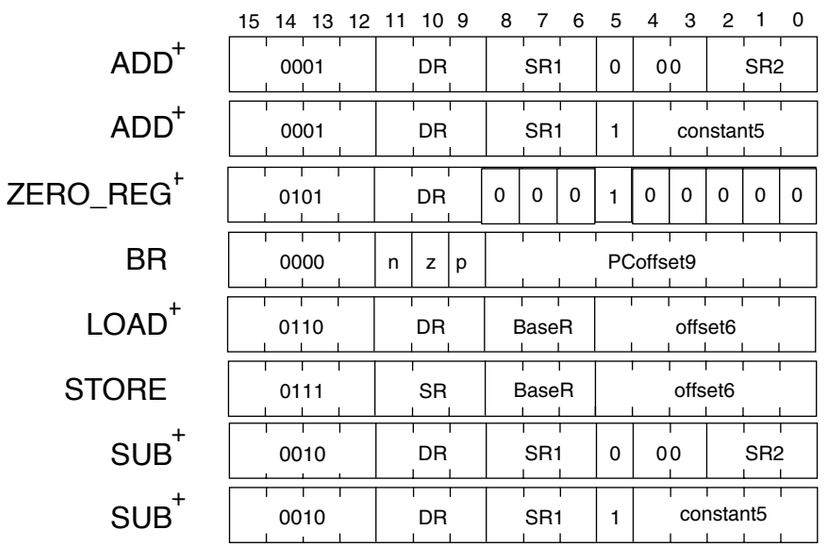
LOAD DESTR <- [BASER + CONSTANT] – add the value of a register to a constant to compute a memory address and copy 4 bytes starting at that address to the destination register.

STORE SRCR1 -> [BASER + CONSTANT] – add the value of a register to a constant to compute a memory address and copy the source register to 4 bytes of memory starting at that address.

BR.___ PCOFFSET – the branch instruction specifies a combination of condition codes (n, z, p); if any of the specified condition codes holds a 1, the PC is set to PC + 2 + 2(*PCOFFSET)*. Otherwise PC is set to PC + 2.

For all instructions other than the branch, PC is set to PC + 2. Any instruction that writes a general-purpose register also set the condition code bits: if new value is negative then n=1, else n=0; if new value is zero then z=1, else z=0; if new value is positive then p=1, else p=0.

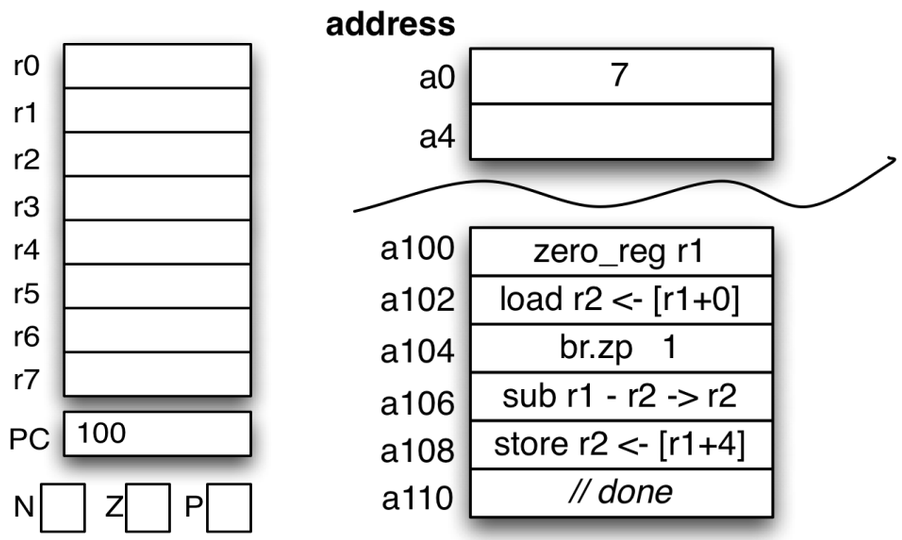
3. Decoding an instruction:



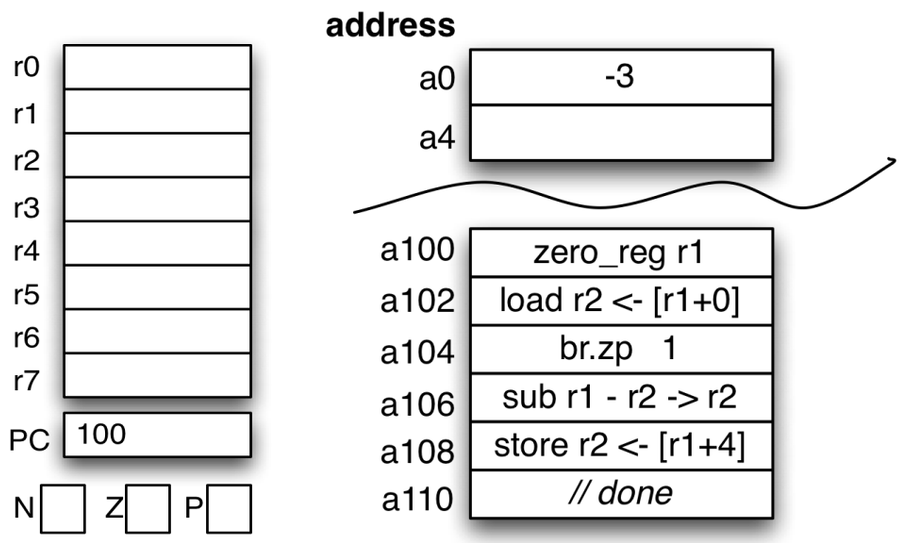
4. Decoding 16 bit-string instructions:

```
0111011001000100

0001100011000010

0101101000100000

0000110000001100
```

6. Execute a machine code program - me:

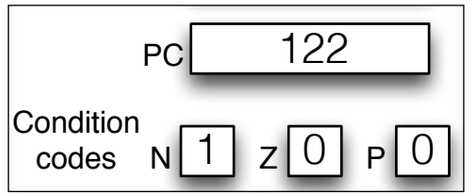


7. Execute a machine code program - you:



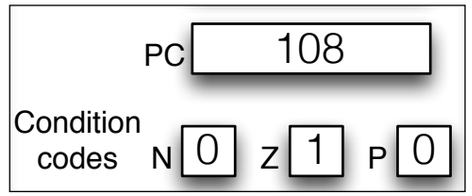
13. What does this code do?

5. All about that branch, 'bout that branch, 'bout that branch …



BR.N 6 PC =

BR.NZ -6 PC =



BR.NZP 22 PC =

BR.ZP -10 PC =

Workspace:

8. Complete the **Java source code** below for a program that displays the following message: Boing! followed by a newline. Your code must compile and work exactly as described.

Pseudo code: Print "Boing!" to the screen, followed by a newline

Java Code:

```
public class BoingPrinter {
```

14. Complete the **Java source code** below for a program that displays the following message: Boing! followed by a newline. Your code must compile and work exactly as described.

Pseudo code: Print “Boing!” to the screen, followed by a newline

Java Code:

```
public class BoingPrinter {
```

9. Why is the list in a residential telephone book sorted by name?

10. If the number of residents doubled why does it *not* take twice as long to lookup a number for a given name?

11. Why must you use a different search algorithm to find a name given a number?

12. If the number of residents doubled why does it take twice as long to lookup a name?

13. Why are some algorithms more efficient than others? When and why is this efficiency important? How should we measure or describe efficiency?

15. Wooden toy abstraction demo:

15. Java as a high-level language: What happens 'under the covers' in the following code? How often do we read 'score' ? ____,
write to score? ___
How many bytes are used to hold the value of score?

```
int score=0;
score = score + 1;
if(score>0) ...
```

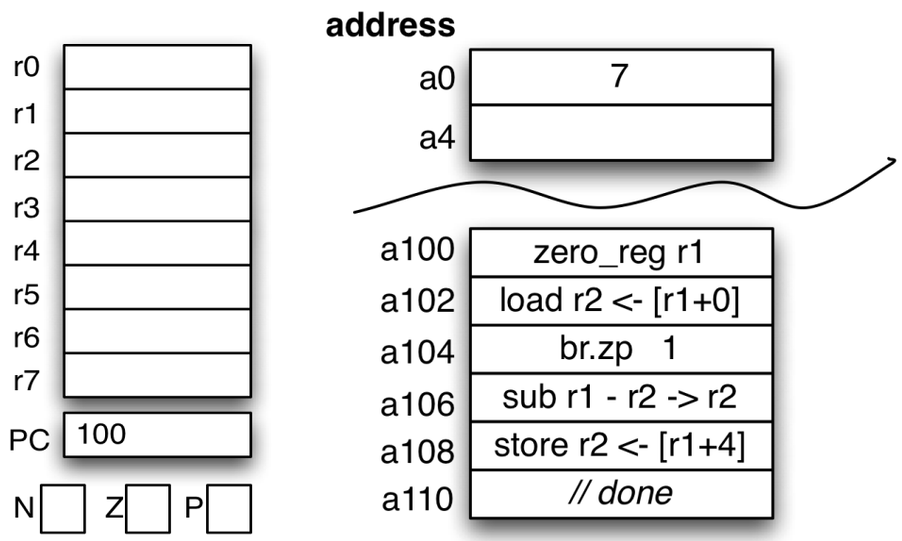
16. Map the variables in the code to the block of memory at right –->

Computing a Quiz Average: Pseudo-code to calculate a quiz average.

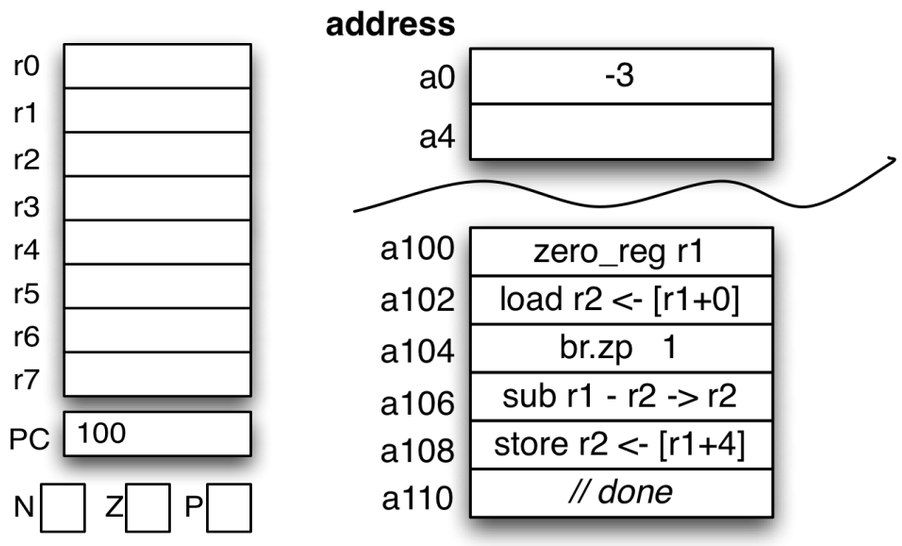
1. get number of quizzes
2. sum := 0
3. count := 0
4. while count < number of quizzes
   - get quiz grade
   - sum = sum + quiz grade
   - count = count + 1
5. average = sum / number of quizzes
6. display average

Memory Address	Value
11	
10	
9	
8	
7	
6	
5	
4	
3	
2	
1	
0	

9. Execute a machine code program - me:



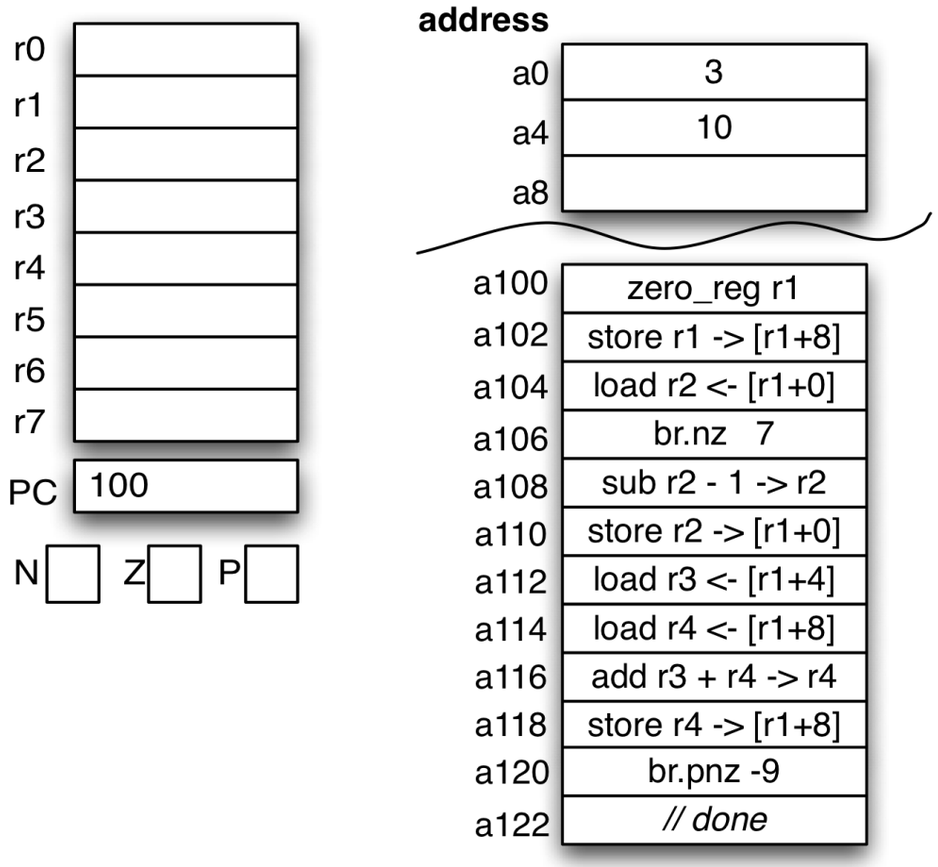
10. Execute a machine code program - you:



11. What does this code do?

Workspace:

4. Execute a machine code program:



5. What does this code do?

Workspace:

A simple machine language:

ZERO_REG DESTR – puts a zero in the specified register.

ADD SRCR1 + SRCR2 -> DESTR – add two registers together and write the result in the destination register.

ADD SRCR1 + CONSTANT -> DESTR – add a register to a constant and write the result in the destination register.

SUB SRCR1 - SRCR2 -> DESTR – subtract one register from another and write the result in the destination register.

SUB SRCR1 - CONSTANT -> DESTR – subtract a constant from a register and write the result in the destination register.

LOAD DESTR <- [BASER + CONSTANT] – add the value of a register to a constant to compute a memory address and copy 4 bytes starting at that address to the destination register.

STORE SRCR1 -> [BASER + CONSTANT] – add the value of a register to a constant to compute a memory address and copy the source register to 4 bytes of memory starting at that address.

BR.___ PCOFFSET – the branch instruction specifies a combination of condition codes (n, z, p); if any of the specified condition codes holds a 1, the PC is set to PC + 2 + 2(*PCOFFSET)*. Otherwise PC is set to PC + 2.

For all instructions other than the branch, PC is set to PC + 2. Any instruction that writes a general-purpose register also set the condition code bits: if new value is negative then n=1, else n=0; if new value is zero then z=1, else z=0; if new value is positive then p=1, else p=0.