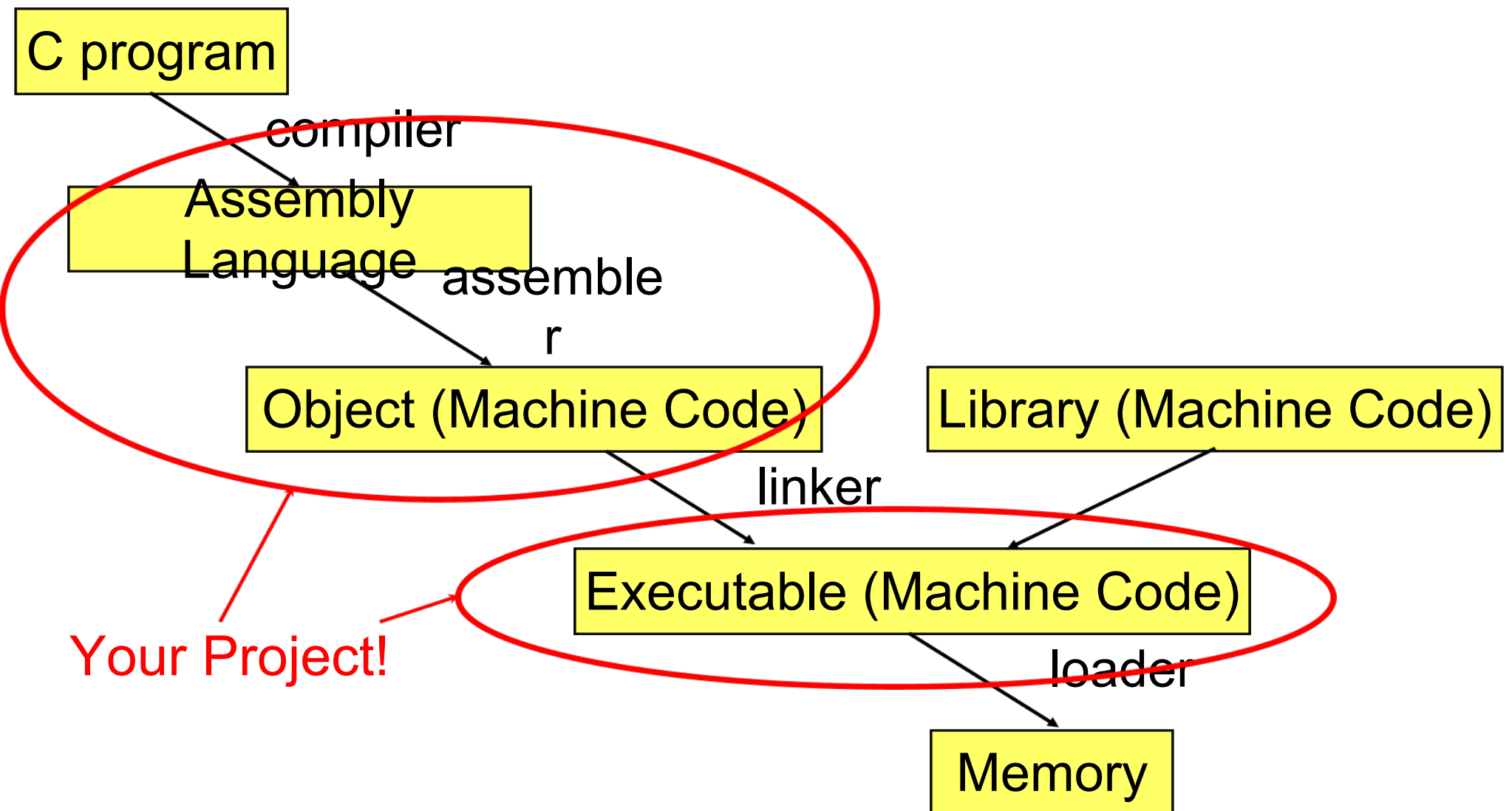


ISA & CSE 141 Project

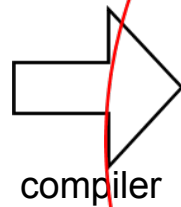
Slides by Hung-Wei Tseng

From Code to Running Programs



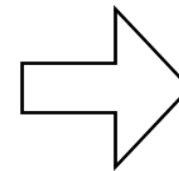
From Code to Running Programs

```
int main()  
{  
    printf("hello, world\n");  
    return 0;  
}
```



compiler

```
ldah gp,8192(t12)  
lda gp,28464(gp)  
lda sp,-16(sp)  
stq ra,0(sp)  
stq fp,8(sp)  
mov sp,fp  
ldah a0,-1(gp)  
lda a0,32648(a0)  
ldq t12,-32584(gp)  
jsr ra,(t12),120006b40  
ldah gp,8192(ra)  
lda gp,28424(gp)  
clr v0  
br 120001198  
mov fp,sp  
ldq ra,0(sp)  
ldq fp,8(sp)  
lda sp,16(sp)  
ret zero,(ra),0x1  
nop
```



assembler

```
00 20 bb 27  
30 6f bd 23  
f0 ff de 23  
00 00 5e b7  
08 00 fe b5  
0f 04 de 47  
ff ff 1d 26  
88 7f 10 22  
b8 80 7d a7  
6e 56 5b 6b  
00 20 ba 27  
08 6f bd 23  
00 04 ff 47  
00 00 e0 c3  
1e 04 ef 45  
00 00 5e a7  
08 00 fe a5  
10 00 de 23  
01 80 fa 6b  
1f 04 ff 47
```

Your Project!

Assembly Language

- Text representation of machine instructions

keyword • Sample:

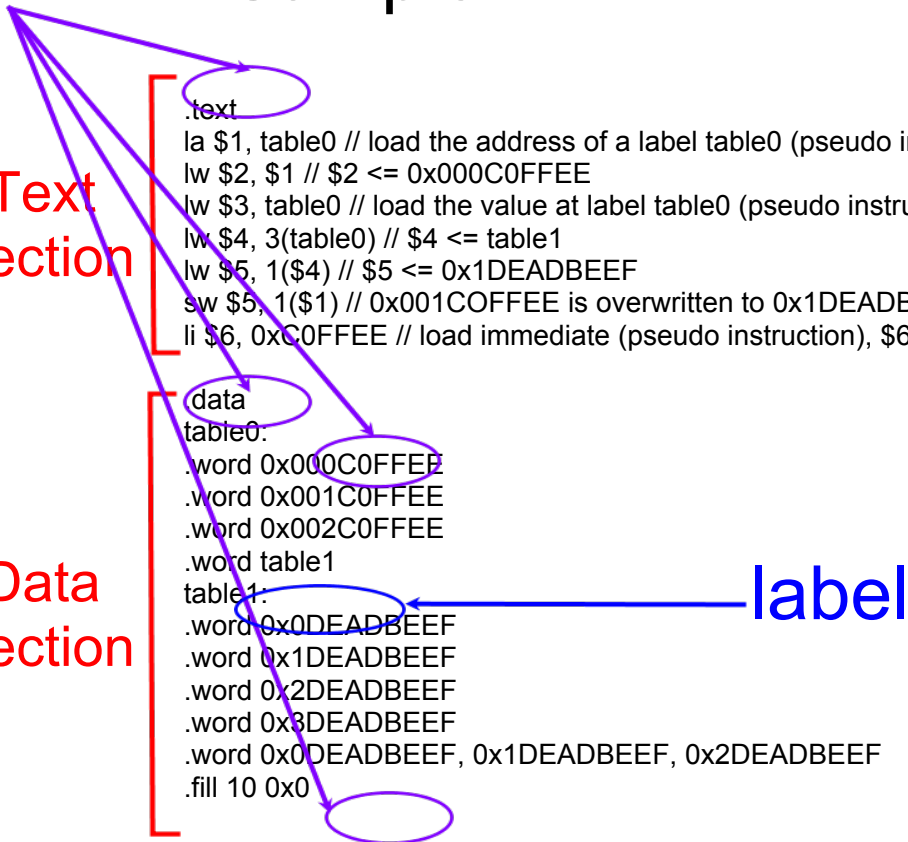
Text
Section

```
.text  
la $1, table0 // load the address of a label table0 (pseudo instruction)  
lw $2, $1 // $2 <= 0x000C0FFEE  
lw $3, table0 // load the value at label table0 (pseudo instruction), $3 <= 0x000C0FFEE  
lw $4, 3(table0) // $4 <= table1  
lw $5, 1($4) // $5 <= 0x1DEADBEEF  
sw $5, 1($1) // 0x001C0FFEE is overwritten to 0x1DEADBEEF  
li $6, 0xC0FFEE // load immediate (pseudo instruction), $6 <= 0xC0FFEE
```

Data
Section

```
.data  
table0:  
    .word 0x000C0FFEE  
    .word 0x001C0FFEE  
    .word 0x002C0FFEE  
    .word table1  
table1:  
    .word 0x0DEADBEEF  
    .word 0x1DEADBEEF  
    .word 0x2DEADBEEF  
    .word 0x3DEADBEEF  
    .word 0x0DEADBEEF, 0x1DEADBEEF, 0x2DEADBEEF  
    .fill 10 0x0
```

label



Assembly Language

- Sections
 - Text Section: Instructions of the program
 - Data Section: Initialized data storage
- Keyword
 - Indicate the change of sections, how to deal with the data or ...
- Label
 - A name for an instruction/data address
- Pseudo Instruction
 - The shortcut to a sequence of instructions

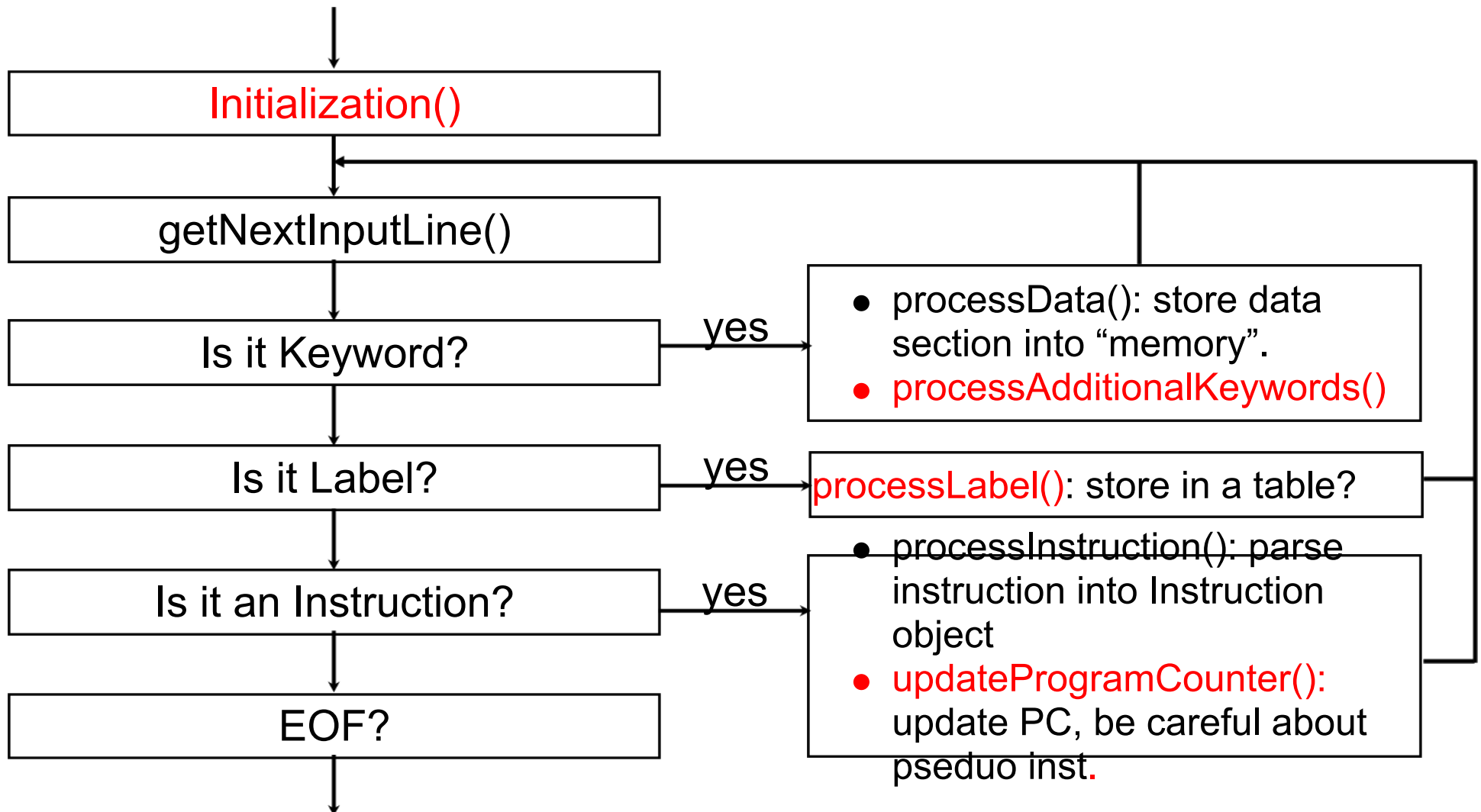
Assembler

- Translate assembly code to machine code
 - From `add $s0, $a1, $t7` to `0x00AF8020`
- Develop an assembler
 - You may use the java-based [Assembler framework](#)
 - Accept your assembly code as input
 - Output
 - `[$prefix]_i.coe`: machine code
 - `[$prefix]_d.coe`: initial data memory

Tips of Using Framework

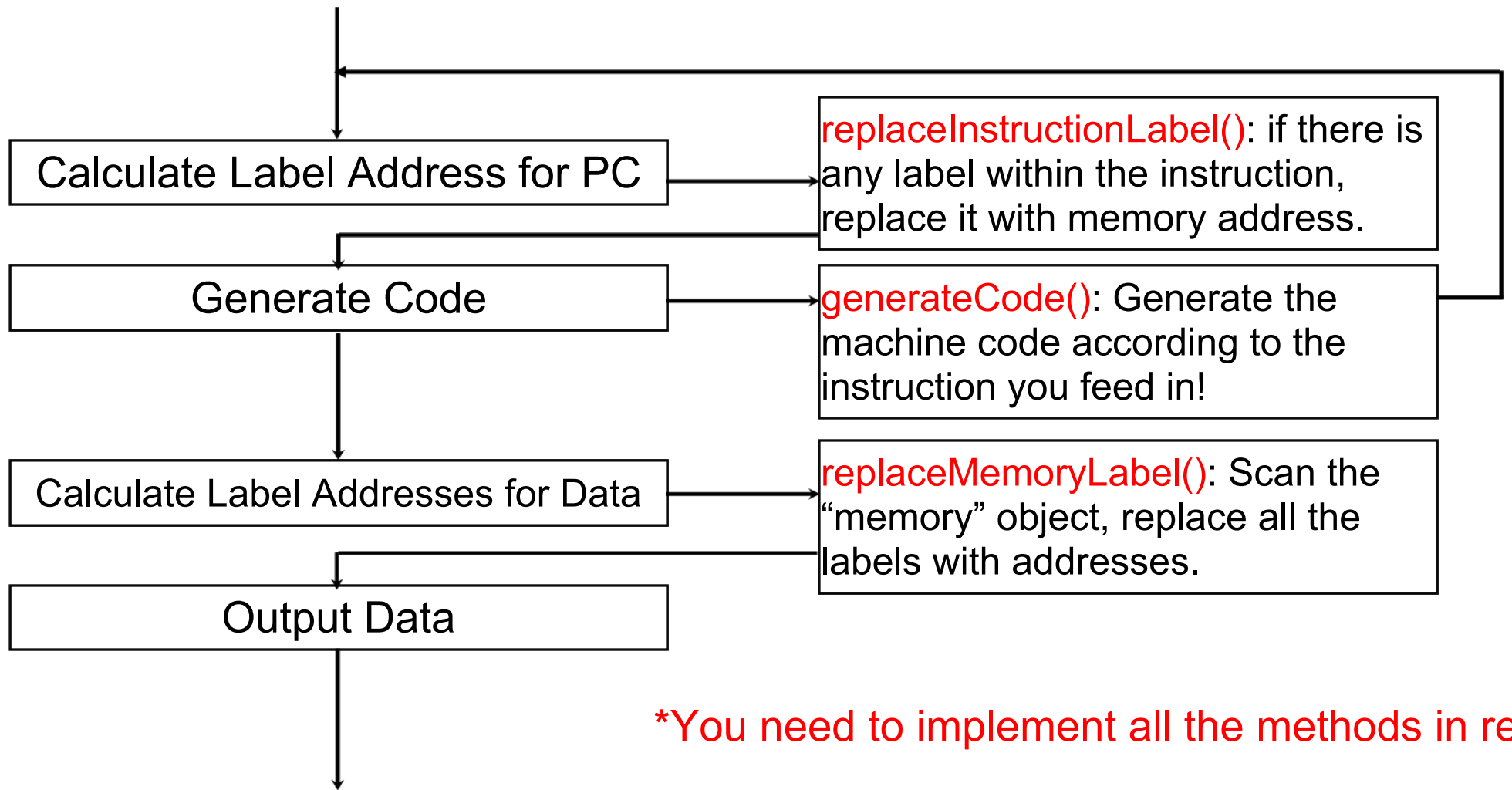
- Extend the Assembler class
 - AssembleCode() contains a two parse algorithm
 - parse #1: scan the code and labels
 - parse #2: replace labels with offsets (or addresses) and generate code
- Implement virtual methods:
 - processLabel, generateCode, updateProgramCounter, initialization, replaceInstructionLabel, and replaceMemoryLabel

AssemblyCode() 1st Parse



*You need to implement all the methods in red

AssemblyCode() 2nd Parse



Simulator

- Simulate the behavior of a processor running your own ISA
 - Input: the assembled machine code
 - Output: what the code supposed to do

Simulator

- Develop a simulator
 - You may use the provided [simulator framework](#)
 - Tip:
 - Starting by modify the switch statement in the "execute" method in the provided ISASimulator class

Benchmarks

- Programs used to validate the functionality and performance of a computer system
 - Fibonacci numbers
 - Function calls
 - SuperGarbage
 - A virtual machine
 - Be careful of branch instructions.