5.22 Compilers

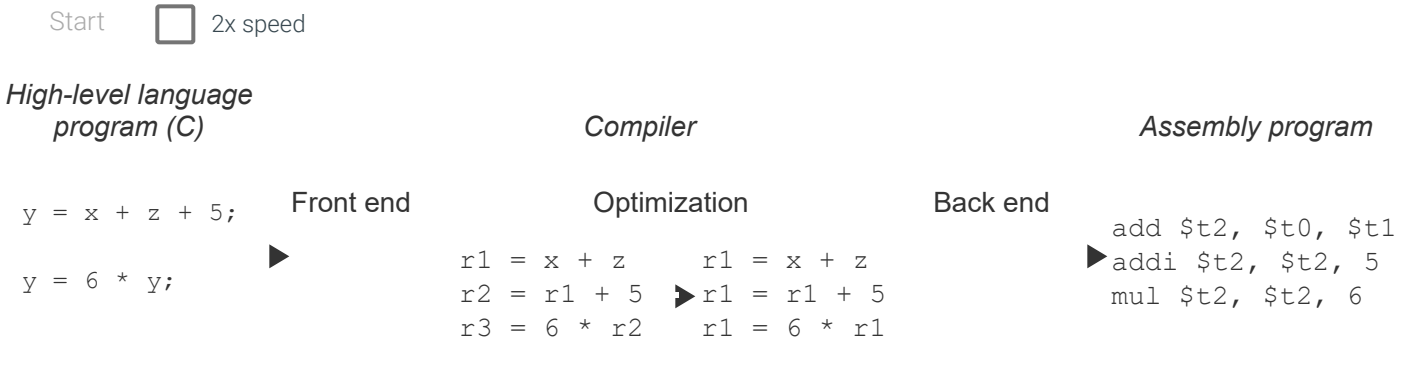
Basic compiler operations

A ***compiler*** is a program that converts a program in a high-level language, like C, C++, or Java, into assembly instructions.

A modern compiler typically has three parts:

- A compiler's ***front end*** ensures the program is valid according to the language's rules, and converts the program to ar representation (IR). Ex: `y = 6 * y;` is valid in C, but `y = 6y;` is not.
- A compiler's ***optimizations*** simplify the intermediate representation.
- A compiler's ***back end*** converts the intermediate representation to a processor's assembly instructions.

PARTICIPATION ACTIVITY 5.22.1: A compiler converts a high-level language program into assembly instructions.



PARTICIPATION ACTIVITY 5.22.2: Compiler.

Which compiler would be responsible for the following tasks?

1) Implement the operation `x = y * 2` as the instruction `add $t3, $t2, $t2`.
   - ○ Front end
   - ○ Optimization
   - ○ Back end

2) Detect and report a syntax error for the following C statement.

```
if numVal < 5 {
```

   - ○ Front end
   - ○ Optimization
   - ○ Back end

3) Determine which MIPS register should hold the result of the operation `w = x + 100`.
   - ○ Front end
   - ○ Optimization
   - ○ Back end

4) Reduce the number of operations needed to implement the following.

```
x = (a * b)
```

```
y = (a * c)
z = x + y
```

- ○ Front end
- ○ Optimization
- ○ Back end

Memory organization

Program memory refers to all memory contents used by a program, including both instructions and data. Program memory organized into several regions (or segments):

- The ***code*** region contains a program's instructions. The code region is also called the ***text*** region.
- The ***static*** region contains global variables (variables defined outside any function) and static local variables (variable functions starting with the keyword "static").
- The ***stack*** contains values used to call functions and may also contain a function's local variables.
- The ***heap*** contains all dynamically allocated memory.. Ex: The malloc() function allocates memory in the heap, and th deallocates memory in the heap.

When compiling a C program, the compiler determines to which memory region each variable should be located, by determ memory address for each variable, and generating the instructions to initialize the variable, if needed.

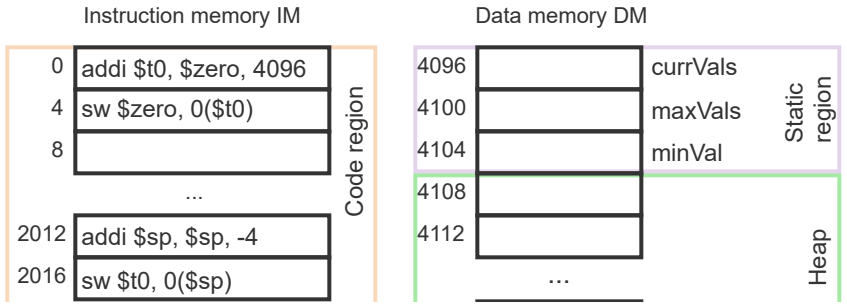
PARTICIPATION ACTIVITY 5.22.3: Program memory organization.

Start ☐ 2x speed

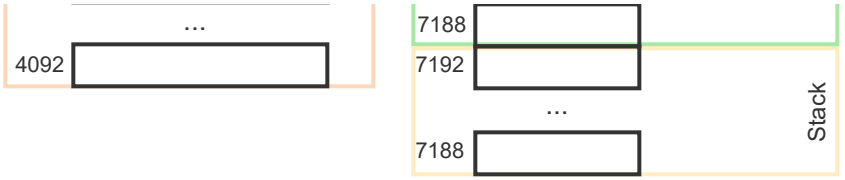
```
int currVals = 0;
int maxVals = 7;
int minVal = -1;

int CalcMax(int x, int y) {
   int max;

   // ...
```



```
    return max;
}
```



PARTICIPATION ACTIVITY 5.22.4: Program memory organization.

Refer to the animation above.

1) In what memory location is minVal located?

- 4096
- 4100
- 4104

2) CalcMax's local variable max will be contained in the static region.

- True
- False

3) Which instructions might the compiler generate to initialize maxVals to 7?

-
```
addi $t2, $zero, 7
sw $t2, 0($t1)
```
-
```
addi $t1, $zero, 4100
addi $t2, $zero, 7
sw $t2, 0($t1)
```
-

```
addi $t1, $zero, 4100
addi $t2, $zero, -1
sw $t2, 0($t1)
```

4) For the following function, in which region would the variable lastVal be located?

```
int DiffFromLast(int newVal) {
   static int lastVal = 0;
   int valDiff = 0;

   valDiff = lastVal - newVal;
   lastVal = newVal;

   return valDiff;
}
```

- Code
- Stack
- Static

data and bss regions

*The static region is often divided into two regions: data and bss. The **data** region contains data that is initialized. Ex: A global variable declaration `int currSize = 4;` initializes the variable to 4, so would be allocated to the data region. The **bss** region (short for block started by symbol -- a term carried over from early assemblers) contains data that is uninitialized. Ex: A static variable declared as `static int lastReading;` does not initialize the variable to 4, so would be allocated to the bss region.*

Provide feedback on this section