**Name: Shah Sarmad Hamdani**

**Reg. No: fa21-bcs-021**

**Q3: Give oprimization example of optimization used in mini.**

**1. Constant Folding**

**What It Does**: Evaluates constant expressions at compile time instead of runtime.

**Example**:  
Input Code:

c

Copy code

int x = 5 \* 3 + 2;

Optimized Code:

c

Copy code

int x = 17;

**Explanation**: Instead of calculating 5 \* 3 + 2 during execution, the compiler computes the result (17) at compile time, reducing runtime computations.

**2. Dead Code Elimination**

**What It Does**: Removes code that does not affect the program’s output.

**Example**:  
Input Code:

c

Copy code

int x = 10;

x = 20;

return x;

Optimized Code:

c

Copy code

int x = 20;

return x;

**Explanation**: The assignment x = 10; is unnecessary as x is immediately reassigned a new value.

**3. Loop Invariant Code Motion**

**What It Does**: Moves calculations inside loops that do not depend on the loop variable to outside the loop.

**Example**:  
Input Code:

c

Copy code

for (int i = 0; i < n; i++) {

int x = 10 \* 20;

arr[i] = x + i;

}

Optimized Code:

c

Copy code

int x = 10 \* 20;

for (int i = 0; i < n; i++) {

arr[i] = x + i;

}

**Explanation**: The computation of 10 \* 20 does not depend on the loop variable i, so it is moved outside the loop to avoid redundant calculations.

**4. Peephole Optimization**

**What It Does**: Improves small code segments by replacing inefficient sequences with simpler ones.

**Example**:  
Input Assembly Code:

css

Copy code

LOAD A, 0

ADD A, 0

STORE A, B

Optimized Assembly Code:

css

Copy code

STORE 0, B

**Explanation**: Adding 0 is unnecessary, so the redundant instructions are removed.

**Significance:**

* Reduces execution time and memory usage.
* Improves performance, especially for resource-constrained systems.
* Ensures the generated code is both efficient and functional.