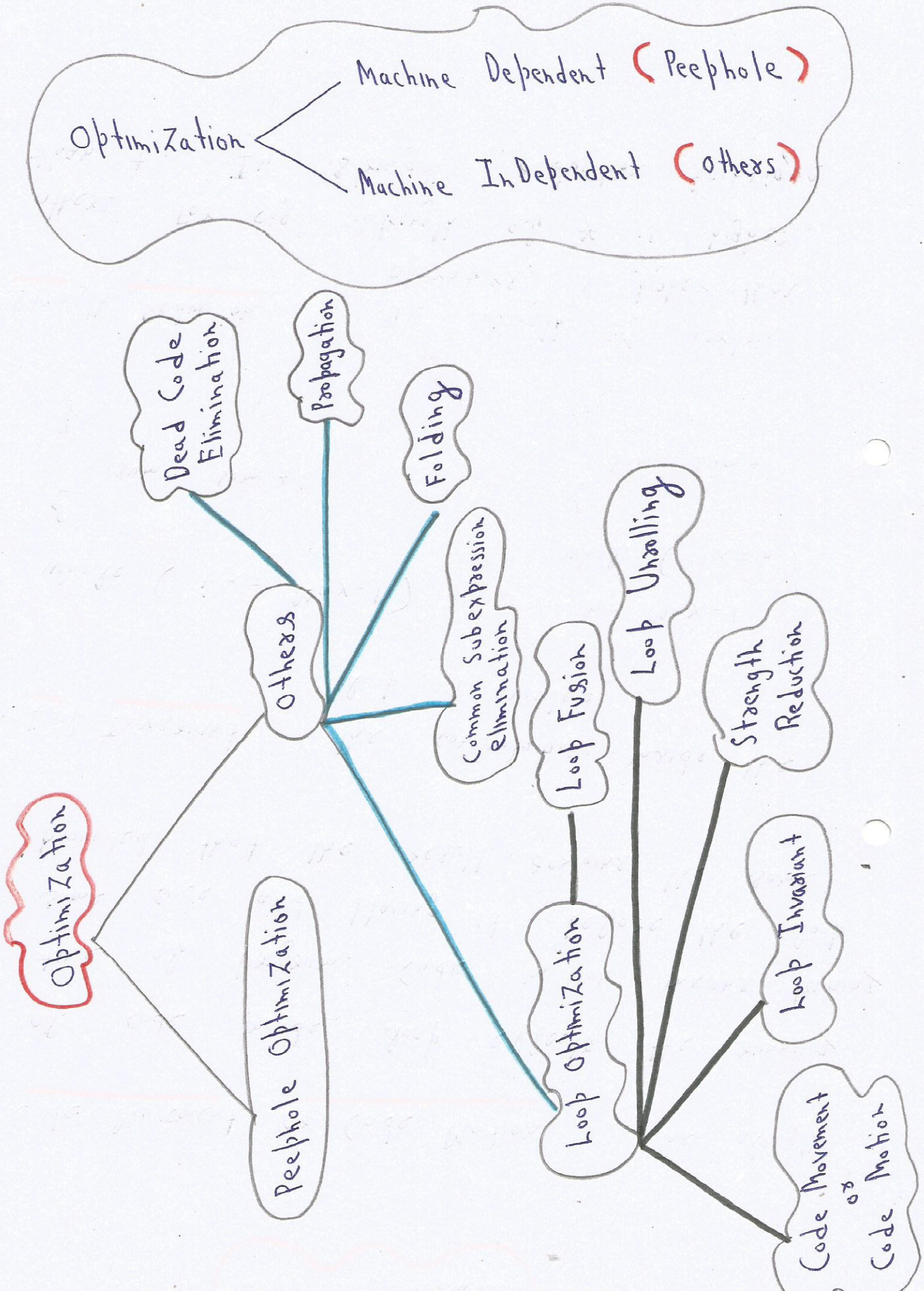
Optimization
Machine Dependent (Peephole)
Machine InDependent (others)
Optimization
Peephole Optimization
Others
Dead Code Elimination
Propagation
Folding
Common Subexpression elimination
Loop Optimization
Loop Fusion
Loop Unrolling
Strength Reduction
Loop Invariant
Code Movement or Code Motion

Loop Optimization

Code Movement or Code Motion The length of code inside loop effects the running time of program. Code motion means taking some code and placing it before the loop provided that the result remains the same

Loop Invariant The computation inside the loop is avoided

```
while ( i <= Max-1 )
{
    sum = sum + 1
}
```

⟹

```
n = Max-1
while ( i <= n )
{
    sum = sum + 1
}
```

Strength Reduction The strength of certain operators is higher than others. For e.g strength of * is higher than +. In strength reduction technique

the higher strength operators can be replaced by lower strength operators.

e.g

```
for (i=1; i<=50; i++)
    count = i * 7;
```

⟹

```
temp = 7
for (i=1; i<=50; i++)
    count = temp
    temp = temp + 7
```

Loop Unrolling In this method number of jumps and tests can be reduced by writing the code two times.

```
int i=1
while ( i<=100)
{
   a[i] = b[i]
   i++
}
```

⟹

```
int i=1
while (i<=100)
{
   a[i] = b[i];
   i++;
   a[i] = b[i];
   i++;
}
```

Loop Fusion

In Loop Fusion method several loops are merged in one loop

```
for i=1 to n do
for j= 1 to m do
a[i,j] = 10
```

⟹

```
for i = 1 to n*m do
a[i] = 10
```

Common SubExpression Elimination

DAG is used

Folding - The computation of constant is done at compile time instead of execution time

e.g length = 22/7 * d. Perform computation of 22/7 at compile time.

Propagation

- Variable Propagation
- Constant Propagation

Constant Propagation In this technique value of variable is replaced and computation of expression is done at the compilation time.

e.g

```
pi = 3.14
r = 5
Area = pi * r * r
```

Here at the compilation time value of pi is replaced by 3.14 and r by 5 then computation of 3.14 * r * r is done during compilation

Variable Propagation

. Use of one variable instead of another. i.e

if there is a copy statement $x = y$

```
c = a * b                    c = a * b
x = a          ⟹            d = a * b + y
d = x * b + y
```

Here variable x is eliminated

Dead Code Elimination

A variable is said to be live in a program if the value contained in it is used subsequently. On the other hand variable is said to be dead at a point in program if the value contained in it is never been used

```
a = x + y                    a = x + y
b = a          ⟹            b = a + 2
b = a + 2
```

Peephole Optimization

Peephole Optimization is applied for optimization of Assembly Code.

Statement by Statement Code generation strategy produces target code (assembly code) that contains redundant things.

① Redundant Instruction Elimination

$temp2 = id_3 * 60.0$

$id_1 = id_2 + temp2$

Assembly code for this

```
Mov  R1, 60.0
Mov  R2, id3
MUL  R1, R2
Mov  temp2, R1
Mov  R1, temp2
Mov  R2, id2
ADD  R1, R2
Mov  id1, R1
```

Redundant

```
Mov R1, 60.0
Mov R2, id3
MUL R1, R2
Mov R2, id2
ADD R1, R2
Mov id1, R1
```

(2) Flow Of Control Optimization

(i) Folding JMP to JMP

```
JMP L1          ⟹      JMP L2
L1: JMP L2
```

(ii) Folding JMP to RETURN

```
JMP L1          ⟹      RET
L1: RET
```

(3) Replacing slow instructions with faster one

```
a load 1                a load 1
a load 1        ⟹      dup
mul                     mul
```

(4) Algebraic simplification

```
X = X + 0
X = X * 1
```

Any statements like this then simply remove