Assignment No. 2

Q.1) Translate following statement into three address code if $x<y$ and $a>b$.

→ 0) $t_1 = x<y$

1) $t_2 = a>b$

2) $t_3 = t_1$ $$ t_2

3) if t_3 goto 4

4) .....

Explanation:→

① $t_1 = x<y$ compares the values of x & y and assigns the result to a new temporary variable t_1.

② $t_2 = a>b$ compares the values of a and b and assigns the results to new temporary variable etc..

③ $t_3 = t_1$ $$ t_2 checks if both t_1 and t_2 are true (ie. have a value of 1) and assign the result to a new temp. variable t_3. If both are true t_3 will be assigned the value 1 otherwise 0.

④ It checks if t_3 is true (ie. has a value of 1). If is true, the control jumps to the inst. labeled 5. otherwise, the program continues to the next inst. often this block.

Q.2 Differentiate betn loop optimization and local optimization.

	loop optimization	local optimization
①	Optimizes loop in the code	Optimize code within a single function.
②	Reduce the number of instruction executed in loops	Reduce the number of instruction executed in a function.
③	maximizes the use of processor registers in loops	Eliminates redundent code with a function
④	Improves performance of loop - intensive code	Improve performance of code by minimizing memory accesses.
⑤	Techniques used includes loop unrolling loop fusion loop invariend code motion and loop vectorization	Techniques used include constant folding dead code elimination strength reduction and common subexpression . elimination.

Page No.
Date: / /20

Q.3 Explain peephal optimization with their characteristics.

→* peephal optimization :→

peephal optimization is a local optimization technique used in compiler to improve code performance. It focuses on eliminating redundent or inefficient code sequence in a small and fixed size window of instruction, known as a "peephole".

* characteristics of peephole optimization :→

① localized optimization:

peephole optimization is a localized optimization techniques that focusses on optimizing a small and fixed sized window of instruction. This window typically betn three and seven instruction large.

② Iterative process:

peephole optimization is an iterative process that examines each instruction in the window and looks for optimization to eliminate redundent.

③ machine Independent:

peephole optimization is a machine-independent optimization technique which means that it can be applied to code generated for any target machine architecture.

Page No.:
Date: / /20

Q.4 What are issues in generating target code?

→ ① Target machine architecture :→

The target architecture may not support all the feature and construct of the source language. This can result in a loss of functionality in the generated code.

② Instruction set limitations :→

The target machine instruction set may have limitations that make it difficult to generate efficient code. For ex. some machine may not support certain types of memory access or have limited supported for floating point arithmetic.

③ Register allocation :→

Register allocation is a critical issue in generating efficient code. If the compiler does not allocate register optimally, it can result in inefficient code that requires excessive memory access.

④ Debugging :→

Debugging generated code can be challenging, especially if the generated code does not match the source code exactly.

⑤ optimization tradeoffs :→

The compiler must make trade offs betⁿ generating efficient code and generating code quality. This can result in suboptimal code in some cases.

⑥ Code size :→

The size of generated code is an imp. consideration, especially for embedded system or mobile device with limited memory. If the generated code is too large it may not fit in the available memory or require excessive memory access, leading to performance degradation.

Q.5) Construct the given expression into DAG.

$a + a * (b-c) + (b-c) * d$

→ steps for constructing a DAG :→

① d_1 = leaf (id, entry - a)

② d_2 = leaf (id, entry - a) = d_1

③ d_3 = leaf (id, entry - b)

④ d_4 = leaf (id, entry - c)

⑤ d_5 = node ('-', d_3, d_4)

⑥ d_6 = node ('*', d_1, d_5)

⑦ d_7 = node ('+', d_1, d_6)

⑧ d_8 = leaf (id , entry - b) = d_3

⑨ d_9 = leaf (id, entry - c) = d_4

(10) d_{10} = node ('-', d_3, d_4) = d_5

(11) d_{11} = leaf (id, entry-d)

(12) d_{12} = node ('*', d_6, d_{11})

(13) d_{13} = node ('+', d_7, d_{12})

Now,

t_1 = b-c

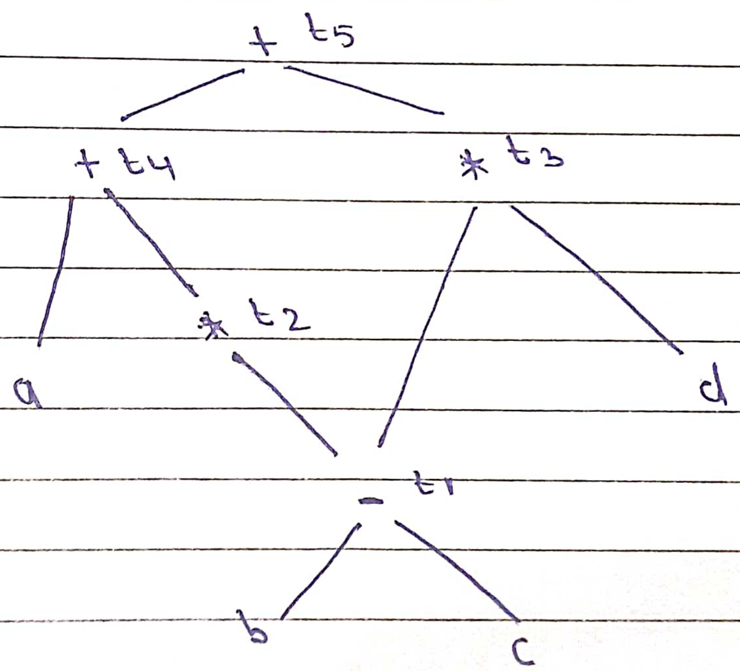
t_2 = a * t_1

t_3 = t_1 * d

t_4 = a + t_2

t_5 = t_4 + t_3

DAG:



+ t5
+ t4
* t3
* t2
a
d
- t1
b
c