Compiler Design

- SCSA1604

Name: Mohnish Devaraj

Reg No: 39110636

Assignment-II

PART-A

(1) c. loop-invariant code

(2) Dead

(3) c. Cross compiler

(4) used to represent the variables or parameters.

(5) strength reduction.

PART-B

(1) The difference between dependent and machine independent code optimization is that the machine dependent optimization is applied to object code whereas, the machine independent code optimization is applied to intermediate code.

(2) Constant folding is an optimization technique that eliminates expressions that calculate a value that can already be determined before code execution. These are typically calculations that only reference constant values or expressions that reference variables whose value are constant.

For example,

Consider this statement

i ← 320 * 200 * 32

Most compilers would not actually generate two multiple instructions. Instead, they identify constructs such as these and substitute the computed values (in this case, 2048000). This way, the code will be replace by

i ← 2048000

③ DAG is used in:

- Determining the common sub-expression
- Determining which names are used inside the block and computed outside the block.
- Determining which statements of the block could have their computed value outside the block.
- Simlifying the list of quadruples by eliminating the common su-expresions and not performing the assignment of the form x := y unless and until it is a must.

④ The most common way of sdving the data-flow equations is by using an iterative algorithm. It starts with an approximate of the in-state of each block. The out-states are then computed by applying the transfer functions on the in-states.

$out_b = trans_b(in_b)$ $OUT[B] = IN[B] - KILL[B]$ $GEN[B]$

$in_b = \bigwedge$ join $IN[B] = OUT[B]$ $IN[B] \wedge \neg KILL[B]$

(5) 90-10 rule:

states that 90% of the time is spent in the 10% of the code

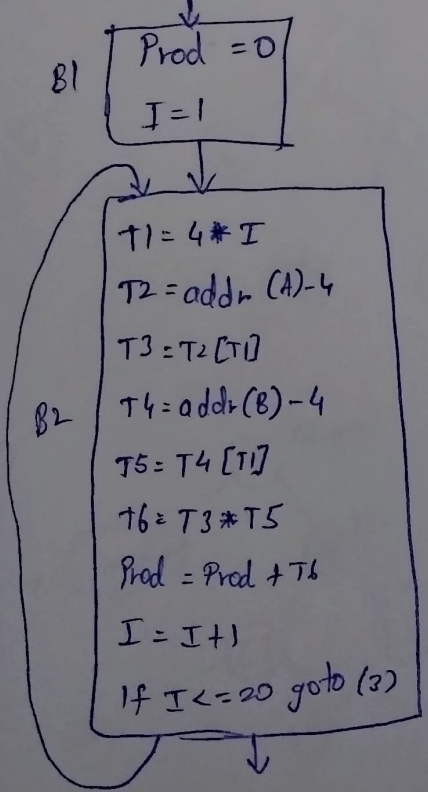
PART-C

(1)

```
begin
    prod := 0;
    i := 1;
    do begin
    prod := prod + a[i] * b[i]
    i := i+1;
    end
    while i<=20;
end
```

(1) PROD = 0
(2) I = 1
(3) T1 = 4xI
(4) T2 = addr(A)-4
(5) T3 = T2[T1]
(6) T4 = addr(B)-4
(7) T5 = T4[T1]
(8) T6 = T3*T5
(9) PROD = PROD + T6
(10) I = I+1
(11) If I<=20 goto (3)

Flow graph



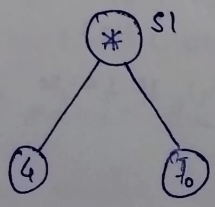
- B1 is the initial node
- B2 immediately follows B1, so there is an edge B1 to B2.
- The target of jump from last statement B2, so there is an edge from B2 (last statement) to B2 (first statement)
- B1 is the predecessor of B2, and B2 is a successor of B1.

Direct Acyclic Graph (DAG)

```
S1 := 4 * I
S2 := a[S1]
S3 := 4 * I
S4 := b[S3]
S5 := S2 * S4
S6 := prod + S5
prod := S6
S7 := I + 1
I := S7
if I <= 20 goto (1)
```

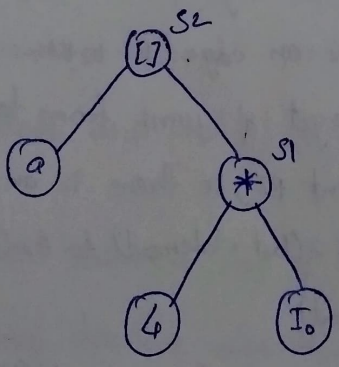
Stages in DAG Construction:

(a)



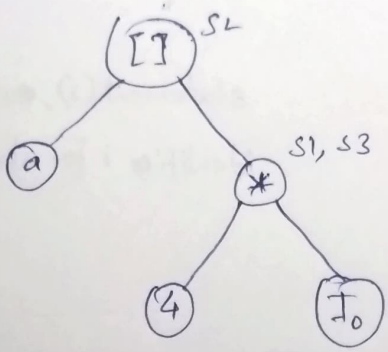
statement (1)

(b)



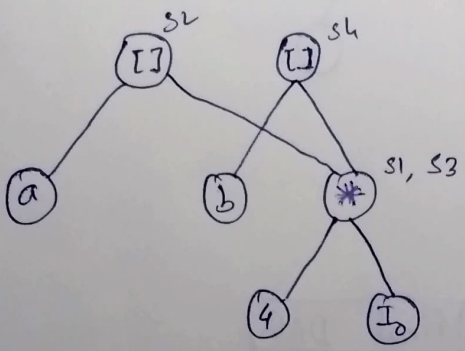
statement (2)

(c)



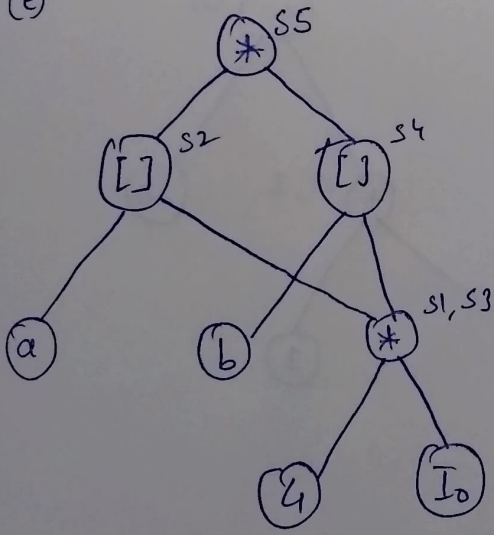
$4 * I_0$ node exist already hence identifier S3 to the existing node for statement (3)

(d)



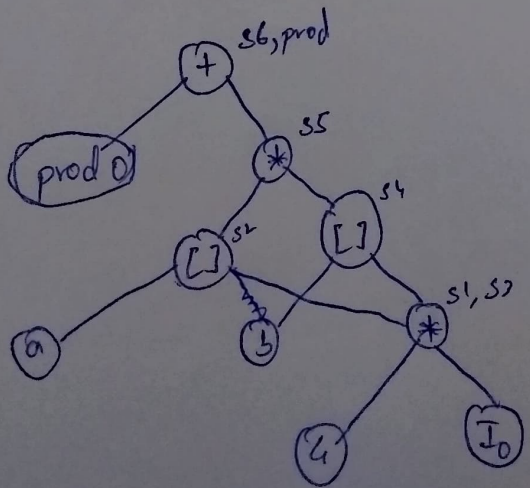
statement (4)

(e)



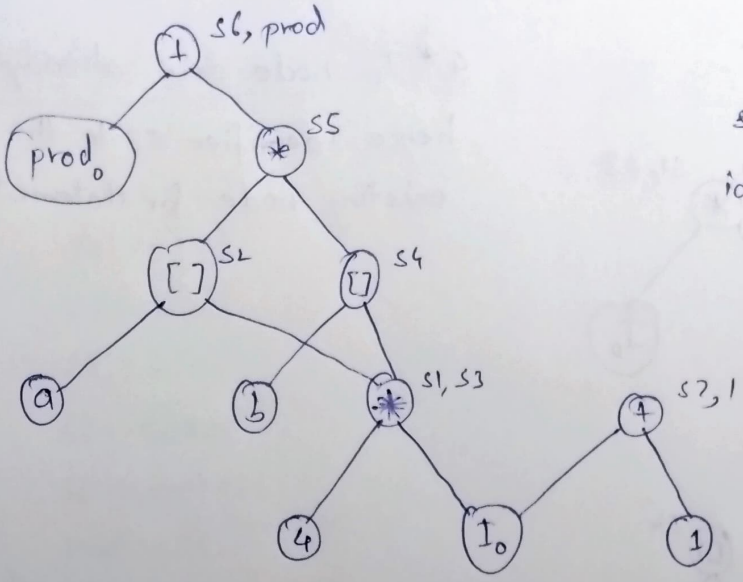
statement (5)

(f)



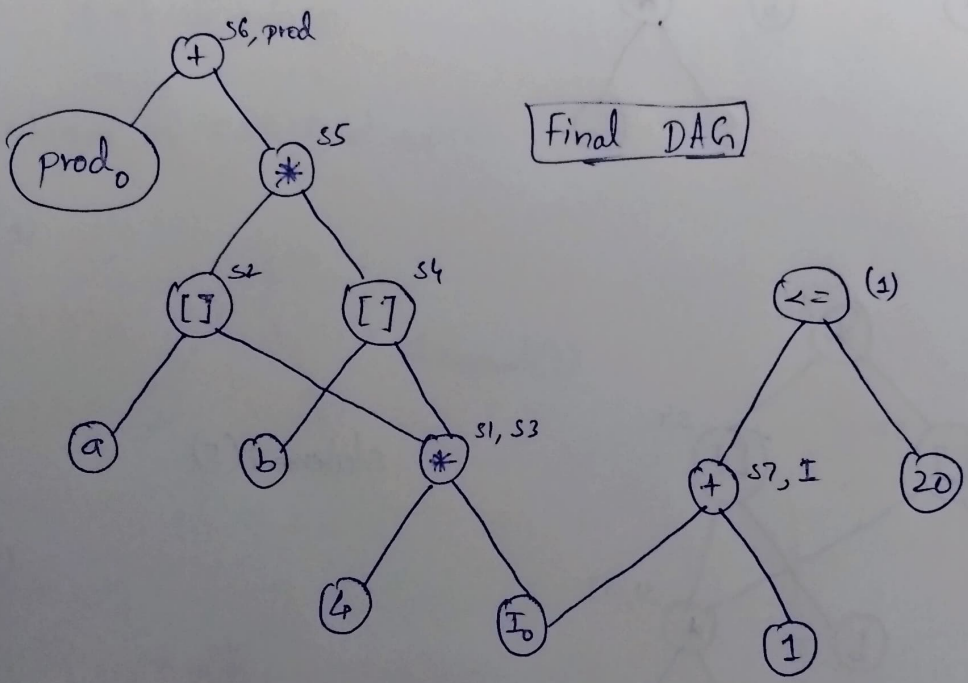
statement (6), attach identifier prod for statement (7)

(g)



statement (8), attach identifier i for statement (9)

(h)



Final DAG