ASSIGNMENT # 05

(Compiler Construction)

Submitted By:

Muhammad Abdullah	(SP17-BCS-025)
Afaq Masood	(SP17-BCS-034)
Yusra Fatima	(SP17-BCS-063)
Umer Khan Wazir	(SP17-BCS-098)
Abdul Mannan	(FA16-BCS-103)

Submitted to:

Mr. Salman Aslam

Question no # 1

Case 1 :-

```
i = 0
if (i == 1)
{
    a = x + 5
}
```

Optimization technique in this case is "Dead Case Elimination"

Revised code : i=0.

Case 2 :-

```
for (int j=0 ; j<mn ; j++)
{
    x = y+z
    a[j] = 6*j;
}
```

The code optimization technique is code movement

Revised Code :

```
x = y+z
for (int j=0 ; j<n ; j++)
{
    a[j] = 6*j
}
```

Question 1 : part 2:-

Case 3:-

The technique used is common "sub-expression elimination".

Revised Code:-

$S_1 = 4*i$
$S_2 = a[S_1]$
$S_3 = 4*j$
$S_5 = n$
$S_6 = b[S_2] + S_5$

Case 4:-

Technique used is "hoisting"

Revised Code:-

t = 20 * PI
DO F I = 1, 100
ARRAY (I) = t * I
END DO.

Question 1 part 3:-

Case 5:-

The technique is "machine idiom"

```
i = i+1  →  i++
i = i-1  →  i--
```

Case 6:-

The technique is "redundant instruction elimination"

```
MOV  R0, index      value of index → R0.
SHL  R0             double value in R0.
MOV  LA[R0], 6      contant 6 → address of a + R0.
```

Case 7:-

The optimization technique is "loop unrolling"

```
int  i = 0
color. map [n+i] = i.
  i++.
color  map [n+i] = i
  i++
color  map [n+i] = i
```

Question 1 part 4:-

Case 8:-

The technique is algebraic identities.

```
u = u+0 = u
u = u*1 = u
```

So, use 'u'.

Case 9:-

Optimization technique is "code motion"

```
t = limit - 2
while (i <= t)
do
    { loop code }
```

Case 10:-

The optimization is "Strength Reduction"

$B = A \times A.$

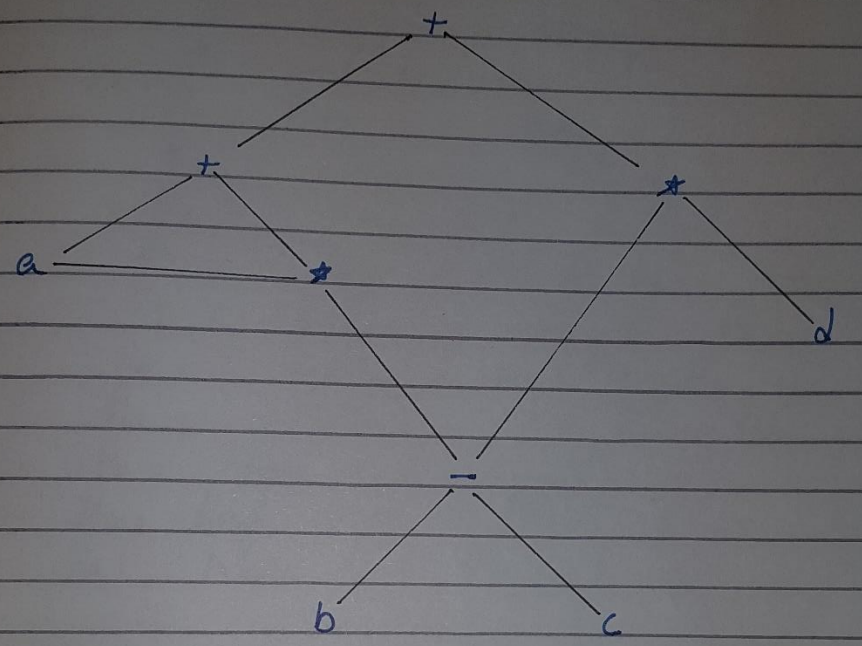
Question 2 part 1:-

i) $E \rightarrow E + T \mid E - T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id \mid num.$

Product	Semantic Rule
$E \rightarrow E_1 + T$	E.node = new node ('+', E_1.node, T.node)
$E \rightarrow E_1 - T$	E.node = new node ('-', E_1.node, T.node)
$E \rightarrow T$	E.node = T.node
$T \rightarrow T_1 * F$	T.node = new node ('*', T_1.node, F.node)
$T \rightarrow F$	T.node = F.node
$F \rightarrow (E)$	F.node = E.node
$F \rightarrow id$	F.node = new leaf (id, id.entry)
$F \rightarrow num$	F.node = new leaf

Question 2, part 2:-

(ii) Directed Acyclic Graph:-



(iii) The leaf for 'a' has two parents because 'a' appears twice in the expression. Moreover, the two occurrences of the common subexpression ~~are~~ ~~repre~~ 'b', 'c' are represented by one node, the node labelled '-'. That node b has two parents, representing its two uses in the subexpression a * (b-c) and (b-c) * d. Even though b & c appear twice in the complete expression.

Question 2, part 3:-

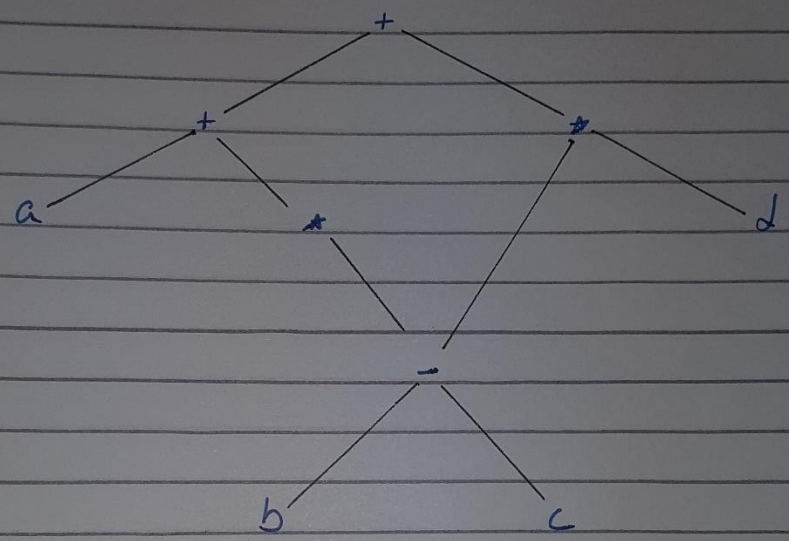
Steps:-

P_1 = leaf (id, entry - a)
P_2 = leaf (id, entry - a) = P_1
P_3 = leaf (id, entry - b)
P_4 = leaf (id, entry - c)
P_5 = Node ('-', P_3, P_4)
P_6 = Node ('*', P_1, P_5)
P_7 = Node ('+', P_1, P_6)
P_8 = leaf (id, entry - b) = P_3
P_9 = leaf (id, entry - c) = P_4
P_{10} = Node ('-', P_3, P_4) = P_5
P_{11} = leaf (id, entry - id)
P_{12} = leaf ('*', P_5, P_{11})
P_{13} = Node ('+', P_7, P_{12})

The node created by the previous call is returned.

Question 42 part 4:-

(iv) DAG :-



Three address Code :-

$t_1 = b - c$

$t_2 = a * t_1$

$t_3 = a + t_2$

$t_4 = t_1 * d$

$t_5 = t_3 + t_4$

Question # 3

a) 2+3+4+5

Three Address Code

$t_1 = 2+3$
$t_2 = t_1+4$
$t_3 = t_2+5$

P code

ide 2
ide 3
adi
ide 4
adi
ide 5
adi

b) 2+(3+(4+5))

Three Address code

$t_1 = 4+5$
$t_2 = 3+t_1$
$t_3 = 2+t_2$

P code

ide 4
ide 5
adi
ide 3
adi
ide 2
adi

c) a*b + a*b*c

$t_1 = a * b$
$t_2 = t_1 * c$
$t_3 = t_1 * t_2$

P code

ida a
ida b
mpi
sto
ida
id b
mpi
ida c
mpi
iod a
api

Question 4

a) $(x=y=2)+3*(x=4)$

Three address code

$x = 2$
$y = 2$
$x = 4$

$t_1 = 3 * 4$
$t_2 = 2 + t_1$

P code

```
ida x
ida y
ida 2
Stn
 idc 3
 ida x
 ida 4
Stn
 mpi
 adi
```

b) $a[a[i]] = b[i=2]$

Question #5

a) $a = b^* - c + b^* - c$.

Three address code:

$t_1 = \text{uminus } c$
$t_2 = b^* t_1$
$t_3 = t_1 + t_2$
$a = t_3$

.

Quadruples

	OP	arg1	arg2	Results
0	uminus	c		t_1
1	*	b	t_1	t_2
2	+	t_2	t_2	t_3
3	=	t_3		a

Triples

Obs	OP	arg 1	arg2	
0	uminus	c		
1	*	b	(0)	
2	+	(1)	(1)	
3	=	a	(2)	

P-code

```
ida   b
ida   c
ncg   c
mpi
lad   b
shl   b, 2
```

(b) $a = -b * (c+d) / e$

Three address code:

$t_1 = \text{uminus } b$
$t_2 = c + d$
$t_3 = t_2 / c$
$t_4 = t_1 + t_3$
$= t_4.$

Quadruple

	OP	arg1	arg2	Result
0	uminus	b		t_1
1	+	c	d	t_2
2	/	t_2	e	t_3
3	*	t_1	t_3	t_4
4	=	t_4		a.

triple

	OP	arg1	arg2
0	uminus	b	
1	+	c	d
2	/	(1)	e
3	*	(1)	(2)
4	=	a	(3)

P-code

```
lda c        :    lod c
lda d             lda e
api               dpi
ftoa              lod b
lda b             lod c
neg b             mpi
```

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

Three address code

$t_1 = a+b$
$t_2 = -t_1$
$t_3 = c+d$
$t_4 = t_2 * t_3$
$t_5 = t_1 + c$
$t_6 = t_4 + t_5$
$x = t_6$

Quadruple:

	OP	arg1	arg2	Results
0	+	a	b	t_1
1	−	t_1		t_2
2	+	c	d	t_3
3	*	t_2	t_3	t_4
4	+	t_1	c	t_5
5	+	t_4	t_5	t_6
6	=	t_6		x.

Triple

	op	arg1	arg2
0	+	a	b
1	−	(0)	
2	+	c	d
3	*	(1)	(2)
4	+	(2)	c
5	+	(3)	(4)
6	=	(4)	(5)

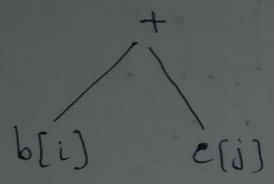
Pcode

```
ida a
ida b
api
sto
neg a
ida c
ida d
api
lod a
lod c
api
lod a
lod c
mpi
loda
api
```

Question # 06 :-

1. $a = b[i] + c[j]$

(a). Syntax Tree :-



(b). Three-address Code :-

$t_1 = b[i]$

$t_2 = c[j]$

$t_3 = t_1 + t_2$

$a = t_3$

(c). Quadruple :-

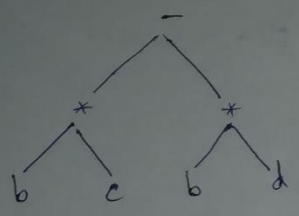
	Op.	arg_1	arg_2	Results
0	[]	b	i	t_1
1	[]	c	j	t_2
2	+	t_1	t_2	
3	=	t_3		a

(d). Triple :-

	Op	arg_1	arg_2
0	[]	b	i
1	[]	c	j
2	+	(0)	(1)
3	=	a	(2)

2. a[i] = b*c - b*d.

(a). Syntax Tree :-



(b). Three-address Code :-

$t_1 = b * c$

$t_2 = b * d$

$t_3 = t_1 - t_2$

$t_4 = a[i]$

$t_4 = t_3$

(c). Quadruple :-

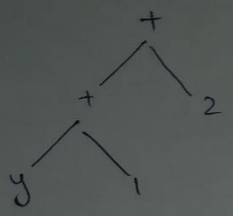
	Op	arg_1	arg_2	Results
0	*	b	c	t_1
1	*	b	d	t_2
2	-	t_1	t_2	t_3
3	[]	a	i	t_4
4	=	t_3		t_4

(d). Triple :-

	Op	arg_1	arg_2
0	*	b	c
1	*	b	d
2	-	(0)	(1)
3	[]	a	i
4	=	(3)	(2)

3. $x = f(y+1) + 2$.

(a). Syntax Tree :-



(b). Three-address Code :-

$t_1 = y + 1$, Param(t_1)
t_2 = call f(t_1)
$t_3 = t_2 + 2$
$x = t_3$

(c). Quadruple :-

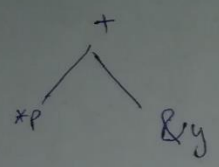
	Op	arg_1	arg_2	Results
0	+	y	1	t_1
1	Param	t_1		
2	call	t_1	1	t_2
3	+	t_2	2	t_3
4	=	t_3		x

(d). Triples :-

	Op	arg_1	arg_2
0	+	y	1
1	Param	(0)	
2	Call	(0)	1
3	+	(2)	2
4	=	x	(3)

4. x = *p + &y

(a). Syntax Tree :-



(b). Three-address Code :-

$t_1 = *p$

$t_2 = \&y$

$t_3 = t_1 + t_2$

$x = t_3$

(c). Quadruple :-

	Op	arg_1	arg_2	Results
0	*	p		t_1
1	&	y		t_2
2	+	t_1	t_2	t_3
3	=	t_3		x

(d). Triples :-

	Op	arg_1	arg_2
0	*	p	
1	&	y	
2	+	(0)	(1)
3	=	(x)	(2)

Question #07 :-

Three - address Code :-

```
read x
t1 = x > 0
if-false t1 goto t1
fact = 1
label t2
t2 = fact * x
fact = t2
t3 = x - 1
x = t3
t4 = x == 0
If-false t4 goto t2
write fact
label L1
halt
```

Pcode :-

```
lda x
rdi
lod x
lde 0
grt
fjp L1
lda fact
lde 1
sto
lab t2
ida fact
lod fact
lod x
mpi
sto
lda x
lod x
lde 1
sbi
sto
load x
lde 0
equ
fjp t2
lod x
mpi
sto
lda x
load fact
wri
lab t1
stp
```

Question # 08 :-

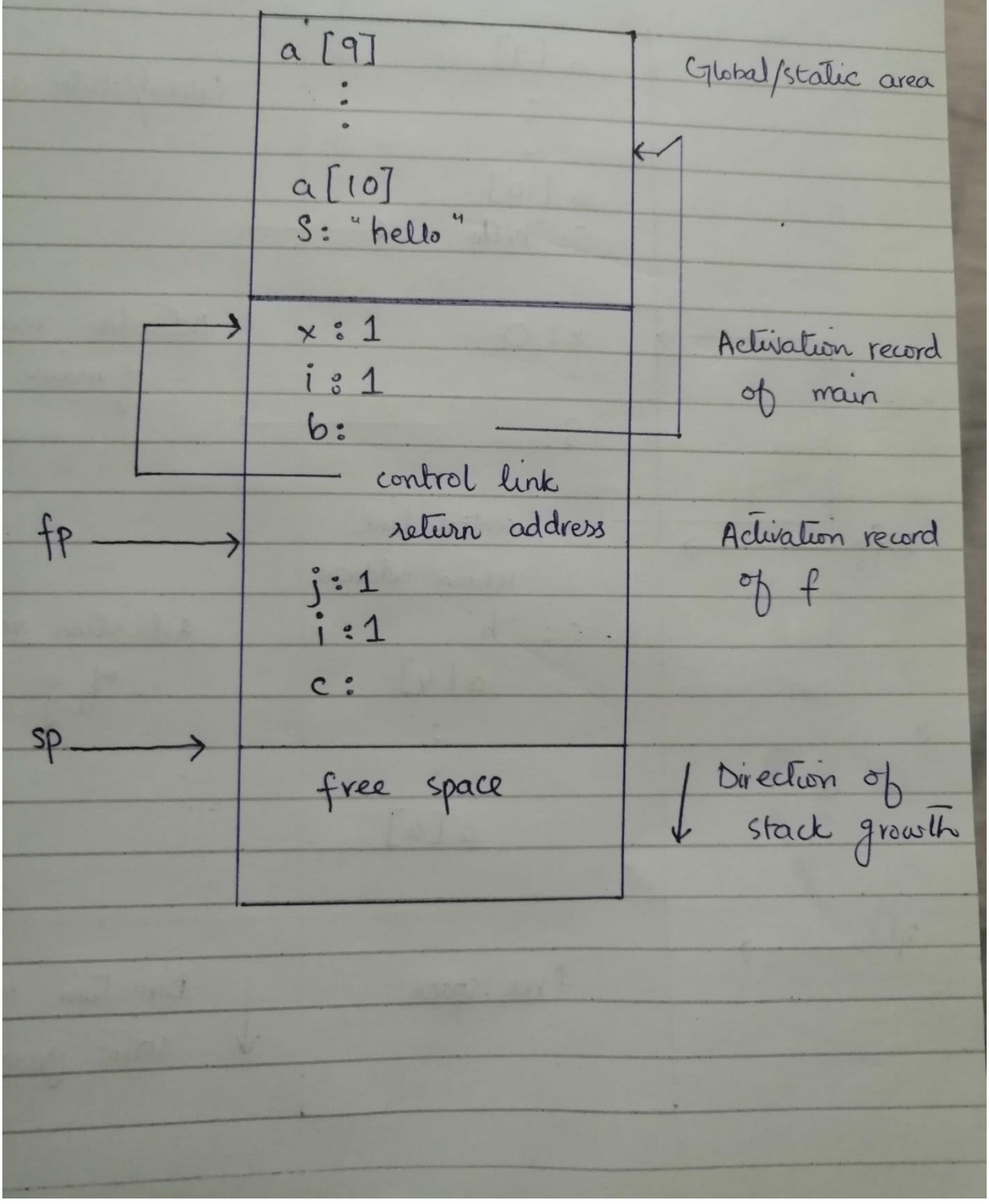
Three-address Code:-

```
Read x
Read v
t1 = v = 0
if false t1 goto L1
  Label L2
  temp = v
  t2 = v1 * v
  t3 = u | t2
  t4 = u - t3
  u = temp
  t4 = r = 0
  if-false t4 goto L2
    write u
  Label L1
    v = 0
    halt
```

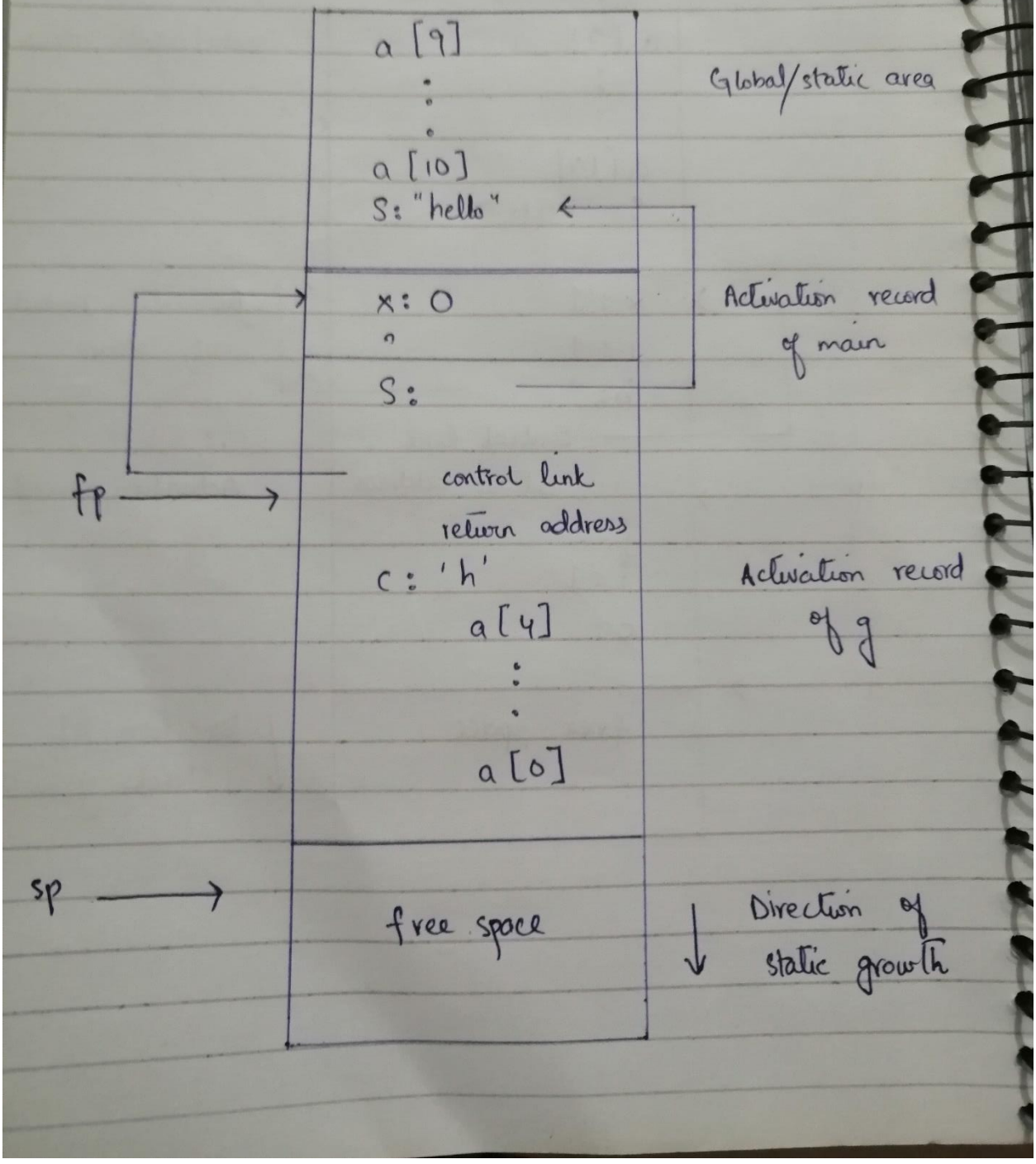
Pcode:-

```
lda x
rdi
lda v
rdi v
ide 0
eq v
fjp L1
Lab L2
lda temp
lod v
equ
lda v
mpi
lda u
lod v
dpi
lda u
lod u
spi
lda u
sto
lda v
lod temp
equ
lod v
ldc 0
equ
fjp L2
wri
Lab L1
stp.
```

Q9 Part (A)

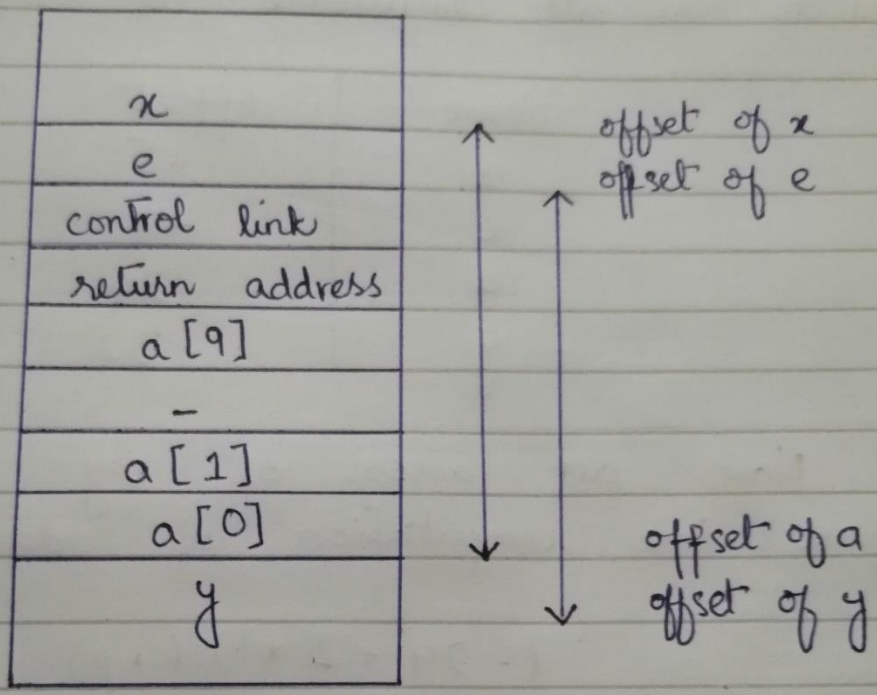

a [9]
a[10]
S: "hello"
Global/static area
x : 1
i : 1
b:
Activation record of main
control link
return address
fp
j : 1
i : 1
c :
Activation record of f
sp
free space
Direction of stack growth

Q9- Part-(b)


a [9]
a [10]
S: "hello"
Global/static area
x: 0
S:
Activation record of main
fp
control link
return address
c: 'h'
a [4]
a [0]
Activation record of g
sp
free space
Direction of static growth

Q10

a) Activation Record



Q10 b)

We would have following offset values which are all computable at compile time.

Name	offset
x	+5
c	+4
a	-24
y	-32

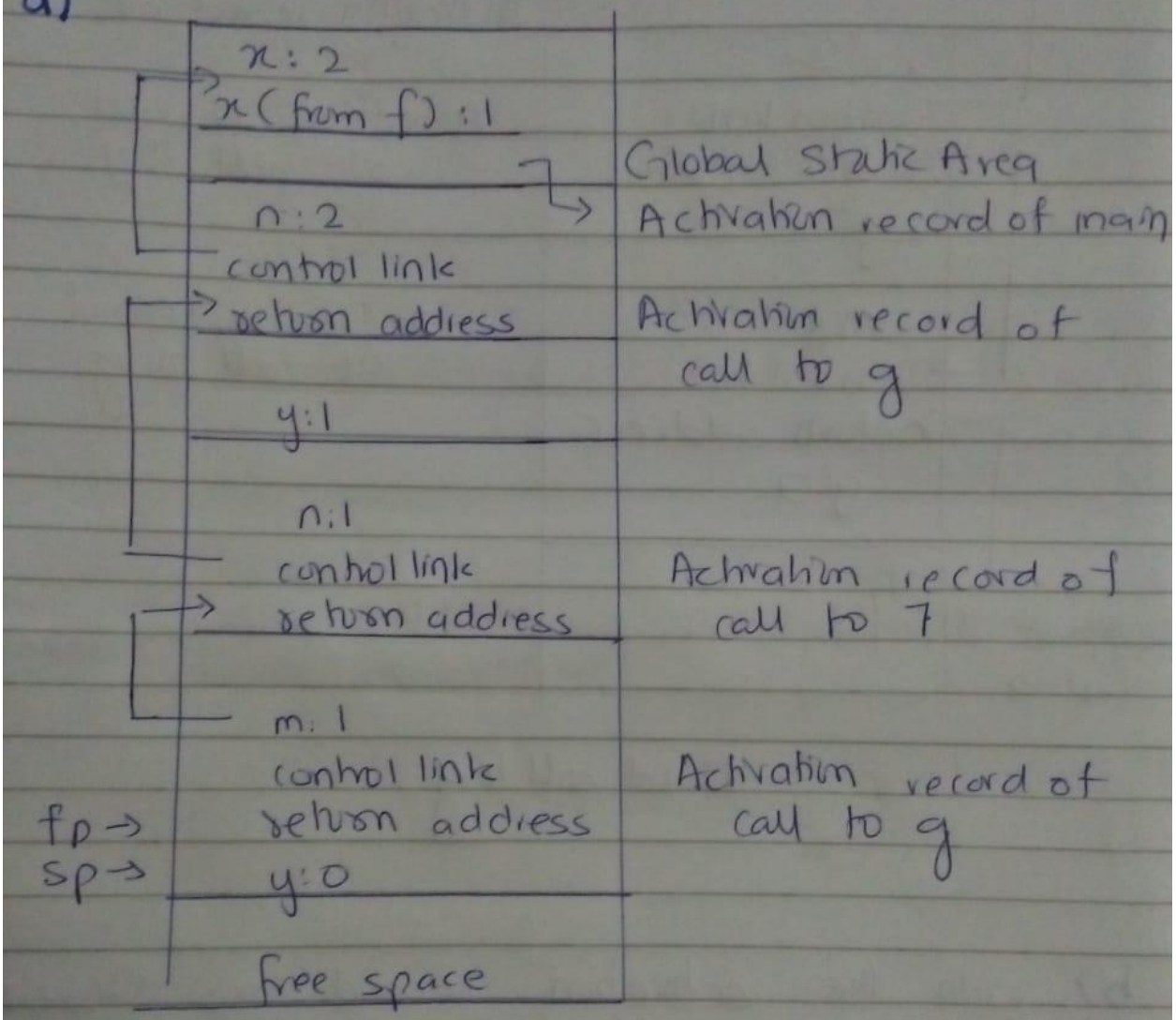
Now, an access of soy a[i] would require the computation of address.

$$(-24 + 2 * 1)\ (+p)$$

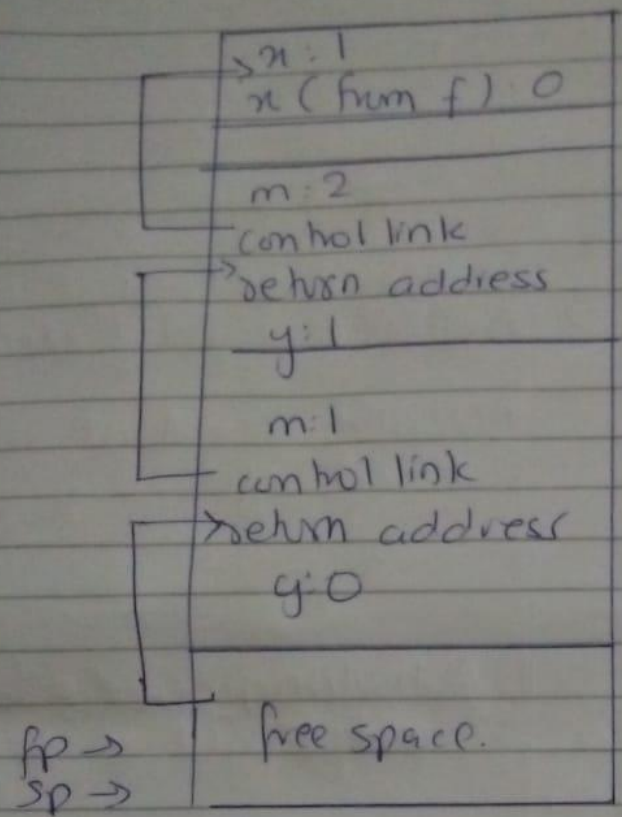
Here factor of 2 in the product $2 \times i$ is the factor.

Q NO 11:

a)



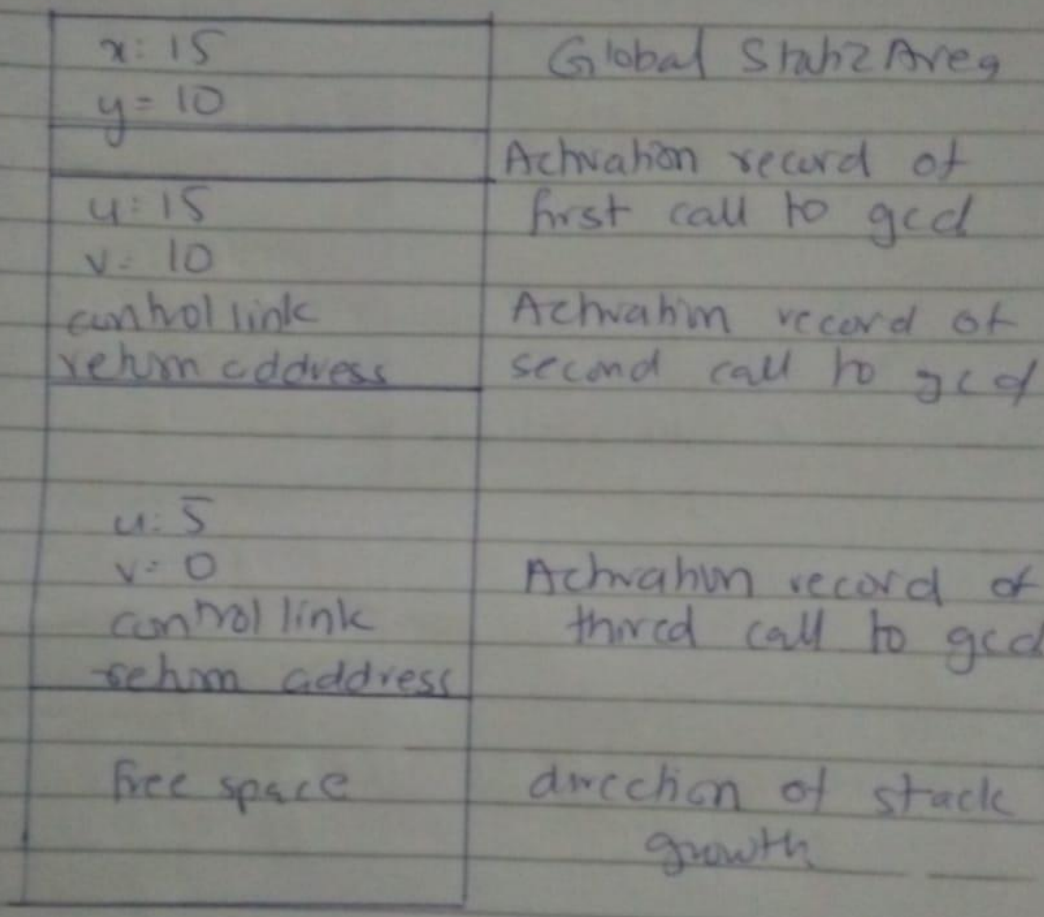
Second call to g



third call to g

b) With the activation record as shown we have offset = +4 and y offset = -6 and reference to many can be written in machine code, as 4(fp) and -6(fp)

Question 12:


x: 15
y: 10
Global Static Area
Activation record of first call to gcd
u: 15
v: 10
control link
return address
Activation record of second call to gcd
u: 5
v: 0
control link
return address
Activation record of third call to gcd
free space
direction of stack growth