Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Compiler Theory: CS31003
3rd year CSE, 5th Semester

Laboratory Quiz - 3 : Symbol Table *Marks: 15*
Date: November 12, 2020

1. A symbol table is implemented using hash table. What will be the entry in the symbol table for the following declaration? [1]

   `static int x;`

   a) <x, int, static>

   b) <x, static, int>

   c) <x, int>

   d) <int, static, x>

   Answer: a)

2. Consider the below grammar for ternary operator. [1]

   E -> E1 N1 ? M1 E2 N2 : M2 E3
   M -> ϵ
   N -> ϵ

   where E is an arithmetic expression. E1 should be of type **bool** instead of **integer** in the ternary operator. Choose the translation rule for the ternary operator grammar considering **int** to **bool** type conversion.

   a)
   ```
   {
     E.loc = gentemp();
     E.type = E2.type;
     emit(E.loc ’=’ E3.loc);
     l = makelist(nextinstr);
     emit(goto .... );
     backpatch(N2.nextlist, nextinstr);
     emit(E.loc ’=’ E2.loc);
     l = merge(l, makelist(nextinstr));
     emit(goto .... );
     backpatch(N2.nextlist, nextinstr);
   ```

```
convInt2Bool(E1);
backpatch(E1.truelist, M2.instr);
backpatch(E1.falselist, M1.instr);
backpatch(l, nextinstr);
}
```

b)
```
{
E.loc = gentemp();
E.type = E2.type;
emit(E.loc '=' E3.loc);
l = makelist(nextinstr);
emit(goto .... );
backpatch(N2.nextlist, nextinstr);
emit(E.loc '=' E2.loc);
l = merge(l, makelist(nextinstr));
emit(goto .... );
backpatch(N1.nextlist, nextinstr);
convInt2Bool(E1);
backpatch(E1.truelist, M1.instr);
backpatch(E1.falselist, M2.instr);
backpatch(l, nextinstr);
}
```

c)
```
{
E.loc = gentemp();
E.type = E2.type;
l = makelist(nextinstr);
emit(E.loc '=' E3.loc);
emit(goto .... );
emit(E.loc '=' E2.loc);
l = merge(l, makelist(nextinstr));
emit(goto .... );
backpatch(N2.nextlist, nextinstr);
convInt2Bool(E1);
backpatch(E2.truelist, M1.instr);
backpatch(E2.falselist, M2.instr);
backpatch(l, nextinstr);
}
```

d) None of the options are correct.

Answer: b)
Source: Module 5 PPT.

3. Consider the below three address code.

```
1)  i = 1
2)  j = 1
3)  t1 = 10 * i
4)  t2 = t1 + j
5)  t3 = 8 * t2
6)  t4 = t3 - 88
7)  a[t4] = 0.0
8)  j = j + 1
9)  if j <= 10 goto (3)
10) i = i + 1
11) if i <= 10 goto (2)
12) i = 1
13) t5 = i - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) i = i + 1
17) if i <= 10 goto (13)
```

Identify the valid leader instructions (First instruction of each basic block) for the given three-address code. [1]

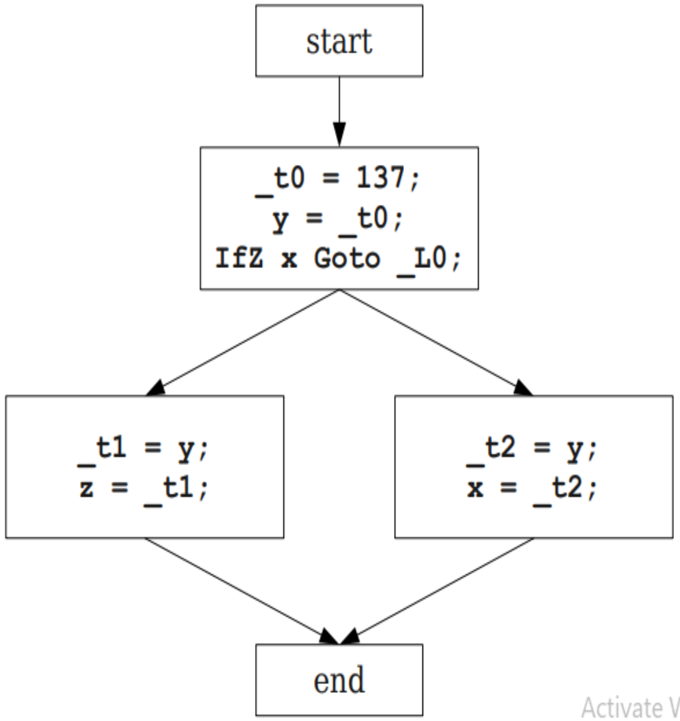
(a) 1, 2, 3, 9, 11, 17
(b) 1, 3, 8, 10, 13, 16
(c) 1, 2, 3, 10, 12, 13
(d) 1, 2, 3, 10, 12, 16

Answer: c)

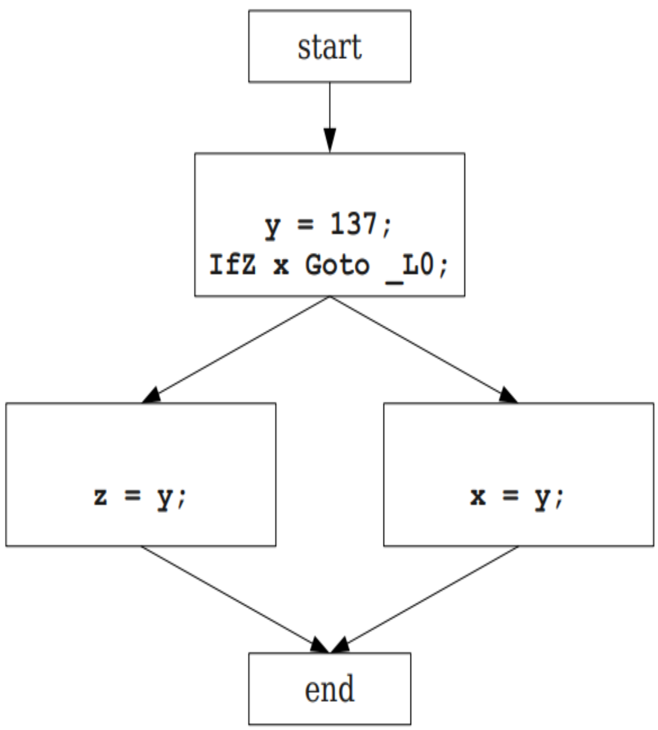
4. Consider the below code segment and corresponding control flow graph. [1]

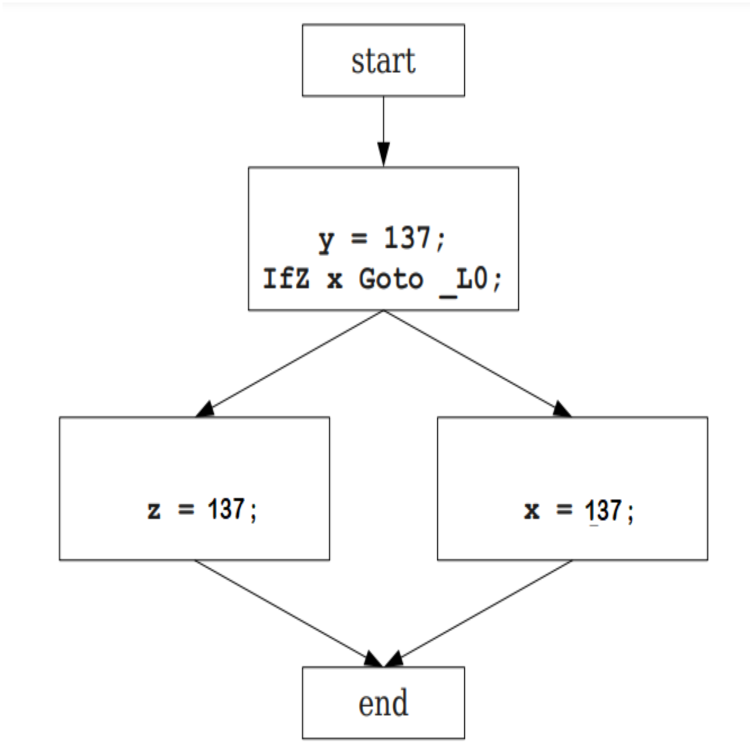
```
int main() {
    int x;
    int y;
    int z;

    y = 137;
    if (x == 0)
        z = y;
    else
        x = y;
}
```

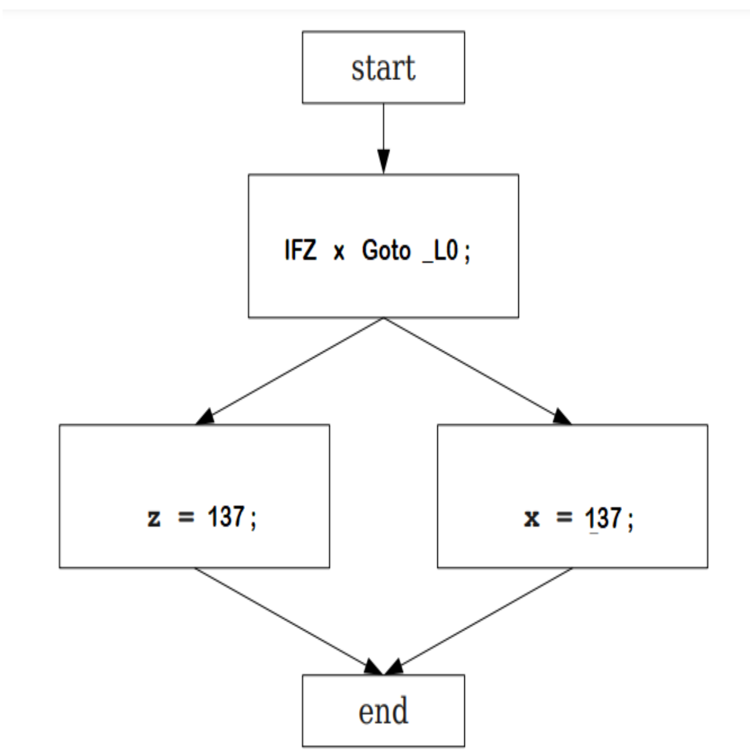


What will be the optimized CFG after doing local optimization and global optimization?

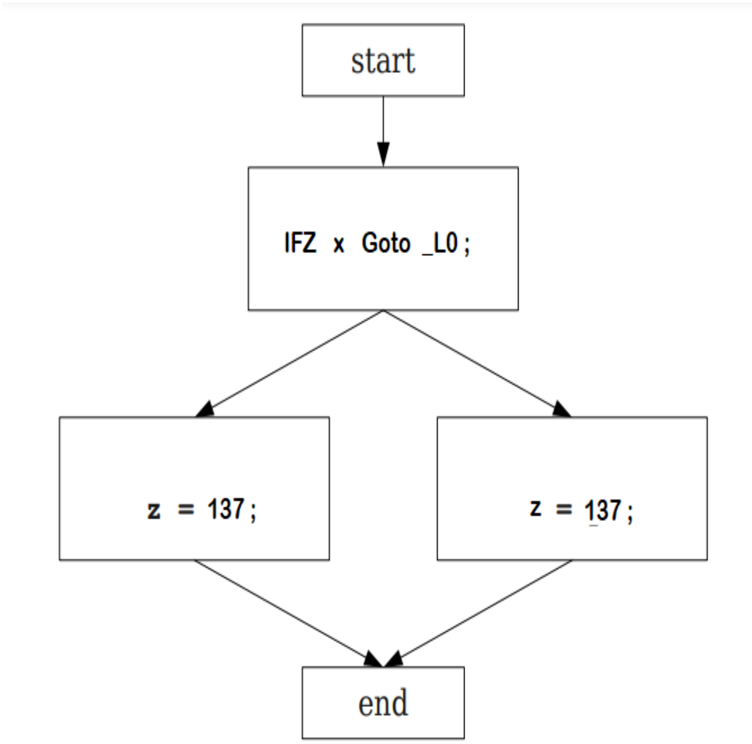
(a) 


start
y = 137;
IfZ x Goto _L0;
z = 137;
x = 137;
end

(b)


start
IFZ x Goto _L0;
z = 137;
x = 137;
end

(c)



(d)

Answer: b)

5. Consider the below x86 assembly code. [1]

```
CONST SEGMENT
__real@400b333333333333 DQ
0400b333333333333r   ; 3.4
__real@4004000000000000 DQ
04004000000000000r   ; 2.5
CONST ENDS

_TEXT SEGMENT
_z$ = -8 ; size = 8
_x$ = 8  ; size = 8
_y$ = 16 ; size = 8

push ebp
mov  ebp, esp
sub  esp, 8
mov  DWORD PTR [ebp-8], 0xccccccccH
mov  DWORD PTR [ebp-4], 0xccccccccH

fld   QWORD PTR __real@4004000000000000
fstp  QWORD PTR _x$[ebp]
fld   QWORD PTR __real@400b333333333333
fstp  QWORD PTR _y$[ebp]
fadd  QWORD PTR _x$[ebp]
```

```
fstp  QWORD PTR _z$[ebp]
fld   QWORD PTR _z$[ebp]
```

What is the return value (in decimal) of the above procedure call?

a) 5.9
b) 0.9
c) -0.9
d) 0.0

Answer: a)

6. Consider below x86 assembly code instructions. [1]

```
a) FINIT
b) FTST
c) FLDPI
d) FILD
e) FCHS
```

and instruction descriptions

```
i) Data transfer instructions.
ii) Arithmetic instructions.
iii) Comparison instructions.
iv) Control instructions.
v) Load constant instructions.
```

Match instructions according to their descriptions.

(a) (a-iv), (b-iii), (c-ii), (d-i), (e-v)
(b) (a-iv), (b-i), (c-v), (d-iii), (e-ii)
(c) (a-ii), (b-i), (c-v), (d-i), (e-iii)
(d) (a-iv), (b-iii), (c-v), (d-i), (e-ii)

Answer: d)
Source: https://docs.oracle.com/cd/E18752_01/html/817-5477/ennbz.html♯eoizl

7. Consider the below C function.

```
char greater (int *a, int *b)
{
    if (*a>*b)
        return 1;
    else
        return 0;
}
```

What will be the corresponding x86 assembly program for the above C function? [1]

a)
```
greater:
movl (%rdi), %ecx
movl (%rsi), %edx
cmpl %ecx, %edx
jle les
movb $1, %al
ret
les:
movb $0, %al
ret
```

b)
```
greater:
movl (%rdi), %ecx
movl (%rsi), %edx
cmpl %ecx, %edx
jl les
movb $0, %al
ret
les:
movb $1, %al
ret
```

c)
```
greater:
movl (%rdi), %ecx
movl (%rsi), %edx
cmpl %ecx, %edx
jg gtr
movb $0, %al
ret
gtr:
movb $1, %al
ret
```

d)
```
greater:
movl (%rdi), %ecx
movl (%rsi), %edx
cmpl %ecx, %edx
jmp gtr
movb $0, %al
ret
gtr:
movb $1, %al
ret
```

Answer : a), c)
Source: https://www.cs.purdue.edu/homes/cs250/LectureNotes/AssemblyExamples.pdf

8. Which of the following **mov** instruction definition is/are incorrect in a modern x86 compatible processor? [1]

   a) *mov eax, [esi-4]*

b) *mov edx, [esi+4*ebx]*

c) *mov eax, [ebx-ecx]*

d) *mov [eax+esi+edi], ebx*

Answer : c), d)
source: https://www.cs.dartmouth.edu/ sergey/cs258/tiny-guide-to-x86-assembly.pdf

9. Consider the below for loop.

```
for(i=0;i<n;++i){
    \\code
}
```

Write x86 assembly code for the above code segment. Consider all local variables used in for loop are already declared in x86 assembly code. Also consider the value of n is 3. [2]

a)
```
mov   DWORD PTR _i$[ebp], 0
jmp   SHORT $LN3@main

$LN2@main:
mov   eax, DWORD PTR _i$[ebp]
add   eax, 1
mov   DWORD PTR _i$[ebp], eax

$LN3@main:
mov   ecx, DWORD PTR _i$[ebp]
cmp   ecx, DWORD PTR _n$[ebp]
jge   SHORT $LN1@main

; Code for for loop body

jmp + SHORT $LN2@main

$LN1@main:
```

b)
```
mov   DWORD PTR _i$[ebp], 0
jmp   SHORT $LN3@main

$LN1@main:
mov   eax, DWORD PTR _i$[ebp]
add   eax, 1
mov   DWORD PTR _i$[ebp], eax

$LN2@main:
mov   ecx, DWORD PTR _i$[ebp]
cmp   ecx, DWORD PTR _n$[ebp]
jge   SHORT $LN1@main

; Code for for loop body
```

```
jmp + SHORT $LN2@main

$LN3@main:
```

c)
```
mov   DWORD PTR _i$[ebp], 0
jge   SHORT $LN3@main

$LN2@main:
mov   eax, DWORD PTR _i$[ebp]
add   eax, 1
mov   DWORD PTR _i$[ebp], eax

$LN3@main:
mov   ecx, DWORD PTR _i$[ebp]
cmp   ecx, DWORD PTR _n$[ebp]
jge   SHORT $LN1@main

; Code for for loop body

jge + SHORT $LN2@main

$LN1@main:
```

d)
```
mov   DWORD PTR _i$[ebp], 0
jmp   SHORT $LN3@main

$LN2@main:
mov   eax, DWORD PTR _i$[ebp]
add   eax, 1
mov   DWORD PTR _i$[ebp], eax

$LN3@main:
mov   ecx, DWORD PTR _i$[ebp]
cmp   ecx, DWORD PTR _n$[ebp]
jmp   SHORT $LN1@main

; Code for for loop body

jmp + SHORT $LN2@main

$LN1@main:
```

Answer : d)

10. Consider the following C code segment. [2]

```
int a = 10;
int fun(int x){  \\function scope fun
    int t, u;
```

```
    t = x;
    {  \\un-named block scope fun_1
        int p, q, t;
        p = a;
        t = 4;
        {  \\un-named block scope fun_1_1
            int p;
            p = 5;
        }
        q = p;
    }
    return u = t;
}
```

Corresponding symbol table for the above code block after nested block flattening is given below.

ST.f(): ST.f.parent = ST.glb					
x	int	param	4	0	null
t	int	local	4	4	null
u	int	local	4	8	null
p#1	int	blk-local	4	0	null
q#2	int	blk-local	4	4	null
t#3	int	blk-local	4	8	null
p#4	int	blk-local	4	0	null

Where,
p♯1 is for variable p in scope fun_1.
q♯2 is for variable q in scope fun_1.
t♯3 is for variable t in scope fun_1.
p♯4 is for variable p in scope fun_1_1.

What will be the x86 assembly instructions generated from the above code and symbol table?

a)
```
mov    eax, DWORD PTR _x$[ebp]
mov    DWORD PTR _t$[ebp], eax
mov    ecx, DWORD PTR _a
mov    DWORD PTR _p$4[ebp], ecx
mov    DWORD PTR _t$2[ebp], 4
mov    DWORD PTR _p$1[ebp], 5
mov    edx, DWORD PTR _p$4[ebp]
mov    DWORD PTR _q$3[ebp], edx
mov    eax, DWORD PTR _t$[ebp]
mov    DWORD PTR _u$[ebp], eax
mov    eax, DWORD PTR _u$[ebp]
mov    esp, ebp
pop    ebp
ret    0
```

b)
```
mov    eax, DWORD PTR _x$[ebp]
```

```
mov    DWORD PTR _t$[ebp], eax
mov    eax, DWORD PTR _a
mov    DWORD PTR _p$[ebp], eax
mov    DWORD PTR _t$[ebp], 4
mov    DWORD PTR _p$[ebp], 5
mov    eax, DWORD PTR _p$[ebp]
mov    DWORD PTR _q$[ebp], eax
mov    eax, DWORD PTR _t$[ebp]
mov    DWORD PTR _u$[ebp], eax
mov    eax, DWORD PTR _u$[ebp]
mov    esp, ebp
pop    ebp
ret    0
```

c)
```
mov    eax, DWORD PTR _x$[ebp]
mov    DWORD PTR _t$[ebp], eax
mov    ecx, DWORD PTR _a
mov    DWORD PTR _p$4[ebp], ecx
mov    DWORD PTR _t$2[ebp], 5
mov    eax, DWORD PTR _t$[ebp]
mov    DWORD PTR _u$[ebp], eax
mov    DWORD PTR _p$1[ebp], 4
mov    edx, DWORD PTR _p$4[ebp]
mov    DWORD PTR _q$3[ebp], edx
mov    eax, DWORD PTR _u$[ebp]
mov    esp, ebp
pop    ebp
ret    0
```

d) None is correct.

Answer: a)

11. The **data** section in an assembly program is used for [1]

   a) Declaring identifiers used in program.
   b) Declaring initialized data or constant which do not change in runtime.
   c) Declaring function prototype.
   d) Declaring parameters of user-defined function.

   Answer : b)

12. _________ segment of an assembly program is used for keeping the actual assembly code instructions. (Fill in the blank with exact answer which we use while writing any assembly code) [1]
   Answer: _TEXT

13. State whether TRUE or FALSE. The reason behind an uninitialized variable of a C program is initialized with 0xccccccccH in x86 assembly code is "if we accidentally try to execute this data as instruction, it will lead to a system call for breakpoint as 'cc' is the opcode for "INT

3" instruction". [1]
Answer: TRUE