CS-1319-1: Programming Language Design and Implementation (PLDI)

Assignment 6

Gautam Ahuja

18 December 2023

```c
// IO Library header -- as defined in Assignment 5
int printStr(char *s);
int printInt(int n);
int readInt(int *eP);
// Swap two integers
void swap(int *a, int *b) {
    int t;
    t = *a;
    *a = *b;
    *b = t;
}
// Works for 3 digit numbers only
int kt(int n) {
    int p; // Previous number
    int d1; // Largest digit
    int d2; // Second largest digit
    int d3; // Smallest digit
    int m; // Next number
    p = n; // Remember current number
    // Extract digits in sorted order
    d1 = n % 10;
    n = n / 10;
    d2 = n % 10;
    n = n / 10;
    if (d1 < d2)
        swap(&d1, &d2);
    d3 = n % 10;
    if (d2 < d3) {
        swap(&d2, &d3);
        if (d1 < d2)
            swap(&d1, &d2);
    }
    // Check digits to debug
    printInt(d1);
    printInt(d2);
    printInt(d3);
    printStr("\n");
    // Compute the diff of largest and smallest
    // three digit numbers with the given digits
    m = (d1 - d3) * 99;
    // Check for the fixed point
    if (m == p)
        return m; // Should return 495 if n != 0
    else
        return kt(m); // Continue search for fixed point
}
int main() {
int n;
int m;
while (1) {
    n = readInt(0);
    m = kt(n);
    printStr("Constant = ");
    printInt(m);
    printStr("\n");
}
    return 0;
}
```

Q1

A. Global Symbol Table

Global Symbol table is as follows:

ST.glb				Parent: *Null*
Name	**Type**	**Category**	**Size**	**Offset**
printStr	ptr(char) -> int	function	0	ST.printStr
printInt	char -> int	function	0	ST.printInt
readInt	ptr(int) -> int	function	0	ST.readInt
swap	ptr(int) x ptr(int) -> void	function	0	ST.swap
kt	int -> int	function	0	ST.kt
main	void -> int	function	0	ST.main

B. Array of Quads

We generate array of quads starting at 100:

1. swap

```
100: t01 = *a
101: t = t01
102: t02 = *b
103: *a = t02
104: t03 = t
105: *b = t03
106: return
```

2. kt

```
offset _s1 "\n"

100: p = n
101: t01 = 10
102: t02 = n % t01
103: d1 = t02
104: t03 = 10
105: t04 = n / t03
106: n = t04
107: t05 = 10
108: t06 = n % t05
109: d2 = t06
110: t07 = 10
111: t08 = n / t07
112: n = t08
113: if d1 < d2 goto 115
114: goto 120
115: t09 = &d1
116: t10 = &d2
117: param t09
118: param t10
119: call swap, 2
120: t11 = 10
121: t12 = n % t11
122: d3 = t12
123: if d2 < d3 goto 125
124: goto 137
125: t13 = &d2
126: t14 = &d3
127: param t13
128: param t14
129: call swap, 2
```

```
130: if d1 < d2 goto 132
131: goto 137
132: t15 = &d1
133: t16 = &d2
134: param t15
135: param t16
136: call swap, 2
137: param d1
138: t17 = call printInt, 1
139: param d2
140: t18 = call printInt, 1
141: param d3
142: t19 = call printInt, 1
143: param _s1
144: t20 = call printStr, 1
145: t21 = d1 - d3
146: t22 = 99
147: t23 = t21 * t22
148: m = t23
149: if m == p goto 151
150: goto 152
151: return m
152: param m
153: t24 = call kt, 1
154: return t24
```

3. `main`

```
offset _s1 "\n"
offset _s2 "Constant = "
100: t01 = 1
101: if t01 == 0 goto 117
102: goto 103
103: t02 = 0
104: param t02
105: t03 = call readInt, 1
106: n = t03
107: param n
108: t04 = call kt, 1
109: m = t04
110: param _s2
111: t05 = call printStr, 1
112: param m
113: t06 = call printInt, 1
114: param _s1
115: t07 = call printStr, 1
116: goto 101
117: t08 = 0
118: return t08
```

C. Symbol Tables for `swap`, `kt`, `main`

Symbol Tables

1. `swap`

ST.swap				Parent: *ST.glb*
Name	**Type**	**Category**	**Size**	**Offset**
b	`ptr(int)`	param	4	+12
a	`ptr(int)`	param	4	+8
t	`int`	local	4	-4
t01 .. t03	`int`	temp	4	-8 ... -16

2. `kt`

ST.kt				Parent: *ST.glb*
Name	**Type**	**Category**	**Size**	**Offset**
n	`int`	param	4	+8
p	`int`	local	4	-4
d1	`int`	local	4	-8
d2	`int`	local	4	-12
d3	`int`	local	4	-16
m	`int`	local	4	-20
t01 ... t24	`int`	temp	4	-24 ... -116

3. `main`

ST.main				Parent: *ST.glb*
Name	**Type**	**Category**	**Size**	**Offset**
n	`int`	local	4	-4
m	`int`	local	4	-8
t01 .. t08	`int`	temp	4	-12 ... -40

4. Offset for constants

```
offset _s1 "\n"
offset _s2 "Constant = "
```

They will look as follows in memory:

Text
Constant
Static
Heap
...
Stack

The strings will be present in the constant segment of memory at a particular address. Those addresses are marked by the variable names `_s1` and `_s2` during run time and then referred when used by a function.

D. IO Library

The IO library function are essentially generating the required assembly code directly to produce a `syscall` for a input-output operation. The symbol table will be as follows:

ST.printStr				Parent: *ST.glb*
Name	**Type**	**Category**	**Size**	**Offset**
s	`ptr(char)`	para	4	+8
t01 ..	`int`	temp	4	-4 ...

ST.printInt				Parent: *ST.glb*
Name	**Type**	**Category**	**Size**	**Offset**
n	`int`	para	4	+8
t01 ..	`int`	temp	4	-4 ...

ST.readInt				Parent: *ST.glb*
Name	**Type**	**Category**	**Size**	**Offset**
eP	`ptr(int)`	para	4	+8
t01 ..	`int`	temp	4	-4 ...

Q2 - Peephole Optimization

Before optimization:

```
offset _s1 "\n"

100: p = n
101: t01 = 10                    ; def-use propagation; Deadcode
102: t02 = n % t01               ; Copy: t01 -> 10
103: d1 = t02                    ; value copy propagation: t02 -> n % 10
104: t03 = 10                    ; def-use propagation; Deadcode
105: t04 = n / t03               ; Copy: t03 -> 10
106: n = t04                     ; value copy propagation: t04 -> n / 10
107: t05 = 10                    ; def-use propagation; Deadcode
108: t06 = n % t05               ; Copy: t05 -> 10
109: d2 = t06                    ; value copy propagation: t06 -> n % 10
110: t07 = 10                    ; def-use propagation; Deadcode
111: t08 = n / t07               ; Copy: t07 -> 10
112: n = t08                     ; value copy propagation: t08 -> n / 10
113: if d1 < d2 goto 116         ; jump-over-jump
114: goto 120                    ; dead-code
115: t09 = &d1
116: t10 = &d2
117: param t09
118: param t10
119: call swap, 2
120: t11 = 10                    ; def-use propagation; Deadcode
121: t12 = n % t11               ; Copy: t11 -> 10
122: d3 = t12                    ; value copy propagation: t12 -> n % 10
123: if d2 < d3 goto 125         ; jump-over-jump
124: goto 137                    ; dead-code
125: t13 = &d2
126: t14 = &d3
127: param t13
128: param t14
129: call swap, 2
130: if d1 < d2 goto 132         ; jump-over-jump
131: goto 137                    ; dead-code
132: t15 = &d1
133: t16 = &d2
134: param t15
135: param t16
136: call swap, 2
137: param d1
138: t17 = call printInt, 1
139: param d2
140: t18 = call printInt, 1
141: param d3
142: t19 = call printInt, 1
143: param _s1
144: t20 = call printStr, 1
145: t21 = d1 - d3
146: t22 = 99                    ; def-use propagation; Deadcode
147: t23 = t21 * t22             ; Copy: t22 -> 99
148: m = t23                     ; value copy propagation: t23 -> t21 * 99
149: if m == p goto 151          ; jump-over-jump
150: goto 152                    ; dead-code
151: return m
152: param m
153: t24 = call kt, 1
154: return t24
```

After Optimization:

```
offset _s1 "\n"

100: p = n
101: d1 = n % 10
102: n = n / 10
103: d2 = n % 10
104: n = n / 10
105: if d1 > d2 goto 111
106: t09 = &d1
107: t10 = &d2
108: param t09
109: param t10
110: call swap, 2
111: d3 = n % 10
112: if d2 > d3 goto 124
113: t13 = &d2
114: t14 = &d3
115: param t13
116: param t14
117: call swap, 2
118: if d1 > d2 goto 124
119: t15 = &d1
120: t16 = &d2
121: param t15
122: param t16
123: call swap, 2
124: param d1
125: t17 = call printInt, 1
126: param d2
127: t18 = call printInt, 1
128: param d3
129: t19 = call printInt, 1
130: param _s1
131: t20 = call printStr, 1
132: t21 = d1 - d3
133: m = t21 * 99
134: if m != p goto 136
135: return m
136: param m
137: t24 = call kt, 1
138: return t24
```

After Cleaning:

```
offset _s1 "\n"

100: p = n
101: d1 = n % 10
102: n = n / 10
103: d2 = n % 10
104: n = n / 10
105: if d1 > d2 goto 111
106: t01 = &d1
107: t02 = &d2
108: param t01
109: param t02
110: call swap, 2
111: d3 = n % 10
112: if d2 > d3 goto 124
113: t03 = &d2
```

```
114: t04 = &d3
115: param t03
116: param t04
117: call swap, 2
118: if d1 > d2 goto 124
119: t05 = &d1
120: t06 = &d2
121: param t05
122: param t06
123: call swap, 2
124: param d1
125: t07 = call printInt, 1
126: param d2
127: t08 = call printInt, 1
128: param d3
129: t09 = call printInt, 1
130: param _s1
131: t10 = call printStr, 1
132: t11 = d1 - d3
133: m = t11 * 99
134: if m != p goto 136
135: return m
136: param m
137: t12 = call kt, 1
138: return t12
```

Q3 - Control Flow Graph

A. Leaders

The following are the leader quads in `kt` (cleaned).

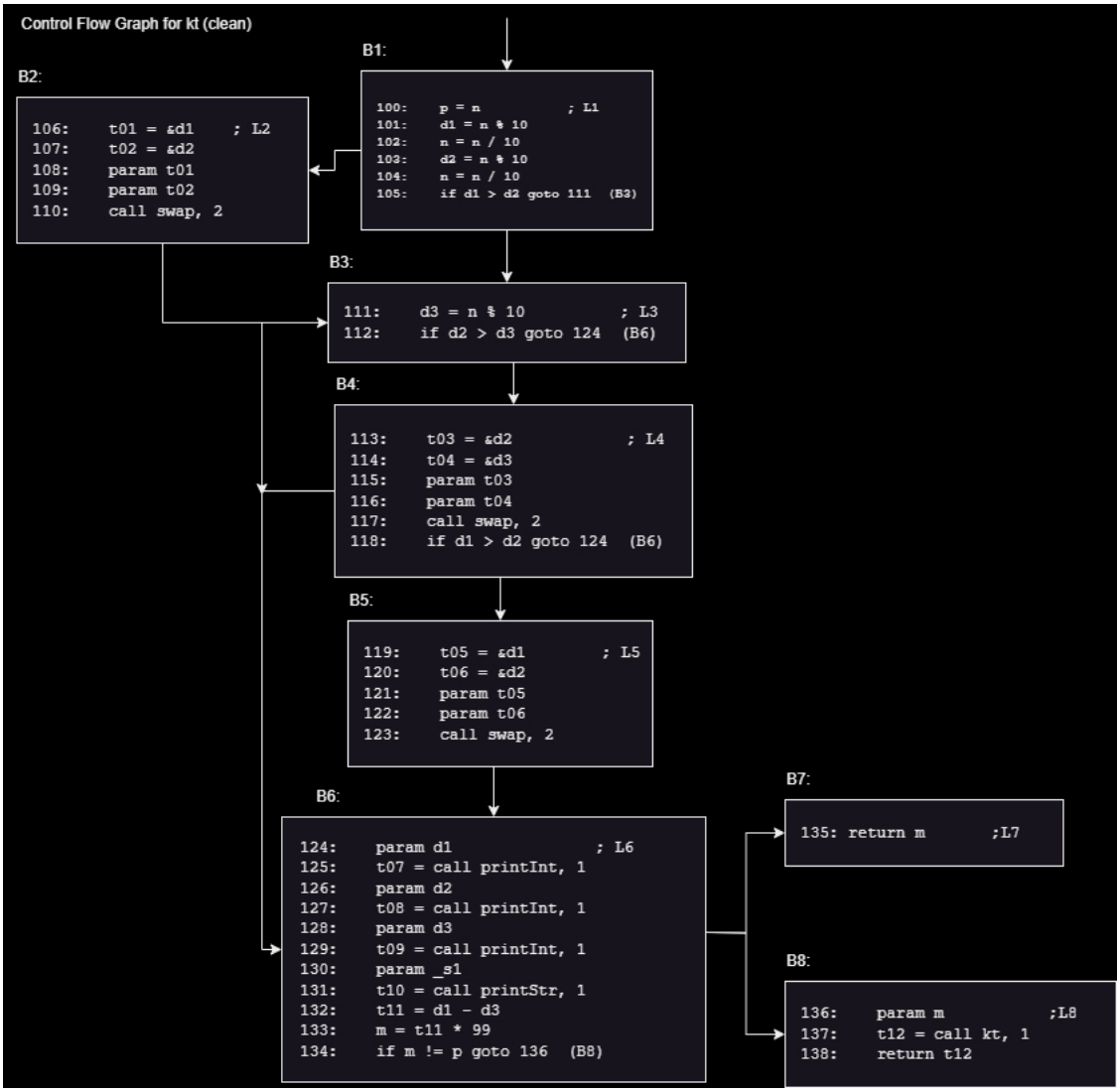
1. `100:  p = n`
2. `106:  t01 = &d1`
3. `111:  d3 = n% 10`
4. `113:  t03 = &d2`
5. `119:  t05 = &d1`
6. `124:  param d1`
7. `135:  return m`
8. `136:  param m`

B. Basic Blocks

```
offset _s1 "\n"
; Block 1 (B1)
100: p = n                          ; Leader 1 (L1) - By rule 1
101: d1 = n % 10
102: n = n / 10
103: d2 = n % 10
104: n = n / 10
105: if d1 > d2 goto 111  (B3)

; Block 2 (B2)
106: t01 = &d1                      ; Leader 2 (L2) - By rule 3
107: t02 = &d2
108: param t01
109: param t02
110: call swap, 2

; Block 3 (B3)
111: d3 = n % 10                    ; Leader 3 (L3) - By rule 2
112: if d2 > d3 goto 124  (B6)

; Block 4 (B4)
113: t03 = &d2                      ; Leader 4 (L4) - By rule 3
114: t04 = &d3
115: param t03
116: param t04
117: call swap, 2
118: if d1 > d2 goto 124  (B6)

; Block 5 (B5)
119: t05 = &d1                      ; Leader 5 (L5) - By rule 3
120: t06 = &d2
121: param t05
122: param t06
123: call swap, 2

; Block 6 (B6)
124: param d1                       ; Leader 6 (L6) - By rule 2
125: t07 = call printInt, 1
126: param d2
127: t08 = call printInt, 1
128: param d3
```

```
129: t09 = call printInt, 1
130: param _s1
131: t10 = call printStr, 1
132: t11 = d1 - d3
133: m = t11 * 99
134: if m != p goto 136  (B8)

; Block 7 (B7)
135: return m                          ; Leader 7 (L7) - By rule 3

; Block 8 (B8)
136: param m                           ; Leader 8 (L8) - By rule 2
137: t12 = call kt, 1
138: return t12
```

The Flow Graph Looks as follows:



Q4 - Optimize CFG

We optimize CFG for every block using Value Numbering method.

1. **Block 1**

```
; Block 1 (B1)
100: p = n                                  ; Leader 1 (L1) - By rule 1
101: d1 = n % 10
102: n = n / 10
103: d2 = n % 10
104: n = n / 10
105: if d1 > d2 goto 111  (B3)
```

Value	**Table**	**Name**		**Table**	**Hash**	**Table**
Name	*Value*	*Index*	*Name*	*Value*	*Expression*	*Value*
n	1	1	n	-	-	-
p	2	2	p	-	-	-
d1	3	3	d1	-	n%10	3
n	4	4	n	-	n/10	4
d2	5	5	d2	-	n%10	5
n	6	6	n	-	n/10	6

No Optimizations in B1.

2. **Block 2**

```
; Block 2 (B2)
106: t01 = &d1                              ; Leader 2 (L2) - By rule 3
107: t02 = &d2
108: param t01
109: param t02
110: call swap, 2
```

Value	**Table**	**Name**		**Table**	**Hash**	**Table**
Name	*Value*	*Index*	*Name*	*Value*	*Expression*	*Value*
d1	1	1	d1	-	-	-
t01	2	2	t01	-	-	-
d2	3	3	d2	-	-	-
t02	4	4	t02	-	-	-

No Optimizations in B2.

3. **Block 3**

```
; Block 3 (B3)
111: d3 = n % 10 ; Leader 3 (L3) - By rule 2
112: if d2 > d3 goto 124 (B6)
```

Value	**Table**	**Name**		**Table**	**Hash**	**Table**
Name	*Value*	*Index*	*Name*	*Value*	*Expression*	*Value*
n	1	1	n	-	-	-
d3	2	2	d3	-	n%10	2

No Optimizations in B3.

4. **Block 4**

```
; Block 4 (B4)
113: t03 = &d2 ; Leader 4 (L4) - By rule 3
114: t04 = &d3
115: param t03
116: param t04
117: call swap, 2
118: if d1 > d2 goto 124 (B6)
```

Value	**Table**	**Name**		**Table**	**Hash**	**Table**
Name	*Value*	*Index*	*Name*	*Value*	*Expression*	*Value*
d2	1	1	d2	-	-	-
t03	2	2	t03	-	-	-
d3	3	3	d3	-	-	-
t04	4	4	t04	-	-	-

No Optimizations in B4.

5. **Block 5**

```
; Block 5 (B5)
119: t05 = &d1 ; Leader 5 (L5) - By rule 3
120: t06 = &d2
121: param t05
122: param t06
123: call swap, 2
```

Value	**Table**	**Name**		**Table**	**Hash**	**Table**
Name	*Value*	*Index*	*Name*	*Value*	*Expression*	*Value*
d1	1	1	d1	-	-	-
t05	2	2	t05	-	-	-
d2	3	3	d2	-	-	-
t06	4	4	t06	-	-	-

No Optimizations in B5.

6. **Block 6**

```
; Block 6 (B6)
124: param d1 ; Leader 6 (L6) - By rule 2
125: t07 = call printInt, 1
126: param d2
127: t08 = call printInt, 1
128: param d3
129: t09 = call printInt, 1
130: param _s1
131: t10 = call printStr, 1
132: t11 = d1 - d3
133: m = t11 * 99
134: if m != p goto 136 (B8)
```

Value	**Table**	**Name**		**Table**	**Hash**	**Table**
Name	*Value*	*Index*	*Name*	*Value*	*Expression*	*Value*
d1	1	1	d1	-	-	-
t07	2	2	t07	-	-	-
d2	3	3	d2	-	-	-
t08	4	4	t08	-	-	-
d3	5	5	d3	-	-	-
t09	6	6	t09	-	-	-
t10	7	7	t10	-	-	-
t11	8	8	t11	-	d1-d2	8
m	9	9	m	-	t11*99	9

No Optimizations in B6.

7. **Block 7**

```
; Block 7 (B7)
135: return m ; Leader 7 (L7) - By rule 3
```

Value	Table	Name		Table	Hash	Table
Name	*Value*	*Index*	*Name*	*Value*	*Expression*	*Value*
m	1	1	m	-	-	-

No Optimizations in B7.

8. **Block 8**

```
; Block 8 (B8)
136: param m ; Leader 8 (L8) - By rule 2
137: t12 = call kt, 1
138: return t12
```

Value	Table	Name		Table	Hash	Table
Name	*Value*	*Index*	*Name*	*Value*	*Expression*	*Value*
m	1	1	m	-	-	-
t12	2	1	t12	-	-	-

No Optimizations in B9.

The quad array remains the same as there were no optimizations in local block.

```
offset _s1 "\n"
; Block 1 (B1)
100: p = n                              ; Leader 1 (L1) - By rule 1
101: d1 = n % 10
102: n = n / 10
103: d2 = n % 10
104: n = n / 10
105: if d1 > d2 goto 111  (B3)

; Block 2 (B2)
106: t01 = &d1                          ; Leader 2 (L2) - By rule 3
107: t02 = &d2
108: param t01
109: param t02
110: call swap, 2

; Block 3 (B3)
111: d3 = n % 10                        ; Leader 3 (L3) - By rule 2
112: if d2 > d3 goto 124  (B6)

; Block 4 (B4)
113: t03 = &d2                          ; Leader 4 (L4) - By rule 3
114: t04 = &d3
115: param t03
116: param t04
117: call swap, 2
118: if d1 > d2 goto 124  (B6)

; Block 5 (B5)
119: t05 = &d1                          ; Leader 5 (L5) - By rule 3
120: t06 = &d2
```

```
121: param t05
122: param t06
123: call swap, 2

; Block 6 (B6)
124: param d1                          ; Leader 6 (L6) - By rule 2
125: t07 = call printInt, 1
126: param d2
127: t08 = call printInt, 1
128: param d3
129: t09 = call printInt, 1
130: param _s1
131: t10 = call printStr, 1
132: t11 = d1 - d3
133: m = t11 * 99
134: if m != p goto 136  (B8)

; Block 7 (B7)
135: return m                          ; Leader 7 (L7) - By rule 3

; Block 8 (B8)
136: param m                           ; Leader 8 (L8) - By rule 2
137: t12 = call kt, 1
138: return t12
```

Q5 - Live Variables

Liveliness analysis of variables in `kt`

```
offset _s1 "\n"
; B1                                    <- {n}
100: p = n                              <- {p, n}
101: d1 = n % 10                        <- {d1, p, n}
102: n = n / 10                         <- {d1, p, n}
103: d2 = n % 10                        <- {d2, d1, p, n}
104: n = n / 10                         <- {d2, d1, p, n}
105: if d1 > d2 goto 111  (B3)          <- {d2, d1, p, n}

; B2                                    <- {d2, d1, p, n}
106: t01 = &d1                          <- {t01, d2, d1, p, n}
107: t02 = &d2                          <- {t02, t01, d2, d1, p, n}
108: param t01                          <- {t02, d2, d1, p, n}
109: param t02                          <- {d2, d1, p, n}
110: call swap, 2                       <- {d2, d1, p, n}

; B3                                    <- {d2, d1, p, n}
111: d3 = n % 10                        <- {d3, d2, d1, p}
112: if d2 > d3 goto 124  (B6)          <- {d3, d2, d1, p}

; B4                                    <- {d3, d2, d1, p}
113: t03 = &d2                          <- {t03, d3, d2, d1, p}
114: t04 = &d3                          <- {t04, t03, d3, d2, d1, p}
115: param t03                          <- {t04, d3, d2, d1, p}
116: param t04                          <- {d3, d2, d1, p}
117: call swap, 2                       <- {d3, d2, d1, p}
118: if d1 > d2 goto 124  (B6)          <- {d3, d2, d1, p}

; B5                                    <- {d3, d2, d1, p}
119: t05 = &d1                          <- {t05, d3, d2, d1, p}
120: t06 = &d2                          <- {t06, t05, d3, d2, d1, p}
121: param t05                          <- {t06, d3, d2, d1, p}
122: param t06                          <- {d3, d2, d1, p}
123: call swap, 2                       <- {d3, d2, d1, p}

; B6                                    <- {d3, d2, d1, p}
124: param d1                           <- {d3, d2, d1, p}
125: t07 = call printInt, 1             <- {t07, d3, d2, d1, p}
126: param d2                           <- {d3, d1, p}
127: t08 = call printInt, 1             <- {t08, d3, d1, p}
128: param d3                           <- {d3, d1, p}
129: t09 = call printInt, 1             <- {t09, d3, d1, p}
130: param _s1                          <- {d3, d1, p}
131: t10 = call printStr, 1             <- {t10, d3, d1, p}
132: t11 = d1 - d3                      <- {t11, p}
133: m = t11 * 99                       <- {m, p}
134: if m != p goto 136  (B8)           <- {m}

; B7                                    <- {m}
135: return m                           <- {}

; B8                                    <- {m}
136: param m                            <- {}
137: t12 = call kt, 1                   <- {t12}
138: return t12                         <- {}
```

Liveliness analysis of variables in `main` (optimized)

```
offset _s1 "\n"
offset _s2 "Constant = "
100: t01 = 1                          <- {t01}
101: if t01 != 0 goto 103             <- {}
102: t02 = 0                          <- {t02}
103: param t02                        <- {}
104: n = call readInt, 1              <- {n}
105: param n                          <- {}
106: m = call kt, 1                   <- {m}
107: param _s2                        <- {m}
108: t05 = call printStr, 1           <- {t05, m}
109: param m                          <- {}
110: t06 = call printInt, 1           <- {t06}
111: param _s1                        <- {}
112: t07 = call printStr, 1           <- {t07}
113: goto 101                         <- {}
114: t08 = 0                          <- {t08}
115: return t08                       <- {}
```

Liveliness analysis of variables in `swap` (optimized)

```
100: t01 = *a                         <- {t01, b}
101: t = t01                          <- {t, b}
102: t02 = *b                         <- {t02, t}
103: *a = t02                         <- {a, t}
104: t03 = t                          <- {t03}
105: *b = t03                         <- {b}
106: return                           <- {}
```

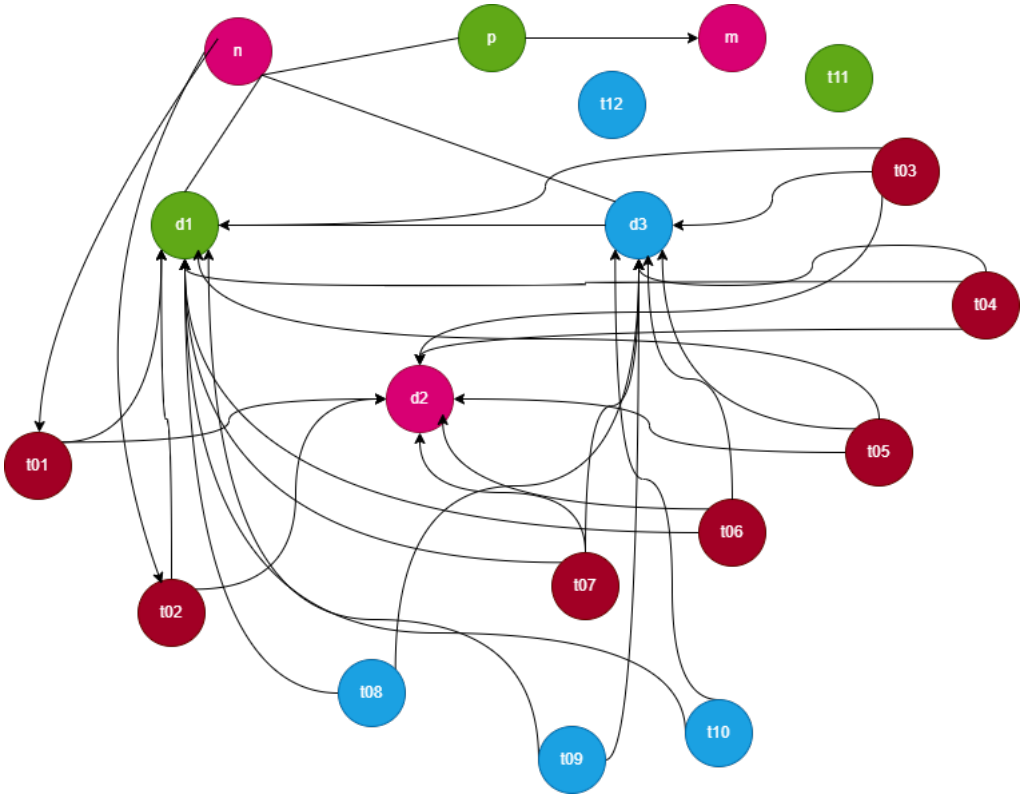
Q6 - Target Code Generation

To fit the code for `kt` in four variables we spill one variable – `r0` – and modify the code as follows. The new liveliness analysis will be:

```
    offset _s1 "\n"
    ; B1                                    <- {n}
    100: p = n                              <- {p, n}
    101: store p, pa                        <- {n}
    101: d1 = n % 10                        <- {d1, n}
    102: n = n / 10                         <- {d1, n}
    103: d2 = n % 10                        <- {d2, d1, n}
    104: n = n / 10                         <- {d2, d1, n}
    105: if d1 > d2 goto 111  (B3)          <- {d2, d1, n}

    ; B2                                    <- {d2, d1, n}
    106: t01 = &d1                          <- {t01, d2, d1, n}
    107: param t01                          <- {d2, d1, n}
    108: t02 = &d2                          <- {t02, d2, d1, n}
    109: param t02                          <- {d2, d1, n}
    110: call swap, 2                       <- {d2, d1, n}

    ; B3                                    <- {d2, d1, n}
    111: d3 = n % 10                        <- {d3, d2, d1}
    112: if d2 > d3 goto 124  (B6)          <- {d3, d2, d1}

    ; B4                                    <- {d3, d2, d1}
    113: t03 = &d2                          <- {t03, d3, d2, d1}
    114: param t03                          <- {d3, d2, d1}
    115: t04 = &d3                          <- {t04, d3, d2, d1}
    117: call swap, 2                       <- {d3, d2, d1}
    116: param t04                          <- {d3, d2, d1}
    118: if d1 > d2 goto 124  (B6)          <- {d3, d2, d1}

    ; B5                                    <- {d3, d2, d1}
    119: t05 = &d1                          <- {t05, d3, d2, d1}
    120: param t05                          <- {d3, d2, d1}
    121: t06 = &d2                          <- {t06, d3, d2, d1}
    122: param t06                          <- {d3, d2, d1}
    123: call swap, 2                       <- {d3, d2, d1}

    ; B6                                    <- {d3, d2, d1}
    124: param d1                           <- {d3, d2, d1}
    125: t07 = call printInt, 1             <- {t07, d3, d2, d1}
    126: param d2                           <- {d3, d1}
    127: t08 = call printInt, 1             <- {t08, d3, d1}
    128: param d3                           <- {d3, d1}
    129: t09 = call printInt, 1             <- {t09, d3, d1}
    130: param _s1                          <- {d3, d1}
    131: t10 = call printStr, 1             <- {t10, d3, d1}
    132: t11 = d1 - d3                      <- {t11}
    133: m = t11 * 99                       <- {m}
    134: p = load pa                        <- {m, p}
    134: if m != p goto 136  (B8)           <- {m}

    ; B7                                    <- {m}
    135: return m                           <- {}

    ; B8                                    <- {m}
    136: param m                            <- {}
    137: t12 = call kt, 1                   <- {t12}
```

```
138: return t12                                    <- {}
```

We perform the graph colouring method to find out what registers to assign to the variables. The RIG (After colouring the graph) is as follows:



The assembly instructions will be as follows for the function `kt()`:

```
offset _s1 "\n"
; B1
100: r0 = n
101: SW r0, $_pa          ; store word
101: MOD r0, n, $10
102: DIV n, n, $10
103: MOD r2, n, $10
104: DIV n, n, $10
105: GRT r0, r2, goto 111 ; (B3)

; B2
106: LEA r3, r0
107: param r3
108: LEA r3, r2
109: param r3
110: call swap, 2

; B3
111: MOD r1, n, $10
112: GRT r2, r1, goto 124 ;  (B6)

; B4
113: LEA r3, r2
114: param r3
115: LEA r3, r1
117: call swap, 2
116: param r3
118: GRT r0, r2, goto 124 ; (B6)
```

```
; B5
119: LEA r3, r0
120: param r3
121: LEA r3, r2
122: param r3
123: call swap, 2

; B6
124: param r0
125: r3 = call printInt, 1
126: param r2
127: r1 = call printInt, 1
128: param r1
129: r1 = call printInt, 1
130: param _s1
131: r1 = call printStr, 1
132: SUB r0, r0, r1
133: MULT r2, r0, $99
134: LW r0, $_pa                    ; load word
134: NEW r2, r0, goto 136  ; (B8)

; B7
135: return r2

; B8
136: param r2
137: r1 = call kt, 1
138: return r1
```