Register Allocation Exercises

1

Problem 1

Consider the three-address code below:

```
 1:      i = 0;
 2:      a = p1;
 3:      b = i * 4;
 4:      c = a + b;
 5: L1: if (i > 100) goto L2
 6:      c = a + 1;
 7:      i = i + 1;
 8:      b = i * 4;
 9:      e = p1;
10:      if (c <= p1) goto L3
11:      c = e - b;
12:      a = e;
13:      goto L4
14: L3: d = p1;
15:      c = d + b;
16:      a = d;
17: L4: if (i <= 100) goto L1
18: L2: return
```

a) Determine the interference graph assuming a given variable is live at the end of a specific instruction if after the use in that instruction the value is still used elsewhere in the code. You should assume the variable `p1` is defined upon entry of the code as it corresponds to a parameter of the procedure and is used after the return instruction.
b) Determine the minimum number of registers needed (without spilling, of course).
c) Assuming you were short of one register, which register(s) would you spill and at which points in the program would you insert the spill code? Explain the rationale of your choice.
d) Now redo the register allocation using the top-down method in which variables used inside a loop are weighted more that variables outside the loop.

a) Live ranges:

p1: {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}

i: {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}

a: {2,3,4,5,6} + {12,13,16,17}

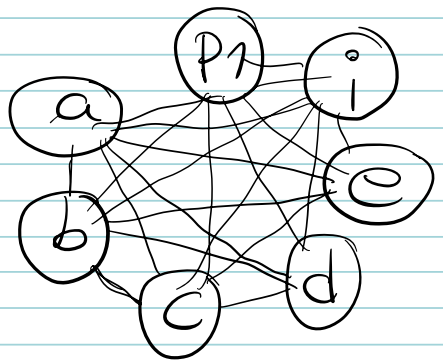
b: {3,4} + {8,9,10,11} + {14,15}

c: {4} + {6,7,8,9,10} + {11} + {15}

d: {14,15,16}

e: {9,10,11,12}

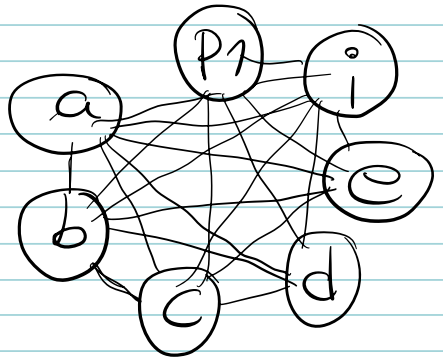
Interference Graph



d e e

são os únicos nodes sem interferência

b)



→ clique de 6

↓

grafo de 6 nodes em que todos se interligam

↓

ex.: {a, b, c, d, i, P1}

ou

{a, b, c, e, i, P1}

↓

Conclusão

São necessárias 6 cores

↓

d e e podem ser pintados com a mesma já que não têm interferência.

c) O melhor processo seria reverer os registos destinados a d e a e, pois são os que possuem menores live ranges.

↓

1 - ver em que linha é definida

2 - é utilizada até ser re-definida?

3 - se sim fazer {init, ..., ..., última utilização}

2

Problem 2

Consider the following 3-address format representation of a computation. Here we have used the Frame Poiner (`$fp`) register to load local variables into temporary variables introduced in the intermediate code generation process. We have also used compile-time offsets to load the values of several local variables as they are located and fixed locations in the current procedure Activation Record (AR).

```
01:    t1 = k * 8
02:    t2 = $fp + offset_A
03:    t3 = t1 + t2
04:    t4 = *t3
05:    t5 = t4 * x
06:    t6 = $fp + offset_B
07:    t7 = t6 * 8
08:    t8 = *t7
09:    t9 = t5 + t8
10:    *t7 = t9
11:    k = k + 1
```

Questions:

(a) Using the bottom-up register allocator described in class assign the various temporary variables and variables to actual registers. For the purpose of this section, assume you only have 4 physical registers. At each point when choosing to reuse a register indicate why do you pick each one.

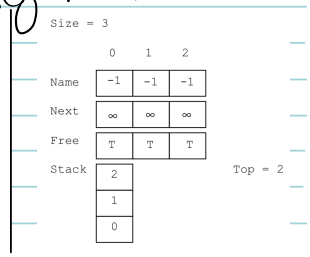
(b) Use the graph-coloring based algorithm for doing register allocation instead. In this section we are to explore the use of interference webs for different definitions of interference as mentioned in class.

a. In the first web use the definition that two variables interfere if there is at least one instruction in which they participate. Derive the interference web between variables using this definition.
b. In the second definition there is no interference if the two webs either do not intersect at all or if they do intersect at a single instruction the web that ends at that instruction participates as the argument of the instruction and the web that begins at that instruction corresponds to the destination value of the instruction.
c. For both definitions determine the minimum number of required registers.
d. Using the graph-coloring heuristic described in class determine the number of required registers for each of the two interference definitions. Why do they differ, or why not?

a)

```
01:    r0 = $fp + offset_K //: r0 ← k; r1 ← empty; r2 ← empty; r3 ← empty;
02:    r1 = r0 * 8         //: r0 ← k; r1 ← t1; r2 ← empty; r3 ← empty;
03:    r2 = $fp + offset_A //: r0 ← k; r1 ← t1; r2 ← t2; r3 ← empty;
04:    r3 = r1 + r2        //: r0 ← k; r1 ← t1; r2 ← t2; r3 ← t3;
05:    r1 = (r3)           //: r0 ← k; r1 ← t4; r2 ← t2; r3 ← t3;
06:    r2 = FP + offset_X  //: r0 ← k; r1 ← t4; r2 ← x; r3 ← t3;
07:    r1 = r1 * r2        //: r0 ← k; r1 ← t5; r2 ← x; r3 ← t3;
08:    r2 = FP + offset_B  //: r0 ← k; r1 ← t5; r2 ← t6; r3 ←t7;
09:    r3 = r2 * 8         //: r0 ← k; r1 ← t5; r2 ← t6; r3 ← t7;
```

Algo that uses this



3

Problem 3

Consider the code fragment depicted below.

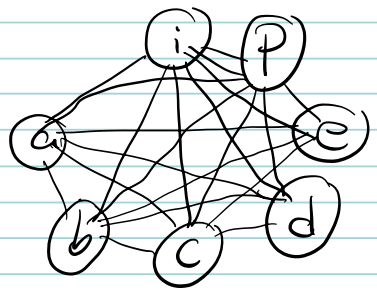
```
01:       i = 0
02:       a = 1
03:       b = i * 4
04:       c = a + b
05: L1:   if i > n goto L2
06:       c = a + 1
07:       i = i + 1
08:       b = i * 4
09:       if c <= p goto L3
10:       e = 1
11:       c = e - b
12:       a = e
13:       goto L4
14: L3:   d = 2
15:       c = d + b
16:       a = d
17: L4:   goto L1
18: L2:   return
```

For this code determine the following:

a) The live ranges for the variables i, a, b, c, d, e, p.
b) The interference graph using the simplest definition of interference where the webs include the live range in terms of the line numbers where an access to a variable (either a read or a write operation) occurs.
c) Determine the minimum number of registers needed (without spilling, of course) using the graph coloring algorithm described in class.
d) Assuming that you were short of one register, which register(s) would you spill and at which points in the program would you insert the spill code? Explain.

a) i : {1 - 17}
a : {2, 3, 4, 5, 6} + {12, 13, 16, 17}
b : {3, 4} + {8, 9, 10, 11, 14, 15}
c : {4} + {6, 7, 8, 9} + {11} + {15}
d : {14, 15, 16}
e : {10, 11, 12}
p : {1 - 17}

b)



Únicos nós sem interferência são d e e.

c) Como 2 nós não têm interferência, podemos pintá-los com a mesma cor. Ficamos com um clique de 6, logo precisamos de 6 cores para pintar o gráfico, o que resulta em 6 registos necessários.

d) Se apenas tivesse 5 registos, removia o nó com menor live range (neste caso seria o nó d ou e, já que os dois possuem o mesmo comprimento de range.

Problem 7. Control-Flow Analysis and Register Allocation

Consider the three-address code below for a procedure with input/output arguments p0 and p1 and using several temporary variables, named `t0` through `t5`.

```
01:      t5 = 0
02:      t0 = p1
03:      t2 = 0
04:      t1 = p2
05:      if (t1 > 0) goto L1
06:      t5 = t0
07:      t0 = 1
08:      t2 = 0
09: L1:  t1 = t0
10:      t4 = t0
11:      if (t4 > t5) goto L2
12:      t4 = t1 + 1
13:      t2 = t2 + 1
14:      goto L1
15: L2:  t0 = p1
16:      t3 = t0
17:      t1 = t2 + t3
18:      ret t1
```

Questions:

For this code determine the following:

a. Basic blocks and the corresponding control-flow graph (CFG) indicating for each basic block the corresponding line numbers of the code above.

b. Dominator tree and the natural loops in this code (if any) along with the corresponding back edge(s).

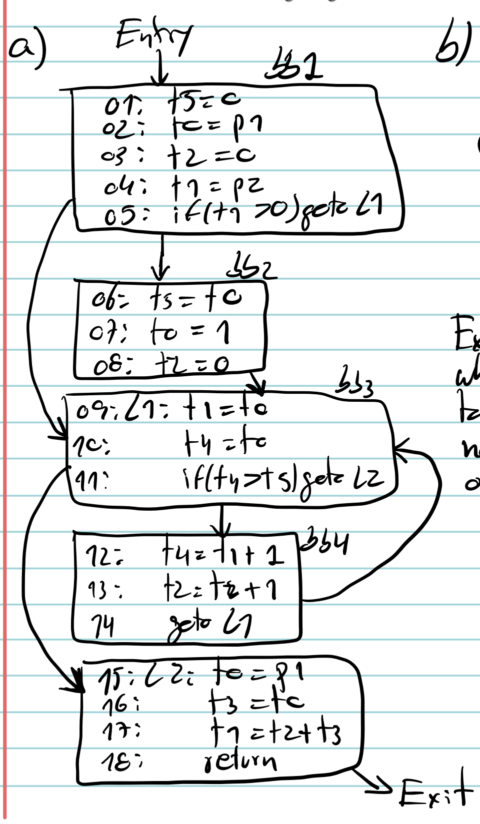
c. Determine the live ranges for the variables `t0`, `t1`, `t2`, `t3`, `t4` and `t5`, the corresponding webs and interference graph, for the refined notion of interference discussed in class. Assume that you do not need registers for the parameters p0 and p1 and assume that on exit of the last basic block (the one ending with the return instruction) `t2` is live but the remainder temporaries are dead on exit of the procedure. In this analysis you should ignore the parameter variables `p0` and `p1`.

d. Can you color the resulting interference graphs with 4 colors? Why not? Suggest a modification to the code, possibly using source-level code transformations, that allows for the coloring with 4 colors. Present a coloring assignment for 4 colors.

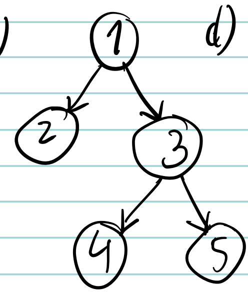
porque ter o goto para e 10 Again.

{2 – 14}

a)



b)



Exists a back-edge 4-3, whose head dominates its tail and, the corresponding natural loop is composed of the basic blocks 3 and 4.

d)
t0: {2,3,4,5,6,9,10} + {7,9,10} + {15,16}
t1: {4,5} + {9,10,11,12} + {17,18}
t2: {3,17}
t3: {16,17}
t4: {10,11} + {12}
t5: {1–5, 9–14} + {6–14}

{1–14}

d) The figure below depicts the interference graph for the temporaries `t0` through `t0` using the refined notion of interference as discussed in class. As it can be seen there is a clique of size 5 (W0a, W1a, W5, W4b and W2), so clearly it cannot be colored using 4 colors.

```
web0a: {02-14}
web0b: {15-16}
web1a: {09-13}
web1b: {17-18}
web2: {02-18}
web3: {16-18}
web4a: {10-11}
web4b: {12}
web5: {01-14}
```