TD11 : Register Allocation - Thuesday, December 7

Exercise 1 Register allocation in a basic block

Consider the following assembly code :

```
Load b -> R1
Load b -> R2
Mult R1 R2 -> R2
Make 4 -> R3
Load a -> R4
Load c -> R5
Mult R4 R5 -> R5
Mult R5 R3 -> R3
Sub  R2 R3 -> R3
```

Question 1 What does this code compute and how many registers does it use ?

Solution : By analysing the state of the register at the end of the program :

Register	Content
R1	b
R2	b^2
R3	$b^2 - 4ac$
R4	a
R5	ac

we can say that it computes $b^2 - 4ac$ using the 5 registers. □

Question 2 Write this code in the SSA form.

Solution :

```
Load b -> V1
Load b -> V2
Mult V1 V2 -> V3
Make 4 -> V4
Load a -> V5
Load c -> V6
Mult V5 V6 -> V7
Mult V4 V7 -> V8
Sub V3 V8 -> V9
```

□

Question 3 Do the register allocation with fixed ordering and a minimum number of registers. Use the greedy algorithm seen in course.

Solution :

The next table represents the *live range* of registers during the execution of the previous code, using the convention that a register dies before its last use :

code	V1	V2	V3	V4	V5	V6	V7	V8	V9
`Load b -> V1`	|								
`Load b -> V2`	|	|							
`Mult V1 V2 -> V3`			|						
`Make 4 -> V4`			|	|				
`Load a -> V5`			|	|	|			
`Load c -> V6`			|	|	|	|		
`Mult V5 V6 -> V7`			|	|		|		
`Mult V4 V7 -> V8`			|					|	
`Sub V3 V8 -> V9`									|

As we can observe, 4 variables are alive at the same time, therefore, we need to use at least 4 registers in order not to spill. The greedy algorithm, working from upper left to bottom right on the above table yields the optimal solution. Moreover, we can remark that the interference graph of a linearized code is an interval graph. As register allocation is a graph coloring problem, this makes register allocation trivially polynomial due to the greedy algorithm. The obtained code is :

```
Load b -> R1
Load b -> R2
Mult R1 R2 -> R1
Make 4 -> R2
Load a -> R3
Load c -> R4
Mult R3 R4 -> R3
Mult R2 R4 -> R4
Sub R1 R4 -> R2
```

□

Exercise 2

We consider the program with the control flow graph given below. We consider that there are no live variables at the output of the program.

Question 4 Give the alive variables for points 1 to 5 in the program.

Solution :

$live(1) = \{a, b, d\}$
$live(2) = \{a, b, c, d, f\}$
$live(3) = \{a, b, d\}$
$live(4) = \{e\}$
$live(5) = \{a, b, k, d\}$

□

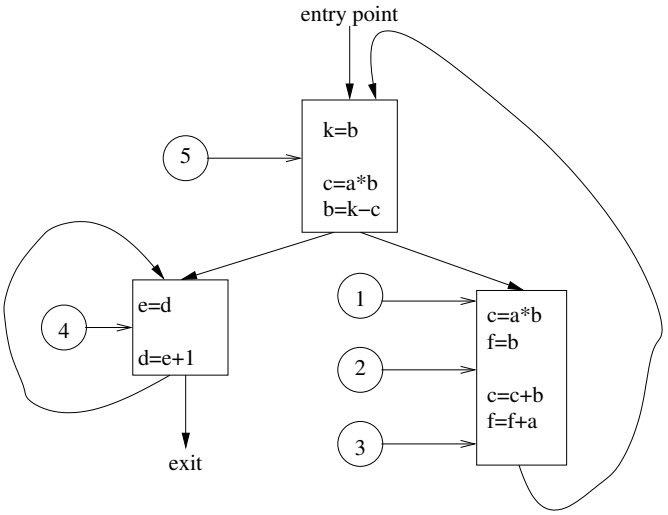

FIGURE 1 – Control flow graph

Question 5 Build the interference graph of the variables (nodes represent variables and edges represent the interference between the variables). Using the interference graph, explain the difference if the code would have been in SSA.

Solution : Here, the variable c introduces two disjoint live ranges. By representing these two by one single node, we force the two variables to be stored in the same register. In our case, nothing changes because the the interferences on c come from the second live range.

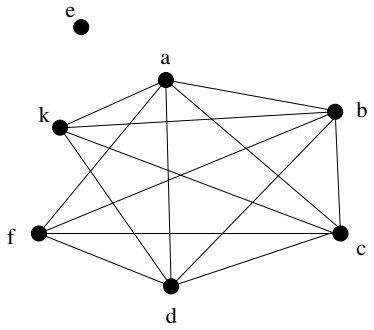

FIGURE 2 – Interference graph

□

Question 6

In order to perform a register allocation, propose a coloring of the obtained graph. How many registers are there needed in order to execute the above code without needing to spill ?

Solution :

We obviously need 5 registers. □

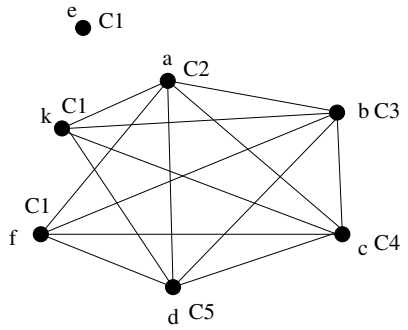

FIGURE 3 – Colored interference graph

Question 7 Consider now that we exchange the instructions : $f = b$ and $c = c + b$ so that the content of the basic block becomes :

```
c=a*b
c=c+b
f=b
f=f+a
```

What are now the alive variables before the instruction $f = b$? What about the number of colors needed to color the interference graph ?

Solution :

$Live(2) = \{a, b, d\}$

We still need 5 colors. There is no interference between f and c, but the group $\{a, b, c, d, k\}$ still forms a clique. □

Question 8 The instruction exchange operation performed above is legal, because it does not change the result of the program execution. Explain the conditions that the following two instructions must verify such that they may be exchanged while preserving the same semantics of the execution :

```
x1 = y1 op z1
x2 = y2 op z2
```

Solution :

The instructions may be swapped if there is no dependency between them : no data dependency $x_1 \neq y2, x_1 \neq z_2$, no output dependence $x_1 \neq x_2$, and no anti-dependence $x_2 \neq y_1, x_2 \neq z_1$. □

Question 9 One way to force 2 variables to share the same register is to join their nodes in the interference graph (this allows reducing the number of moves between registers). Explain in what conditions is it legal to join nodes in this way.

Solution : We can join them if the live ranges of the variables are disjoint (no edge in the graph between them) □

Exercise 3 Node deletion problem (on the interference graph)

Typically, *spilling* is done during the coloring phase : if the greedy algorithm is blocked, one or several variables are splilled in order to unblock it and lower the register pressure. In practice, on a basic block in SSA form, we can find, beforehand a sufficient set of nodes to be removed such that the resulting interference graph is k-colorable : it is enough to decrease the maximum number of variables alive simultaneously, MAXLIVE, to k (the number of registers). In other words, we have to remove live intervals until MAXLIVE is k.

Question 10 Consider a set of live intervals $\{I_1, ..., I_n\}$ on a basic block ; k the number of registers. Give a polynomial algorithm which eliminates a minimum number of intervals s.t. MAXLIVE becomes smaller than k. **Solution :**

Theorem 0.1 *(Furthest First). The spill everywhere problem for an interval graph is polynomially solvable, with a greedy algorithm, if $w(v) = 1$ for all v even if r is not fixed.*

The algorithm behind this theorem is the well-known furthest use strategy described by Belady (L. A. Belady. A study of replacement algorithms for a virtual storage computer. IBM Systems Journal, 5(2) :78–101, 1966.). This strategy is very interesting for designing spilling heuristics on the dominance tree. We give here a constructive proof for completeness.

Proof *An interval graph is the intersection graph of a family of sub-sequences of a (graph) chain. For convenience, we denote the chain as B, vertices of B are called points, and sub-sequences of B are called variables. Consecutive points are denoted by $p_1, \ldots, p_m$, and the set of variables is denoted by V. Once variables are removed (spilled), the remaining set of variables V' is called an allocation. An allocation is said to fit B if, for each point p of B, the number of remaining variables intersecting p is at most r. The goal is to remove a minimum number of variables such that the remaining allocation fits B. The greedy algorithm can be described as follows :*

1. *(init) Let $V_0 = V'$ and $i = 1$;*
2. *(find first) Let $p(i)$ be the first point from the beginning of the chain such that more than r remaining variables, i.e. in V_{i-1}, intersect intersect $p(i)$;*
3. *(remove furthest) Select a variable v_i that intersects p and ends the furthest and remove it, i.e., let $V'_i = V_{i-1} \backslash \{v_i\}$;*
4. *If V'_i fits B, stop, otherwise increment i by 1 and go to Step 1.*

Let us prove that the solution obtained by the greedy algorithm is optimal.

Consider an optimal solution S (described by a set V_S of spilled variables) such that V_S contains the maximum number of variables v_i selected by the greedy algorithm.

Suppose that S does not spill all of them and denote by v_{i_0} the variable with smallest index such that $v_{i_0} \notin V_S$. By definition of p_{i_0} in the greedy algorithm, there are at least $r+1$ variables not in $\{v_1, \ldots, v_{i_0-1}\}$ intersecting $p(i_0)$. As S is a solution, there is a variable v in V_S (thus $v \neq v_{i_0}$) that intersects $p(i_0)$. We claim that spilling $W = V_S \cup \{v_{i_0}\} \backslash \{v\}$, i.e., spilling v_{i_0} instead of v, is a solution too. Indeed, for all points before $p(i_0)$ (excluded), the number of variables in $V'_{i_0-1} = V \backslash \{v_1, \ldots, v_{i_0-1}\}$ is at most r. Since $\{v_1, \ldots, v_{i_0}\} \subseteq W$, this is true for $V \backslash W$ too. Furthermore, each point p after $p(i_0)$ (included), intersected by v, is also intersected by v_{i_0} by definition of v_{i_0}. Thus, as p is intersected by at most r variables in $V \backslash V_S$, the same is true for $V \backslash W$. Finally, this solution spills more variables v_i than S , which is not possible by definition of S . Thus V_S contains all variables v_i and, by optimality, only those. This proves that the greedy algorithm gives an optimal solution.

Proof is given in : F. Bouchez, A. Darte and F. Rastello, "On the complexity of spill everywhere under SSA form", 2007.

□

Question 11 Spilling a variable which has many uses is usually more expensive than spilling a variable which has few uses. In fact we can ponderate the intervals with an approximation of the splilling cost. Is there a polynomial time algorithm for solving this problem ? Hint : Express this problem as an ILP problem. We know that ILP problems with totally unimodular matrices are solvable in polynomial time. See also, F. Bouchez, A. Darte and F. Rastello, "On the complexity of spill everywhere under SSA form", 2007.