





Register Allocation

Adapted from Lectures by
Prof. Alex Aiken and George Necula (UCB)

Lecture Outline

- Memory Hierarchy Management
- Register Allocation
 - Register interference graph
 - Graph coloring heuristics
 - Spilling
- Cache Management

The Memory Hierarchy

	Registers	1 cycle	256-8000 bytes
	Cache	3 cycles	256k-1M
	Main memory	20-100 cycles	32M-1G
	Disk	0.5-5M cycles	4G-1T

Managing the Memory Hierarchy

- Programs are written as if there are only two kinds of memory: main memory and disk
- Programmer is responsible for moving data from disk to memory (e.g., file I/O)
- Hardware, under OS control, is responsible for moving data between memory and caches
- Compiler is responsible for moving data between memory and registers

Current Trends

- Cache and register sizes are growing slowly
- Processor speed improves faster than memory speed and disk speed
 - The cost of a cache miss is growing
 - The widening gap is bridged with more caches
- It is very important to:
 - Manage registers properly
 - Manage caches properly
- Compilers are good at managing registers

The Register Allocation Problem

- Recall that intermediate code uses as many temporaries as necessary
 - This complicates final translation to assembly
 - But simplifies code generation and optimization
 - Typical intermediate code uses too many temporaries
- **The register allocation problem:**
 - Rewrite the intermediate code to use fewer temporaries than there are machine registers
 - **Method:** assign more temporaries to a register
 - But without changing the program behavior



History

- Register allocation is as old as intermediate code
- Register allocation was used in the original FORTRAN compiler in the '50s
 - Very crude algorithms
- A breakthrough was not achieved until 1980 when Chaitin invented a register allocation scheme based on graph coloring
 - Relatively simple, global and works well in practice

CS781(Prasad)

L28RA

7

An Example

- Consider the program


```
a := c + d
e := a + b
f := e - 1
```

 - with the assumption that *a* and *e* die after use
- Temporary *a* can be "reused" after *e := a + b*
- Same with temporary *e*
- Can allocate *a*, *e*, and *f* all to one register (*r₁*):

```
r1 := r2 + r3
r1 := r1 + r4
r1 := r1 - 1
```

CS781(Prasad)

L28RA

8

Basic Register Allocation Idea

- The value in a dead temporary is not needed for the rest of the computation
 - A dead temporary can be reused
- Basic rule:
 - Temporaries *t₁* and *t₂* can share the same register if at any point in the program at most one of *t₁* or *t₂* is live!

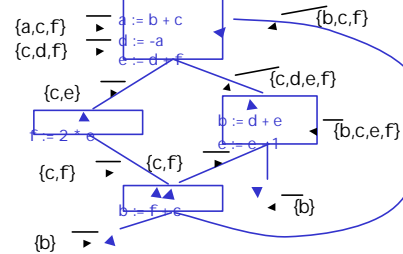
CS781(Prasad)

L28RA

9

Algorithm: Part I

- Compute **live** variables for each point:



10

The Register Interference Graph

- Two temporaries that are **live** simultaneously cannot be allocated in the same register
- We construct an undirected graph
 - A node for each temporary
 - An edge between *t₁* and *t₂* if they are **live** simultaneously at some point in the program
- This is the register interference graph (RIG)
 - Two temporaries can be allocated to the same register if there is no edge connecting them

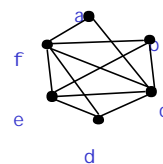
CS781(Prasad)

L28RA

11

Register Interference Graph. Example.

- For our example:



- E.g., *b* and *c* cannot be in the same register
- E.g., *b* and *d* can be in the same register

CS781(Prasad)

L28RA

12

Register Interference Graph. Properties.

- It extracts exactly the information needed to characterize legal register assignments
- It gives a global (i.e., over the entire flow graph) picture of the register requirements
- After RIG construction the register allocation algorithm is architecture independent

Graph Coloring. Definitions.

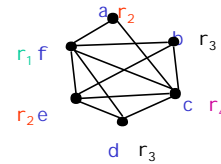
- A **coloring of a graph** is an assignment of colors to nodes, such that nodes connected by an edge have different colors
- A graph is **k-colorable** if it has a coloring with k colors

Register Allocation Through Graph Coloring

- In our problem, colors = registers
 - We need to assign colors (registers) to graph nodes (temporaries)
- Let k = number of machine registers
- If the RIG is k -colorable then there is a register assignment that uses no more than k registers

Graph Coloring. Example.

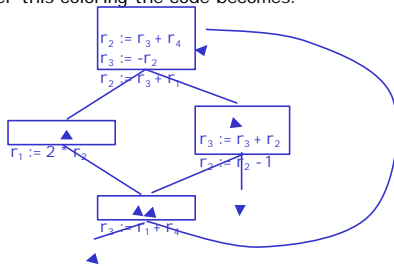
- Consider the example RIG



- There is no coloring with less than 4 colors
- There are 4-colorings of this graph

Graph Coloring. Example.

- Under this coloring the code becomes:



Computing Graph Colorings

- The remaining problem is to compute a coloring for the interference graph
- But:
 1. This problem is very hard (NP-hard). No efficient algorithms are known.
 2. A coloring might not exist for a given number or registers
- The solution to (1) is to use heuristics
- We'll consider later the other problem

Graph Coloring Heuristic

- **Observation:**
 - Pick a node t with fewer than k neighbors in RIG
 - Eliminate t and its edges from RIG
 - If the resulting graph has a k -coloring then so does the original graph
- **Why:**
 - Let c_1, \dots, c_n be the colors assigned to the neighbors of t in the reduced graph
 - Since $n < k$ we can pick some color for t that is different from those of its neighbors

CS781(Prasad)

L28RA

19

Graph Coloring Heuristic

- The following works well in practice:
 - Pick a node t with fewer than k neighbors
 - Put t on a stack and remove it from the RIG
 - Repeat until the graph has one node
- Then start assigning colors to nodes on the stack (starting with the last node added)
 - At each step pick a color different from those assigned to already colored neighbors

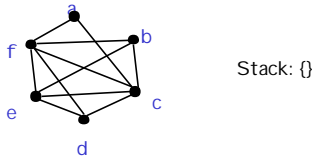
CS781(Prasad)

L28RA

20

Graph Coloring Example (1)

- Start with the RIG and with $k = 4$:



- Remove a and then d

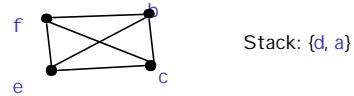
CS781(Prasad)

L28RA

21

Graph Coloring Example (2)

- Now all nodes have fewer than 4 neighbors and can be removed: c, b, e, f



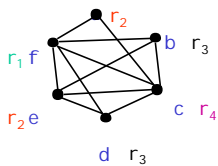
CS781(Prasad)

L28RA

22

Graph Coloring Example (2)

- Start assigning colors to: f, e, b, c, d, a



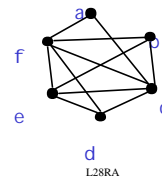
CS781(Prasad)

L28RA

23

What if the Heuristic Fails?

- What if during simplification we get to a state where all nodes have k or more neighbors?
- **Example:** try to find a 3-coloring of the RIG:



CS781(Prasad)

L28RA

24

Conclusions

- Register allocation is a “must have” optimization in most compilers:
 - Because intermediate code uses too many temporaries
 - Because it makes a big difference in performance
- Graph coloring is a powerful register allocation scheme
- Register allocation is more complicated for CISC machines