

CS352 - Compilers: Principles and Practice
Homework 3 - One Reg, Two Reg, Red Reg, Blue Reg
Due: 2003 December 11, beginning of lecture

Consider this simple translation of our favorite Fibonacci function:

```
fib:  a := r3
      f := r1
      n2 := M[f]
      n1 := M[f + 4]
      x := M[f + 8]
      n0 := n1 + n2
      M[f + 4] := n0
      M[f] := n1
      if x <= 3 goto L1
      x := x - 1
      M[f + 8] := x
      r1 := f
      call fib
      n0 := r1
L1:   r1 := n0
      r3 := a
      return
```

The code has been compiled for a target architecture with four (4) registers:

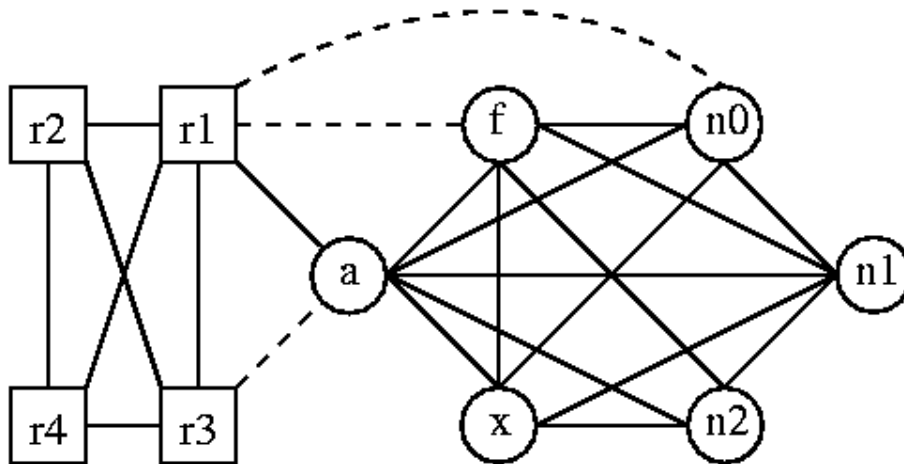
- $r1$ is a (caller-save) argument/result register,
- $r2$ and $r4$ are caller-save temporaries.
- $r3$ is a callee-save register.

The code defines and uses temporaries $n0$, $n1$, $n2$, a , x , and f .

- Build the control flow graph for this program fragment, and perform liveness analysis on it. Show the *in* and *out* sets for each node in the graph. (There should be one node for each line of code.)

<i>n</i>	<i>def</i> (<i>n</i>)	<i>use</i> (<i>n</i>)	<i>in</i> (<i>n</i>)	<i>out</i> (<i>n</i>)
fib: a := r3	{a}	{r3}	{r1, r3}	{r1, a}
f := r1	{f}	{r1}	{r1, a}	{a, f}
n2 := M[f]	{n2}	{f}	{a, f}	{n2, a, f}
n1 := M[f + 4]	{n1}	{f}	{n2, a, f}	{n1, n2, a, f}
x := M[f + 8]	{x}	{f}	{n1, n2, a, f}	{n1, n2, a, f, x}
n0 := n1 + n2	{n0}	{n1, n2}	{n1, n2, a, f, x}	{n0, n1, a, f, x}
M[f + 4] := n0	{}	{n0, f}	{n0, n1, a, f, x}	{n0, n1, a, f, x}
M[f] := n1	{}	{n1, f}	{n0, n1, a, f, x}	{n0, a, f, x}
if x <= 3 g...	{}	{x}	{n0, a, f, x}	{n0, a, f, x}
x := x - 1	{x}	{x}	{a, f, x}	{a, f, x}
M[f + 8] := x	{}	{f, x}	{a, f, x}	{a, f}
r1 := f	{r1}	{f}	{a, f}	{r1, a}
call fib	{r1, r2, r4}	{r1}	{r1, a}	{r1, a}
n0 := r1	{n0}	{r1}	{r1, a}	{n0, a}
L1: r1 := n0	{r1}	{n0}	{n0, a}	{r1, a}
r3 := a	{r3}	{a}	{r1, a}	{r1, r3}
return	{}	{r1, r3}	{r1, r3}	{r1, r3}

- Construct the interference graph for the program fragment, using solid lines to show interference, and dotted lines to show *MOVE* relations.

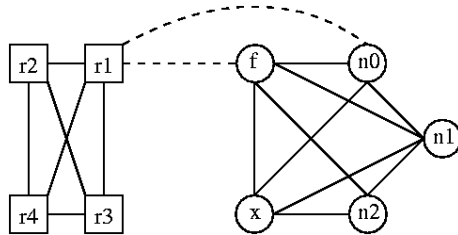


3. Show the steps for register allocation (graph coloring with conservative coalescing) for this program in detail, as described in the notes and pages 229–232 of the textbook. When choosing nodes for potential spills, use the spill priority heuristic (*number of defs + number of uses*)/*degree*. Use the George criteria for conservative coalescing.

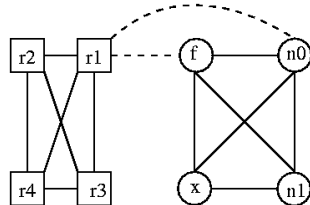
- No simplifies possible. (All non-move-related nodes significant.)
- No coalesces possible. (All move-related pairs fail conservative criteria.)
- No freezes possible. (No move-related nodes of low degree.)
- Must choose potential spill. Use heuristic:

	<i>defs + uses</i>	/	<i>degree</i>	=	<i>spill priority</i>
<i>n0</i>	4	/	4	=	1
<i>n1</i>	3	/	5	=	0.6
<i>n2</i>	2	/	4	=	0.5
<i>a</i>	2	/	6	=	0.33
<i>f</i>	8	/	5	=	1.6
<i>x</i>	5	/	5	=	1

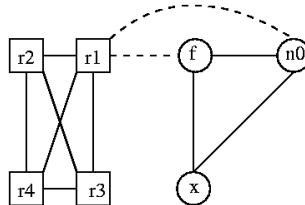
Node *a* wins. Push *a* on stack as potential spill.



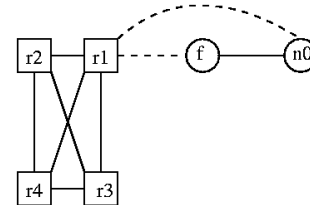
Simplify *n2*.



Simplify *n1*.



Simplify *x*.

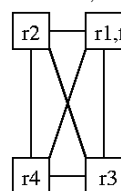
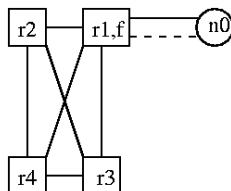


- Can't simplify – all nodes are either move-related or precolored.

Can coalesce *r1, f* or *r1, n0*.

Last move *constrained*.

Eliminate move, simplify *n0*.



- Select phase reveals no coloring for node a . Becomes *actual spill*. Add code into original program fragment for spilling:

```

a := r3  →  a1 := r3
           M[a_loc] := a1

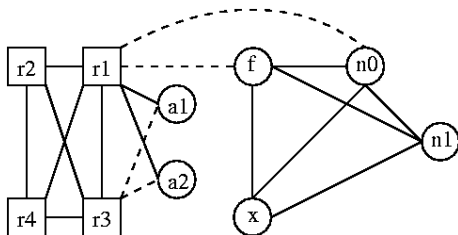
```

```

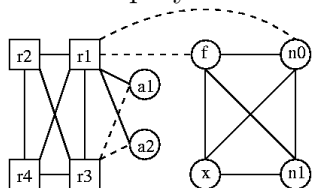
r3 := a  →  a2 := M[a_loc]
           r3 := a2

```

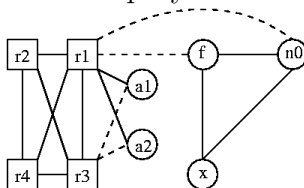
- Recalculate liveness ranges, build new interference graph:



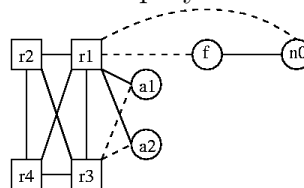
Simplify $n2$.



Simplify $n1$.

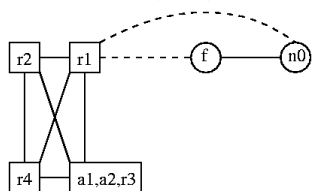


Simplify x .

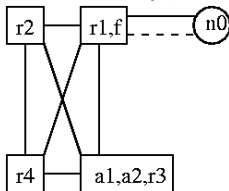


-

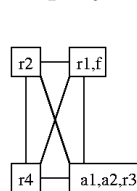
Coalesce $a1$, $a2$, and $r3$.



Coalesce $f, r1$.



Simplify $n0$.



- Select colors:

$r1$	$r2$	$r3$	$r4$	$n0$	$n1$	$n2$	$a1$	$a2$	f	x
1	2	3	4	2	4	2	3	3	1	3

4. Show the final program after register allocation,

```
fib:  r3 := r3
      M[a_loc] := r3
      r1 := r1
      r2 := M[r1]
      r4 := M[r1 + 4]
      r3 := M[r1 + 8]
      r2 := r4 + r2
      M[r1 + 4] := r2
      M[r1] := r4
      if r3 <= 3 goto L1
      r3 := r3 - 1
      M[r1 + 8] := r3
      r1 := r1
      call fib
      r2 := r1
L1:   r1 := r2
      r3 := M[a_loc]
      r3 := r3
      return
```

... with redundant moves removed.

```
fib:  M[a_loc] := r3
      r2 := M[r1]
      r4 := M[r1 + 4]
      r3 := M[r1 + 8]
      r2 := r4 + r2
      M[r1 + 4] := r2
      M[r1] := r4
      if r3 <= 3 goto L1
      r3 := r3 - 1
      M[r1 + 8] := r3
      call fib
      r2 := r1
L1:   r1 := r2
      r3 := M[a_loc]
      return
```