10. Register Allocation

Eva Rose    Kristoffer Rose

NYU Courant Institute
Compiler Construction (CSCI-GA.2130-001)
http://cs.nyu.edu/courses/spring15/CSCI-GA.2130-001/lecture-10.pdf

April 20, 2015
1. Basic Block Code Generation
2. Cross-BB Register Allocation
3. Interference Graphs
4. HACS & Project Milestone 3
Sixth compilation phase

source program

Tokens
Lexical Analysis

Syntax Analysis

Symbol Table

Tree

Semantic Analysis

Intermediate Representation Generator

Tree

Optimizer

IR

Code Generator

Assembly

IR

Machine-Dependent Code Optimizer

target machine code
Basic Block Code Generation

Cross-BB Register Allocation

Interference Graphs

HACS & Project Milestone 3
Next-Use

How many registers are needed inside a basic block?
Now generate the instructions but . . .

- *Maintain mapping between variables and registers.*
- Avoid loading values already in a register.
- Store only as needed.
- Reuse registers that have no further use.
Register Allocation: *getReg*

We have value $V$ how do we get it in a register?

1. If we got it all is already well!
2. If we have an empty register then use that.
3. If there is no free register we have to make one—
   1. If we have a redundant one, use that;
   2. If we know our instruction will destroy a register, use it;
   3. If we have no next-use then it is as free;
   4. Otherwise we have to spill.

Keep track of:

- Register to variable map.
- Variable to register/memory map.
Register Allocation: \textit{getReg}

We have value $V$ how do we get it in a register?

1. If we got it all is already well!
2. If we have an empty register then use that.
3. If there is no free register we have to make one—
   1. If we have a redundant one, use that;
   2. If we know our instruction will destroy a register, use it;
   3. If we have no next-use then it is as free;
   4. Otherwise we have to spill.

Keep track of:

- Register to variable map.
- Variable to register/memory map.
Let’s translate

t = a - b
u = a - c
v = t + u
a = d
d = v + u

Reconstruct Dragon Book Figure 8.16 from this.
Let’s translate

\[
\begin{align*}
t &= a - b \\
u &= a - c \\
v &= t + u \\
a &= d \\
d &= v + u
\end{align*}
\]

Reconstruct Dragon Book Figure 8.16 from this.
1. Basic Block Code Generation
2. Cross-BB Register Allocation
3. Interference Graphs
4. HACS & Project Milestone 3
Example

```c
int mult(int a, int b)
{
    int i = a;
    int r = 0;
    while (i>0) {
        r += b;
        i--;
    }
    return r;
}
```
Example (IR)

```c
int mult(int a, int b)
{
    i = a
    r = 0
    goto Test

Next:
    r = r + b
    i = i - 1

Test:
    if i>0 goto Next

return r;
}
```
Example (ARM32)

mult: ; "a"=R0, "b"=R1
   MOV R2, R0 ; "i"=R2
   MOV R3, #0 ; "r"=R3
   B Test

Next: ADD R3, R3, R1
      SUB R2, R2, #1

Test: CMP R2, #0
      BGT Next

      MOV R0, R3
      MOV pc, lr
Register Allocation for Multiple Basic Blocks?

Determine which variables are required for each jump...
Example (ARM32 Subset, naïve forward register allocation)

```
# R0  R1  R2  R3  a    b    r   i

MOV R3,R0 # a,i,b   a,i  R0,R3 R1  R0,R3
MOV R2,#0 # a,i,b   r   a,i  R0,R3 R1  R2  R0,R3
B Test   # a,i,b   r   a,i  R0,R3 R1  R2  R0,R3

Next: ADD R2,R2,R1
       SUB R3,R3,#1

Test: CMP R3,#0
       BGT Next

MOV R0,R2 # r   b   r   i   R0  R1  R2 R0 R3
```
Example (ARM32 Subset, naïve forward register allocation)

```
mult:   STMFD sp!,{R4-R11,lr}  # a  b
        MOV  R3,R0  # a,i  b  a,i  R0,R3  R1  R0,R3
        MOV  R2,#0  # a,i  b  r  a,i  R0,R3  R1  R2  R0,R3
        B   Test

Next:  ADD  R2,R2,R1
        SUB  R3,R3,#1

Test:  CMP  R3,#0
        BGT  Next

        MOV  R0,R2  # r  b  r  i  R0  R1  R2  R0  R3
```
Example (ARM32 Subset, naïve forward register allocation)

```
# R0  R1  R2  R3  a   b   r   i

mult:  STMFD sp!,{R4-R11,lr} # a  b

      MOV R3,R0 # a,i  b  a,i  R0  R3
      MOV R2,#0 # a,i  b  r   a,i  R0  R3
      B  Test  # a,i  b  r   a,i  R0  R3  R1  R2  R0,R3

Next:  ADD  R2,R2,R1
      SUB  R3,R3,#1

Test:  CMP  R3,#0
      BGT Next

      MOV  R0,R2 # r  b  r  i  R0  R1  R2  R0  R3
```
Example (ARM32 Subset, naïve forward register allocation)

```
# R0  R1  R2  R3  a  b  r  i

mult:  STMFD sp!,{R4-R11,lr}  # a  b  R0  R1
       MOV  R3,R0  # a,i  b  a,i  R0,R3  R1  R0,R3
       MOV  R2,#0  # a,i  b  r  a,i  R0,R3  R1  R2  R0,R3
       B  Test  # a,i  b  r  a,i  R0,R3  R1  R2  R0,R3

Next:  ADD  R2,R2,R1
       SUB  R3,R3,#1

Test:  CMP  R3,#0
       BGT  Next

MOV  R0,R2  # r  b  r  i  R0  R1  R2,R0  R3
```
Example (ARM32 Subset, naïve forward register allocation)

```
# R0  R1  R2  R3  a   b   r   i

mult:  STMFD sp!,{R4-R11,lr}  # a  b       R0  R1

    MOV  R3,R0             # a,i b   a,i  R0,R3  R1   R0,R3
    MOV  R2,#0             # a,i b   r   a,i  R0,R3  R1   R2   R0,R3
    B  Test               # a,i b   r   a,i  R0,R3  R1   R2   R0,R3

Next:    ADD    R2,R2,R1
         SUB    R3,R3,#1

Test:    CMP    R3,#0
         BGT    Next

    MOV    R0,R2          # r   b   r   i  R0  R1  R2,R0  R3
```
Example (ARM32 Subset, naïve forward register allocation)

```assembly
# R0  R1  R2  R3  a  b  r  i

mult:  STMFD sp!,{R4-R11,lr}  # a  b  R0  R1

  MOV   R3,R0      # a, i  b  a, i  R0  ,R3  R1  R0  ,R3
  MOV   R2,#0      # a, i  b  r  a, i  R0  ,R3  R1  R2  R0  ,R3
  B    Test       # a, i  b  r  a, i  R0  ,R3  R1  R2  R0  ,R3

Next:  ADD   R2,R2,R1
       SUB   R3,R3,#1

Test:  CMP   R3,#0  # a, i  b  r  a, i  R0  ,R3  R1  R2  R0  ,R3

BGT  Next

MOV   R0,R2      # r  b  i  R0  R1  R2  R0  R3
```
Example (ARM32 Subset, naïve forward register allocation)

```
# R0 R1 R2 R3 a b r i

mult:  STMFD sp!,{R4-R11,lr} # a b R0 R1

MOV  R3,R0 # a,i b a,i R0,R3 R1 R0,R3
MOV  R2,#0 # a,i b r a,i R0,R3 R1 R2 R0,R3
B Test # a,i b r a,i R0,R3 R1 R2 R0,R3

Next:  ADD  R2,R2,R1
SUB  R3,R3,#1

Test:  CMP  R3,#0 # a,i b r a,i R0,R3 R1 R2 R0,R3
BGT Next # a,i b r a,i R0,R3 R1 R2 R0,R3

MOV  R0,R2 # r b r i R0 R1 R2,R0 R3
```
Example (ARM32 Subset, naïve forward register allocation)

# R0  R1  R2  R3  a  b  r  i

mult:  STMFD sp!,{R4-R11,lr}  # a  b  R0  R1

    MOV  R3,R0  # a,i  b  a,i  R0,R3  R1  R0,R3
    MOV  R2,#0  # a,i  b  r  a,i  R0,R3  R1  R2  R0,R3
    B  Test  # a,i  b  r  a,i  R0,R3  R1  R2  R0,R3

Next:  ADD  R2,R2,R1  # a,i  b  r  a,i  R0,R3  R1  R2  R0,R3

    SUB  R3,R3,#1

Test:  CMP  R3,#0  # a,i  b  r  a,i  R0,R3  R1  R2  R0,R3

    BGT  Next  # a,i  b  r  a,i  R0,R3  R1  R2  R0,R3

    MOV  R0,R2  # r  b  r  i  R0  R1  R2,R0  R3
Example (ARM32 Subset, naïve forward register allocation)

```
# R0  R1  R2  R3  a   b   r   i
mult: STMFD sp!,{R4-R11,lr} # a  b
      MOV  R3,R0          # a  i  b  a  i
      MOV  R2,#0          # a  i  b  r  a  i
      B   Test            # a  i  b  r  a  i
Next: ADD  R2,R2,R1     # a  i  b  r  a  i
      SUB  R3,R3,#1       # a  b  r  i
      CMP  R3,#0          # a  i  b  r  a  i
      BGT Next            # a  i  b  r  a  i
      MOV  R0,R2          # r  b  r  i
      MOV  R0,R1          # r  b  i
      MOV  R0,R0,R2       # r  b  r  i
      MOV  R0,R3          # r  b  i
      MOV  R0,R0,R0,R3    # r  b  i
```
Example (ARM32 Subset, naïve forward register allocation)

```assembly
mult:    STMFD sp!,{R4-R11,lr}  # a b R0 R1
         MOV R3,R0 # a,i b a,i R0,R3 R1 R0,R3
         MOV R2,#0 # a,i b r a,i R0,R3 R1 R2 R0,R3
         B Test # a,i b r a,i R0,R3 R1 R2 R0,R3

Next:   ADD R2,R2,R1 # a,i b r a,i R0,R3 R1 R2 R0,R3
         SUB R3,R3,#1 # a b r i R0 R1 R2 R3

Test:   CMP R3,#0 # a b r i R0 R1 R2 R3
         BGT Next # a b r i R0 R1 R2 R3

MOV R0,R2 # r b r i R0 R1 R2 R3 R0 R3
```
Example (ARM32 Subset, naïve forward register allocation)

```
# R0  R1  R2  R3  a   b   r   i

mult:  STMFD sp!,{R4-R11,lr}  # a  b

MOV    R3,R0                   # a,i b   a,i  R0,R3  R1  R0,R3
MOV    R2,#0                   # a,i b   r   a,i  R0,R3  R1  R2  R0,R3
B      Test                     # a,i b   r   a,i  R0,R3  R1  R2  R0,R3

Next:  ADD    R2,R2,R1           # a  b   r   i  R0  R1  R2  R3
       SUB    R3,R3,#1            # a  b   r   i  R0  R1  R2  R3

Test:  CMP    R3,#0              # a  b   r   i  R0  R1  R2  R3
       BGT    Next                # a  b   r   i  R0  R1  R2  R3

MOV    R0,R2                   # r   b   r   i  R0  R1  R2,R0  R3
```
Example (ARM32 Subset, naïve forward register allocation)

```
# R0 R1 R2 R3 a b r i

mult: STMFD sp!,{R4-R11,lr} # a b R0 R1
      MOV R3,R0 # a,i b a,i R0,R3 R1 R0,R3
      MOV R2,#0 # a,i b r a,i R0,R3 R1 R2 R0,R3
      B Test # a,i b r a,i R0,R3 R1 R2 R0,R3

Next:  ADD R2,R2,R1 # a b r i R0 R1 R2 R3
       SUB R3,R3,#1 # a b r i R0 R1 R2 R3

Test:  CMP R3,#0 # a b r i R0 R1 R2 R3
       BGT Next # a b r i R0 R1 R2 R3

MOV R0,R2 # r b r i R0 R1 R2,R0 R3
```
Example (ARM32 Subset, naïve forward register allocation)

```
# R0  R1  R2  R3  a  b  r  i
mult: STMFD sp!,{R4-R11,lr}  # a  b  R0  R1

MOV  R3,R0  # a, i b  a, i  R0, R3  R1  R0, R3
MOV  R2,#0  # a, i b  r  a, i  R0, R3  R1  R2  R0, R3
B  Test  # a, i b  r  a, i  R0, R3  R1  R2  R0, R3

Next: ADD  R2,R2,R1  # a  b  r  i  R0  R1  R2  R3
SUB  R3,R3,#1  # a  b  r  i  R0  R1  R2  R3

Test: CMP  R3,#0  # a  b  r  i  R0  R1  R2  R3
BGT  Next  # a  b  r  i  R0  R1  R2  R3

MOV  R0,R2  # r  b  r  i  R0  R1  R2, R0  R3
LDMFD sp!,{R4-R11,pc}  # r  b  R1  R0
```
Example (ARM32 Subset, naïve forward register allocation)

# R0 R1 R2 R3 a b r i

```
mult:

MOV R3, R0  # a, i b a, i R0, R3 R1 R0, R3
MOV R2, #0  # a, i b r a, i R0, R3 R1 R2 R0, R3
B Test     # a, i b r a, i R0, R3 R1 R2 R0, R3

Next: ADD R2, R2, R1  # a b r i R0 R1 R2 R3
      SUB R3, R3, #1  # a b r i R0 R1 R2 R3

Test: CMP R3, #0  # a b r i R0 R1 R2 R3
      BGT Next  # a b r i R0 R1 R2 R3

MOV R0, R2  # r b r i R0 R1 R2 R0 R3
MOV pc, lr   # r b R1 R0
```
Interference Graph

Track variables that are live at the same time

Analyze backwards, one 3-address instruction at the time:
- When a variable is set then it is removed.
- When a variable is used then it is added.
**Interference Graph**

Track variables that are **live** at the same time

Analyze **backwards**, one 3-address instruction at the time:

1. When a variable is **set** then it is **removed**.
2. When a variable is **used** then it is **added**.
Interference Graph

```c
int mult(int a, int b)
{
    i = a
    r = 0
    goto Test

    Next:
        r = r + b
        i = i - 1

    Test:
        if i>0 goto Next

    return r;
}
```
int mult(int a, int b)
{
    i = a
    r = 0
    goto Test

    Next:
        r = r + b
        i = i - 1

    Test:
        if i>0 goto Next  {r}

        return r;
}
```c
int mult(int a, int b) {
    i = a
    r = 0
    goto Test

    Next:
    r = r + b
    i = i - 1

    Test:       {i,r}
    if i>0 goto Next   {r}

    return r;
}
```
int mult(int a, int b)
{
    i = a
    r = 0
    goto Test {i,r}

    Next:
    r = r + b
    i = i - 1 {i,r}

    Test: {i,r}
    if i>0 goto Next {r}

    return r;
}
Interference Graph

```c
int mult(int a, int b)
{
    i = a
    r = 0
    goto Test {i,r}

    Next:
    r = r + b
    i = i - 1 {i,r}

    Test:
    if i>0 goto Next {r}

    return r;
}
```
### Interference Graph

```c
int mult(int a, int b)
{
    i = a
    r = 0
    goto Test

    Next:
    r = r + b
    i = i - 1

    Test:
    if i>0 goto Next
    return r;
}
```
**Interference Graph**

```c
int mult(int a, int b)
{
    {a}
    i = a
    r = 0
    goto Test {i,r}

Next: {b,i,r}
    r = r + b
    i = i - 1 {i,r}

Test: {i,r}
    if i>0 goto Next {b,i,r}

    return r;
}
```
```c
int mult(int a, int b)
{
    i = a
    r = 0
    goto Test {i,r}

    Next:
    r = r + b
    i = i - 1 {i,r}

    Test:
    if i>0 goto Next {b,i,r}

    return r;
}
```
Interference Graph

```c
int mult(int a, int b)
{
    {a}
    i = a
    r = 0
    goto Test {b,i,r}

    Next:
    {b,i,r}
    r = r + b
    i = i - 1 {b,i,r}

    Test:
    {b,i,r}
    if i>0 goto Next {b,i,r}

    return r;
}
```
Interference Graph

```c
int mult(int a, int b)
{
    {a,b}  // initially
    i = a
    r = 0
    goto Test  // initial edge
    {b,i,r}

    Next:  // incoming edge from Test
    {b,i,r}
    r = r + b
    i = i - 1  // outgoing edge to Next
    {b,i,r}

    Test:  // incoming edge from Next
    {b,i,r}
    if i>0 goto Next  // outgoing edge to Next
    {b,i,r}

    return r;
}
```

Eva Rose, Kristoffer Rose

Compiler Construction (CSCI-GA.2130-001) 10. Register Allocation

April 20, 2015
**Interference Graph**

```c
int mult(int a, int b)
{
    {a,b}
    i = a
    r = 0
    goto Test {b,i,r}

    Next: {b,i,r}
    r = r + b
    i = i - 1 {b,i,r}

    Test: {b,i,r}
    if i>0 goto Next {b,i,r}

    return r;
}
```

Interference Graph

![Interference Graph Diagram](image)
Example (ARM32 Subset, finished)

```assembly
# a,r=R0 b=R1 i=R2

mult: MOV R2,R0
      MOV R0,#0
      B Test

Next: ADD R0,R0,R1
     SUB R2,R2,#1

Test: CMP R2,#0
      BGT Next
      MOV pc,lr
```
Example (Full ARM32)

```
#  a,i=R0  b=R1  r=R2

mult:    MOV S  R2, R0
         MOV  R0, #0
         B    Test

Next:    ADD  R0, R0, R1
         SUBS R2, R2, #1

Test:    BPL  Next
         MOV  pc, lr
```
Basic Block Code Generation  Cross-BB Register Allocation  Interference Graphs  HACS & Project Milestone 3

**Spill Example (IR)**

Start:  
\#{a=R0,b=R1}
\[i = a\]
\[r = 0\]
goto Test

Next:  
\#{b,i,r}
\[r = r + b\]
\[i = i - 1\]

Test:  
\#{b,i,r}
if i>0 goto Next

End:  
\#{r}
return r;
<table>
<thead>
<tr>
<th></th>
<th>Basic Block Code Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Cross-BB Register Allocation</td>
</tr>
<tr>
<td>3</td>
<td>Interference Graphs</td>
</tr>
<tr>
<td>4</td>
<td>HACS &amp; Project Milestone 3</td>
</tr>
</tbody>
</table>
Questions?