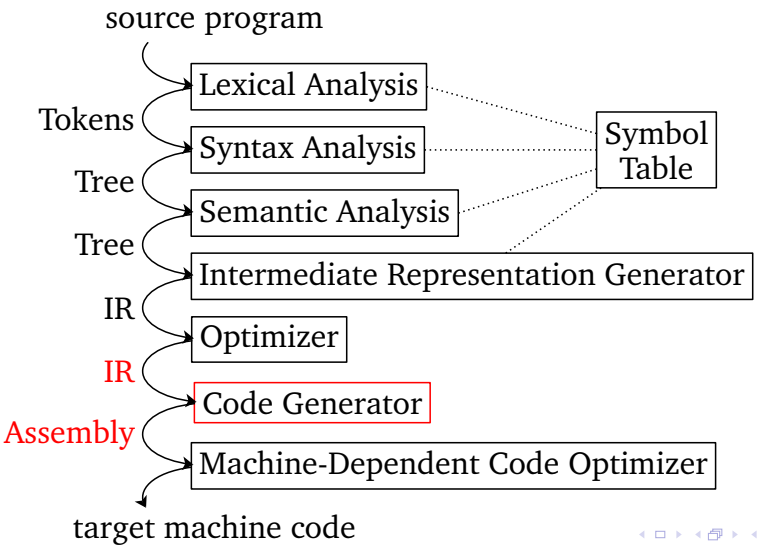
Register Allocation

Eva Rose Kristoffer Rose

NYU Courant Institute
Compiler Construction (CSCI-GA.2130-001)
http://cs.nyu.edu/courses/fall14/CSCI-GA.2130-001/lecture-10.pdf

November 13, 2014

Sixth compilation phase



Next-Use

How many registers are needed inside a basic block?

Idea

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.

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

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

Reconstruct Dragon Book Figure 8.16 from this.

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.

DAG – Directed Acyclic Graph

- One node per initial value.
- One node per statement, with an edge to last node observing every parameter.
- Node labeled by operator and list of variables *not* used further.
- Output nodes are those with *live exit variables*.

DAG uses

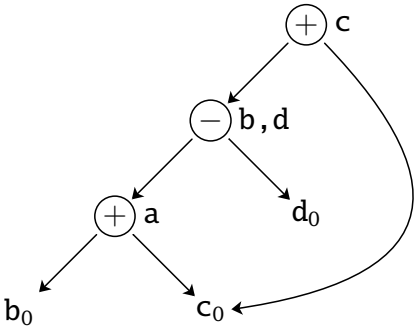
- Local common subexpressions.
- Dead code elimination.
- Apply albegraic simplifications.
- Reorder statements to reduce variable count.
- Register Allocation.

Local Common Subexpressions

```
1  a = b + c
2  b = a - d
3  c = b + c
4  d = a - d
```


Local Common Subexpressions

```
1  a = b + c
2  b = a - d
3  c = b + c
4  d = a - d
```



Dead Code Elimination

Inputs: b,c,d

```
1 a = b + c
2 b = b - c
3 c = c + d
4 e = b + c
```

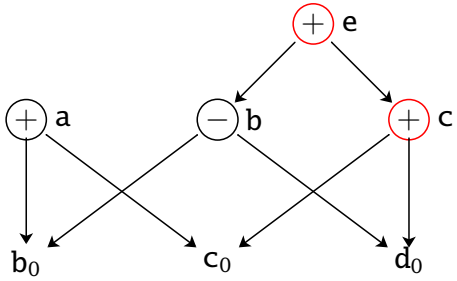
Outputs: a,b

Dead Code Elimination

Inputs: b, c, d

```
1  a = b + c
2  b = b - c
3  c = c + d
4  e = b + c
```

Outputs: a, b



Algebraic Identities

$$x + 0 = 0 + x = x \qquad x - 0 = x$$
$$x \times 1 = 1 \times x = x \qquad x/1 = x$$

Expensive Cheaper

$$x^2 = x \times x$$
$$2 \times x = x + x$$
$$x/2 = x \times 0.5 \; (= x \gg 1)$$

Algebraic Identities

$$x + 0 = 0 + x = x \qquad x - 0 = x$$
$$x \times 1 = 1 \times x = x \qquad x/1 = x$$

Expensive	Cheaper
x^2	$x \times x$
$2 \times x$	$x + x$
$x/2$	$x \times 0.5\ (= x \gg 1)$

Algebraic Identities

- Constant folding
- Commutativity with Local Common Subexpressions
- Associativity with composite expressions

```
1  a = c + b
2  e = c + d + b
```


Algebraic Identities

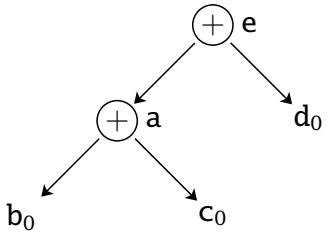
- Constant folding
- Commutativity with Local Common Subexpressions
- Associativity with composite expressions

```
1  a = c + b
2  e = c + d + b
```


Algebraic Identities

- Constant folding
- Commutativity with Local Common Subexpressions
- Associativity with composite expressions

```
1  a = c + b
2  e = c + d + b
```

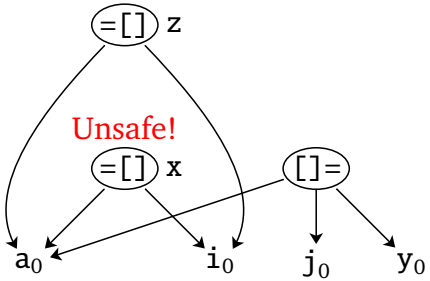


Array References

```
1 x = a[i]
2 a[j] = y
3 z = a[i]
```


Array References

```
1  x = a[i]
2  a[j] = y
3  z = a[i]
```



No Pointing!

```
1  x = *p
2  *q = y
```

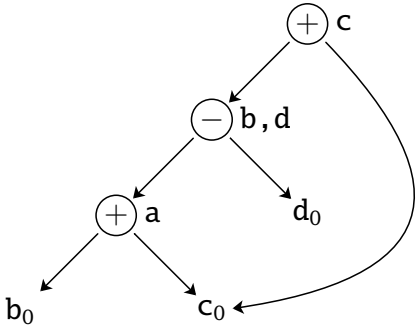
The entire memory is a single array! Needs pointer (“points-to”) analysis!

No Pointing!

```
1 x = *p
2 *q = y
```

The entire memory is a single array! Needs pointer (“points-to”) analysis!

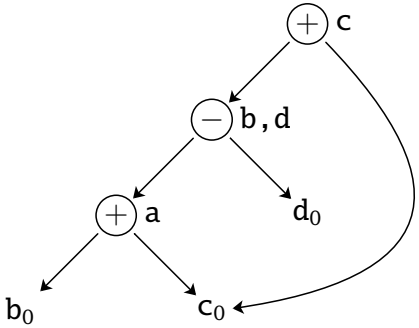
Back to Code




```
1 a = b + c
2 b = a - d
3 c = b + c
4 d = b
```

Respect ordering, pay special attention to overlapping side effects!

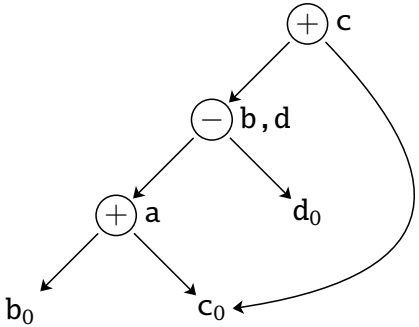
Back to Code




```
1  a = b + c
2  b = a - d
3  c = b + c
4  d = b
```

Respect ordering, pay special attention to overlapping side effects!

Back to Code




```
1  a = b + c
2  b = a - d
3  c = b + c
4  d = b
```

Respect ordering, pay special attention to overlapping side effects!

Example

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

Example (IR)

```
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;
}
```

ARM32 Instruction Subset

```
MOV reg,arg           reg = R0 | R1 | ... | R15 | SP | LR | PC

ADD reg,reg1,arg2     arg = #imm8 | reg | reg,LSL #imm5 | reg,LSR #imm5
SUB reg,reg1,arg2
MUL reg,reg1,arg2     immn = n-bit unsigned constant
AND reg,reg1,arg2
ORR reg,reg1,arg2
EOR reg,reg1,arg2

CMP reg,arg

B   imm12
Bcd imm12             cd = EQ | NE | GT | LT | GE | LE | CS | CC
BL  imm12

LDRb reg,mem          mem = [reg,arg]   b = B?
STRb reg,mem

LDMFD reg!,mreg       mreg = {reg,...,reg}
STMFD reg!,mreg
```

ARM32 Calling Conventions

Register	*On Entry*	*On Return*
r0–3	parameter or unused	return value or unused
r4–11	preserved	same as on entry
r12	undefined	undefined
r13 ‘sp’	stack pointer	same as on entry
r14 ‘lr’	return address	(unconstrained)
r15 ‘pc’	program counter	return address

Example (ARM32 Subset)

```
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: *getReg* for One Basic Block

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.

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)

```
                                # 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)

```
                                # 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)

```
                                # 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  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           # 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           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,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        # 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
```

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:                           # 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
        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:

1. When a variable is set then it is removed.
2. 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

```
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

Interference Graph

```
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;
}
```


Interference Graph

Interference Graph

```
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;
}
```


Interference Graph

Interference Graph

```
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

Interference Graph

```
int mult(int a, int b)
{                       {a}
  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

Interference Graph

```
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      {r}

  return r;
}
```


Interference Graph

Interference Graph

```
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;
}
```


Interference Graph

Interference Graph

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

  return r;
}
```


Interference Graph

Interference Graph

```
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

Interference Graph

```
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

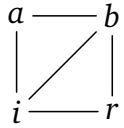
Interference Graph

```
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

Example (ARM32 Subset, finished)

```
                          # 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:   MOVS  r2,r0
        MOV   r0,#0
        B Test
Next:   ADD   r0,r0,r1
        SUBS  r2,r2,#1
Test:   BPL Next
        MOV pc,lr
```

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;
```

Questions?

Bonus Division

```
.global udiv64
udiv64:
    adds      r0,r0,r0
    adc       r1,r1,r1

    .rept 31
        cmp     r1,r2
        subcs   r1,r1,r2
        adcs    r0,r0,r0
        adc     r1,r1,r1
    .endr

    cmp     r1,r2
    subcs   r1,r1,r2
    adcs    r0,r0,r0

    bx      lr
```

Questions?