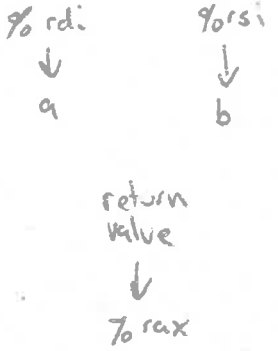
Register Allocation

TACK

```
mult = fun
  (a:int, b, int) -> int
{

  r = 0;
  i = a;

  while i > 0 {

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

  }
  -> r

}
```

"Virtual register" x64

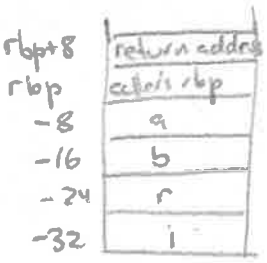
Calling conventions:




```
mult: push %rbp
      mov %rbp, %rsp
      mov a, %rdi
      mov b, %rsi

      mov r, 0
      mov i, a

L1:   cmp i, 0
      jg L2
      jmp L3
L2:   add r, b
      sub i, 1
      jmp L1
L3:   mov %rax, r
      mov %rsp, rbp
      pop %rbp
      ret
```

"spill -all" x64

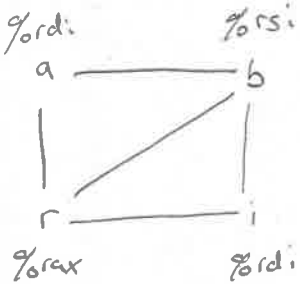
Stack layout:




```
mult: push %rbp
      mov %rbp, %rsp
      mov [%rbp -8], %rdi
      mov [%rbp-16], %rsi

      mov [%rbp -24], 0
      mov %rax, [%rbp -8]
      mov [%rbp-32], %rax

L1:   cmp [%rbp-32], 0
      jg L2
      jmp L3
L2:   mov %rax, [%rbp-16]
      add [%rbp-24], %rax
      sub [%rbp-32], 1
      jmp L1
L3:   mov %rax, [%rbp - 24]
      mov %rsp, %rbp
      pop %rbp
      ret
```

"3-register" x64

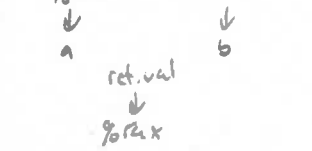
Interferesance graph:




```
mult:

      mov %rax, 0

L1:   cmp %rdi, 0
      jg L2
      jmp L3
L2:   add %rax, %rsi
      sub %rdi, 1
      jmp L1

L3:
      ret
```

"2-register" x64

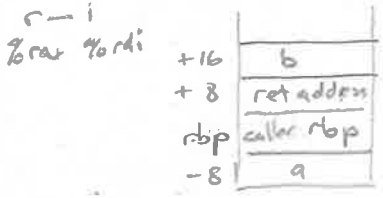
Calling conventions:



Graph coloring

spill a → [%rbp - 8]

spill b → [%rbp -16]




```
mult: push %rbp
      mov %rbp, %rsp
      mov [%rbp-8], %rdi

      mov %rax, 0
      mov %rdi, [%rbp-8]

L1:   cmp %rdi, 0
      jg L2
      jmp L3
L2:   add %rax, [%rbp-16]
      sub %rdi, 1
      jmp L1
L3:   mov %rsp, %rbp
      pop %rbp
      ret
```