

MO615B - Implementação de  
Linguagens II

e

MC900A - Tópicos Especiais em  
Linguagem de Programação

Prof. Guido Araujo

[www.ic.unicamp.br/~guido](http://www.ic.unicamp.br/~guido)

# Aula 6

# *Def-Use Chain*

- Liga cada *definição* aos *usos* que alcança
- Pode ser calculada a partir das *use-def chains* (mais comum). *Use-def chains* são calculadas a partir de *reaching definitions*.
- Pode ser definida como um DFA com as mesmas equações de *live-variable analysis*

# *Def-Use Chain*

- $OUT[B]$ : usos alcançáveis a partir do final de B
- $use_B$ : usos  $u_i(x)$  que aparecem antes de qualquer definição não ambigua à variável  $x$ .
- $def_B$ : todos os usos  $u_i(x)$  para as variáveis  $x$  que forem definidas no bloco básico.
- Equações e algoritmo são idênticos.

# Alocação Global de Registradores

# Grafo de Interferência

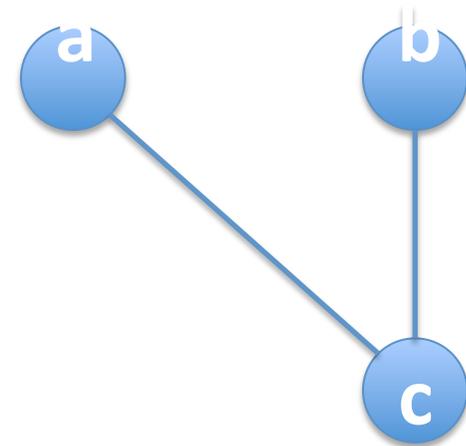
- Informa quais variáveis estão vivas em um mesmo ponto do programa.
- Construído a partir da análise de longevidade.
- Utilizado para alocação de registradores.
- Interferência: ocorre quando a e b não podem ocupar o mesmo registrador
- Outras formas de interferência:
  - Ex: Quando a é gerado por uma instrução que não pode escrever no registrador r1. Neste caso a interfere com r1.

# Grafo de Interferência

- Representação:

Variáveis	a	b	c
a			x
b			x
c	x	x	

Matriz



Grafo de Interferência

# Grafo de Interferência

- Tratamento especial de cópias: é importante não criar falsas interferências entre a fonte e destino

```
t := s          ; cópia
...
x := op s      ; uso de s
...
y := op t      ; uso de t
```

- s e t estiverem vivas após a instrução de cópia
- Podemos aproveitar o mesmo registrador: Não criar interferência entre as duas.

# Grafo de Interferência

1. Definição de  $a$  que não seja move:

– *Live-out* =  $b_1, \dots, b_j$

- Adicione as arestas  $(a, b_1), \dots, (a, b_j)$ .

2. Moves  $a \leftarrow c$ :

– *Live-out* =  $b_1, \dots, b_j$

- Adicione as arestas  $(a, b_1), \dots, (a, b_j)$  para os  $b_i$ 's diferentes de  $c$ .

# Alocação de Registradores

- A RI e a seleção de instruções assumiram que o número de registradores era infinito
- Objetivo:
  - Atribuir registradores físicos (da máquina) para os temporários usados nas instruções.
  - Se possível, atribuir a fonte e o destino de operações de cópia (MOVES) para o mesmo registrador
    - Elimina operações inúteis!

# Alocação de Registradores

- Grafo de Interferência (IG):
  - Temos arestas entre  $t_1$  e  $t_2$  se eles não podem ocupar o mesmo registrador.
  - Estão vivas no mesmo ponto; ou
  - Restrições da arquitetura;
    - P. Ex:  $a = a + b$  não pode ser atribuído ao  $r_{12}$
- O problema se transforma em um problema de coloração de grafos.

# Coloração do IG

- Queremos colorir o IG com o mínimo de cores possíveis, de maneira que nenhum par de nós conectados por uma aresta tenham a mesma cor.
  - Coloração de vértices
  - As cores representam os registradores
  - Se nossa máquina tem  $k$  registradores e encontrarmos uma  $k$ -coloração para o IG a coloração é uma alocação válida dos registradores.

# Coloração do IG

- E se não existir uma  $k$ -coloração?
  - Então teremos que colocar alguns dos temporários ou variáveis na memória.
  - Operação conhecida como *spilling*.
- Coloração de vértices é um problema NP-Completo.
  - Logo, alocação de registradores por coloração também é.
- Existe uma aproximação linear que traz bons resultados

# Coloração por Simplificação

- Principais fases

1. *Build*

2. *Simplify*

3. *Spill*

4. *Select*

# Coloração por Simplificação

## *1. Build:*

- Construir o IG
- Usa a análise de longevidade

# Coloração por Simplificação

## 2. *Simplify*:

- Heurística
- Suponha que o grafo  $G$  tenha um nó  $m$  com menos de  $k$  vizinhos
- $K$  é o número de registradores
- Faça  $G' = G - \{m\}$
- Se  $G'$  pode ser colorido com  $k$  cores,  $G$  também pode

# Coloração por Simplificação

## 2. Simplify:

- Leva a um algoritmo recursivo (pilha)
  - Repetidamente:
    - Remova nós de grau menor que  $K$
    - Coloque na pilha
  - Cada remoção diminui o grau dos nós em  $G$ , dando oportunidades para novas remoções

# Coloração por Simplificação

## 3. Spill:

- Em algum momento não temos um nó com grau  $< k$
- A heurística falha
- Temos que marcar algum nó para *spill*
- A escolha desse nó é também uma heurística
  - Nó que reduza o grau do maior número de outros nós
  - Nó com menor custo relacionado as operações de memória

# Spill em Potencial

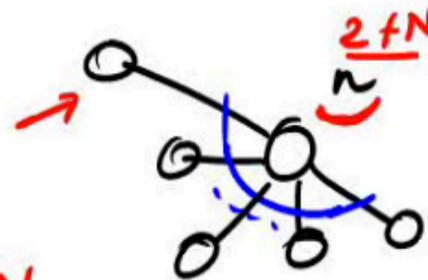
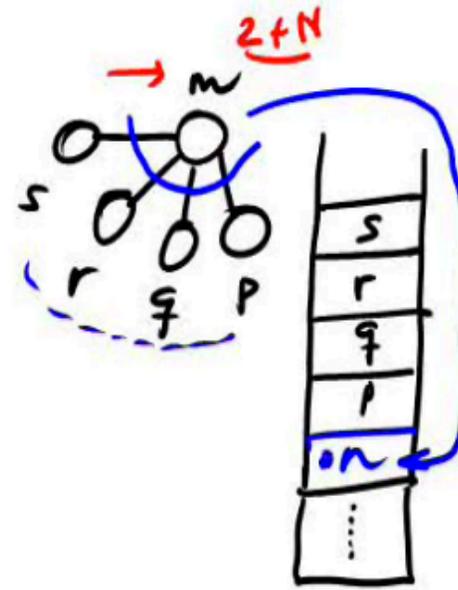
- SPILL  $k=4$ 
  - SELECT  $m$  & PUSH
  - $\text{degree}(m) = 4 > k$

$$\underline{m} := \underline{m} + 1$$

```

for( $i=0; i < N; i++$ ) {
     $s += a[i] + \underline{m};$ 
}
    
```

$$\frac{\sum \text{uses}(i)}{\text{degree}(i)} = \frac{2+N}{5}$$



# Coloração por Simplificação

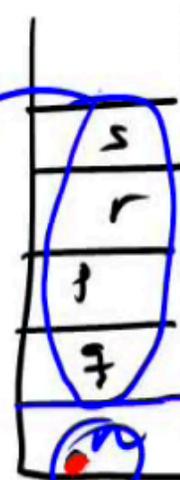
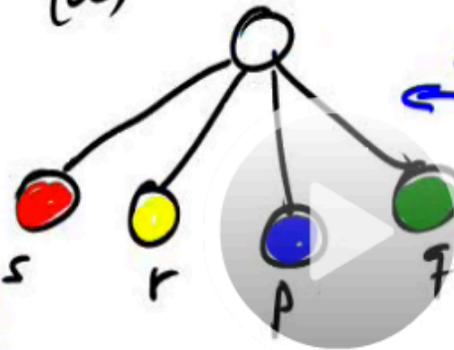
## 4. Select:

- Atribui as cores
- Reconstrói o grafo  $G$  adicionando os nós na ordem determinada pela pilha
- Quando adicionamos um nó, devemos ter uma cor para ele dado o critério de seleção usado para remover
- Isso não vale para os nós empilhados marcados como spill
  - Se todos os vizinhos já usarem  $k$  cores, não adicionamos no grafo
  - Continua o processo

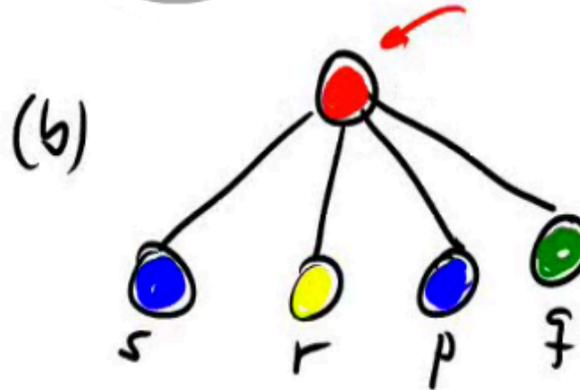
# Select Color

- SELECT  $k=4$   
 - POP & ASSIGN COLOR

$n := n + 1$  (a)  $n$



$\rightarrow t_{20} := M[n]^s$   
 $\rightarrow t_{20} := t_{20} + 1$   
 $\rightarrow M[n] := t_{20}$



# Coloração por Simplificação

## 5. Start Over:

- Pode ser que o *Select* não consiga atribuir uma cor a algum nó
- Reescrever o código para gravar o valor na memória após cada definição e ler da memória antes de cada uso.
- Isso gera novos temporários
  - Com *live ranges* mais curtas
- O algoritmo é repetido desde a construção do IG
- O processo acaba quando *Select* tiver sucesso para todos os vértices

# Exemplo

- Suponha que temos 4 registradores

*live-in:* k j

g := mem[j+12]

→ h := **k** - 1

f := g \* **h**

e := mem[j+8]

m := mem[j+16]

b := mem[f]

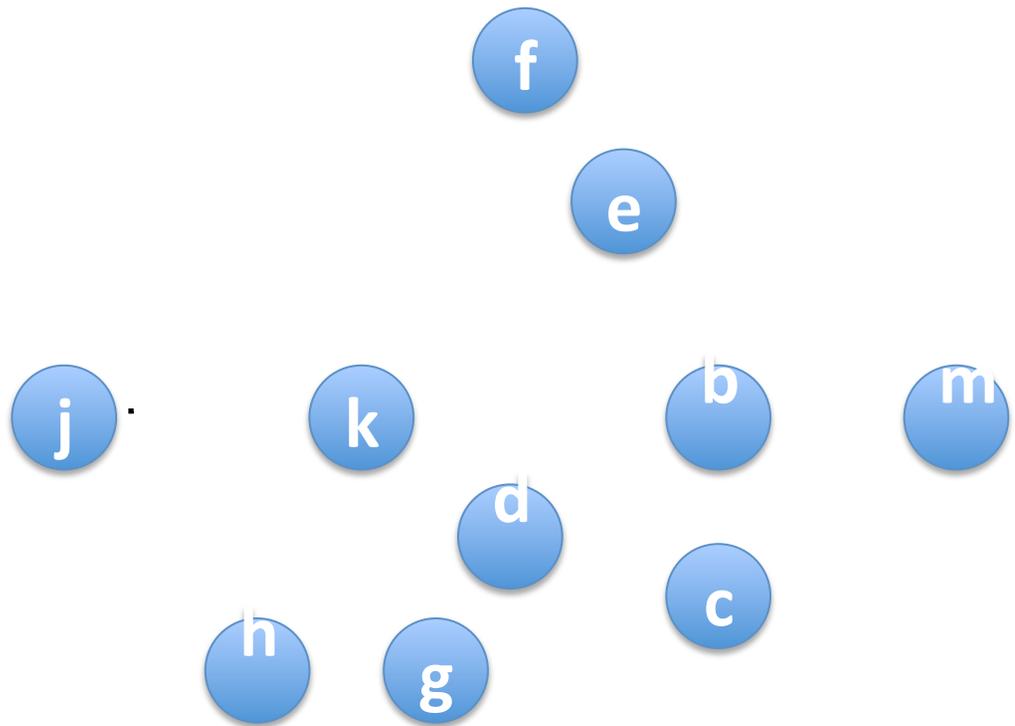
c := e + 8

d := c

k := m + 4

j := b

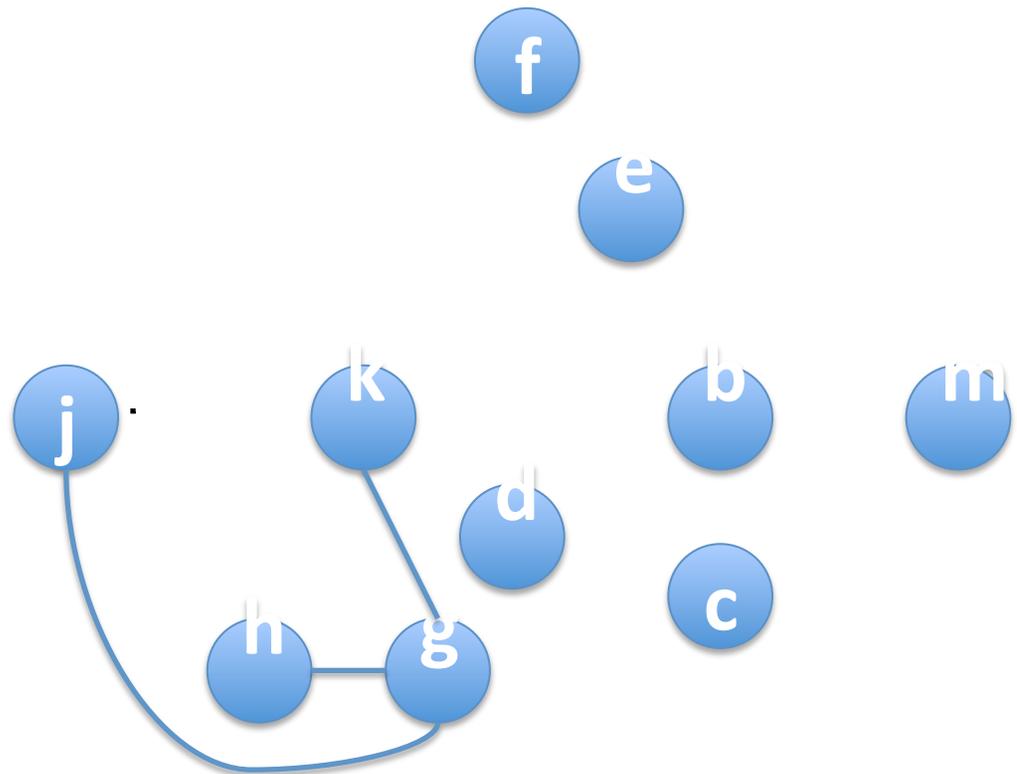
*live-out:* d k j



# Exemplo

- Suponha que temos 4 registradores

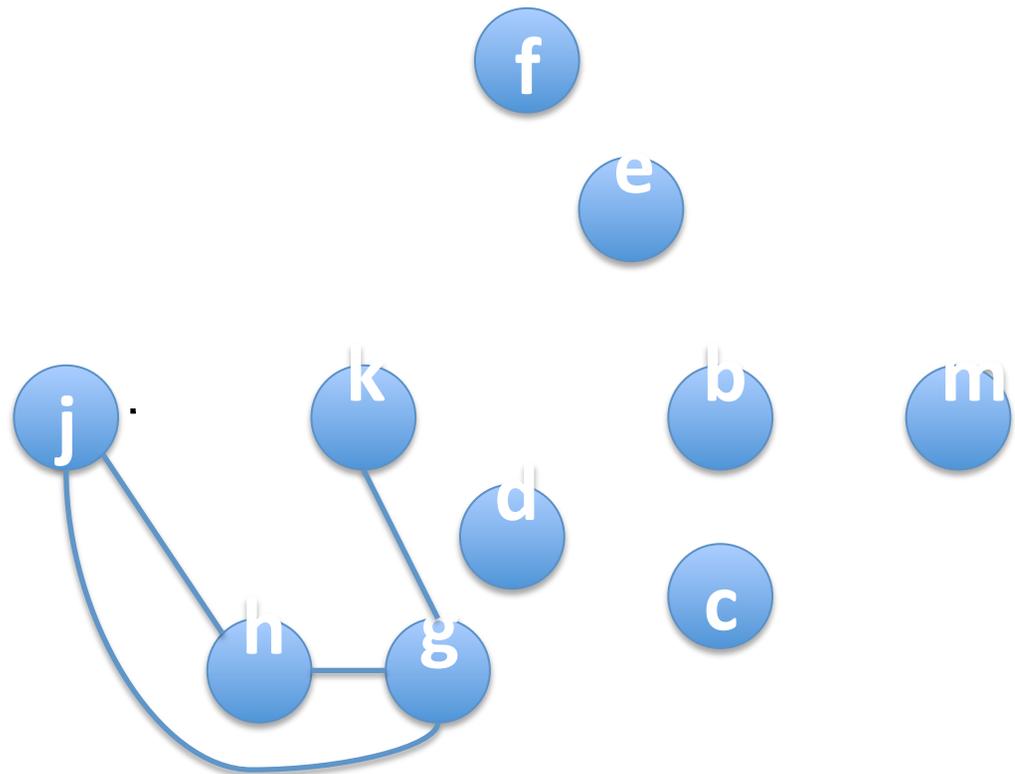
```
live-in: k j
  g := mem[j+12]
  h := k - 1
→ f := g * h
  e := mem[j+8]
  m := mem[j+16]
  b := mem[f]
  c := e + 8
  d := c
  k := m + 4
  j := b
live-out: d k j
```



# Exemplo

- Suponha que temos 4 registradores

```
live-in: k j  
g := mem[j+12]  
h := k - 1  
f := g * h  
→ e := mem[j+8]  
m := mem[j+16]  
b := mem[f]  
c := e + 8  
d := c  
k := m + 4  
j := b  
live-out: d k j
```

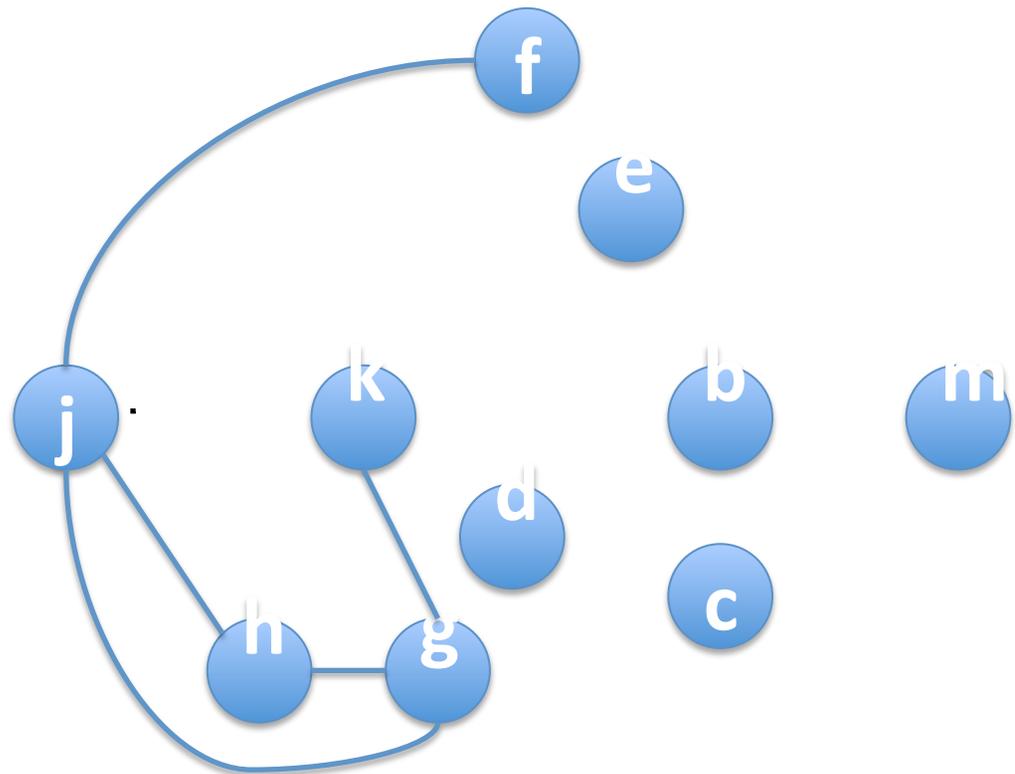


# Exemplo

- Suponha que temos 4 registradores

```
live-in: k j  
g := mem[j+12]  
h := k - 1  
f := g * h  
e := mem[j+8]  
→ m := mem[j+16]  
b := mem[f]  
c := e + 8  
d := c  
k := m + 4  
j := b
```

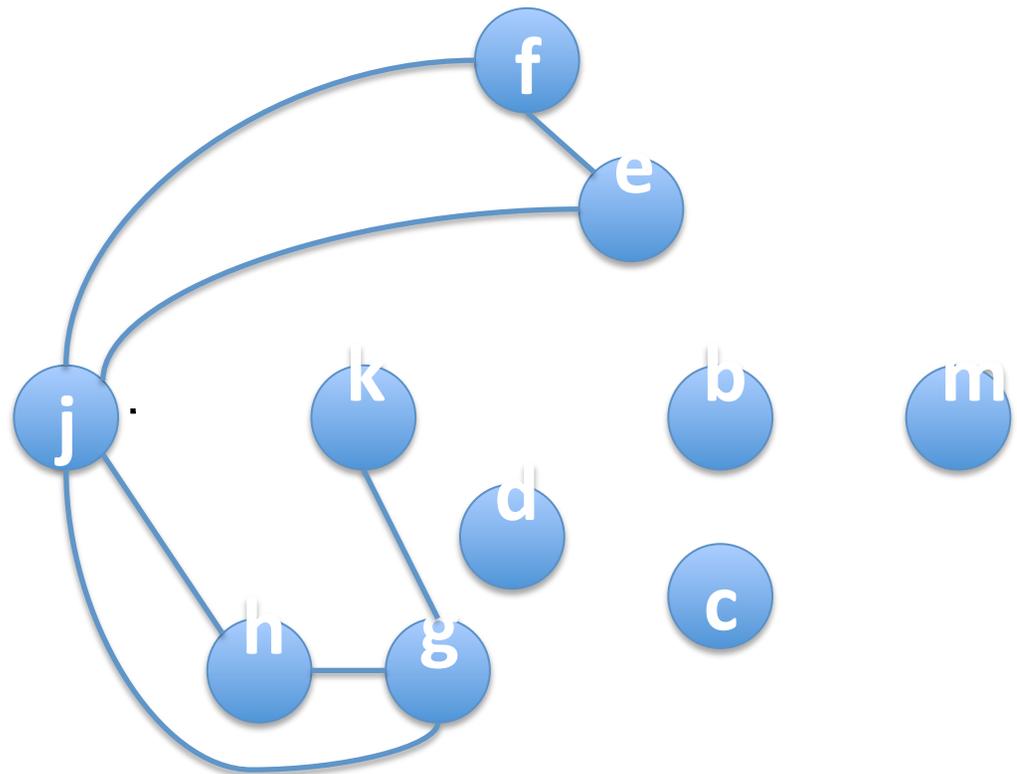
```
live-out: d k j
```



# Exemplo

- Suponha que temos 4 registradores

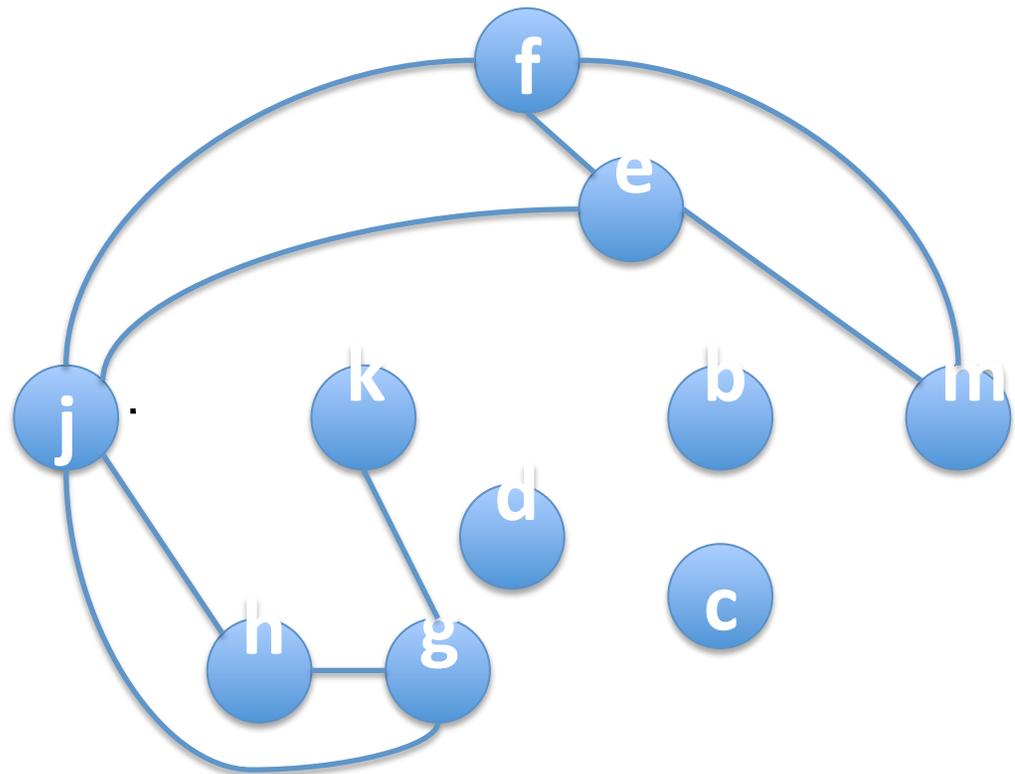
```
live-in: k j
g := mem[j+12]
h := k - 1
f := g * h
e := mem[j+8]
m := mem[j+16]
→ b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live-out: d k j
```



# Exemplo

- Suponha que temos 4 registradores

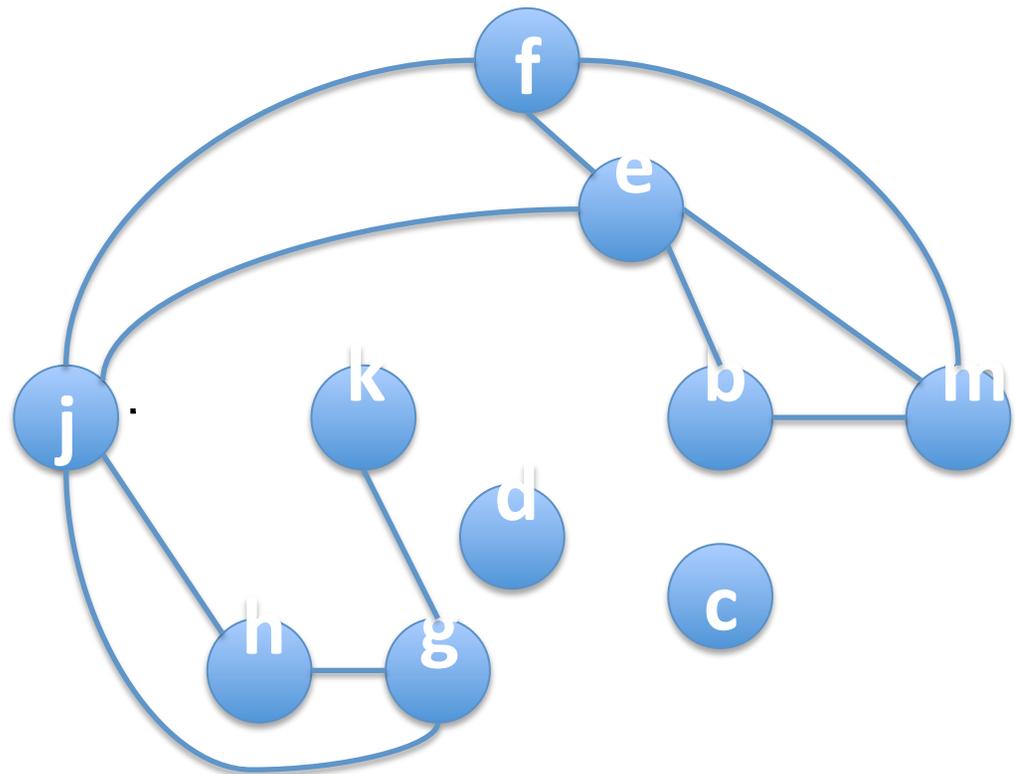
```
live-in: k j
g := mem[j+12]
h := k - 1
f := g * h
e := mem[j+8]
m := mem[j+16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
live-out: d k j
```



# Exemplo

- Suponha que temos 4 registradores

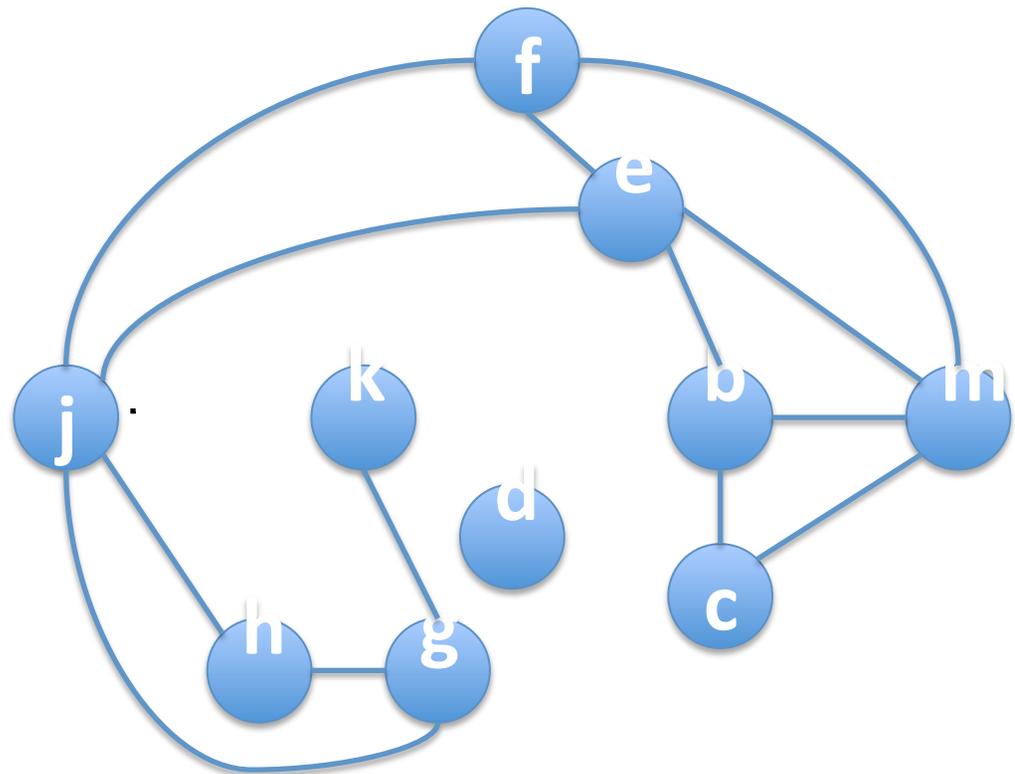
```
live-in: k j  
g := mem[j+12]  
h := k - 1  
f := g * h  
e := mem[j+8]  
m := mem[j+16]  
b := mem[f]  
c := e + 8  
→ d := c  
k := m + 4  
j := b  
live-out: d k j
```



# Exemplo

- Suponha que temos 4 registradores

```
live-in: k j
g := mem[j+12]
h := k - 1
f := g * h
e := mem[j+8]
m := mem[j+16]
b := mem[f]
c := e + 8
d := c
→ k := m + 4
j := b
live-out: d k j
```

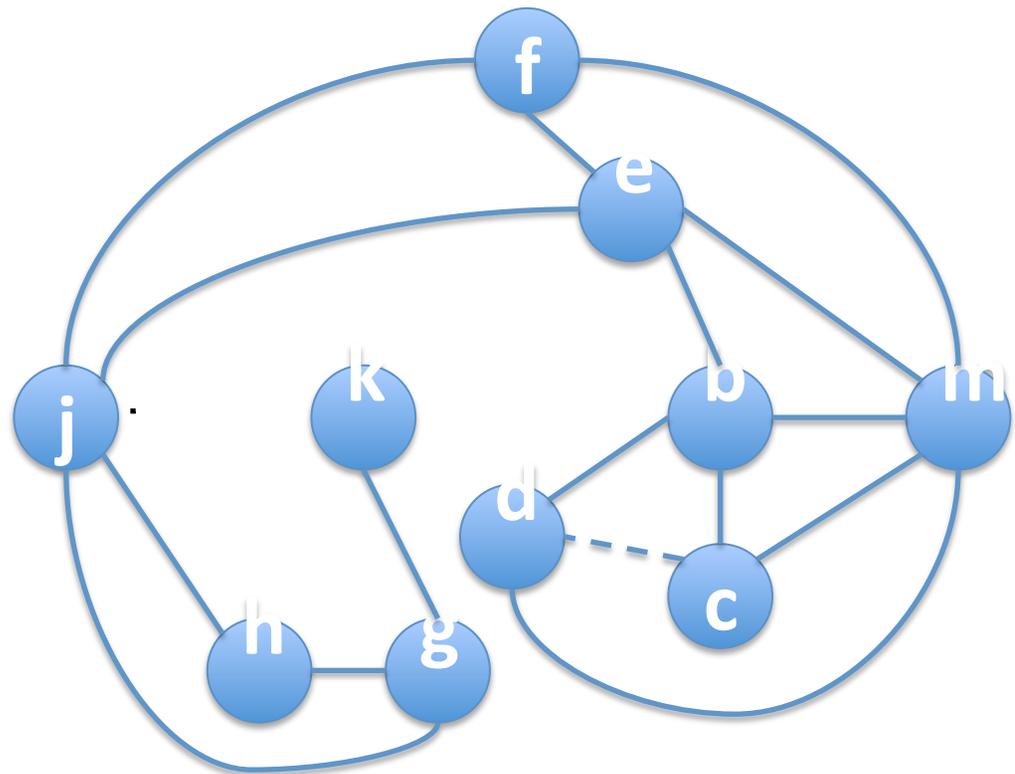


# Exemplo

- Suponha que temos 4 registradores

```
live-in: k j  
g := mem[j+12]  
h := k - 1  
f := g * h  
e := mem[j+8]  
m := mem[j+16]  
b := mem[f]  
c := e + 8  
d := c  
k := m + 4  
→ j := b
```

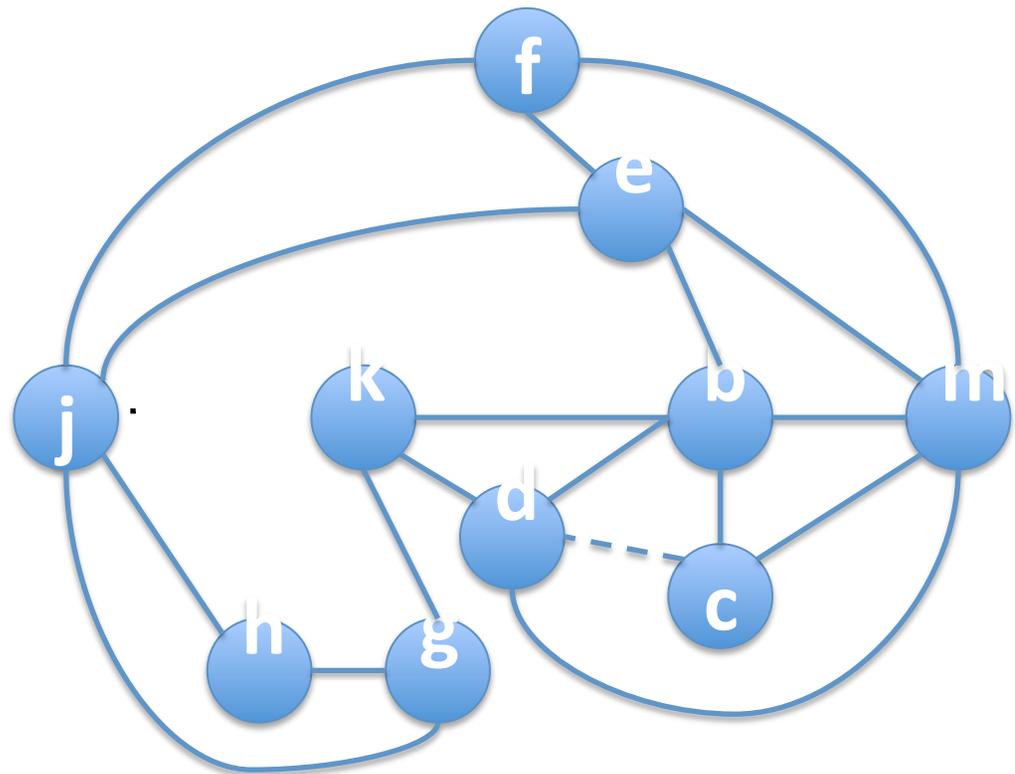
```
live-out: d k j
```



# Exemplo

- Suponha que temos 4 registradores

```
live-in: k j  
g := mem[j+12]  
h := k - 1  
f := g * h  
e := mem[j+8]  
m := mem[j+16]  
b := mem[f]  
c := e + 8  
d := c  
k := m + 4  
j := b
```



→ *live-out: d k j*

# Exemplo

- Suponha que temos 4 registradores

```
live-in: k j  
g := mem[j+12]  
h := k - 1  
f := g * h  
e := mem[j+8]  
m := mem[j+16]  
b := mem[f]  
c := e + 8  
d := c  
k := m + 4  
j := b  
live-out: d k j
```

