



Universidade Federal do Ceará - Campus de Quixadá

**TRABALHO II**  
**Compiladores**  
**Prof. Lucas Ismaily**

**INFORMAÇÕES IMPORTANTES**

O Trabalho II contém três opções para o aluno escolher. As opções são mutuamente exclusivas, isto é, o aluno deve escolher **exatamente uma opção**. A data máxima de entrega do trabalho é **20/09/2024**. Porém, recomendo fortemente que entreguem antes, para evitar imprevistos. **Atenção:** findado o prazo de envio, todos os grupos que não enviaram receberão automaticamente nota **zero**. A entrega será **somente** via e-mail ([ismailybf@ufc.br](mailto:ismailybf@ufc.br), assunto Trabalho II - Compiladores), numa pasta zipada contendo todos os arquivos, e se preciso, instrução para execução.

**Trabalho pode conter no máximo 5 alunos.** Sejam honestos com vocês e comigo. Qualquer fraude será punida com nota zero para todos os envolvidos.

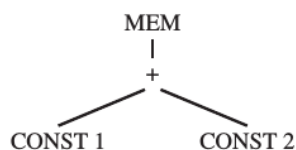


**OPÇÃO 1 – Seleção de instrução**

**Compiladores**

**Prof. Lucas Ismaily**

1. (3,0 pontos) Implemente um algoritmo que recebe como entrada um conjunto de instruções no formato linear e o imprime em formato de árvore. Por exemplo, se a entrada for MEM(+ (CONST 1, CONST 2)), sua saída deve ser algo parecido com a figura a seguir. **Nota:** é algo parecido, pode inclusive, usar o próprio terminal para imprimir uma estrutura nesse formato.



2. (3,5 pontos) Implemente a fase de Seleção de Instrução de um compilador considerando os padrões da arquitetura Jouette, conforme tabela abaixo.

—	$r_i$	TEMP
ADD	$r_i \leftarrow r_j + r_k$	<pre>       +      / \     /   \   </pre>
MUL	$r_i \leftarrow r_j \times r_k$	<pre>       *      / \     /   \   </pre>
SUB	$r_i \leftarrow r_j - r_k$	<pre>       -      / \     /   \   </pre>
DIV	$r_i \leftarrow r_j / r_k$	<pre>       /      / \     /   \   </pre>
ADDI	$r_i \leftarrow r_j + c$	<pre>       +      / \     /   \   CONST  +           / \          /   \         /     \        /       \       /         \      /           \     /             \    /               \   /                 \  /                   \ /                     \ CONST                CONST   </pre>
SUBI	$r_i \leftarrow r_j - c$	<pre>       -      / \     /   \    /     \   /       \  /         \ /           \ CONST        CONST   </pre>
LOAD	$r_i \leftarrow M[r_j + c]$	<pre>       MEM      MEM      MEM      MEM  +        +        +        +      / \      / \      / \      / \     /   \    /   \    /   \    /   \    /     \  /     \  /     \  /     \   /       \ /       \ /       \ /       \  /         \ /         \ /         \ /         \ /           \ /           \ /           \ /           \ CONST        CONST        CONST        CONST   </pre>
STORE	$M[r_j + c] \leftarrow r_i$	<pre>       MOVE      MOVE      MOVE      MOVE       / \      / \      / \      / \      /   \    /   \    /   \    /   \     /     \  /     \  /     \  /     \    /       \ /       \ /       \ /       \   /         \ /         \ /         \ /         \  /           \ /           \ /           \ /           \ /             \ /             \ /             \ /             \ CONST          CONST          CONST          CONST   </pre>
MOVEM	$M[r_j] \leftarrow M[r_i]$	<pre>       MOVE       / \      /   \     /     \    /       \   /         \  /           \ /             \ MEM            MEM                    </pre>



**Nota:** Você deve utilizar o algoritmo baseado em Programação Dinâmica. Para computar os custos de cada instrução, considere os seguintes:

- I. A instrução TEMP (a primeira) tem custo zero, ou seja, um simples carregamento para um registrador, por exemplo,  $R_i \leftarrow \text{TEMPO } X$ , tem custo zero.
- II. O custo de uma instrução MOVEM é dois, ou seja, carregar da memória e atribuir à memória, por exemplo,  $M[R_1] \leftarrow M[R_2]$ , tem custo dois.
- III. Os custos das demais instruções são unitários.

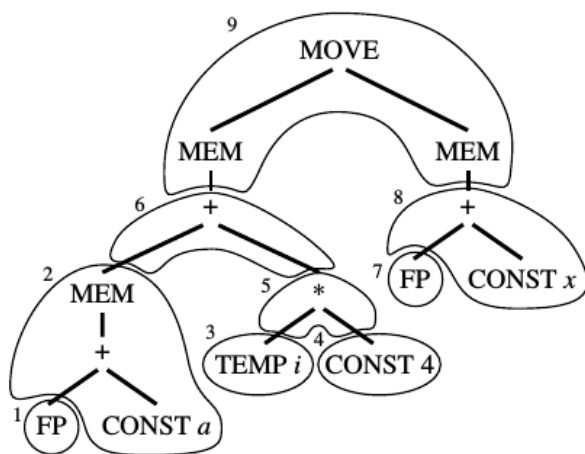
### Entrada

A entrada é composta por um conjunto de instruções no formato linear.

### Saída

Você deve imprimir os padrões selecionados e o custo da solução gerada.

3. (3,5 pontos) Implemente uma função que recebe um conjunto de padrões (Questão 2) e exibe o código equivalente. Por exemplo:



2	LOAD	$r_1 \leftarrow M[\text{fp} + a]$
4	ADDI	$r_2 \leftarrow r_0 + 4$
5	MUL	$r_2 \leftarrow r_i \times r_2$
6	ADD	$r_1 \leftarrow r_1 + r_2$
8	ADDI	$r_2 \leftarrow \text{fp} + x$
9	MOVEM	$M[r_1] \leftarrow M[r_2]$



**OPÇÃO 2 – Alocação de registradores**

**Compiladores**

**Prof. Lucas Ismaily**

1. (3,0 pontos) Implemente a Análise de Longevidade.

**Entrada**

A entrada é composta por um grafo de fluxo de controle no seguinte formato: a primeira linha da entrada é composta por dois inteiros **N** e **M**, sendo **N** o número do bloco básico e **M** a quantidade de linhas de códigos de três endereços. As **M** linhas seguintes são códigos de três endereços. Após as **M** linhas, segue uma última linha contendo uma sequência de inteiros, representando os vértices sucessivos do bloco **N**, caso **N** não tenha sucessores, é dado o valor zero.

**Saída**

Para cada bloco básico do grafo de entrada, seu programa deve computar os conjuntos **IN** e **OUT**. **Nota: nada será impresso na tela.**

**Exemplo**

**Entrada**

```
1 2
a = a+c
b = 4-a
2
2 1
b=20*c
3
3 2
d = a+b
b = 0
0
```

**Saída**

<b>OUT[ 1 ] = { a , c }</b>	<b>IN[ 1 ] = { a , c }</b>
<b>OUT[ 2 ] = { a , b }</b>	<b>IN[ 2 ] = { a , c }</b>
<b>OUT[ 3 ] = { }</b>	<b>IN[ 3 ] = { a , b }</b>

2. (3,0 pontos) Implemente o Grafo de Interferência utilizando a Análise de Longevidade.

**Entrada**

A entrada é composta pela saída da Análise de Longevidade, realizada na Questão 2.

**Saída**

Seu programa deve produzir um Grafo de Interferência. **Nota: nada será impresso na**



tela.

3. (4,0 pontos) Implemente a Alocação de Registradores utilizando o Grafo de Interferência. Considere uma arquitetura com 4 registradores.

#### Entrada

A entrada é composta por um Grafo de Interferência, representado por uma matriz  $n \times n$ , sendo  $n$  o número de variáveis do problema.

#### Saída

Para cada entrada, seu programa deve imprimir a quantidade de variáveis, quantas variáveis cada registrador irá guardar e quantas irão para memória (*spill*). Use a saída no seguinte formato: a primeira linha é a quantidade de variáveis do programa; a segunda linha é a quantidade de variáveis que o registrador que guarda o maior número de variáveis; a terceira é a quantidade de variáveis que o segundo registrador que guarda mais variáveis, e assim por diante. Ou seja, será impresso em ordem decrescente de quantidade de variáveis por registrador (que são quatro). A quinta linha contém quantas variáveis foram para memória, coloque 0 se nenhuma variáveis foi para a memória.

#### Exemplo de saída

```
20
6
5
4
4
1
4
1
1
1
1
1
1
0
```



**OPÇÃO 3 – Análises de Fluxos**

**Compiladores**

**Prof. Lucas Ismaily**

1. (3,0 pontos) Implemente a Análise de Longevidade.
2. (3,5 pontos) Implemente o algoritmo de fluxo de dados Reaching Definitions (definições alcançantes).
3. (3,5 pontos) Implemente o algoritmo de fluxo de dados Available Expressions (Expressões Disponíveis).

Para todas as questões considere o seguinte formato de entrada.

**Entrada**

A entrada é composta por um grafo de fluxo de controle no seguinte formato: a primeira linha da entrada é composta por dois inteiros **N** e **M**, sendo **N** o número do bloco básico e **M** a quantidade de linhas de códigos de três endereços. As **M** linhas seguintes são códigos de três endereços. Após as **M** linhas, segue uma última linha contendo uma sequência de inteiros, representando os vértices sucessivos do bloco **N**, caso **N** não tenha sucessores, é dado o valor zero.

**Saída**

Para cada bloco básico do grafo de entrada, seu programa deve computar os conjuntos **IN** e **OUT** de cada questão.