Universidade Estadual de Londrina

DC Departamento de Computação

1COP029 - Trabalho T3: Alocador de Registradores

Utilizando os conhecimentos adquiridos na disciplina até o momento, implemente um programa que realiza a leitura de arquivos que contêm uma lista de tempos de vida e tenta alocar registradores através da técnica de Linear Scan. Cada arquivo de entrada possui o seguinte formato:

```
K=18
vr1 --> 1 6
vr2 --> 2 8
vr3 --> 4 8
vr4 --> 6 15
vr5 --> 8 17
vr6 --> 9 20
vr7 --> 10 15
vr8 --> 11 12
vr9 --> 13 19
vr10 --> 15 20
vr11 --> 16 21
vr12 --> 17 18
vr13 --> 19 21
vr14 --> 20 21
```

Descrição do Arquivo de Entrada

A linha `K=18` indica o total de registradores físicos que devem ser considerados no processo de alocação. Como no exemplo `K` possui o valor 18, isto quer dizer que existem 18 registradores físicos, numerados de `0` até `17` para realizar o processo de alocação.

As linhas iniciadas por `vr`X, onde X é um número dado pela expressão regular `[0-9]+`, seguido do símbolo `-->` indicam os registradores virtuais a serem alocados e seus respectivos tempos de vida. O tempo de vida de um registrador virtual corresponde ao intervalo entre sua primeira definição e sua última utilização. Considere, por exemplo, a seguinte linha:

```
vr1 --> 1 6
```

Na linha acima é apresentado o tempo de vida do registrador virtual `vr1`. Sua primeira definição ocorre na linha `1` e sua última utilização ocorre na linha `6`. Os registradores virtuais `vr1`, `vr2` e `vr3` não podem ser alocados ao mesmo registrador físico, pois seus tempos de vida interferem entre si, isto é, possuem interseção. Observe que se um tempo de vida se encerra na mesma linha que outro começa eles não interferem entre si, podendo assim serem alocados ao mesmo registrador físico. Desta forma, os registradores

Versão deste texto: V. 2025.01

virtuais `vr1` e `vr4` podem ser alocados ao mesmo registrador físico, visto que o tempo de vida de `vr1` se encerra na linha `6` e o tempo de vida de `vr4` começa na linha `6`.

Programa Alocador e Saída Esperada

A sua implementação deve tentar alocar cada um dos tempos de vida com o número de registradores físicos que é fornecido na linha que contém o valor de `K` até o valor 2 de forma decrescente. Assim, se o valor de `K` for 18, o seu programa deve tentar alocar os tempos de vida utilizando 18 registradores; depois deve tentar alocar os tempos de vida com 17 registradores; depois com 16 registradores e assim sucessivamente até que apenas 2 registradores sejam utilizadas.

Observe que ao tentar alocar os tempos de vida com:

- `K` registradores: você poderá utilizar os registradores físicos de `0` até `K-1`;
- `K-1` registradores: você poderá utilizar os registradores físicos de `0` até `K-2`;
- `K-2` registradores: você poderá utilizar os registradores físicos de `0` até `K-3`;
- ...
- `4` registradores: você poderá utilizar os registradores físicos de `0` até `3`;
- `3` registradores: você poderá utilizar os registradores físicos de `0` até `2`;
- `2` registradores: você poderá utilizar os registradores físicos de `0` até `1`.

De forma que o resultado seja padronizado, algumas regras devem ser seguidas durante a alocação. O programa deve apresentar as seguintes características:

- O programa deve iterar sobre um vetor contendo os tempos de vida ordenados de acordo com o tempo inicial de forma ascendente. Observe que a lista de tempos de vida fornecida no arquivo de entrada já está ordenada nesta ordem;
- Em cada iteração é mantida uma lista de tempos de vida que interferem com o atual e foram atribuídos a um registrador. Para cada novo tempo de vida, a lista é percorrida para remover tempos de vida "expirados", que não interferem mais com o atual;
- Em cada iteração o tempo de vida deve ser alocado ao registrador físico de menor número disponível e adicionado à lista de tempos de vida ativos;
- Caso não restem registradores físicos disponíveis é necessário realizar *spill*. Nessa situação é necessário escolher uma variável para ser armazenada na memória entre o tempo de vida da iteração atual e as presentes na lista de tempos de vida ativos na iteração. Deve ser selecionado para *spill* o registrador virtual com o maior número de linha da última utilização. Caso um ou mais registradores virtuais apresentarem intervalos iguais, selecione o tempo de vida com menor intervalo de tempo de vida, ou seja, a linha da última utilização menos a linha da primeira definição. Caso ainda assim seja necessário decidir entre 2 ou mais candidatos, selecione para *spill* o registrador virtual alocado mais recentemente.

A saída do seu programa deve indicar se cada um dos tempos de vida foi alocado com sucesso ou se gerou um *spill*. Como exemplo, considere que o seu programa recebeu o seguinte arquivo de entrada:

```
K=4
vr1 --> 1 3
vr2 --> 2 4
vr3 --> 3 6
vr4 --> 4 8
vr5 --> 5 9
vr6 --> 6 9
vr7 --> 7 8
vr8 --> 8 10
vr9 --> 9 11
vr10 --> 10 11
```

Esta entrada possui 4 registradores físicos. Você deve tentar alocar os tempos de vida utilizando primeiramente 4 registradores (`0` até `3`), depois 3 registradores (`0` até `2`) e por final 2 registradores (`0` e `1`). Para a entrada apresentado como exemplo, o seu programa deve gerar a seguinte saída:

```
K = 4

vr1: 0
vr2: 1
vr3: 0
vr4: 1
vr5: 2
vr6: 0
vr7: 3
vr8: 1
vr9: 0
vr10: 1
----------------------------------------
K = 3

vr1: 0
vr2: 1
vr3: 0
vr4: 1
vr5: 2
vr6: SPILL
vr7: 0
vr8: 0
vr9: 1
vr10: 0
----------------------------------------
K = 2

vr1: 0
vr2: 1
vr3: 0
vr4: 1
vr5: SPILL
```

```
vr6: SPILL
vr7: 0
vr8: 1
vr9: 0
vr10: 1
---------------------------------------
---------------------------------------
K = 4: Successful Allocation
K = 3: SPILL on interation(s): 6
K = 2: SPILL on interation(s): 4, 6
```

Na saída apresentada existe uma linha que indica o total de registradores físicos disponíveis (valor de `K`), que no arquivo de entrada é 4. A seguir, a saída apresenta as alocações dos tempos de vida com 4 registradores, depois 3 registradores, e finalmente 2 registradores. A cada alocação é indicado o valor de `K` sendo utilizado.

Se um tempo de vida for alocado com sucesso, a saída indica o registrador físico associado ao registrador virtual. Caso um registrador virtual sofra *spill* é indicado na saída, como por exemplo, como ocorreu na linha que contém o texto `vr6: SPILL`, entre outras.

Ao final de todas as tentativas, a saída contém um pequeno resumo de todo o processo de alocação. No exemplo de saída apresentado, este resumo contém o seguinte texto:

```
K = 4: Successful Allocation
K = 3: SPILL on interation(s): 6
K = 2: SPILL on interation(s): 4, 6
```

Neste resumo é mostrado o resultado da alocação para cada valor de `K`. Quando a alocação é bem sucessida a mensagem `Successful Allocation` é mostrada. Caso contrário é mostrada a mensagem indicando as iterações nas quais ocorreram *spill*, como a mensagem `K = 2: SPILL on interation(s): 4, 6`. A primeira iteração é considerada a iteração 0.

Implementação e Critérios de Correção

O seu programa deve ser implementado exclusivamente em `C` ou `C++`.

O seu programa deverá ler a entrada padrão do sistema e escrever a sua saída na saída padrão do sistema. Em sua implementação você pode usar as ferramentas `flex` e `bison` se desejar.

A saída gerada pelo seu programa será comparada com uma saída padrão e deve possuir **EXATAMENTE** o **MESMO** conteúdo para ser considerada correta.

Recomendações

Por favor, evite escrever código da seguinte forma:

```
for(int i = 0; ...)
{
    ...
}
```

onde uma variável local está sendo declarada dentro de um comando. Se ainda sim quiser utilizar tal estilo de programação, não se esqueça de colocar no `Makefile` as devidas opções para que o compilador aceite tal construção, pois nem todos os compiladores a aceitam por padrão.

Em geral a opção `-std=c99` é o suficiente para que tal construção seja aceita e compilada sem maiores problemas. Sua utilização, em geral, é da seguinte forma:

```
$gcc -std=c99 teste.c -o teste
```

Se você programar utilizando `C++ 11`, por favor, utilize também a opção adequada do compilador para habilitar a compilação de tal versão do `C++`.

IMPORTANTE: Se ficou com alguma dúvida em relação a qualquer item deste texto, não hesite em falar com o professor da disciplina, pois ele está à disposição para sanar eventuais dúvidas, além do que, isso faz parte do trabalho dele.

SUGESTÃO: Espera-se que cada aluno gere sempre o mesmo resultado de saída, desta forma uma maneira de validar o trabalho é comparar a sua saída com a saída de um outro colega. Cada aluno possui a sua implementação, mas todas devem chegar ao mesmo resultado final. Se dois alunos tiverem resultados diferentes, isso implica que um deles (ou ambos) gerou um resultado incorreto, bastando analisar em mais detalhes o teste que gerou a divergência e assim corrigir qualquer problema de implementação.

Especificações de Entrega

O trabalho deve ser entregue no *AVA* em um arquivo .zip com o nome `linearscan.zip`. Este arquivo .zip deve conter somente os arquivos necessários à compilação, sendo que deve haver um `Makefile` para a geração do executável.

A entrega deve ser feita exclusivamente no *AVA* até a data/hora especificada. Não serão aceitas entregas atrasadas ou por outro meio que não seja o *AVA*.

Observação: o arquivo .zip não deve conter pastas, para que quando descompactado, os fontes do trabalho apareçam no mesmo diretório do .zip. O nome do executável gerado pelo `Makefile` deve ser `linearscan`.

O programa gerado deve ler os tempos de vida da entrada padrão do sistema e imprimir as saídas na saída padrão do sistema. Um exemplo de execução para uma entrada chamada `entrada.txt` seria a seguinte:

```
$./linearscan < entrada.txt
```

IMPORTANTE: Arquivos ou programas entregues fora do padrão receberão nota **ZERO**. Entende-se como arquivo fora do padrão aquele que tenha um nome diferente de `linearscan.zip`, que contenha subpastas ou que não seja um `.zip` por exemplo. Entende-se como programa fora do padrão aquele que não contiver um `Makefile`, que apresentar erro de compilação, que não ler da entrada padrão, não imprimir na saída padrão ou o nome do executável for diferente de `linearscan`, por exemplo. Uma forma de verificar se seu arquivo ou programa está dentro das especificações é testar o mesmo com o `script` de testes que é fornecido no *AVA*. Se o seu arquivo/programa **não** funcionar com o `script`, significa que ele está **fora** das especificações e, portanto, receberá nota **ZERO**.