Universidade Federal do Rio Grande do Norte
DIM0127 – Arquitetura de Computadores
Descrição dos Trabalhos da Disciplina

Item 1. Grupos de NO MÁXIMO 2 alunos.

Item 2. Cada trabalho poderá ser selecionado por, no máximo, 3 grupos.

Item 3. Todos os integrantes do grupo serão avaliados individualmente, e devem saber responder as perguntas relativas a todo o trabalho.

Item 4. A entrega do trabalho final é considerada completa com a entrega de três itens:

1. Implementação
2. Apresentação
3. Relatório final

A NÃO entrega de um dos itens listados acima implica em desclassificação do grupo e nota **0,0** para todos os integrantes.

Item 5. O relatório final deve conter:

a. Introdução: descrição do trabalho e sua contextualização com o estudo de arquiteturas de computadores.

b. Solução Implementada: detalhes sobre a implementação.

c. Análise de resultados: resultados de simulação, tempo, área e demais aspectos a serem considerados devem ser reportados e analisados. Gráficos de análise, como barras, pizza, dentre outros podem (e devem) ser utilizados para auxiliar na análise dos resultados.

d. Conclusão: breve descrição do que foi implementado com análise, também breve, dos resultados.

e. Todo material utilizado: livros, links, códigos, e qualquer outro material deve ser descrito da seção de Referências.

f. Descrição de organização do código: pastas, subpastas, etc.

g. Descrição de como fazer para compilar e executar o projeto

Item 6. Casos de plágio e cópias não referenciadas devidamente serão tratados com reprovação dos integrantes do grupo e denúncia aos órgãos competentes.

Item 7. Exceções e casos omissos devem ser tratados diretamente com o professor da disciplina.

Trabalho 1: Simulador de Árbitro de Barramento

Descrição: implementar um simulador de árbitro de barramento que recebe como entrada as solicitações de uso de barramento e a prioridade de cada solicitação e retorna como saída a ordem que as solicitações foram atendidas para as seguintes políticas:

1) Daisy Chaining

2) Prioridade fixa: a prioridade é definida pelo usuário

3) Prioridade rotativa: a prioridade é definida pelo usuário

4) Justiça: a prioridade e o tempo são definidos pelo usuário

Trabalho 2: Ferramenta de reordenamento de instruções e renomeação de registradores

Descrição: Implementar uma ferramenta que recebe como entrada a informação sobre a dependência de dados de todas as instruções (grafo de dependência de dados) e, realiza o reordenamento de instruções para reduzir essas dependências.

A ferramenta também deve ser capaz de renomear os registradores, considerando o conjunto de registradores do MIPS, para reduzir os conflitos causados por dependências falsas.

A saída da ferramenta são

1) As instruções reordenadas
2) As instruções reordenadas e renomeadas
3) O grafo de dependência de dados das instruções reordenadas.

Exemplo de entrada da ferramenta:

#inst	tipo_inst	dest	op1	op2	#inst_recebe_resultado
1	add	$s3	$s1	$s2	2,3
2	sub	$s4	$s3<-1	$s5	3
3	add	$t0	$s3<-1	$s4<-2	6,7
4	mult	lo	$s5	$s6	5
5	mov	$t1	lo<-5	$zero	
6	lw	$t2	100	$t0<-3	7
7	add	$t3	$t2<-6	$t0<-3	

Trabalho 3: Simulador de Memória Virtual

Descrição: implementar um simulador de memória virtual que realize acessos à memória, tanto leitura quanto escrita, com base no endereço virtual, tabela de páginas e acesso à endereço físico.

O simulador receberá como entrada um arquivo que conterá a sequência de endereços de memória acessados. Esses endereços estarão escritos como números hexadecimais, seguidos por uma letra R ou W, para indicar se o acesso foi de leitura ou escrita.

Ao iniciar o programa, será definido o tamanho da memória (em quadros - *frames*) para aquele programa e qual o algoritmo de substituição de páginas a ser utilizado. O programa deve, então, processar cada acesso à memória para atualizar os bits de controle de cada *frame*, detectar faltas de páginas (*page faults*) e simular o processo de carga e substituição de páginas. Durante todo esse processo, estatísticas devem ser coletadas, para gerar um relatório curto ao final da execução.

Forma de operação

O programa deverá ser iniciado com quatro argumentos:

virtual lru arquivo.log 4 128

Esse argumentos representam, pela ordem:

1. o algoritmo de substituição a ser usado (lru, fifo ou random);
2. o arquivo contendo a sequência de endereços de memória acessados (arquivo.log, nesse exemplo);
3. o tamanho de cada página/frame de memória, em kilobytes -- faixa de valores razoáveis: de 2 a 64;
4. o tamanho total da memória física disponível para o processo, também em kilobytes -- faixa de valores razoáveis: de 128 a 16384 (16 MB).

Formato da saída

Ao final da simulação, quando a sequência de acessos à memória terminar, o programa deve gerar um pequeno relatório, contendo:

- a configuração utilizada (definida pelos quatro parâmetros);
- o número total de acessos à memória contidos no arquivo;
- o número de *page faults*;
- o número de páginas ”sujas” que tiveram que ser escritas de volta no disco (lembrando-se que páginas sujas que existam no final da execução não precisam ser escritas).

Um exemplo de saída poderia ser da forma (valores completamente fictícios):

```
prompt> virtual lru arquivo.log 4 128
Executando o simulador...
Arquivo de entrada: arquivo.log
Tamanho da memoria: 128 KB
Tamanho das páginas: 4 KB
Tecnica de reposicao: lru
Paginas lidas: 520
Paginas escritas: 352
```

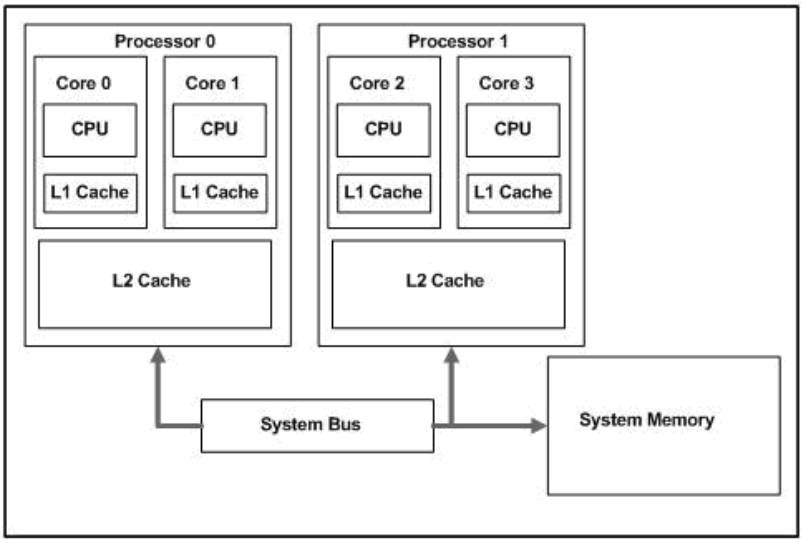
Formato do arquivo de entrada

Como mencionado anteriormente, cada linha contém um endereço de memória acessado, seguido das letras R ou W, indicando um acesso de leitura ou escrita, respectivamente. Por exemplo, as linhas a seguir foram retiradas de um dos arquivos utilizados:

```
0785db58 W
000652d8 R
0005df58 W
000652e0 R
0785db50 W
000652e0 R
31308800 R
00062368 R
```

Trabalho 4: Simulador de hierarquia de memória em Multicore

Descrição: o sistema de memória terá dois níveis de cache (L1 e L2) e a memória principal. Cada core terá uma cache L1. Uma cache L2 será compartilhada entre dois cores. A memória principal será compartilhada por todos os cores. Observar figura abaixo.



No simulador, o usuário informará quantos cores o sistema terá (somente quantidades múltiplas de 2 são permitidas). O sistema automaticamente cria as caches e a memória principal. Um arquivo, lido pelo simulador, carrega os dados na memória principal. O usuário informa o endereço da memória principal para leitura e o core que irá utilizar aquele dado. O sistema automaticamente carrega o dado na cache L2 e na cache L1 do respectivo core. O sistema também deve ser capaz de atualizar o dado, caso o core modifique esse valor, e atualizar todos os níveis da hierarquia imediatamente (*write-through*).

Atenção:

- Assumir que os dados são números inteiros
- O sistema deverá dar opção de ler ou alterar o dado, indicando qual core fará a operação
    - Em caso de leitura: o dado será carregado nas caches (se ainda não estiver)
    - Em caso de escrita: o dado será alterado e atualizado em todos os níveis da hierarquia

Trabalho 5: Implementar um jogo em Assembly-MIPS usando o MARS

Jogos possíveis: gênius, resta 1, pong

É obrigado usar o simulador de interface gráfica do MARS.