

Universidade Federal do Rio Grande do Norte
IMD0041 – Introdução a Organização e 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 na 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 três políticas diferentes:

- 1) Daisy Chaining
- 2) Prioridade: a prioridade é definida pelo usuário
- 3) Justiça: a prioridade e o tempo são definidos pelo usuário

Trabalho 2: Simulador de Associatividade Adaptativa

Descrição: Implementar um simulador de acesso à memória cache que recebe como entrada os endereços a serem acessados. Esses endereços estarão escritos como números hexadecimais. O simulador irá buscar a palavra na cache, com o mapeamento que estiver sendo utilizado no momento. Se a palavra não for encontrada (miss), ela será escrita na cache.

Cada vez que a palavra não for encontrada, o contador de miss é atualizado. Quando a taxa de miss estiver acima de $x\%$, é necessário trocar o mapeamento.

Exemplo: o sistema inicia a execução com uma memória com mapeamento direto. Se a taxa de miss aumentar ($>x\%$), a associatividade é mudada para conjunto de 4 blocos, ou para totalmente associativa.

O simulador deve permitir visualizar o estado da memória a cada leitura e escrita.

O limite para a taxa de miss é definido pelo usuário.

Trabalho 3: 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 4: 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 conterà 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
```

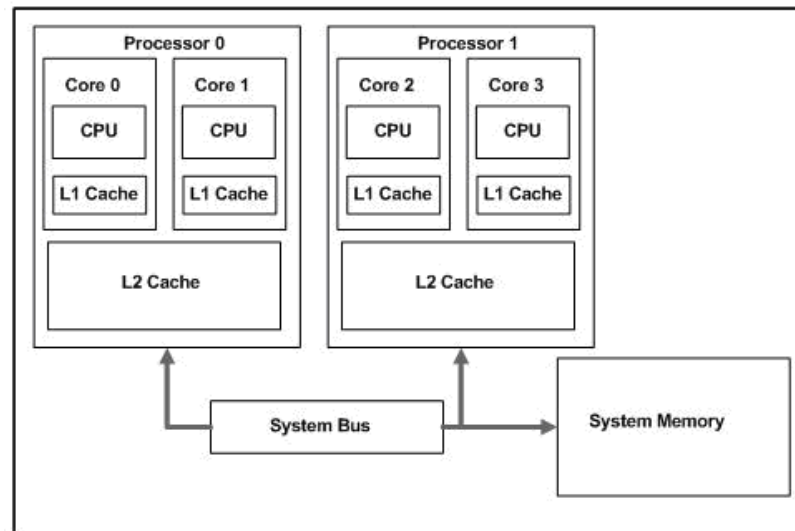
Formato do arquivo de entrada

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 5: 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 6: Implementar um modelo de Coerência de Cache baseada em Snoop

Descrição: Considerando o cenário da figura abaixo, criar a simulação de um sistema com caches, barramento, memória e um componente conectado às caches que realiza operações de coerência de cache. Nesta implementação, o mecanismo Snoop

implementa o protocolo MESI que identifica quatro estados possíveis para uma linha da cache:

M (Modificada): A linha foi modificada pela cache e encontra-se “suja”. Nenhuma outra cache a possui.

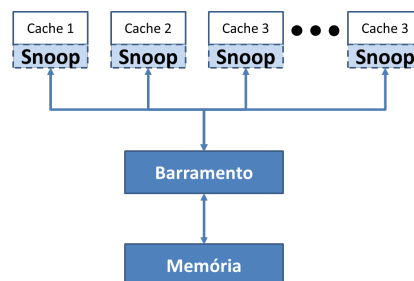
E (Exclusiva): A linha **não** foi modificada pela cache e encontra-se “limpa”. Nenhuma outra cache a possui.

S: (Compartilhada – Shared): A linha está limpa porém existe uma cópia em pelo menos uma outra cache.

I: (Inválida): Esta linha da cache é inválida.

Requisitos:

1. Opção do usuário para determinar o número de caches.
2. Entrada de qual endereço deve ser lido/escrito por que cache.
 - a. Ex: *prompt:> read cache2 0x349A82F*
3. O sistema deve escrever na tela quando ocorrem leituras, escritas ou invalidações.
4. O sistema deve considerar as caches com política de escrita *Write-Back*



Trabalho 7: Simulador de pipeline considerando conflitos de dados e estruturais

Descrição: Implementar uma ferramenta que simule a execução em pipeline 5 estágios do MIPS. A ferramenta recebe como entrada um conjunto de instruções em Assembly MIPS, simula a execução dessas instruções considerando todas as dependências de dados e conflitos estruturais e, tem como saída, a quantidade de ciclos necessários para executar essas instruções e a informação sobre os estágios em cada ciclo.

Para considerar os conflitos estruturais, a ferramenta deve ter como entrada também a informação de quantas memórias existem (modelo Harvard ou von Neumann).

Exemplo de Saída:

São necessários 8 ciclos para executar essas instruções.

Ciclo 1

BI: 4
DI: 0
EX: 3
MEM: 2
WB: 1

Ciclo 2

BI: 5
DI: 4
EX: 0
MEM: 3
WB: 2

Ciclo 3 ...

Ciclo 4 ...

Ciclo 5 ...

Ciclo 6 ...

Ciclo 7 ...

Ciclo 8 ...

Trabalho 8: Simulador de Processador Superscalar

Descrição: Implementar uma ferramenta que simule a execução de um processador Superscalar com n pipelines em paralelo, cada um com 5 estágios. A ferramenta recebe como entrada um conjunto de instruções em Assembly MIPS, e a informação do n (quantidade de pipelines em paralelo). Em seguida, simula a execução dessas instruções considerando todas as dependências de dados, e tem como saída, a quantidade de ciclos necessários para executar essas instruções e a informação sobre os estágios em cada ciclo.

Para o caso de n instruções seguidas sem dependência de dados, as instruções serão executadas em paralelo.

Exemplo de Saída:

São necessários 4 ciclos para executar essas instruções.

Ciclo 1

Pipeline 1	Pipeline 2
BI: 7	BI: 8
DI: 0	DI: 6
EX: 4	EX: 5
MEM: 3	MEM: 0
WB: 1	WB: 2

Ciclo 2		
Pipeline 1		Pipeline 2
BI: 9		BI: 10
DI: 7		DI: 8
EX: 0		EX: 6
MEM: 4		MEM: 5
WB: 3		WB: 0
Ciclo 3: ...		
Ciclo 4: ...		

Trabalho 9: Simulador de Processador VLIW

Descrição: Implementar uma ferramenta que simule a execução de um processador VLIW com 5 estágios de pipeline, com n issues em paralelo. A ferramenta recebe como entrada um conjunto de instruções em Assembly MIPS, e a informação do n (quantidade de unidades funcionais em paralelo). Em seguida, simula a execução dessas instruções considerando todas as dependências de dados, e tem como saída, a quantidade de ciclos necessários para executar essas instruções e a informação sobre os estágios em cada ciclo.

Para o caso de n instruções seguidas sem dependência de dados, as instruções serão executadas em paralelo.

Exemplo de Saída:

São necessários 5 ciclos para executar essas instruções.

Ciclo 1	
BI: 5	
DI: 4	
EX: 3 – add \$s0, \$s1, \$s2 sub \$t0, \$t1, \$t2 lw \$s3, 100(\$s1) add \$t3, \$s2, \$t2	
MEM: 2	
WB: 1	
Ciclo 2	
BI: 6	
DI: 5	
EX: 4 – add \$t4, \$s6, \$s7 sw \$s5, 124(\$s1)	
MEM: 3	
WB: 2	
Ciclo 3: ...	
Ciclo 4: ...	
Ciclo 5: ...	

Trabalho 10: Simulador de Predição de Desvio

Descrição: Implementar um simulador de predição de salto que recebe como entrada um arquivo contendo os saltos e tem como saída

- 1) a própria entrada;
- 2) o resultado da predição de cada salto;
- 3) a percentagem de acerto em cada predição;
- 4) qual o melhor algoritmo de predição para aquela entrada.

O simulador deve conter dois sistemas de predição de salto que utilizem correlação e/ou informações globais.