Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
IMD0021 – Organização e Arquitetura de Computadores

Descrição dos Trabalhos da Disciplina

Instruções:

1. Grupos de NO MÁXIMO 2 alunos – DUPLA.
2. São permitidos no MÁXIMO dois projetos iguais.
3. A entrega do trabalho será considerada completa com a entrega dos seguintes itens: **Implementação**, **Relatório Final** e **Apresentação do trabalho**.
4. Os integrantes do grupo serão avaliados individualmente, e devem saber responder as perguntas relativas a todo o trabalho.
5. O relatório final deve seguir as orientações do documento "Regras para avaliação".
6. As regras para avaliação estão no documento "Regras para avaliação".

7. Prazos:

	Data
Definição dos grupos e trabalhos	13/10/2015
Apresentação do projeto final	01/12/2015

Trabalho 1: Implementação de um processador monociclo e um multiciclo de 32 bits com banco de registradores que executa somente operações lógicas e aritméticas.

Descrição: Implementar a parte operativa e o controle de dois processadores: um que executa instruções em multiciclo e outro em pipeline. Os processadores executa apenas operações lógicas e aritméticas e lê e escreve apenas no banco de registradores.
As instruções são semelhantes às instruções do MIPS e as operações são ADD, SUB, MUL, AND, OR, XOR.

Deve ter:

1. Parte Operativa: ULA, multiplexadores
2. Parte de Controle: sinais para controlar os componentes em máquina de estados
3. Registradores: banco de registradores, contador de programa (PC)
4. Resultados comparativos quando executando o mesmo código nos dois processadores

Linguagem: VHDL ou SystemC

Trabalho 2: Implementação de um processador pipeline de 32 bits com banco de registradores.

Descrição: Implementar a parte operativa e o controle de um processador que executa instruções em pipeline. O processador executa operações lógicas, aritméticas e saltos e lê e escreve apenas no banco de registradores.
As instruções são semelhantes às instruções do MIPS e as operações são ADD, SUB, MUL, DIV, AND, OR, XOR, BEQ, BNE e JUMP.

Deve ter:

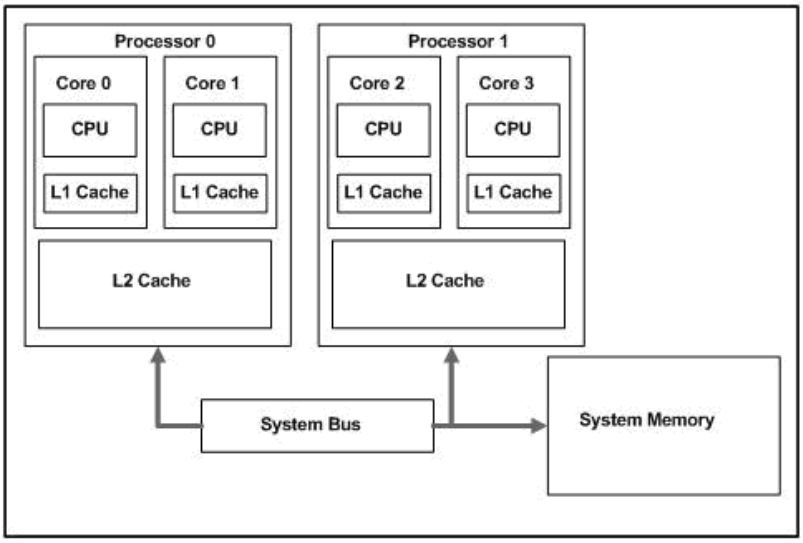
5. Parte Operativa: ULA, multiplexadores
6. Parte de Controle: sinais para controlar os componentes em máquina de estados
7. Registradores: banco de registradores, contador de programa (PC)

Intel Galileo ou Arduíno: o grupo deve desenvolver o pipeline utilizando as plataformas Intel Galileu ou Arduíno. Neste projeto as funções de cada estágio do pipeline devem ser realizadas em um Galileo ou Arduíno diferente e os 5 componentes devem se comunicar para realizar a troca de informações de um estágio para o outro.

Linguagem: C utilizando Galileo ou Arduíno.

Trabalho 3: 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

Deve ter:

- Pode ser modo texto ou interface gráfica, contanto que o conteúdo de todas as memórias sejam exibidos
- Em caso de cache cheia, o algoritmo de substituição será o LRU (menos recentemente utilizado)

Linguagem: Qualquer linguagem de programação de alto nível

Trabalho 4: Simulador de sistema de predição de salto

Descrição: Implementar um simulador de predição de salto que recebe como entrada um arquivo contendo os saltos e tem como saída 1) a entrada; 2) o resultado da predição de cada salto; 3) a percentagem de acerto em cada predição e 4) qual o melhor algoritmo de predição para aquela entrada. O simulador deve conter quatro sistemas de predição de salto: “1 bit”, “2 bits”, “adaptativo de dois níveis” e mais um proposto pela equipe.

Exemplo de arquivo de entrada: T N T T N N N T N T T T

Deve ter:

- Obrigatório ter interface gráfica
- O algoritmo de predição de salto proposto não pode ser do tipo *n*-bits

Linguagem: Qualquer linguagem de programação de alto nível

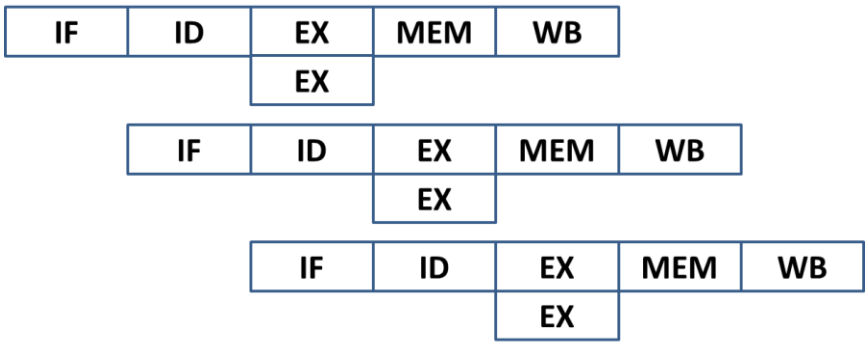
Trabalho 5: Implementação de um processador pipeline VLIW

Descrição: Implementação de um processador para executar duas instruções de ULA em paralelo. Para funcionar como VLIW, deve ser necessário alterar o formato da instrução para conter duas operações, ao invés de uma; controle e parte operativa.

Deve ter:

1. Parte Operativa: ULA; multiplexadores
2. Parte de Controle: sinais de controle dos componentes em máquina de estados
3. Registradores
4. Modificação no conjunto de instruções

Linguagem: VHDL ou SystemC



Trabalho 6: Implementação do controle de uma máquina de café.

Descrição: A máquina trabalha com quatro tipos de café de preços diferentes e aceita moedas de R$ 0,10; R$ 0,50 e R$ 1,00. Um sinal de entrada indica a entrada de moedas, enquanto a outra entrada indica a solicitação do tipo de café, o qual deve ser previamente escolhido através da entrada de seu código. A máquina está preparada para devolução do troco, caso o valor da(s) moeda(s) exceda o valor do café escolhido, ou caso valor seja inferior ao seu preço. Para isso foram implementados os sinais de saída: "libera troco", "troco" e "libera café".

Tipos de café e preços:

- Café espresso – R$ 2,00
- Mocaccino – R$ 3,00
- Cappuccino – R$ 3,50
- Chocolate quente – R$ 2,50

Deve ter:
1. Fluxograma do funcionamento do sistema
2. Parte Operativa: ULA, comparador, multiplexadores
3. Parte de Controle: sinais de controle dos componentes em máquina de estados
4. Registradores: dados de entrada, dados de saída

Linguagem: VHDL ou SystemC

Trabalho 7: Implementar interrupção com salvamento de contexto no processador

Descrição: Implementação de um processador pipeline que atenda interrupções. Para atender interrupções é necessário acrescentar uma instrução de interrupção ao conjunto de instruções, adicionar a etapa de salvar e restaurar contexto e criar uma rotina de interrupção. A rotina de interrupção pode ser apenas uma instrução de soma.

Deve ter:

1. Parte Operativa: ULA; multiplexadores
2. Parte de Controle: sinais de controle dos componentes em máquina de estados
3. Registradores
4. Modificação no conjunto de instruções

Linguagem: VHDL ou SystemC

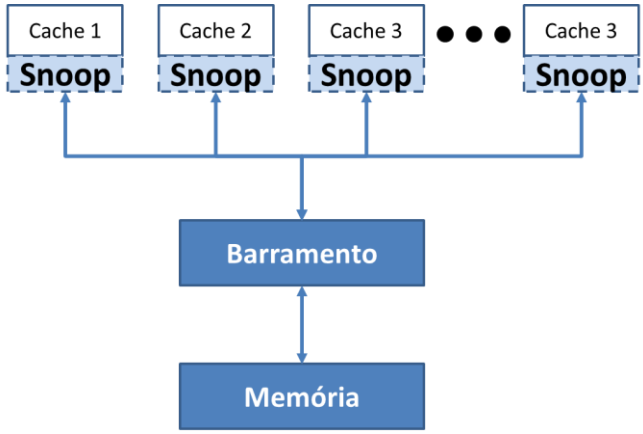
Trabalho 8: Implementar um modelo de Coerência de Cache baseada em Snoop

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 politicia de escrita *Write-Back*



Linguagem: Qualquer linguagem de programação de alto nível

Trabalho 9: Implementar um modelo de Coerência de Cache baseado em Diretório

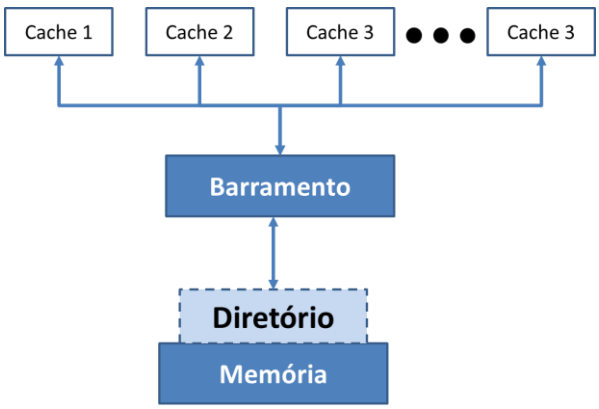
Descrição: Considerando o cenário da figura abaixo, criar a simulação de um sistema com caches, barramento, memória e um componente conectado à memória que realiza operações de coerência de cache. Nesta implementação, o Diretório identifica se as linhas das caches encontram-se em estado "limpo" ou "sujo" e realiza ações dependendo da solicitação e estado das linhas.

Restrições:

1. Quando um bloco está sujo:
    a. Uma solicitação de leitura gera um pedido de write-back + a leitura do bloco
    b. Uma solicitação de escrita gera um pedido de write-back com invalidação + a leitura/escrita do bloco (para manter a exclusividade).
2. Quando um bloco está limpo:
    a. Uma solicitação de leitura não gera nada a além da própria leitura do bloco.
    b. Uma solicitação de escrita gera um pedido de invalidação para todas as outras caches que possuem a linha + a leitura/escrita do bloco (para manter a exclusividade).

Deve ter:

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, invalidações ou write-backs.



Linguagem: Qualquer linguagem de programação de alto nível

Trabalho 10: Implementar um jogo no Assembly ARM usando o simulador ARMSim#

Descrição: Implementar um dos jogos: Campo Minado, BlackJack ou Batalha Naval, em Assembly ARM usando o simulador ARMSim#

Deve ter:

- Interface gráfica
- Download do Simulador:
  - http://armsim.cs.uvic.ca/DownloadARMSimSharp.html