Exercício Programa 3
Minha memória já não é como antigamente...

Arquitetura de Computadores - 2021.Q1
http://professor.ufabc.edu.br/~e.francesquini/2021.q1.ac/

Prof. Emilio Francesquini
e.francesquini@ufabc.edu.br

14 de abril de 2021

1 O Projeto

Nesta terceira fase do projeto vamos adicionar alguns componentes no Minips que vão nos permitir entender muito melhor como a hierarquia de memória funciona. Além disto, é claro, há algumas poucas novas instruções que precisarão ser adicionadas ao seu emulador.

Como de praxe, para tirar a nota máxima neste EP, é condição necessária (mas não suficiente) que seu programa execute corretamente todos os programas de entrada disponibilizados. Eles incluem, naturalmente, aqueles da fase 1 e 2 assim como alguns adicionais para a fase 3.

O foco desta última fase é adição da simulação da memória e dos níveis de cache. Para isto o seu emulador deverá ser capaz de rodar com diferentes configurações de memória e também fazer uma estimativa (simplificada) do tempo de execução levando em conta as latências de cada nível.

Na fase 2 adicionamos a possibilidade de executar códigos escritos em C no Minips. Atendendo a pedidos, modifiquei o arquivo `minips.h` e o `Makefile` para que o programa também possa ser rodado nativamente na sua máquina. O arquivo `.c` é compilado com o `gcc` padrão do seu sistema gerando arquivos com o sufixo `.elf.native` que podem ser executados diretamente.

As dicas das fases anteriores continuam valendo: implemente o emulador, instrução a instrução, conforme elas forem aparecendo nos programas de teste. Depois que todos os programas estiverem rodando comece a fazer a modelagem do sistema de memória. Os programas desta vez são mais

simples do que os das fases anteriores, logo o foco principal é o sistema de memória.

2 Convenções

Todas as convenções das fases 1 e 2 sem mantêm. Além disto:

- Como falamos na introdução, o `Makefile` agora compila uma versão do binário nativa para o seu computador, permitindo que você a execute lado a lado com o seu emulador. Este arquivo binário nativo para a máquina tem a extensão `.elf.native`. A lista completa de extensões fica então:

Extensão	**Descrição**
`.asm`	Código fonte da entrada de teste escrito em MIPS assembly
`.c`	Código fonte da entrada de teste ecrito em C
`.elf`	Arquivo binário executável compilado para MIPS32 Little Endian
`.elf.native`	Arquivo binário executável compilado para a ISA do `gcc` padrão no sistema (provavelmente, x86-64)
`.text`	Seção de texto extraída do `.elf` para MIPS
`.data`	Seção de dados extraída do `.elf` para MIPS
`.rodata`	Seção de dados *read only* extraída do `.elf` para MIPS

- Pelo menos as seguintes configurações de memória deverão ter suporte do seu emulador.

Conf.	**Níveis**	**Tipo**	**Tamanho**	**Map.**	**Tam./Linha**	**Política**
1	0	-	-	-	-	-
2	1	Unificada	1024	Direto	32	Aleatória
3	1	Split	512/cada	Direto	32	Aleatória
4	1	Split	512/cada	Direto	32	LRU
5	1	Split	512/cada	4 vias	32	LRU
6	2	Split Unificada	L1 512/cada L2 2048	4 vias 8 vias	64 64	LRU LRU

  - A configuração 1 é a padrão caso nenhuma seja especificada. Ela é equivalente a um processador sem caches.
  - As latências padrões de acessos para cada um dos níveis são:

Nível	Latência
L1	1
L2	10
RAM	100

    - **Dica**: a implementação de cache pode ser única para todos os tipos de mapeamento (direta/associativa/completamente associativa) bastando para isso ajustar o parâmetro da associatividade.
    - As caches com configuração *split* devem ser coerentes. Você perceberá que alguns programas como o `09.contador` não funcionarão caso contrário.
    - Todos os níveis funcionam com política de *write-back*.

- Faremos uma estimativa de tempo de execução bem capenga nessa versão do Minips (mas muito melhor do que tínhamos antes!). Assuma que todas as instruções levam um ciclo para executar. Exceto, é claro, aquelas que fazem acesso à memória. Neste caso o número de ciclos gastos para executar a instrução é o tempo de acesso a memória contando eventuais acumulos de tempos devido ao percurso da hierarquia de memória até achar o nível que contém o dado desejado.
    - O custo de cache miss em um nível será simplificado como sendo o custo da busca no próximo nível de memória mais o custo de substituição de uma linha no nível atual (que pode variar dependendo da necessidade fazer *write-back* para o nível seguinte).

3 Recursos e cuidados

Continuam os mesmos da fase 1. Além disto, o emulador de referência implementado pelo professor foi atualizado. Você pode baixar a nova versão nos links abaixo:

- Linux (x86-64): minips
- Windows (x86-64): minips.exe

4 Entradas e saídas esperadas

Nesta nova versão teremos **duas novas opções de execução**. Assim, as opções são:

`minips {run|decode|trace|debug} [conf] arquivo`

Opção	Descrição
`run`	Executa o programa e imprime estatísticas básicas ao final.
`decode`	Decodifica o programa e imprime o assembly. `conf`, caso fornecida é ignorada.
`trace`	O mesmo que `run`, além disso escreve no arquivo `minips.trace` os acessos feitos à memória.
`debug`	O mesmo que `trace`, contudo com muito mais informações sobre os acessos às caches/memória.

A depender do programa executado, o arquivo `minips.trace` pode ficar grande, especialmente com a opção `debug`. A seguir descrevemos as saídas esperadas:

4.1 Opção `run`

O seu programa deve se comportar como o programa de exemplo fornecido. Se for informada uma configuração de memória, ela deve ser utilizada para execução. Caso nenhuma seja fornecida, a configuração 1 deve ser assumida.

É obrigatório o fornecimento de estimativas de tempo para a versão monociclo e para a versão em *pipeline* (já levando em conta as latências de memória discutidas acima). Assuma as mesmas frequências do programa de referência. Também é obrigatória a apresentação de estatísticas de *hits* e *misses* para cada um dos níveis de memória.

O hardware que "inspirou" essas estatísticas é o MIPS que era usado pelo Playstation 1. Assumam que o processador funciona a uma frequência de 33.8688 MHz, ou seja, um período de 29525,6991686 picossegundos com um pipeline de 5 estágios, tal qual visto em aula. Assumam também que a versão monociclo funciona com uma frequência 4 vezes menor.

4.1.1 Exemplo 1

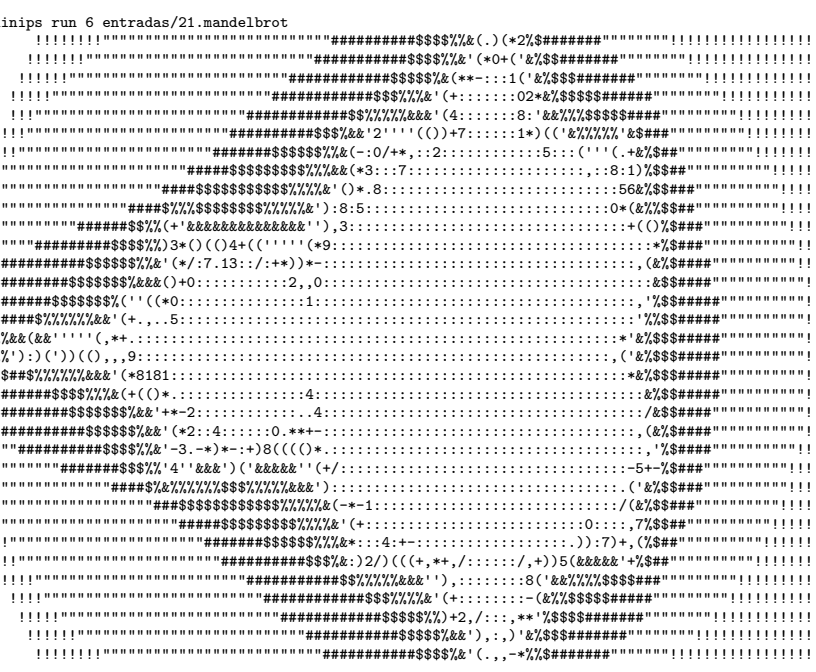
```
$ ./minips run entradas/09.contador
123456789
Execution finished successfully
-------------------------
Instruction count: 164 (R: 64 I: 82 J: 18 FR: 0 FI 0)
Simulation Time: 0.01 sec.
Average IPS: 12199.02


Simulated execution times for:
-------------------------
Monocycle
    Cycles: 18182
    Frequency: 8.4672 MHz
    Estimated execution time: 0.0021 sec.
    IPC: 0.01
    MIPS: 0.08
Pipelined
    Cycles: 18186
    Frequency: 33.8688 MHz
    Estimated execution time: 0.0005 sec.
    IPC: 0.01
    MIPS: 0.31
Speedup Monocycle/Pipeline: 4.00x


Memory Information
------------------
Level  Hits          Misses        Total          Miss Rate
-----  ------------  ------------  ------------   ---------
  RAM           182             0           182       0.00%
```

4.1.2 Exemplo 2

```
$ ./minips run 6 entradas/21.mandelbrot
```



```
Execution finished successfully
--------------------------
Instruction count: 589009 (R: 102590 I: 115107 J: 3470 FR: 328188 FI 39654)
Simulation Time: 1.28 sec.
Average IPS: 461032.53


Simulated execution times for:
--------------------------
Monocycle
    Cycles: 589999
    Frequency: 8.4672 MHz
    Estimated execution time: 0.0697 sec.
    IPC: 1.00
    MIPS: 8.45
Pipelined
    Cycles: 590003
    Frequency: 33.8688 MHz
    Estimated execution time: 0.0174 sec.
    IPC: 1.00
    MIPS: 33.81
Speedup Monocycle/Pipeline: 4.00x


Memory Information
------------------
Level  Hits          Misses        Total          Miss Rate
-----  ------------  ------------  -------------  ---------
  L1i        589002            14         589016       0.00%
  L1d            10             2             12      16.67%
   L2             0             9              9     100.00%
  RAM             9             0              9       0.00%
```

Em caso de dúvidas, envie para o Discord da disciplina.

4.2 Entradas

Além das entradas anteriores, as seguintes entradas foram incluídas. Todas entradas de exemplo podem ser baixadas aqui: entradas.zip.

Entrada	Descrição
`18.naive_dgemm`	Multiplicação de matrizes 64x64
`19.regular_dgemm`	Multiplicação de matrizes 64x64
`20.blocking_dgemm`	Multiplicação de matrizes 64x64
`21.mandebrot`	Calcula e imprime um fractal

4.3 Opção `decode`

A opção `decode` continua como era na fase 2. Contudo agora é obrigatório que seja apresentado o endereço da instrução, do hexadecimal que a representa além é claro, da instrução decodificada.

4.4 Opções `trace` e `debug`

A opção `trace` executa o programa (assim como `run` pode receber diferentes configurações de memória) e além disso gera um arquivo `minips.trace`. Este arquivo contém uma lista de todos os acessos à memória feitos pelo programa em execução. As linhas podem ter os seguintes formatos:

- `R 0x00400008 (line# 0x00020000)` - Leitura de um dado no endereço `0x00400008` que, neste exemplo, está localizado na linha de memória `0x00020000`. O endereço não varia de execução para execução, mas a linha pode mudar de acordo com o tamanho da linha de cache da configuração utilizada.
- `I 0x00400010 (line# 0x00020000)` - Leitura de uma instrução no endereço `0x00400010`. Leituras de instruções devem ser feitas através da cache de instruções (caso exista). Leituras de instruções são feitas no estágio de *fetch* da CPU e devem se diferenciar das leituras efetuadas como resultado da execução de uma instrução como, por exemplo, `lw`.
- `W 0x00400008 (line# 0x00020000)` - Escrita de um dado no endereço `0x00400008`. Escritas são sempre feitas através da cache de dados (caso exista) precisando haver o cuidado de manter a coerência caso o endereço escrito seja o de uma instrução que esteja na cache de instruções.

A opção `debug` traz além das informações do trace, informações adicionais sobre o funcionamento da hierarquia da memória. O formato é livre e pode ser usado por vocês durante o desenvolvimento.

O formato de saída da opção `trace` é rígido, já o formato do `debug` é livre para incluirem as informações que acharem relevantes.

5 Bônus

5.1 Bonus 1 - 1 a 2 pontos

Você terá 1 ponto adicional na nota se você achar um bug na implementação de referência feita pelo professor. Não contam como bugs: passar instruções inválidas, reclamar de mensagens de erros toscas, passar parâmetros inválidos... Basicamente, é um programa que tem garantia vitalícia limitada! Máximo de 2 pontos por cliente. Obrigatório expor o bug no Discord!

5.2 Bônus 2 - 1 ponto

Se você fizer uma temporização mais acurada do tempo do Minips levando em conta cache miss penalty, encontrar valores reais de tamanhos de caches, latências, associatividade, etc de processadores MIPS por aí e integrar ao seu emulador.

5.3 Bônus 3 - 1 ponto

Se o seu código executar corretamente e for o mais rápido de todos.

5.4 Bônus 4 - 1 ponto

Explique no seu relatório/vídeo usando o seu emulador:

- Porque misses de um nível não batem exatamente como o número de acessos (hits + misses) do nível seguinte?
- Porque o número de ciclos não bate com o número de instruções executadas?
- Porque os tempos de execução dos programas `18.naive_dgemm`, `19.regular_dgemm` e `20.blocking_dgemm` variam tanto entre si quando eles efetivamente fazem a mesma coisa?

6 Entrega

Junto com o seu código você deve entregar um brevíssimo relatório (máximo de 2 páginas) contendo obrigatoriamente:

- Nome completo
- RA
- Usuário do GitHub
- Link para vídeo no YouTube ou qualquer outro lugar acessível pelo professor.

O seu vídeo deve ter duração máxima de 5 minutos, deve mostrar o seu código executando e deve destacar os pontos fortes da sua implementação assim como as eventuais limitações e dificuldades que você teve durante a implementação.

A entrega do código e do relatório deve ser feita pelo GitHub Classroom através do link `https://classroom.github.com/a/r8MsBwVm`. Será considerado como entrega o último *commit* (não esqueça de dar *push*) no repositório até a **data limite de 02/05/2021**.

Projetos entregues com atraso sofrerão descontos seguindo a seguinte tabela:

Dias em atraso	Nota máxima
1 dia	7
2 dias	6
3 dias	5
>3 dias	0

Para discussões, dúvidas e comentários utilize o Discord em `https://discord.gg/9RtRcx3`.