ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES II
LABORATÓRIO 3

OBJETIVO

Implementar um simulador para a arquitetura MIPS32 4K, com suporte à previsão de desvio e execução fora de ordem.

DOCUMENTAÇÃO

A documentação necessária para este laboratório se encontra em: www.mips.com.
Você precisará dos seguintes documentos para implementar o simulador:

1. MIPS32® 4K® Processor Core Family Software User's Manual
2. The MIPS32 Instruction Set v6.06
3. MIPS32 Instruction Set Quick Reference v1.01

ATIVIDADES

Como os laboratórios anteriores, as atividades são incrementais. Portanto, precisam ser concretizadas sequencialmente.

Atividade 1: Implementar um tradutor de Assembly para binário

Não será necessário simular todo o conjunto de instruções da arquitetura MIPS32. O simulador deverá simular as instruções apresentadas na Tabela 1.

Tabela 1: Instruções simuladas.

ADD	BNE	MTHI
ADDI	DIV	MTLO
AND	J	MUL
ANDI	JR	MULT
B	LUI	NOP
BEQ	MADD	NOR
BEQL	MFHI	OR
BGEZ	MFLO	ORI
BGTZ	MOVN	SUB
BLEZ	MOVZ	XOR
BLTZ	MSUB	XORI

Vamos primeiro analisar um simples trecho de código, para entender as limitações por não gerenciar todo o conjunto de instruções. O código em C:

```
i = (N + 3)*N
```

deveria ser traduzido para:

```
lw      $t0, 4($gp)     # carrega N
add     $t1, $t0, $zero # copia N para $t1
```

```
addi    $t1, $t1, 3      # N+3
mul     $t1, $t1, $t0    # (N+3)*N
sw      $t1, 0($gp)      # i = ...
```

Contudo, o simulador não gerencia as instruções lw (load) e sw (store). Então um código como este deverá ser escrito utilizando registradores e imediatos, da seguinte forma:

```
add     $t0, $zero,$zero      # $t0 = 0
addi    $t0, $t0, 10          # copia o valor 10 para $t0
add     $t1, $t0, $zero       # copia o valor 10 para $t1
addi    $t1, $t1, 3           # 10+3
mul     $t2, $t1, $t0         # $t2 = (10+3)*10
```

Como pode ser observado no código anterior, a expressão i = N*N + 3*N foi transformada em $t2 = (10 + 3)*10.

Além das instruções apresentadas na Tabela 1, o simulador deverá gerenciar a instrução *syscall* com código 0. Esta instrução irá indicar para o simulador final de execução.

A Tabela 2 apresenta o código binário relativo ao código MIPS32 acima.

Tabela 2: Código binário

Binário	Decimal	Hexadecimal
00000000000000000100000000100000	16416	4020
00100001000010000000000000001010	554172426	2108000A
00000001000000000100100000100000	16795680	1004820
00100001001010010000000000000011	556335107	21290003
01110001001010000101000000000010	1898467330	71285002

Portando para o código Assembly anterior, o tradutor irá gerar como saída um arquivo texto contendo:

```
4020
21008000A
1004820
21290003
71285002
```

Um dos parâmetros do simulador será um arquivo contendo o programa, escrito em Assembly, que será simulado. Portanto, em uma atividade posterior será necessário modificar o tradutor para gerar o código final em memória.

Atividade 2: Implementar o estágio de busca

Atividade 3: Implementar o estágio de execução

Atividade 4: Implementar o estágio de busca da memória

Atividade 5: Alterar o simulador para suportar bypassing

Atividade 6: Implementar o estágio de alinhamento

Atividade 7: Implementar o estágio de escrita

Atividade 8: Alterar o simulador para suportar previsão de desvio

Altere o estágio de busca, para suporte a previsão de desvio sempre tomado.

Atividade 9: Alterar o simulador para suportar scoreboarding

Atividade 10: Alterar o simulador para suportar especulação

IMPLEMENTAÇÃO

O simulador deverá ser implementado em C; exceto o tradutor que deverá ser implementado em Assembly para 64 bits.

EXECUÇÃO

O simulador deverá ser executado da seguinte forma:

./mips32sim -i <prog>.asm [-s <prog>.s] [--detail] [-h]

As possíveis opções são:

-h: help
-i: arquivo assembly contendo programa a ser simulado

-o: arquivo de saída contendo programa convertido para binário
--detail: execução detalhada

Ao final da execução, o simulador deverá gerar o arquivo <prog>.out. Este contém informações sobre a execução. Caso não seja especificado a opção --detail, o arquivo de saída terá o seguinte formato:

Programa:
```
add     $t0, $zero,$zero
addi    $t0, $t0, 10
add     $t1, $t0, $zero
addi    $t1, $t1, 3
mul     $t2, $t1, $t0
...
```

Binário:
```
4020
21008000A*
1004820
21290003
71285002
...
```

Previsão:
```
Total de saltos:    X
Acertos:            K (k%)
Erros:              W (w%)
```

Ciclos:
```
N ciclos
```

Instruções:
```
Emitidas:    Y
Efetivadas:  H (h%)
```

Para uma saída detalhada (--detail), o arquivo de saída terá a seguinte formato:

Programa:
```
add     $t0, $zero,$zero
addi    $t0, $t0, 10
add     $t1, $t0, $zero
addi    $t1, $t1, 3
mul     $t2, $t1, $t0
...
```

Binário:
```
4020
21008000A
1004820
21290003
71285002
```

...

Previsão:
Total de saltos: X
Acertos: K (k%)
Erros: W (w%)

Ciclos:
N ciclos

Ciclo 1:
Busca:
add $t0, $zero,$zero
addi $t0, $t0, 10
...

Execução:
… <insts>

Busca da memória:
… <insts>

Alinhamento:
… <insts>

Escrita:
… <insts>

Efetivando:
… <insts>

Bypassing:
… <registradores>

Ciclo C

…

Instruções:
Emitidas: Y
Efetivadas: H (h%)

SIMULAÇÕES

Para validar seu simulador implemente os seguintes códigos:

1. fatorial de N
2. k = 10*5+3/(6-2354+87)*45
3. x = (10 +2) ^ 5

Futuramente, serão disponibilizados os programas que serão utilizados para validar seu simulador.

CONSIDERAÇÕES

- O laboratório poderá ser feito em equipe de até 3 alunos.
- O laboratório deverá ser entregue até as 23:00 do dia 4 de julho, por e-mail.
- Seu e-mail deverá ter o seguinte título: ARQII – L3 – RA1 – RA2 – RA3.
- O e-mail deverá conter um pacote contendo os fontes do simulador e um arquivo makefile, para compilar o simulador utilizando a ferramenta make.
- A avaliação do trabalho considerará, completude e corretude.
- Para a completude, as atividades serão contempladas com percentuais distintos, da seguinte forma:
    1. Atividades 1 a 7 até 40%,
    2. Atividade 8 até 10%,
    3. Atividade 9 até 40%,
    4. Atividade 10 até 10%.
- A corretude irá analisar ser seu simulador executa corretamente. Neste sentido observe que as Atividades 8, 9 e 10 formam um grupo indivisível, ou seja, uma depende da outra para simular corretamente um determinado programa. Portanto, se meu laboratório implementa apenas as Atividades 1 a 8, o simulador irá gerar uma saída errada, pois nem todos saltos deveriam ser tomados.
- A especificação do laboratório não pode ser alterada. Caso ocorra alguma alteração (por menor que seja), tal laboratório não será considerado para avaliação.