Especificação do Projeto –
Simulador MIPS Superescalar

CES- 25 Arquiteturas para Alto Desempenho
Prof. Paulo André Castro
Equipe: até três alunos
Última atualização: 5/junho/2018

1. Objetivo
   Exercitar e fixar conhecimentos adquiridos sobre processadores superescalares com exploração de paralelismo entre instruções através do algoritmo de Tomasulo, dependências entre instruções e comparação de desempenho de sistemas através de benchmarks, assuntos estudados durante o curso de Arquiteturas para Alto Desempenho.

2. Descrição do Trabalho

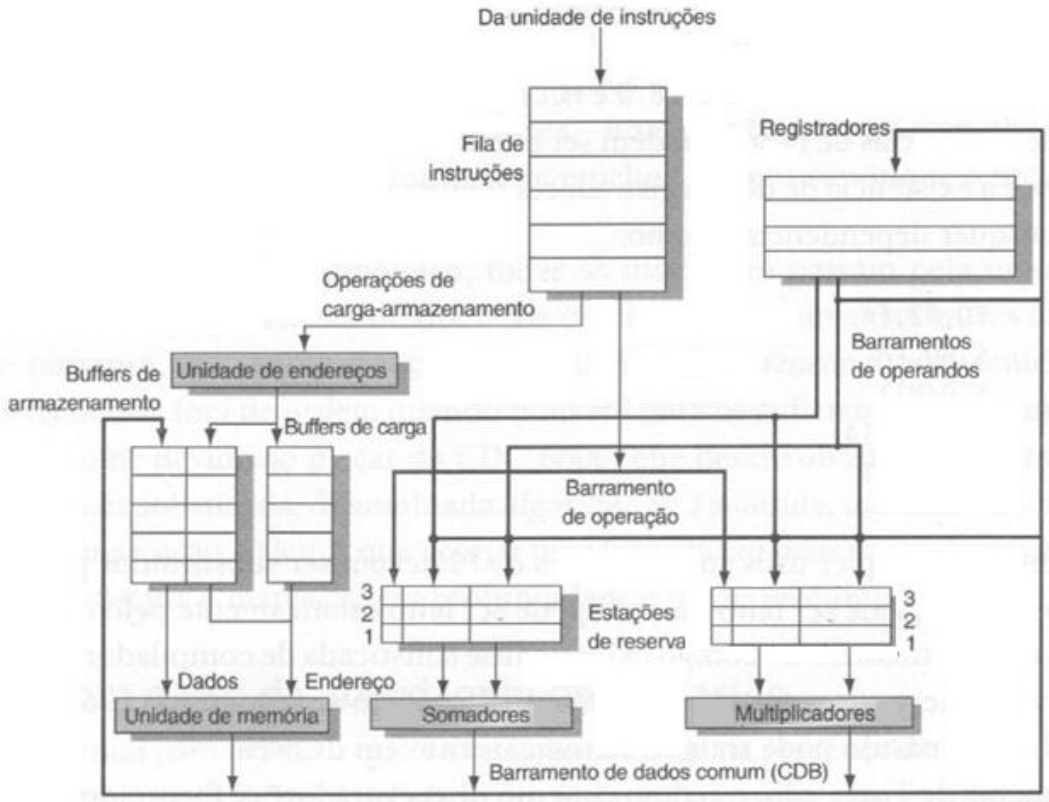


Figura 1. Datapath do MIPS Superescalar CES25.2018SE

Criar um simulador para uma versão do processador MIPS superescalar (MIPS CES25.2018SE) com conforme apresentado na figura 1. O simulador deverá ler um programa conforme formato especificado na seção 3 e executar suas instruções apresentando em cada estação de reserva, o identificador da instrução, seu estado (Emitida, Executando, Gravando), os sinais de controle gerados alem do clock corrente da máquina, o valor dos registradores e suas dependências. Deve ser assumido o tempo da fase de execução de cada instrução, conforme a tabela 1. O tempo de gravação e emissão correspondem a um clock para qualquer instrução. Apenas uma instrução pode ser consolidada por clock. Da mesma forma, apenas uma instrução pode ser emitida por clock, porém emissão e gravação podem ocorrer em paralelo.

Tabela 1. Tempos na Fase de Execução

Instrução	Tempo na Fase de Execução
Load/Store	4
MUL	3
DIV	5
Add/Sub/outras	1

A lógica de funcionamento do MIPS CES25.2018SE deve seguir o explicado em sala e como descrito nas notas de aula. Por exemplo, uma instrução só pode avançar para a fase de execução caso o hardware necessário esteja livre e as dependências já tiverem sido eliminadas.

A memória do computador de dados poderá ser simplificadamente representada por um vetor de 4K posições, onde cada posição contém 32 bits. A memória de dados pode ser utilizada pelo programa através de instruções lw e sw, tendo como base um registrador, como por exemplo: lw R2,20(R0) e sw R3,30(R0).

A memória de instruções será representada por outro vetor, com as instruções lidas do arquivo de programa e armazenado neste vetor. Ao chegar ao final do vetor de instruções o processador deve parar mantendo seu estado. Cada instrução tem também 32 bits, conforme formato MIPS CES25.2018SE (ver apêndice).

O simulador deve ter uma interface gráfica que facilite o acompanhamento da evolução do estado do processador e quais o estado de cada uma das instruções em execução, tal como descrito em sala de aula.

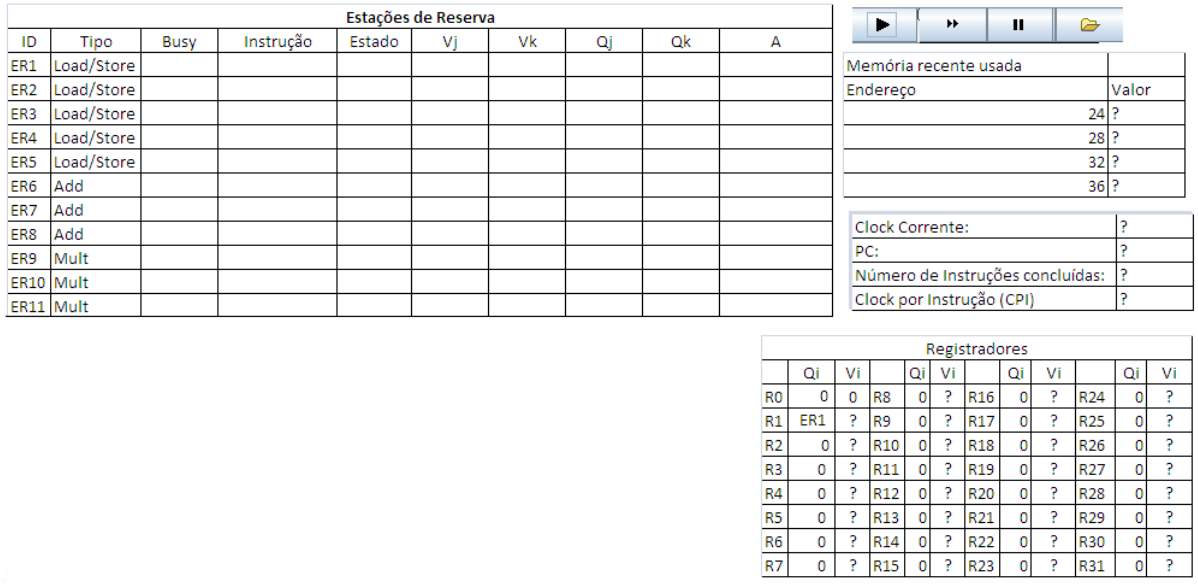


Figura 2. Sugestão de interface gráfica para o simulador

O simulador deve ser capaz de contabilizar o tempo necessário para executar o programa (em número de clocks) e o número de clocks médio por instrução (CPI).

3. Programas de teste

Os programas são armazenados no mesmo formato especificado para o primeiro projeto deste semestre. Isto é, os programas são armazenados em arquivos texto com uma instrução por linha, onde a instrução é definida em 32 caracteres 0's ou 1's e opcionalmente após um sinal de ponto e vírgula é colocado um comentário sobre a instrução, chamado de identificador da instrução. O endereço das instruções inicia em zero e é incrementado com passo igual a quatro. A instrução I1 está no endereço zero, I2 no endereço 4, I3 no endereço em 8 e assim por diante. Veja exemplo abaixo.

Figura 3. Programa Exemplo em formato texto MIPS CES25.2018SE

```
00100000000010100000000001100100        ; I1: addi R10,R0,100
10101100000000000000000000011000        ; I2: sw R0,24(R0)
10101100000000000000000000011100        ; I3:sw R0,28(R0)
10001100000001100000000000011100        ; I4: lw R6,28(R0)
00000000110001100011100000011000        ; I5: mul R7,R6,R6
10001100000000010000000000011000        ; I6: lw R1,24(R0)
00000000001001110100100000100000        ; I7: add R9,R1,R7
10101100000010010000000000011000        ; I8: sw R9,24(R0)
00100000110001100000000000000001        ; I9: addi R6,R6,1
10101100000001100000000000011100        ; I10: sw R6,28(R0)
00011100110010100000000000010100        ; I11: ble R6,R10,20
```

Figura 4. Programa de Benchmark a ser utilizado nos experimentos.

```
ADDI R1,R0,3
ADDI R2,R0,1
ADDI R4,R0,1
P1:
ADDI R5,R0,1
P2:
MUL R6,R1,R4
ADD R6,R6,R5
SW R2,0(R6)
ADDI R2,R2,1
ADDI R5,R5,1
BLE R5,R1,P2
ADDI R4,R4,1
BLE R4,R1,P1
ADDI R2,R0,0
ADDI R4,R0,1
P3:
MUL R6,R4,R1
ADD R6,R6,R4
LW R6,0(R6)
ADD R2,R2,R6
ADDI R5,R4,1
ADDI R9,R1,1
BEQ R5,R9,P5
P4:
MUL R6,R4,R1
ADD R6,R6,R5
LW R3,0(R6)
MUL R7,R5,R1
ADD R7,R7,R4
LW R8,0(R7)
```

```
SW R8,0(R6)
ADD R2,R2,R8
SW R3,0(R7)
ADD R2,R2,R3
ADDI R5,R5,1
BLE R5,R1,P4
ADDI R4,R4,1
BLE R4,R1,P3
P5:
ADDI R6,R0,0
```

Deve ser desenvolvido ainda pela equipe um outro programa simples (fatorial, MDC, etc.) para testes esse programa deve constar do relatório em linguagens C, MIPS CES25.2018SE e representação binária.

4. Aprimoramento do Processador

A **versão 1** do simulador é igual a descrita nesse documento. Você deve propor uma alteração de arquitetura com o objetivo de melhorar o desempenho do processador de modo significativo. Por exemplo, aumentar o número de estações de reservas, aumentar número de unidades funcionais, realizar execução especulativa ou outra qualquer que você julgar que pode melhorar o desempenho do processador, porém sem utilizar "hardware" mais rápido, isto é os tempos da tabela 1 permanecem. A implementação dessa proposta constituirá a **versão 2** do simulador.

5. Experimentos Simulados

Teste a correção do seu simulador no programa de benchmark da Figura 4, variando o parâmetro de entrada dado em R1 de acordo com os dados fornecidos na tabela abaixo. Para realizar esses testes, altere a primeira linha do programa conforme necessário.

Tabela 2. Configurações (ou benchmarks) de Testes

	Valor inicial em R1	Valor Final esperado em R2
Configuração 1	3	45
Configuração 2	5	325
Configuração 3	6	666

Calcule o ganho de desempenho em relação a versão 1 comparada a versão 2, para as três configurações apresentadas na tabela 1. Apresente os dados de forma gráfica, como no exemplo abaixo. Observe que o gráfico é apenas um exemplo, não necessariamente os ganhos de desempenho serão dessa ordem de grandeza e pode até não ocorrer ganho.

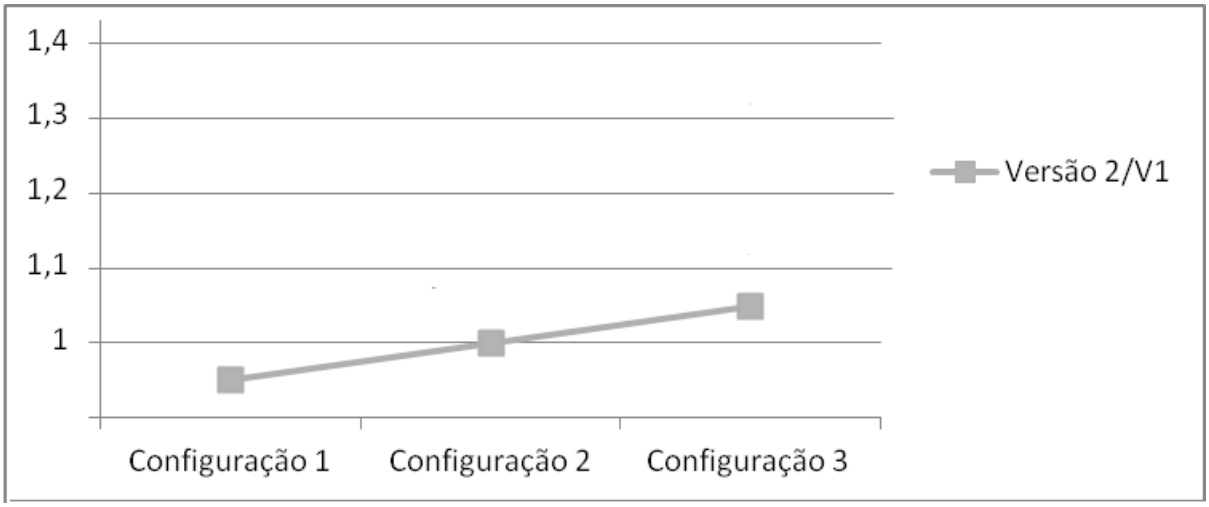


Figura 5. Gráfico de Desempenho relativo a versão 1 nas três configurações.

Qual sua expectativa de sistema mais rápido versão 1 ou versão 2? Analise os resultados obtidos, comparando com a sua expectativa de ganho ou perda de desempenho entre as versões. O que significaria na realidade construir um computador com sistema de memória tal como o da versão 1? Qual versão (1 ou 2) seria preferível na sua visão para implementação real e por quê?

Conjunto de Experimentos e Análises Adicionais

O que faz o programa de benchmark?

Quais os resultados em R2 para R1 iguais a 10,15 e 20?

Apresente o gráfico similar a figura 5 porém com as seis configurações.

6. Apresentação da Primeira Versão

O grupo deve fazer uma apresentação dia 30/5 contemplando toda a versão 1 (ver seção 4) e apresentar sua proposta para a versão 2 do projeto. Deve ser apresentado um relatório parcial contendo os mesmos itens do Relatório final, porém limitado a versão 1 e a proposta da versão 2.

7. Material a ser Entregue

Relatório Versão 1
Prazo de Entrega: 6/junho/2018

Relatório do Projeto Versão 1(A ser entregue impresso e em formato .pdf) com

Integrantes do Grupo:

Objetivo: objetivo do trabalho, citar a linguagem utilizada para implementar o simulador.

Descrição: detalhamento de hipóteses adicionais adotadas (casos omissos) e a da alteração sugerida para o sistema. Descrição do programa elaborado pela equipe, em linguagens C, MIPS CES25.2018SE e representação binária.

Resultados Obtidos: resultados obtidos com a simulação nas duas configurações nos dois programas.

Análise dos Resultados: discussão dos resultados (ganho, perda ou manutenção) do desempenho da Versão 1, discutir outras questões apontadas no item 5.

Proposta de Alteração para Versão 2: Descreva sua proposta de alteração do processador simulado e argumente porque você acredita que esta alteração trará ganhos de desempenho. Explicite se vc deseja que a segunda versão seja o seu exame. Nesse caso, a proposta deve ter complexidade compatível com a implementação de execução especulativa. No caso de optar, por uma alteração mais simples, isto pressupõe que o grupo aceita fazer o exame escrito.

Conclusões: Comentários sobre o trabalho (complexidade/facilidade, sugestões, etc.) e contribuições do trabalho para o entendimento sobre processadores.

Código-fonte e Executável do Simulador (em C, C++ , Java ou Python).

Relatório Versão 2
Prazo de Entrega: 22/junho/2018
OBS.: Trata-se do relatório versão 1 atualizado para incluir os resultados da implementação da proposta de alteração

Relatório do Projeto Versão 2 (A ser entregue impresso e em formato .pdf) com:

Integrantes do Grupo:

Objetivo: objetivo do trabalho, citar a linguagem utilizada para implementar o simulador.

Descrição: detalhamento de hipóteses adicionais adotadas (casos omissos) e a da alteração sugerida para o sistema. Descrição do programa elaborado pela equipe, em linguagens C, MIPS CES25.2018SE e representação binária.

Resultados Obtidos: resultados obtidos com a simulação nas duas configurações nos dois programas.

Análise dos Resultados: discussão dos resultados (ganho, perda ou manutenção) do desempenho, confirmação ou refutação da sua expectativa em relação a sua proposta de alteração, discutir outras questões apontadas no item 5.

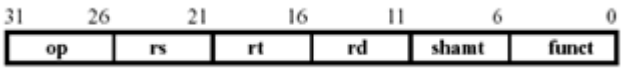
Conclusões: Comentários sobre o trabalho (complexidade/facilidade, sugestões, etc.) e contribuições do trabalho para o entendimento sobre processadores.

Código-fonte e Executável do Simulador (em C, C++ , Java ou Python).

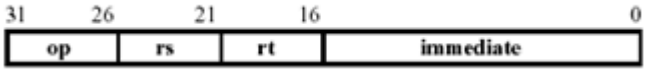
Apêndice – Instruções MIPS CES25.2018SE e Algoritmo Tomasulo

Apêndice – Instruções MIPS CES25.2018

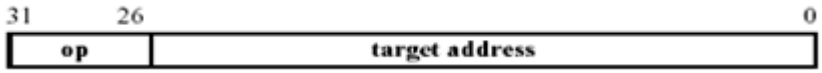
Instrução Formato R



Instrução Formato I



Instrução Formato J



Os campos ***immediate*** guardam números de 16 bits em complemento de dois, com bit mais a esquerda representando o sinal (1-negativo, 0 – positivo).

Os identificadores de registradores ***rs,rt*** e ***rd*** são números que identificam um registrador de R0 a R31 (00000-R0, 00001-R1,....).

O campo shamt não é usado no MIPS CES25.2018. Os campos op e funct tem o significado apresentado na tabela abaixo com a lista de instruções mínimas a serem implementadas no MIPS CES25.2018. O registrador R0 pode ser usado em instruções é um falso registrador, tendo valor igual a zero em qualquer situação. Tentativas de escrever em R0 devem ser desconsideradas.

Conjunto de Instruções a serem implementadas no MIPS CES25.2018						
mnemônico	**Tipo**	**opcode**	**Funct**	**Exemplo**	**Significado**	**Microprograma**
Add	R	"000000"	"100000"	add R9,R1,R7	R9=R1+R7	R[rd]=R[rs]+R[rt]
Addi	I	"001000"	-	addi R6,R5,1	R6=R5+1	R[rt]=R[rs]+ImmExt
Beq	I	"000101"	-	beq R1,R2,10	If(R1==R2) {PC=PC+1+10}	If(R[rs]=R[rt]) { PC=PC+4+Imm}
Ble	I	"000111"	-	ble R1,R2,10	If(R1<=R2) { PC=10 }	If(R[rs]<=R[rt]) { PC=Imm }
Bne	I	"000100"	-	bne R1,R2,10	If(R1!=R2) { PC=PC+1+10}	If(R[rs]!=R[rt]) { PC=PC+4+Imm}
Jmp	J	"000010"	-	jmp 200	PC=200	PC=EndJump
Lw	I	"100011"	-	lw R1,24(R0)	R1=MEM[24+0]	R[rt]=MEM[R[rs]+ImmExt]
Mul	R	"000000"	"011000"	mul R9,R1,R7	R9=R1*R7	R[rd]=R[rs]*R[rt]
Nop	R	"000000"	"000000"	nop	-	no operation
Sub	R	"000000"	"100010"	sub R9,R1,R7	R9=R1-R7	R[rd]=R[rs]-R[rt]
Sw	I	"101011"	-	sw R9,24(R0)	MEM[24+0]=R9	MEM[R[rs]+ImmExt]=R[rt]
LI	Instrução falsa, LI R1,100 é equivalente a ADDI R1,R0,100					

Bom Trabalho!
Prof. Paulo André Castro

Algoritmo de Tomasulo

Extraído de Hennesy & Patterson. Arquitetura de Computadores: Uma Abordagem Quantitativa. Tradução da 3ª. edição americana. Editora Campus-Elsevier.

Estado de Instrução		Esperar até	Ação ou contabilidade
Emitir	Operação de FP	Estação r vazia	`if (RegisterStat[rs].Qi≠0) {RS[r].Qj ← RegisterStat[rs].Qi} else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0}; if (RegisterStat[rt].Qi≠0) {RS[r].Qk ← RegisterStat[rt].Qi else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0}; RS[r].Busy ← yes; RegisterStat[rd].Qi=r;`
	Load ou Store	Buffer r vazio	`if (RegisterStat[rs].Qi≠0) {RS[r].Qj ← RegisterStat[rs].Qi} else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0}; RS[r].A ← imm; RS[r].Busy ← yes;`
	Load somente		`RegisterStat[rt].Qi=r;`
	Store somente		`if (RegisterStat[rt].Qi≠0) {RS[r].Qk ← RegisterStat[rs].Qi} else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0};`
Executar	Operação de FP	(RS[r].Qj = 0) e (RS[r].Qk = 0)	Calcular resultado: operandos estão em Vj e Vk
	Load-Store etapa 1	RS[r].Qj = 0 & r é o início da fila de carga-armazenamento	`RS[r].A ← RS[r].Vj + RS[r].A;`
	Load etapa 2	Load etapa 1 concluída	Ler de Mem[RS[r].A]
Gravar Resultado	Operação de FP ou Load	Execução concluída em r e CDB disponível	`∀x(if (RegisterStat[x].Qi=r) {Regs[x] ← result; RegisterStat[x].Qi ← 0}); ∀x(if (RS[x].Qj=r) {RS[x].Vj ← result;RS[x].Qj ← 0}); ∀x(if (RS[x].Qk=r) {RS[x].Vk ← result;RS[x].Qk ← 0}); RS[r].Busy ← no;`
	Store	Execução concluída em r e RS[r].Qk = 0	`Mem[RS[r].A] ← RS[r].Vk; RS[r].Busy ← no;`

Bom Trabalho!

Prof. Paulo André Castro
pauloac@ita.br