**Universidade Federal Fluminense**
**Disciplina: Arquitetura de Computadores**
**Professor: Leandro Santiago**
**Projeto 2 - Simulador do MIPS**

## Simulador do MIPS

Desenvolva um programa (C, Python, MIPS ou Javascript) que simule o comportamento do MIPS com pipelining. O programa deve receber como entrada um arquivo contendo o código binário de um programa (ou arquivo .asm) e executá-lo, exibindo, a cada ciclo:

a) Qual instrução está rodando em cada um dos 5 estágios do MIPS

b) O valor armazenado em cada um dos 32 registradores do MIPS

c) O conteúdo da memória (em hexadecimal) do endereço X ao Y

**Observação para implementação com MIPS:** Programa desenvolvido com MIPS deve rodar no simulador MARS e deve utilizar **macros** para modularizar o código. As informações exibidas a cada ciclo indicadas nos itens a) à c) devem ser escritas num arquivo de saída com o nome **saida.out** para não poluir o terminal. No caso do item c), não precisa imprimir todos os 1000 bytes, somente as posições de memória que contém posições preenchidas, indicando o endereço da memória e o conteúdo do endereço.

O simulador deve comportar as seguintes instruções no formato do ISA do MIPS 32 bits (`www.mips.com`) :

| Instrução | Tipo | Significado | Opcode em hexa | Funct em hexa |
|---|---|---|---|---|
| ADD | R | $R[rd] = R[rs] + R[rt]$ | 0 | 20 |
| ADDI | I | $R[rt] = R[rs] + SignExt(Imm)$ | 8 | N/A |
| AND | R | $R[rd] = R[rs\&R[rt]$ | 0 | 24 |
| BEQ | I | If (R[rs] == R[rt])<br>$PC = PC{+}4{+}(SignExt(Imm) << 2)$ | 4 | N/A |
| BNE | I | If (R[rs] != R[rt])<br>$PC = PC{+}4{+}(SignExt(Imm) << 2)$ | 5 | N/A |
| J | J | $PC = ((PC{+}4)[31{-}28])\|(Addr << 2)$ | 2 | N/A |
| JAL | J | $R[31] = PC + 4; PC =$<br>$((PC + 4)[31 - 28])\|(Addr << 2);$ | 3 | N/A |
| JR | R | $PC = R[rs]$ | 0 | 8 |
| LW | I | $R[rt] = M[R[rs] + SignExt(Imm)]$ | 23 | N/A |
| OR | R | $R[rd] = R[rs]\|R[rt]$ | 0 | 25 |
| SLL | R | $R[rd] = R[rs] << shamt$ | 0 | 0 |
| SRL | R | $R[rd] = R[rs] >> shamt$ | 0 | 2 |
| SW | I | $M[R[rs] + SignExt(Imm)] = R[rt]$ | 2b | N/A |
| SUB | R | $R[rd] = R[rs] - R[rt]$ | 0 | 22 |


O banco de registradores deve ser implementado por um vetor R de 32 posições inteiras (para facilitar a implementação), numeradas de 0 a 31. A tabela a seguir mostra o nome de cada registrador, seu número correspondente no MIPS e o uso típico:

| Número | Nome | Uso típico |
|---|---|---|
| 0 | $zero | Armazena a constante 0 e não pode ser escrito |
| 1 | $at | Assembler temporário |
| 2 | $v0 | Armazenar retorno de função e avaliação de expressões |
| 3 | $v1 | Armazenar retorno de função e avaliação de expressões |
| 4 | $a0 | Argumento de função |
| 5 | $a1 | Argumento de função |
| 6 | $a2 | Argumento de função |
| 7 | $a3 | Argumento de função |
| 8 | $t0 | Temporário |
| 9 | $t1 | Temporário |
| 10 | $t2 | Temporário |
| 11 | $t3 | Temporário |
| 12 | $t4 | Temporário |
| 13 | $t5 | Temporário |
| 14 | $t6 | Temporário |
| 15 | $t7 | Temporário |
| 16 | $s0 | Temporário salvo |
| 17 | $s1 | Temporário salvo |
| 18 | $s2 | Temporário salvo |
| 19 | $s3 | Temporário salvo |
| 20 | $s4 | Temporário salvo |
| 21 | $s5 | Temporário salvo |
| 22 | $s6 | Temporário salvo |
| 23 | $s7 | Temporário salvo |
| 24 | $t8 | Temporário |
| 25 | $t9 | Temporário |
| 26 | $k0 | Reservado para o kernel do SO |
| 27 | $k1 | Reservado para o kernel do SO |
| 28 | $gp | Global pointer |
| 29 | $sp | Stack pointer |
| 30 | $fp | Frame pointer |
| 31 | $ra | Return address |

Você pode conferir mais detalhes da especificação da arquitetura de 32 bits do MIPS no link: `https://www.mips.com/products/architectures/mips32-2/`.

A memória de dados será um vetor M de bytes com, no máximo 1000 posições.

O pipeline simulado deve ter as seguintes caracteríticas básicas:

- Instruções J, JR, JAL, BEQ e BNE fazem o desvio no estágio 2 (ID). Sendo assim, deveria existir um stall de 1 ciclo para garantir o funcionamento correto. Você não precisa implementar o stall na versão básica. Isto será feito pelo programador, com uma instrução de NOP logo após as instruções de desvio, ou usando a técnica de delay slot.

- Na versão original não teremos forwarding e nem detecção de hazards. NOPs devem ser inseridos pelo programador para garantir o funcionamento correto do programa.

**EXTRAS:** Implementar também as seguintes opções no simulador (que podem ser ativadas/desativadas em linha de comando):

a) Implementar o mecanismo de previsão de desvio de 2 bits, incluindo o flush do pipe em caso de desvios previstos incorretamente

b) Implementar todos os forwards possíveis. No caso de dependências entre instruções de LW e outras que precisam do resultado do LW no estágio EX, deveria existir um stall de 1 ciclo para garantir o funcionamento correto. Caso não exista unidade de detecção de hazards, o stall será feito pelo programador, com uma instrução de NOP logo após as instruções de desvio, ou escalonando uma instrução para ocupar o lugar da bolha.

c) Implementar unidade de detecção de hazards de dados. Caso não exista unidade de fowarding, todos os hazards de dados devem ser detectados (gerando bolhas). Caso exista unidade de forward, somente detecte os hazards de dados não tratados por forwarding.

O projeto pode ser feito em grupo de até 4 alunos e a entrega do trabalho deve ser feita enviando um arquivo compactado contendo:

a) O código fonte do simulador com todos os arquivos associados

b) O binário e fonte dos programas em assembly usados para teste

c) Um ppt com uma apresentação sobre o projeto

d) Um arquivo README.txt descrevendo como utilizar o simulador