TRABALHO 2: MONTADOR

LINGUAGEM C | COMPILADOR GCC | SISTEMA OPERACIONAL LINUX

Considerando a arquitetura descrita no Anexo I, implemente um programa montador que lê um arquivo em linguagem de montagem .mips e gera um arquivo binário executável .bin para a arquitetura especificada.

Os arquivos texto .mips possuem as seguintes características:

- Cada linha possui uma única instrução;
- Após o nome da instrução haverá um único caracter de espaço;
- Os operandos seguem separados por vírgula, sem espaço em branco entre eles;
- O final do arquivo é detectado por EOF.

exemplo.mips

```
xor r1,r1,r1
xor r2,r2,r2
addi r1,r1,20
addi r2,r2,30
add r0,r1,r2
```

Os arquivos binários .bin devem possuir as seguintes características:

- Cada registro possui 32 bits contendo a codificação da instrução;
- O final do arquivo é detectado por EOF.

O programa montador deve receber como argumento da linha de comando o nome do arquivo .mips e deve gerar o arquivo .bin com o mesmo nome (ex.: teste.mip => teste.bin).

O código deve estar devidamente organizado, identado, comentado e sem *warnings* usando as opções -Wall -Wextra -pedantic do gcc.

Cópias e similares: nota zero para todos os envolvidos.

ANEXO I – A arquitetura

A arquitetura, derivada de MIPS, possui as seguintes características:

- 32 bits com operação em modo dual

- Memória principal: possui **2^{20} bytes** de memória instalada
  - Representada por um vetor MEMORY de **uint8_t**;
  - No início de uma execução todas as células de memória têm valor 0

- Registradores (32 bits cada):
  - 10 de uso geral: r0, ..., r9
  - registradores para divisão/multiplicação: r10 (LO) e r11 (HI)
  - registradores para tratar chamadas ao SO: r12 (número) e r13, r14, r15 (argumentos)
  - ir: registrador de instrução
  - pc: contador de programa
  - base: endereço inicial do processo
  - limit: tamanho em bytes do processo
  - **mode: registrador de modo, onde 0 indica modo kernel e 1 modo usuário**

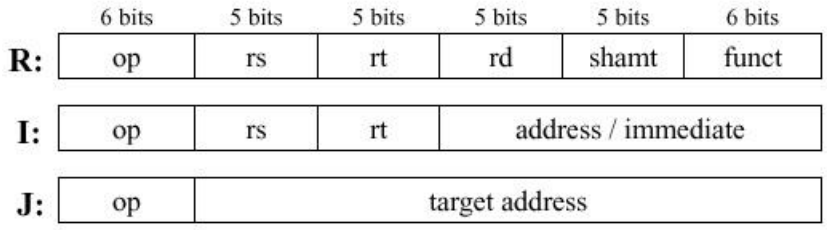
- Todos os registradores possuem valor 0 quando a arquitetura é iniciada, **exceto o registrador limit, que possui valor inicial 2^{20}-1**.

- Estrutura de dados para representar os registradores da CPU:

```
struct s_regs
{
      int32_t r[16];

      uint32_t ir;
      uint32_t pc;
      uint32_t base;
      uint32_t limit;
      uint8_t mode;
};
```

- Formato das instruções (tipo R, I e J):



op: basic operation of the instruction (opcode)
rs: first source operand register
rt: second source operand register
rd: destination operand register
shamt: shift amount
funct: selects the specific variant of the opcode (function code)
address: offset for load/store instructions (+/-2^{15})
immediate: constants for immediate instructions

- Conjunto de instruções:

Instruções	**Tipo de operação**
add, addi, sub, mult, div, and, andi, or, ori, slt, slti, xor, sll, srl	Lógica/aritmética
j, beq, bne, bgez, bgtz	Desvio
lb, sb, lw, sw, lui	Acesso à memória
syscall	Chamada ao SO
readb (**privilegiada** – considerar opcode 1111 00)	Leitura de dispositivo
writeb (**privilegiada** – considerar opcode 1111 01)	Escrita em dispositivo

Na URL <https://phoenix.goucher.edu/~kelliher/f2009/cs220/mipsir.html> é apresentada a descrição, operação, sintaxe e codificação dessas instruções. No entanto, neste trabalho os bits das instruções marcados com '-' ("*don't care*") deverão ter o valor 0.