Universidade Federal de Viçosa

Trabalho Prático 01 – Montador RISC-V
CCF 252 – Organização de Computadores I
Ciência da Computação – Campus UFV-Florestal
Prof. José Augusto Miranda Nacif

Montador RISC-V (versão simplificada)

O trabalho consiste na implementação de uma versão simplificada de um montador RISC-V de acordo com as seguintes especificações:

1. Arquivo de entrada: Deve seguir o padrão de codificação definido no livro texto e possuir a extensão asm.
2. Arquivo de saída: Deve ser compatível com um simulador previamente escolhido.

O montador deve suportar um subconjunto de 7 instruções formado à partir de: `lb`, `lh`, `lw`, `sb`, `sh`, `sw`, `add`, `sub`, `and`, `or`, `xor`, `addi`, `andi`, `ori`, `sll`, `srl`, `bne`, `beq`. A Tabela 01 apresenta qual conjunto de instruções deve ser implementado por cada grupo. Cada grupo deve implementar apenas o seu subconjunto de instruções.

Tabela 01 - Subconjunto de instruções RISC-V a ser implementado por cada grupo.

Grupo	Instruções						
	01	02	03	04	05	06	07
01	`lb`	`sb`	`add`	`and`	`ori`	`sll`	`bne`
02	`lh`	`sh`	`sub`	`or`	`andi`	`srl`	`beq`
03	`lw`	`sw`	`add`	`xor`	`addi`	`sll`	`bne`
04	`lb`	`sb`	`sub`	`and`	`ori`	`srl`	`beq`
05	`lh`	`sh`	`add`	`or`	`andi`	`sll`	`bne`
06	`lw`	`sw`	`sub`	`xor`	`addi`	`srl`	`beq`
07	`lb`	`sb`	`add`	`and`	`ori`	`sll`	`bne`
08	`lh`	`sh`	`sub`	`or`	`andi`	`srl`	`beq`
09	`lw`	`sw`	`add`	`xor`	`addi`	`sll`	`bne`
10	`lb`	`sb`	`sub`	`and`	`ori`	`srl`	`beq`
11	`lh`	`sh`	`add`	`or`	`andi`	`sll`	`bne`

12	lw	sw	sub	xor	addi	srl	beq
13	lb	sb	add	and	ori	sll	bne
14	lh	sh	sub	or	andi	srl	beq
15	lw	sw	add	xor	addi	sll	bne
16	lb	sb	sub	and	ori	srl	beq
17	lh	sh	add	or	andi	sll	bne
18	lw	sw	sub	xor	addi	srl	beq
19	lb	sb	add	and	ori	sll	bne
20	lh	sh	sub	or	andi	srl	beq
21	lw	sw	add	xor	addi	sll	bne
22	lb	sb	sub	and	ori	srl	beq
23	lh	sh	add	or	andi	sll	bne
24	lw	sw	sub	xor	addi	srl	beq
25	lb	sb	add	and	ori	sll	bne
26	lh	sh	sub	or	andi	srl	beq
27	lw	sw	add	xor	addi	sll	bne
28	lb	sb	sub	and	ori	srl	beq
29	lh	sh	add	or	andi	sll	bne
30	lw	sw	sub	xor	addi	srl	beq
31	lb	sb	add	and	ori	sll	bne
32	lh	sh	sub	or	andi	srl	beq

O montador pode ser implementado em qualquer linguagem de programação e em qualquer paradigma, e será testado em uma máquina Linux.

Pontuação extra pode ser dada para:

- Implementação de pseudo instruções;
- Suporte a outras bases numéricas no arquivo .asm;

- Implementação de outras instruções fora do conjunto utilizado para gerar a combinação de cada grupo.

O trabalho prático pode ser feito em grupos de **ATÉ** 2 alunos.

Cópias de trabalhos práticos serão exemplarmente punidas. A punição será a mesma para quem copiou e para quem forneceu o trabalho prático.

Forma de entrega: Github para o código, PVANet Moodle para a documentação em PDF.

O que deve ser entregue:

- Documentação simplificada do trabalho prático em PDF:
  - Utilizar LaTeX no Overleaf e o modelo da Sociedade Brasileira de Computação (SBC)[1];
  - Caso o usuário use uma linguagem diferente de C ou C++, a documentação deve conter instruções com os comandos sobre como instalar o compilador ou o interpretador necessário no Linux.
  - A documentação deve conter instruções sobre como executar o montador, contendo capturas de tela dos resultados.
  - É muito interessante explicar partes importantes do código e mostrá-las na documentação. Evite transliterar o código, mas sim explicá-los em linguagem natural.

- Arquivos fontes do trabalho:
  - Contendo uma makefile que execute a compilação do programa, caso seja necessário (para a geração do binário de linguagens compiladas como C, C++) ou para a geração de bytecodes de linguagens interpretadas que precisam disso (java ou golang);
  - O montador deve ter duas formas de exibir o resultado:
    - No terminal: caso o cliente passe por linha de comando somente o arquivo .asm, o programa deverá exibir na tela do terminal todas as instruções em binário, uma instrução por linha;
    - No arquivo: caso o cliente passe por linha de comando uma entrada convencional de compiladores (./binario_programa entrada.asm -o saida), o programa deverá salvar o código binário, uma instrução por linha, no arquivo depois do parâmetro -o (de output, saída). Caso a linguagem escolhida

[1] https://pt.overleaf.com/latex/templates/sbc-conferences-template/blbxwjwzdngr

seja interpretada, não há problemas em ter a necessidade de colocar, como primeiro argumento, o interpretador (Exemplo: python3 meuMontador entrada.asm -o saida). desde que devidamente documentado.

Exemplo de entrada:

entrada.asm

```
add x2, x0, x1
sll x1, x2, x2
or x2, x2, x1
andi x2, x1, 16
addi x3, x2, -243
```

Exemplo de saída:

arquivo ***saída*** ou escrito no terminal

```
00000000000100000000000100110011
00000000001000010001000010110011
00000000000100010110000100110011
00000001000000000111000100010011
11110000110100010000000110010011
```

Novos exemplos de entrada:

entrada_2.asm

```
add x2, x0, x1
add x6, x0, x2
sub x3, x6, x2
xor x4, x2, x3
srl x0, x2, x2
```

Exemplo de saída:

```
00000000000100000000000100110011
00000000001000000000001100110011
01000000001000110000000110110011
00000000001100010100001000110011
00000000001000010101000000110011
```

Todas as instruções podem ser encontradas no documento de especificação do ISA:
https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf