Relatório de Implementação

Estudante: Cristiano Tolentino Santos
Matrícula: 211028050
Professor: Ricardo Pezzuol Jacobi
Turma: T02 (2024.2 - 35T23)

Apresentação do problema

Este projeto apresenta a implementação de um simulador para a arquitetura RV32I, desenvolvido no IDE Visual Studio Code, em Python 3, com uma máquina utilizando Windows 10 como sistema operacional. A função principal do simulador é a execução de instruções de assembly, permitindo simular o funcionamento de um processador RISC-V. O simulador é alimentado com arquivos binários gerados pelo montador RARS, que traduz programas escritos em Assembly para o formato binário executável pela arquitetura. Esses arquivos binários contêm segmentos de código que são carregados na memória simulada e executados pelo simulador. O código executado foi projetado como uma bateria de testes abrangente, avaliando todas as funcionalidades implementadas no simulador, como instruções aritméticas, lógicas, de memória e de controle de fluxo, validando a precisão da simulação.
O código utilizado implementa uma sequência de testes os quais verificam a execução correta de operações aritméticas (add, addi, sub), lógicas (and, andi, or, ori, xor), de memória (lw, sw, lb, lbu, sb), e de controle de fluxo (beq, bne, jal, jalr).

Descrição das instruções implementadas

- **lb**: Carrega 1 byte com extensão de sinal.
- **lbu**: Carrega 1 byte sem extensão de sinal.
- **lw**: Carrega 4 bytes (palavra).
- **lui**: Carrega um valor imediato nos 20 bits superiores.
- **sb**: Armazena 1 byte na memória.
- **sw**: Armazena 4 bytes (palavra) na memória.
- **add**: Soma dois registradores.
- **addi**: Soma um registrador com um imediato.
- **and**: Faz a operação AND lógico entre dois registradores.
- **andi**: Faz a operação AND lógico entre um registrador e um imediato.
- **auipc**: Soma um imediato ao valor atual do PC e armazena no registrador.
- **sub**: Subtrai dois registradores.
- **bge**: Salta se o primeiro registrador é maior ou igual ao segundo (sinalizado).
- **bgeu**: Salta se o primeiro registrador é maior ou igual ao segundo (não sinalizado).
- **blt**: Salta se o primeiro registrador é menor que o segundo (sinalizado).
- **bltu**: Salta se o primeiro registrador é menor que o segundo (não sinalizado).

- **beq**: Salta se dois registradores forem iguais.
- **bne**: Salta se dois registradores forem diferentes.
- **jal**: Salta para um endereço relativo e armazena o endereço de retorno.
- **jalr**: Salta para um endereço absoluto (com base em um registrador).
- **or**: Faz a operação OR lógico entre dois registradores.
- **ori**: Faz a operação OR lógico entre um registrador e um imediato.
- **slt**: Define 1 se o primeiro registrador é menor que o segundo (sinalizado).
- **sltu**: Define 1 se o primeiro registrador é menor que o segundo (não sinalizado).
- **slli**: Desloca bits para a esquerda.
- **srai**: Desloca bits para a direita com extensão de sinal.
- **srli**: Desloca bits para a direita sem extensão de sinal.
- **xor**: Faz a operação XOR lógico entre dois registradores.
- **ecall**: Realiza uma chamada de sistema (syscall).

Testes e resultados

Teste 1: Carrega-se -2 no registrador t1 e 3 no registrador t2. Em seguida, soma-se esses valores e o resultado é armazenado em t3. Compara-se t3 com 1; caso sejam iguais, o programa segue para t1_ok, indicando sucesso. Caso contrário, o fluxo é direcionado para FAIL. O valor esperado para t3 é 1, visto que -2 + 3 resulta em 1.

Teste 2: Carrega-se -2048 no registrador t1. Utilizando a instrução addi, soma-se -2048 com t1. Verifica-se se o resultado é -4096. Caso correto, o programa segue para t2_ok; se não, direciona-se para FAIL. O valor esperado para t1 após a operação é -4096.

Teste 3: Carrega-se 0x37F no registrador t1 e 0x6E3 no registrador t2. Realiza-se a operação AND entre t1 e t2, e o resultado é armazenado em t3. Compara-se t3 com 0x263; se forem iguais, o programa segue para t3_ok; caso contrário, vai para FAIL. O valor esperado para t3 é 0x263.

Teste 4: Realiza-se uma operação AND com imediato. Carrega-se 0x37F em t1 e realiza-se a operação AND com 0x584. O resultado é armazenado em t1. Compara-se t1 com 0x104; caso seja igual, o programa segue para t4_ok, caso contrário, vai para FAIL. O valor esperado para t1 após a operação é 0x104.

Teste 5: Executa-se a instrução auipc com 0 em t1 e t2. Os valores resultantes refletem o PC somado a zero. Em seguida, realiza-se uma subtração entre t2 e t1, esperando uma diferença de 4 devido ao avanço do PC. O valor esperado para essa diferença é 4.

Teste 6: Mistura-se as instruções beq e bne. Carrega-se -100 em t0 e -200 em t1. Em seguida, ajusta-se t1 em um loop até que se iguale a t0. Quando isso ocorre, verifica-se se as instruções de salto funcionam corretamente. Se tudo ocorrer conforme o esperado, o programa segue para t6_ok; caso contrário, vai para FAIL.

Teste 7: Testam-se as condições de "maior ou igual" com e sem sinal. Valores negativos são carregados nos registradores e as instruções são utilizadas para verificar os comportamentos esperados. O comportamento esperado é que as condições funcionem conforme especificado.

Teste 8: Utilizam-se as instruções blt e bltu para testar a condição "menor que". O comportamento esperado é que as comparações com e sem sinal sejam realizadas corretamente.

Teste 9: Carrega-se o endereço de uma palavra na memória para t0 e utiliza-se a instrução lw para ler o valor dessa palavra em t1. Compara-se t1 com o valor esperado, 0xFAFEF1F0. Se o valor for correto, o programa segue para t9_ok.

Teste 10: Utiliza-se a instrução lb para carregar apenas o primeiro byte da palavra na memória. O valor esperado para t1 é 0xFFFFFFF0.

Teste 11: Utiliza-se a instrução lbu, que não realiza a extensão de sinal. O valor esperado para t1 após a operação é 0xF0.

Teste 12: Utilizam-se as instruções lui e slli. Primeiro, carrega-se um valor grande com lui e, em seguida, desloca-se o valor de t1 12 bits para a esquerda com slli. O valor esperado é que ambos os resultados sejam equivalentes.

Teste 13: Testa-se a instrução jal, que faz o programa saltar para um rótulo, armazenando o endereço de retorno em ra. No rótulo, o programa retorna ra. O comportamento esperado é que o salto e o retorno funcionem corretamente.

Teste 14: Semelhante ao teste 13, mas utilizando a instrução jalr. O endereço de destino é calculado e armazenado em t1. O salto ocorre para t1 e o retorno é realizado corretamente.

Teste 15: Realiza-se uma operação OR entre 0x120 e 0x310. O valor esperado para o resultado é 0x330. Se correto, o programa segue para t15_ok.

Teste 16: Utiliza-se a instrução ori, que realiza uma operação OR com imediato. O valor esperado para t1 após a operação é 0x130.

Teste 17: Utiliza-se a instrução slt. Carrega-se 5 em t1 e 10 em t2. A instrução verifica se t1 é menor que t2, o que é verdadeiro, portanto, t3 deve ser igual a 1.

Teste 18: Utiliza-se a instrução sltu, que realiza a comparação sem sinal. Compara-se 0xFFFFFFF0 com 0x10. Como o primeiro valor não é menor, o valor esperado para t3 é 0.

Teste 19: Realiza-se uma operação XOR entre 0xAAAA e 0x5555. O valor esperado para o resultado é 0xFFFF.

Teste 20: Utiliza-se a instrução ecall, que faz o programa imprimir mensagens, valores e encerrar corretamente. O comportamento esperado é que todas as saídas sejam exibidas conforme planejado antes do término.