Relatório pratica 3 – Algoritmo Tomasulo

Arthur Severo

Victor Le Roy

1. **Introdução:**

Este trabalho consiste na apresentação e construção do algoritmo Tomasulo sem especulação.

1. **Desenvolvimento:**

Para este projeto, optamos por utilizar instruções de 12 bits, sendo eles:

|  |  |  |  |
| --- | --- | --- | --- |
| **3 bits** | **3 bits** | **3 bits** | **3 bits** |
| Op | Rd | Rx | Ry |

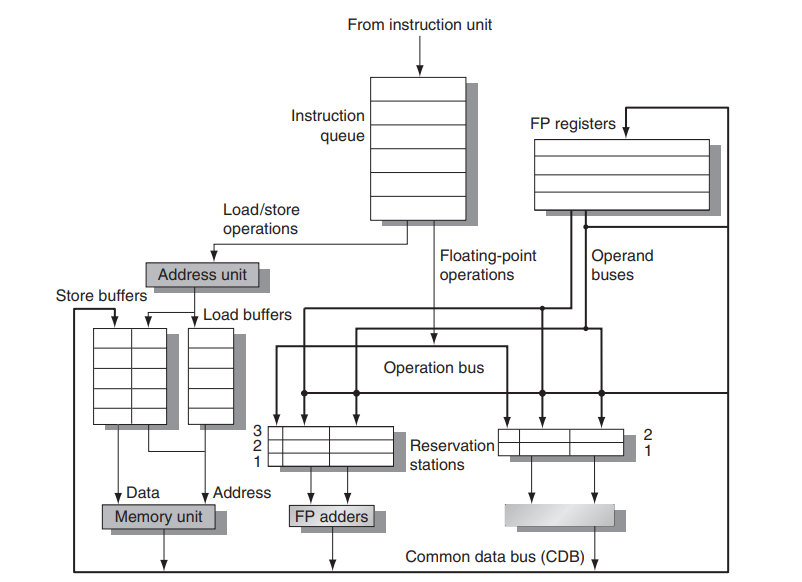
Sendo Op, a operação, Rd, o registrador destino e Rx e Ry, os registradores fonte.

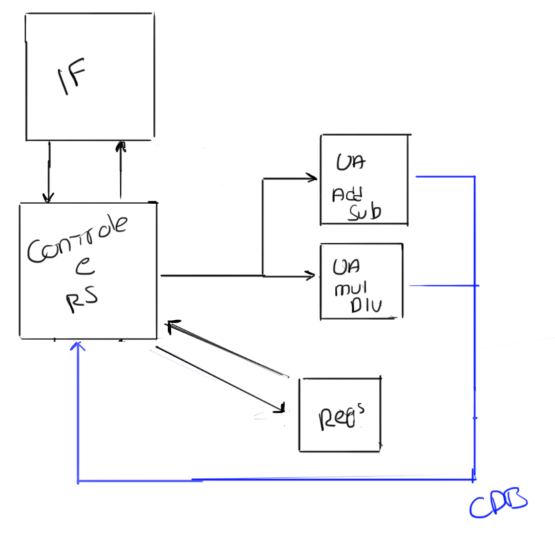
Utilizamos como código de operação:

|  |  |
| --- | --- |
| **Operação** | **Bits** |
| ADD | 000 |
| SUB | 001 |
| MUL | 010 |
| DIV | 011 |

1. **Módulos:**

Foi criado um diagrama a partir do esquema do algoritmo disponibilizado no livro da disciplina (referência 1), sendo este:



A partir do esquema anterior foi criado o seguinte diagrama:

- Unidade funcional/artimética (UA) AddSub:

Realiza operações de soma e subtração e possui a latência 1 ciclo.

- Unidade funcional/artimética (UA) MulDiv:

Realiza operações de multiplicação e divisão e possui a latência 2 ciclos.

- Fila de instruções (IF):

Gerencia os despachos da estação de reserva. Neste módulo, está setado uma memória contendo as instruções, além do contador do processador.

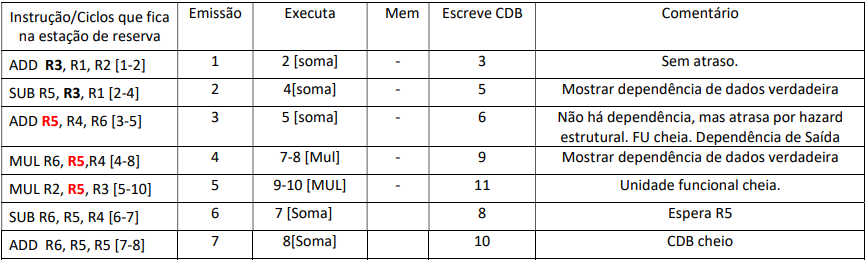
- Controle e RS:

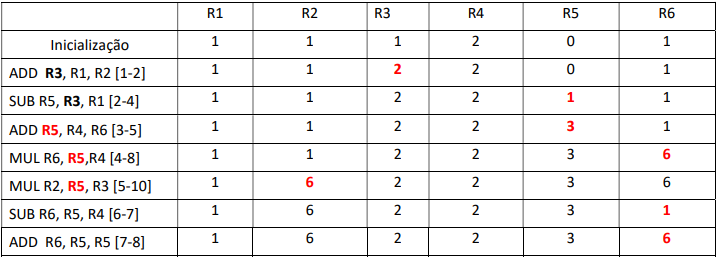
Implementa o processador como um todo. Isto é, implementa o CDB, o controle de hazard e execução das instruções, além da estação de reserva.

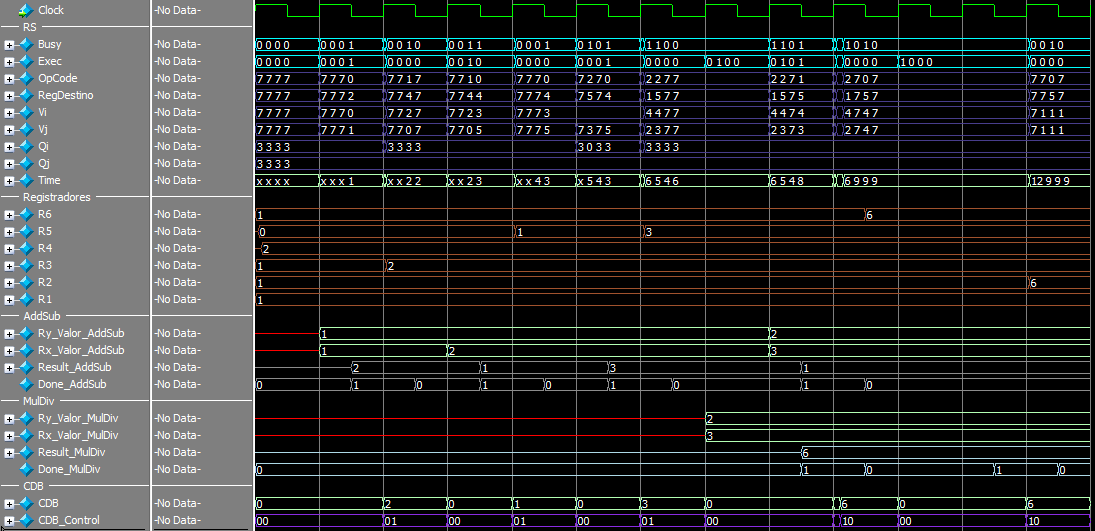
1. **Funcionamento**

Primeiro verificamos se há espaço na estação de reserva. Caso haja, a instrução é despachada da fila e colocada no espaço vazio da estação. Após isso, verifica-se dependência de dados verdadeira, se houver é preenchido os valores e suas dependencias. Se a instrução não tiver nenhuma dependencia, esta é executada e no após a latência, é escrito o valor no registrador.

1. **Testes**

Foi realizado os testes a partir do código a seguir:

E os valores que deveriam acontecer são:

E foi obtido:

É possível perceber uma diferença de 2 ciclos comparado com o previsto. Isso se deu, pois, uma instrução teve sua execução adiada, o que resultou neste problema e no adiamento das instruções subsequentes. Este adiamento fez com que a instrução **SUB R6, R5, R4 [6-7]** tivesse sua execução terminada junto com **MUL R6, R5, R4 [4-8]**, porém ainda é possível ver que, pelo fato de a segunda instrução escrever após a primeira, temos o resultado correto no **R6**. Infelizmente, não conseguimos identificar a solução do problema citado acima.

Além desse problema, foi visto que, em algumas simulações, algumas operações tiveram algum comportamento parecido com o problema citado anteriormente.

1. **Conclusão:**

Este projeto foi com toda certeza o mais desafiador que tivemos até então. Creio que a maior dificuldade veio do fato da falta de conhecimento avançado na linguagem Verilog, o que gerou os problemas citados anteriormente, que foram apenas exemplos dos problemas encontrados, porém, este desafio fez com que pudéssemos aprender bastante sobre a teoria da disciplina, além de aprender mais sobre o algoritmo. Aproveito para agradecer aos veteranos que se disponibilizaram a nos ajudar a resolver este projeto.

1. **Referencias:**

[1] John L Hennessy and David A Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.