Relatório de Projetos Digitais e Microprocessadores

Eloisa Nielsen (GRR20221226)
Ciência da Computação
Universidade Federal do Paraná – UFPR
Curitiba, Brasil

I. Introdução

Este trabalho tem como objetivo a implementação dos circuitos de dados e de controle de um MICO X1 que receba instruções de 32 bits, por meio da ferramenta Logisim Evolution.

Esse processador será composto, principalmente, por um Ponteiro de Instruções, uma Memória de Programa, uma Memória de Controle, um Banco de Registradores (16x32 bits), uma ULA, um Display e circuitos auxiliares, como extensores de sinais, muxes, somadores e portas lógicas. Como dentro dessa implementação haverá uma ULA, que executa operações de lógica e aritmética com números de 32 bits, sendo elas soma, subtração, AND, OR, XOR, NOT, e deslocamentos de bits para esquerda e direita, serão analisados: o somador Ripple Carry, o somador Seletor de Carry e o somador Carry-Lookahead para que seja implementado na ULA o somador que melhor se encaixa nesse processador para a realização da soma e da subtração em relação à área que ele ocupará e ao atraso de sua lógica.

II. Escolha do Somador

O Somador Ripple Carry para 32 bits é feito de 32 Full Adders, onde cada FA depende do resultado do Carry Out do anterior, como dentro dos FA, o Cout é a saída de maior atraso, com 3 portas, o maior atraso será para o resutado do último Cout, e assim o atraso lógico desse circuito será de 32 * 3 portas, ou seja 96. Já sua área, como cada FA é feito de 6 portas, é de 32 * 6 portas, ou seja 192.

O Somador Seletor de Carry de 32 bits faz todas as somas dos FA ao mesmo tempo, em 4 FA que recebem os primeros 4 bits e em mais 7 blocos de 8 FA, onde 4 FA recebem 4 bits de A e B e CIN 1 e os outros 4 recebem os mesmos 4 bits de A e B, mas com CIN 0, e então usa muxes para escolher os 4 bits de resultado e o Cout do blocos baseando-se nos Cout anteriores. Como todas as operações dos FA são feitas ao mesmo tempo, o atraso irá depender do primeiro bloco de 4 FA e dos muxes de cada bloco que dependem dos anteriores. O Cout do primeiro bloco tem atraso de 4 * 3 portas. Um mux tem atraso de 3 portas, como são 7 muxes que escolhem o Cout e dependem um do outro, o atraso será de 7 * 3 + 12, ou seja, 33 portas. Porém, sua área é de 4 FA + 7 * 8 FA + 7 * 5 muxes, que é igual a 60 FA + 35 * 4(número de portas de um mux) portas, então, 500.

Finalmente, o Somador Carry-Lookahead tem um circuito que antecipa o calculo do vai-um usando os bits menos significativos. Sendo constituido de 4 FA e um 4-bit carry-lookahead para calculos de 4 bits de entrada. Com 32 bits, para maior eficiência, são usados 8 desses blocos. Como o atraso de cada bloco é o atraso do 4-bit carry-lookahead, que é de 7, o atraso total é 7 * 8 portas, ou seja 56. Sua área é de 36(dos 4 FA e do 4-bit carry-lookahead) * 8 portas, ou seja, 288.

Nome do Somador	Atraso	Tamanho
Ripple Carry	96	192
Seletor de Carry	33	500
Carry-lookahead	56	288

A partir dos cálculos citados anteriormente, o somador escolhido foi o Somador Carry-lookahead. O Ripple Carry tem a menor área, porém seu grande atraso acaba diminuindo muito sua eficiência, enquanto o Seletor de Carry, mesmo acelerando muito o processo de soma em relação aos outros somadores, acaba não se tornado a escolha ideal por ocupar uma área descomunal em relação ao resto. Então, o Carry-lookahead, como uma média em relação a velocidade e área dos outros somadores, foi considerada a melhor escolha.

III. Código teste de instruções

As instruções do processador são de 32 bits. Os bits de 0 a 15 representam o valor da constante, de 16 a 19 o endereço do registrador c, de 20 a 23 o endereço do registrador b, de 24 a 27 o endereço do registrador a e de 28 a 31 o código da função. As instruções tiveram de ser traduzidas de binário para hexadecimal para serem escritas dentro da Memória de Programa, que ficou organizada da seguinte maneira;

```
000000 00000000 b001000f b0020003 11230000
000004 c3000000 21230000 c3000000 31230000
000008 c3000000 41230000 c3000000 51230000
00000c c3000000 61230000 c3000000 71230000
000010 c3000000 81230000 c3000000 91230000
000014 c3000000 a1230000 c3000000 d0000003
000018 b202000c c2000000 e120000c f0000000
```

Figura 1. Instruções em hexadecimal.

O código teste está representado na tabela a seguir, nele há um teste para cada instrução que o processador pode executar;

Tempo	Instrução
IP→ 0	nop
IP→ 1	R(1) ← R(1) + extZero(15)
IP→ 2	R(2) ← R(2) + extZero(3)
IP→ 3	R(3) ← R(1) + R(2)
IP→ 4	Display R(3)
IP→ 5	R(3) ← R(1) - R(2)
IP→ 6	Display R(3)
IP→ 7	R(3) ← R(1) AND R(2)
IP→ 8	Display R(3)
IP→ 9	R(3) ← R(1) OR R(2)
IP→ 10	Display R(3)
IP→ 11	R(3) ← R(1) XOR R(2)
IP→ 12	Display R(3)
IP→ 13	R(3) ← not R(1)
IP→ 14	Display R(3)
IP→ 15	R(3) ← R(1) ≪ R(2)
IP→ 16	Display R(3)
IP→ 17	R(3) ← R(1) ≫ R(2)
IP→ 18	Display R(3)
IP→ 19	R(3) ← R(1) OR extZero(0)
IP→ 20	Display R(3)
IP→ 21	R(3) ← R(1) XOR extZero(0)
IP→ 22	Display R(3)
IP→ 23	IP ← IP + SignExt(3)
IP→ 24	R(2) ← R(2) + extZero(12)
IP→ 25	Display R(2)
IP→ 26	if R(1)==R(2):IP←IP+SignExt(12) else:IP←IP+1
IP→ 27	IP ← IP + 0

IV. O PROCESSADOR

Ao final, o processador foi organizado seguinte a imagem abaixo;

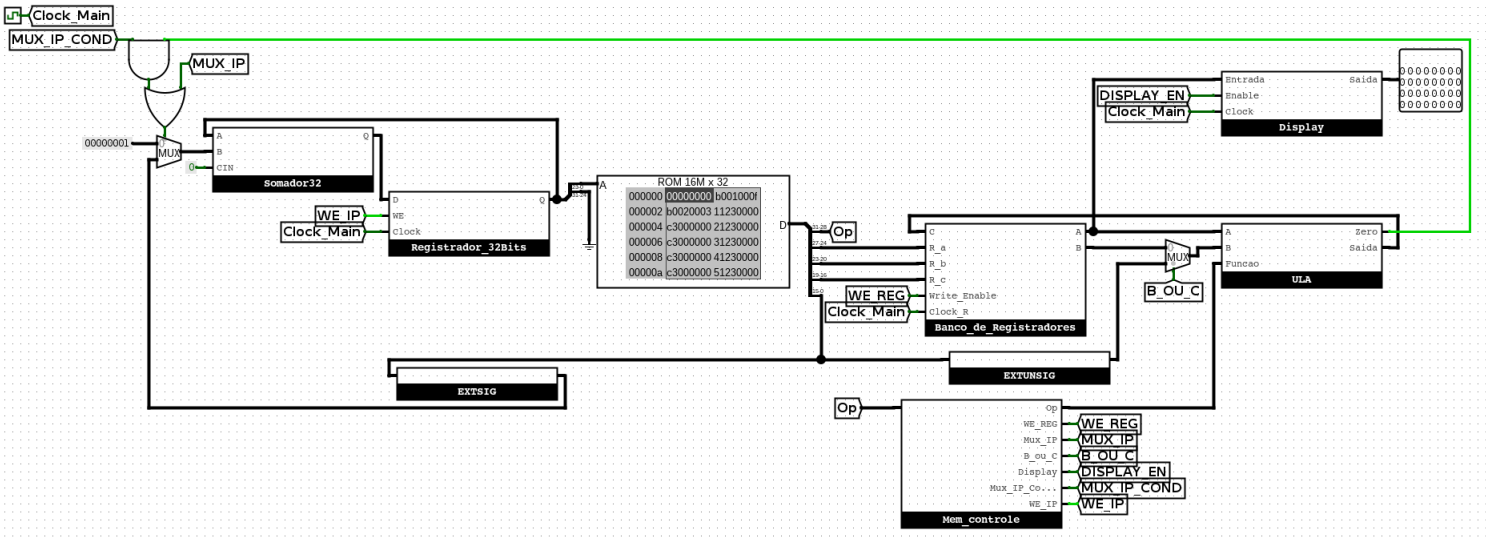

Figura 2. Processador completo.

REFERÊNCIAS

[1]. De Campos, Frederico Oioli. ”Elementos Basicos da Eletronica Digital”. São Paulo: ETE Júlio de Mesquita Arquivo. 2001