Projeto de Circuitos Integrados Digitais - Projeto final

Felipe R. Sobrinho (17/0141764), Luiz Henrique N. de Oliveira (16/0013712) e Kaio Vinicius P. de Carvalho (18/0065416)

I. Introdução

O processador de seis instruções é um dos modelos mais básicos de processador a ser implementado. De fato, processadores comerciais possuem dezenas ou centenas de instruções para a realização de diversas operações, mas esse processador sintetiza bem as instruções fundamentais de um processador, donde as demais são derivadas.

Para implementação de um processador seis instruções, é necessário a implementação de seus blocos fundamentais, que a princípio são dois: bloco(ou unidade) de controle e bloco de operacional. Cada bloco possuí seus componentes com os quais é possível a execução de todos os comandos programados. A figura 1 mostra o esquema básico de um processador de seis instruções.

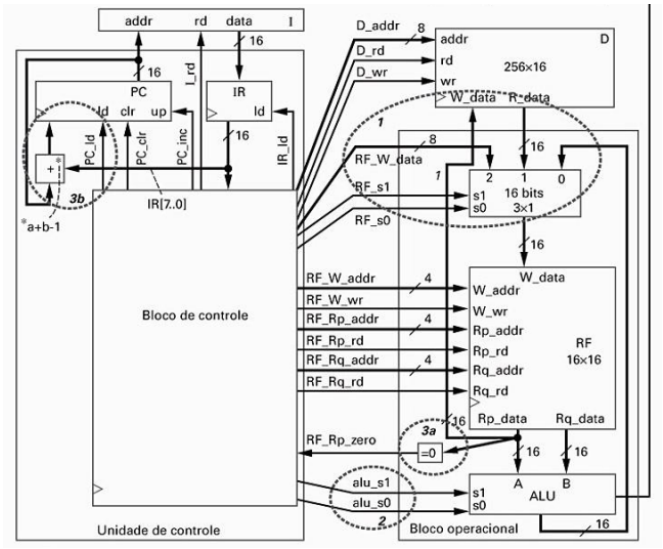

Fig. 1: Processador de seis instruções dividido em dois blocos fundamentais: bloco de controle e bloco operacional.

Uma forma de se descrever a implementação desse processador fisicamente é através da linguagem VHDL ("*VHSIC hardware Description Language*"). O VHDL é uma linguagem utilizada mundialmente para descrição de *hardware*, em conjunto com outras como o Verilog. Após realizada a descrição do *hardware*, é necessário sintetizá-lo, para verificação dos parâmetros reais do circuito, uma vez que para a implementação desse tipo de circuito é utilizado transistores de tecnologia CMOS, que possuí limitações físicas. A sintetização deve ser feita já considerando a tecnologia de fabricação que deverá ser utilizada. Após feita a sintetização do circuito, é possível então realizar a construção do layout no formato de um circuito integrado; processo feito também automaticamente (por conta da alta quantidade de transistores envolvida), mas com o controle de alguns parâmetros.

Tendo todos esses passos em mente, esse projeto visa realizar a implementação de um processador de seis instruções passando por todos os passos para a construção de um circuito integrado. Como software principal para a construção de tal processador, será utilizado o Cadence para a simulação do circuito (Nclaunch) e para a sintetização do circuito (Genus).

II. Descrição do funcionamento dos blocos internos

1) Unidade de controle: A unidade de controle é composta por 4 sub-blocos, são estes: Bloco de controle, somador, PC e IR. Esse formato de blocos funcionais de circuitos permite que seja realizado um *debug* passo-a-passo durante o processo de descrição das estruturas, adicionando confiabilidade e melhor correção das falhas encontradas no processo de teste. Em nível hierárquico, podemos dizer que o bloco de controle se coloca em posição central neste processador de 6 instruções, sendo responsável por comandar todas as variáveis de controle internas que determinam tando os caminhos de dados via MUX quanto as operações lógicas e aritméticas a serem realizadas pela ALU. Não menos importante, os blocos PC e IR componentes dessa unidade de controle são responsáveis pela comunicação correta com a memória de instruções, que ao carregar um arquivo externo .dat deve criar comandos no processador.

- Bloco de controle: Com construção interna simples baseada em uma maquina de estado finita, segue uma ordem pré-definida de estados padrão conforme sugerido na 23, além de interpretar e obedecer a instrução lida na memória, também é responsável por carregar os sinais de controle essenciais para que aconteça o funcionamento esperado do processador.
- Somador: Realiza um deslocamento do contador de programa quando identificado um comando de "saltar se zero". O deslocamento é feito realizando a soma do próprio valor do contador de programa, com o Offset destinado nos últimos 8 bits do programa, e subtraindo uma unidade, para descontar o comando de busca já executado.
- PC: o contador de programa realiza a contabilidade do estado do programa. Ele é responsável por gerenciar os passos dados pelo programa, fazendo saltar as

linhas do código de máquina presente na memória de programa/instrução.
- IR: é um registrador intermediário que guarda o valor do dado da memória de instrução para um determinado valor no contador de programa quando solicitado um comando de busca.

2) Bloco operacional: O bloco operacional é responsável por realizar todas as transformações de dados. Ele é comandado diretamente pelo bloco de controle, e realiza as operações utilizando os dados presentes na memória de dados e no banco de registradores. A memória de dados é utilizada apenas quando solicitado um comando de carregar ou armazenar, enquanto o banco de registradores é utilizado em todos os tipos de instruções. Para realizar as operações aritméticas entre os registradores, é utilizado a unidade lógica aritmética (ULA).

- ALU: Unidade lógica aritmética, responsável por realizar as operações matemáticas solicitadas na instrução, soma, subtração e concatenação.
- Multiplexador 3x1: O bloco multiplexador serve para guiar a informação desejada para o próximo bloco de processamento, neste caso temos um mux 3 entradas que de acordo com as portas de seleção transmite uma das opções de entrada na saída.
- Memória de Dados: a memória de dados funciona como uma memória dinâmica e volátil, ou seja, só guarda os dados quando o processador está em pleno funcionamento. Muito semelhante a memória RAM, possuí uma quantidade razoável de endereços que são utilizados para armazenar o valor presente nos registradores do bloco de registradores.
- Bloco de registradores: também conhecidos como "Register Files (RF)", são um conjunto de registradores utilizados pelo processador de maneira direta, diferentemente dos dados armazenados na memória de dados. Eles são indexados e operados diretamente pelo bloco de controle, num formato ditado pelo arquivo de programa.
- Comparador de Igualdade: responsável por levantar uma *flag* 0 caso o ultimo dado processado tenha todos os bits iguais a zero gerando um indicador útil para alguns processos dentro da maquina de estados do controlador, determinando diferentes caminhos possíveis para a mudança de estado.

III. Metodologia

Para construção das descrições de hardware pela linguagem VHDL, será utilizado o Vivado, para que seja feita uma checagem prévia da sintaxe utilizada, poupando tempo de desenvolvimento.

Como já dito antes, será utilizado o NcLaunch para compilar e realizar a simulação do código e verificar o funcionamento de cada módulo, antes de integrá-los em um bloco único. Quando confirmado que todos os blocos estão em pleno funcionamento, será feito a integração deles utilizando a instanciação, e feita uma nova rodada de simulação utilizando o NClaunch.

Concluindo as etapas anteriores, será feito a síntese das descrições de hardware utilizando o Genos. Com isso, teremos uma boa estimativa da área ocupada, dos atrasos e da potência consumida pelo hardware.

Por último, a etapa de implementação será feita, onde será desenhado todo o layout do circuito integrado resultante das analises anteriores. Feita a implementação, será feita uma nova rodada de simulações para observar o comportamento do circuito. O intuito de se realizar tantas verificações é para que se possa realizar um comparativo entre as simulações de cada uma das etapas.

IV. Resultados

A. Simulação Pré-síntese

1) Unidade de Controle: A simulação do bloco de controle junto com a memória de instrução foi feito utilizando um testbench simples que realiza apenas o estímulo do *clock*. O programa responsável pelo teste está no arquivo *hexFile.mem*, que contém uma instrução de cada tipo, para observar a máquina de estados. A captura de tela referente a unidade de controle pode ser vista na figura 2.

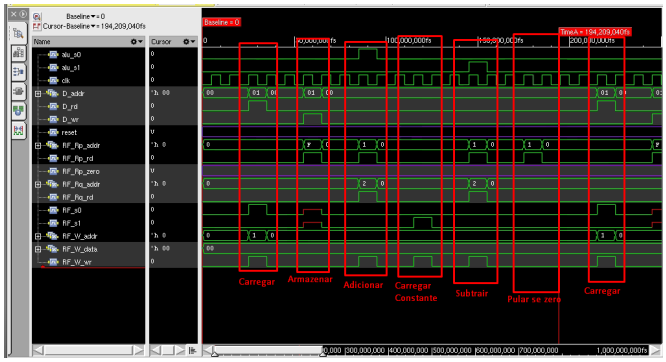

Fig. 2: Teste da unidade de controle.

O código utilizado para realizar cada uma das operações pode ser visto na tabela I.

TABLE I: Tabela do programa utilizado para realizar o teste.

Código de máquina (HEX)	Valor em binário	Operação	Descrição
0x0101	0b0000000 100000001	**Ra[0] = D[1]**	**Carrega o valor de endereço 1 da memória D no registrador Ra[0]**
0x1F01	0b00011111 00000001	D[F] = Ra[1]	Armazena o valor de Ra na memória D
0x2012	0b00100000 00010010	Ra[0] = Rb[1] + Rc[2]	Realiza a soma de Rb[1] e Rb[2] e armazena em Ra[0]
0x3012	0b00110000 00010010	Ra[0] = 0b10010	Carrega a constante 0b10010 no registrador Ra[0]
0x4012	0b01000000 00010010	Ra[0] = Rb[1] - Rc[2]	Subtraí Rb[1] e Rc[2] e armazena em Ra[0]
0x5105	0b01010001 00000101	PC=PC+5 se Ra[1] == 0	Realiza o offset do contador de programa se Ra[1] == 0

Com isso, foi possível verificar que para cada comando, a unidade de controle consegue enviar corretamente os sinais para o bloco operacional, e também consegue realizar o acesso à memória de instrução.

2) Bloco operacional: Para simular o bloco operacional, junto com memoria D, foi criando um testbench, que simula as operações possíveis no do registrado. Primeiro é executado o carregamento de uma constante no registrador 2 do banco de registradores, depois é armazenada essa constante na memoria

D na posição 17 e em seguida carregada, essa mesma posição da memória, em no registrador 2. Podemos ver o resultados da saída desse bloco na figura 3.

Fig. 3: Formas de onda da simulação do bloco operacional e da memoria D

Como descrito anteriormente, o primeira instrução simulada é o carregamento de uma constante no registrador, podemos na figura 4 os valores da saída, onde vale destacar as seguintes entradas RF_W_addr que identifica o endereço do registrador que vai ser armazenado o valor da constante, RF_W_data que o valor que vai ser armazenado no registrador de endereço RF_W_addr, RF_W_Wr que é o sinal que aciona o modo de escrita do banco de registradores e as entrada RF_s1 e RF_s0 que selecionam a saída 3 do multiplexador de 3 entradas.

Fig. 4: Simulação da instrução CONSTANTE (MOV Ra,C)

A segunda instrução simulada é o armazenamento de um registrador na memoria D, podemos ver este passo na figura 5, observe que as entradas D_wr é acionada e a entrada D_addr tem valor 0x11 que o endereço da memoria D.

Fig. 5: Simulação da instrução ARMAZENAR (MOV d, Ra)

A terceira instrução simulada é carregar o valor armazenado na memoria D em um registrador, mostrado na figura 6.

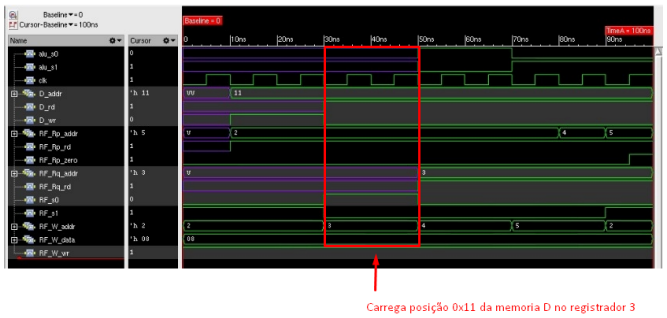

Fig. 6: Simulação da instrução CARREGAR (MOV Ra, d)

Feito os carregamentos vamos a operação de soma e subtração, mostrados na figura 7 e figura 8, respectivamente. Como o registrador 2 tem valor 8 e o registrador 3 também tem valor 8, esses são os registradores que foram carregados nas instruções anteriores, a soma vai ser igual a 16 e a subtração vai ser igual a 0, logo no registrador 4 que é armazenado o resultado da soma tera valor 16 e o registrador 5 que tem o resultado da subtração tera valor 0.

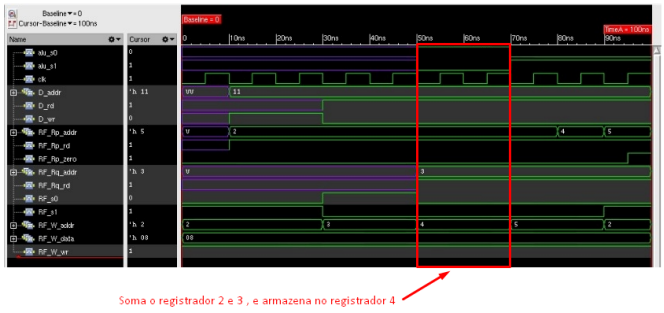

Fig. 7: Simulação da instrução SOMAR (ADD Ra, Rb, Rc)

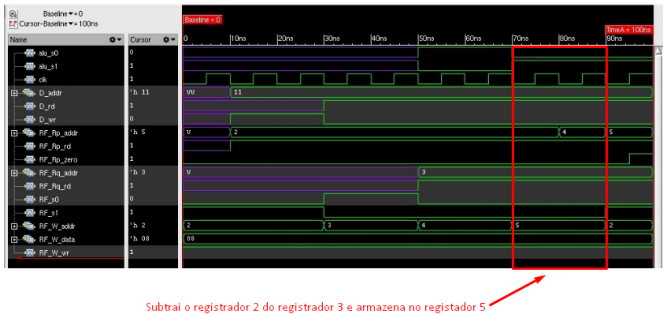

Fig. 8: Simulação da instrução SUBTRAIR (SUB Ra, Rb, Rc)

As ultimas instruções são a de ”saltar-se”, que pula para próxima instrução se o valor do registrador for igual a 0, podemos ver então que na figura 9 como o registrador 4 tem valor igual 16 o sinal de saída RF_Rp_zero tem sinal lógico

igual 0, entretanto na figura 10 como estamos comparando o valor do registrador 5 que tem valor 0 a saída RF_Rp_zero é igual 1, ou seja, neste caso a próxima instrução seria pulada.

Fig. 9: Simulação da instrução ”SALTAR-SE” para o registrador 4 que tem valor 16 (JUMPZ Ra,Offset)

Fig. 10: Simulação da instrução ”SALTAR-SE” para o registrador 5 que tem valor 0 (JUMPZ Ra,Offset)

3) Processador de 6 instruções: Para verificar o funcionamento do processador, com todos os blocos do sistema integrados, foi criado um testbench com um programa que executa-se todas a funções para que ele foi projetado.

Assim foi criado o código em assembly mostrado a abaixo e verificado sua funcionamento no processador.

```
1  MOV R2, #8
2  MOV D(17),R2
3  MOV R3, D(17)
4  ADD R4, R2, R3
5  SUB R5, R2, R3
6  JUMPZ R4, #2
7  JUMPZ R5, #2
8  MOV R5, D(0)
9  MOV R4, D(17)
10 MOV D(0), R4
11 SUB R6, R2, R4
12 JUMPZ R5,#(-4)
```

Podemos ver então na Figura 11 a execução da primeira instrução (MOV R2, 8), onde é carregado o valor 8 no registrador 2. Observe que essa instrução, em código de maquina, é igual 0x3208, como pode ser visto na parte superior do destaque em vermelho no sinal data. Observe também que o valor 8 só armazenado no registrador 2 quando o estado atual da maquina de estados vai para busca.

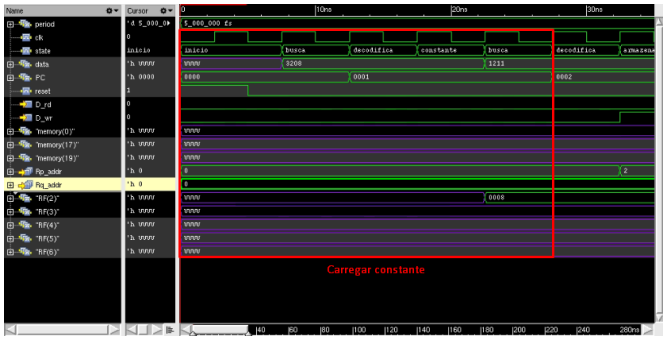

Fig. 11: Carregar constante.

A próxima instrução é a de armazenar o valor do registrador 2 na posição 17 da memoria de dados, essa instrução pode ser conferida na Figura 12, assim como na instrução de carregar constante, o valor da posição 17 da memoria de dados só é atualizada no estado de busca.

Na Figura 12 também temos a instrução de carregar um dado da memoria de dados para o registrador, no nosso caso da posição 17 da memoria para a registrador 3.

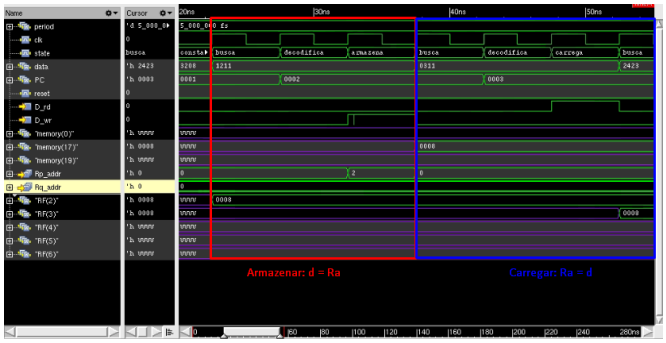

Fig. 12: Instruções de armazenar e carregar na memoria de dados.

Na Figura 13 temos soma e subtração respectivamente, observe que o valor da soma é armazenado no registrador 4 e o da subtração no registrador 5, esses valores serão importantes para a próxima instrução.

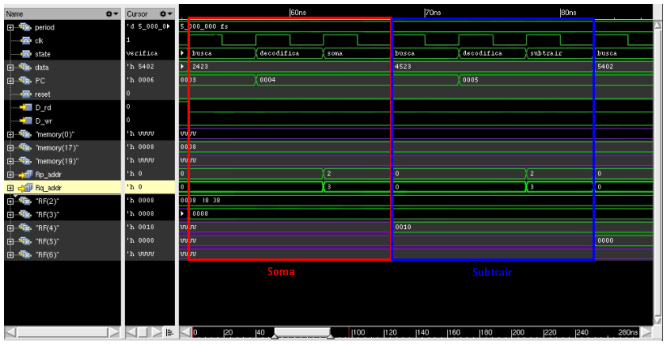

Fig. 13: Instruções de soma e subtração.

Nas Figuras 14 e 15, temos a instrução de ”saltar-se” (JUMPZ R4, 2 e JUMPZ R5, 2), neste caso esperamos que o na primeira instrução de salto que verifica o valor do registrador 4, não salte para o valor de offset definido, já que o registrador 4 tem valor 16. Já na segunda instrução esperamos que a instrução salte para 2 linhas depois, já que o registrador 5 é igual a 0, e como podemos ver na Figura 15, o valor de PC é atualizado para 9 o que indica que a instrução de ”saltar-se” está funcionando corretamente.

Fig. 14: Instrução salta-se para o registrador 4.

Fig. 15: Instrução salta-se para o registrador 5.

Na Figura 16 estamos armazenando na memoria de dados, na posição 19, o valor do registrador 4. Na Figura 17 é demonstrado uma subtração que tenha como resultado valor negativo, neste caso estamos subtraindo o registrador 2 que tem valor igual a 8, do registrador 4 que tem valor 16, e o resultado dessa subtração está sendo armazenada no registrado 6, que é atualizado para o valor -8, como podemos observar.

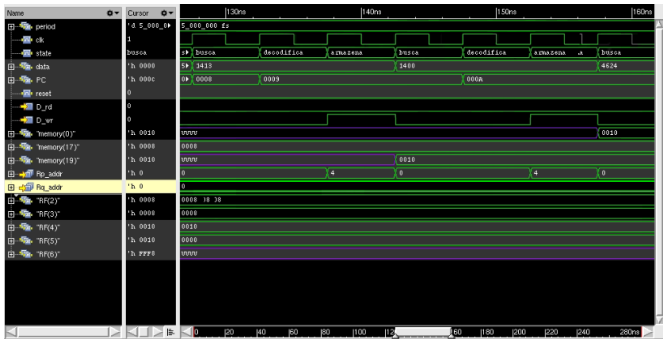

Fig. 16: Instruções de armazenar e carregar.

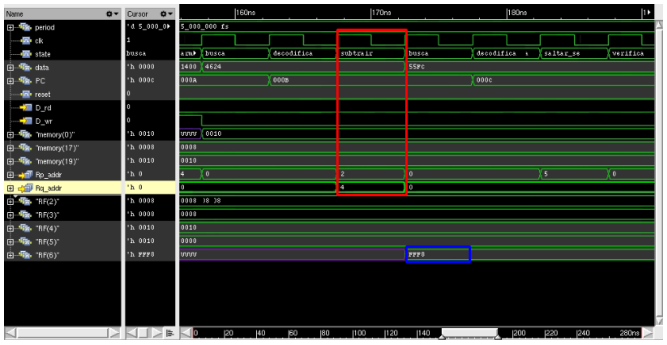

Fig. 17: Subtração com valor negativo.

Na Figura 18, estamos fazendo uma operação de saltar, porém estamos saltando para uma instrução anterior, no caso para instrução da linha 8 do código em assembly apresentado anteriormente, onde estamos carregando o valor da posição 0 da memoria de dados, no registrador 5, como podemos observa na Figura 19, ou seja, nesta instrução o valor do registrador 5 é atualizado para 16, dessa forma quando chegar na operação de saltar, na ultima linha do código, esperamos que não salte e o programa termine.

Na Figura 20 é repetido as instruções anteriores de armazenar, carregar e fazer a subtração com resultado negativo.

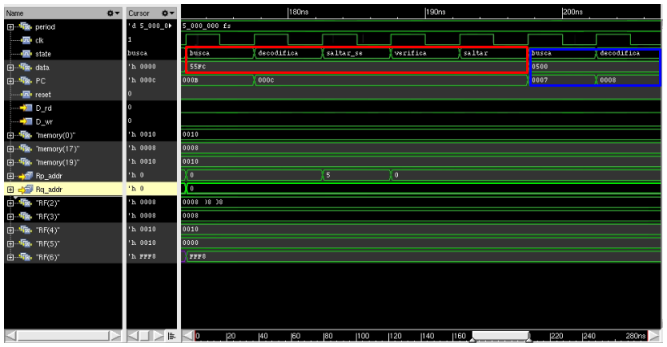

Fig. 18: Operação de salto, com offset negativo.

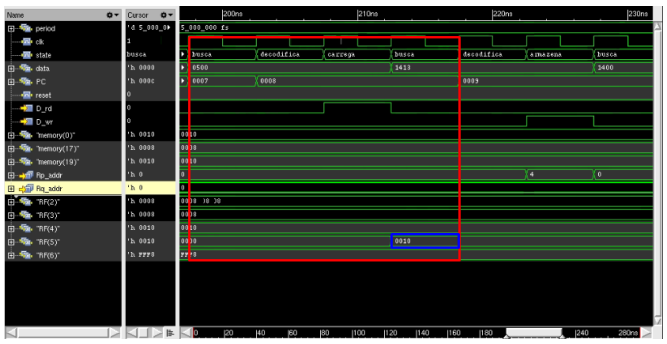

Fig. 19: Atualização do valor do registrador 5.

Fig. 20: Repetição da operações anteriores, armazenar carregar e subtração.

Por fim, na Figura 21 observamos que a instrução JUMPZ não faz o salto para a instrução anterior, como esperado, já que o valor do registrador 5 foi atualizado para 16.

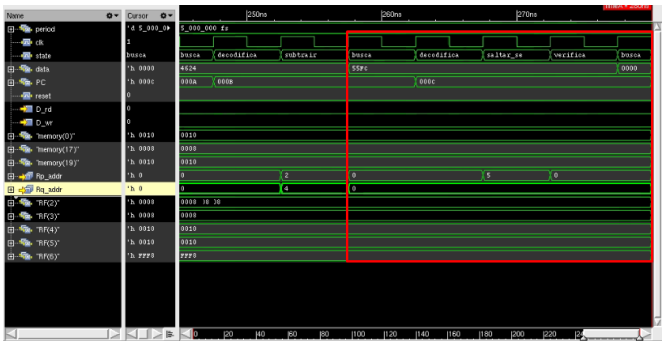

Fig. 21: Ultima instrução de salto.

B. *Síntese*

Comprovado o funcionamento do processador através das simulações vistas na seção IV-A, agora é possível realizar a síntese lógica do processador na tecnologia *tsmc018*. Como resultado da síntese, são gerados, além das *netlists* sintetizadas, relatórios de área, consumo e de atraso.

O relatório de atraso pode ser visto na tabela II. Observe que há um tempo de folga considerável, de $507ps$. Essa folga é importante pois o circuito sintetizado de fato ainda não está posto no layout de circuito impresso, o que conferirá um atraso a mais devido a impedância das trilhas.

TABLE II: Tempo de propagação das portas lógicas.

Clock [clk] (ps)	**Tempo de propagação total (ps)**	**Timing Slack (ps)**
5000	4493	507

O relatório resumido de área pode ser visto na tabela III. Observe que, em comparação ao visto em outros processadores como o MIPS, há um consumo maior de área. Isso se deve ao fato de que o processador implementado possuí um barramento de memória de 16 bits, o que faz com que a quantidade de área seja de, pelo menos, o dobro para um processador de 8 bits, por exemplo.

TABLE III: Parâmetros de consumo de área do circuito.

Tipo	**Instâncias**	**Área(μm)**	**Área %**
Sequencial	292	$15239,078$	$58,6$
Inversoras	60	$412,698$	$1,6$
Buffers	10	$107,565$	$0,4$
Tristates	32	$702,464$	$2,7$
Lógica abstrata	1	$0,000$	$0,0$
Lógica	631	$9542,534$	$36,7$
Células Físicas	0	$0,000$	$0,0$
Total	1026	$26004,339$	$100,0$

O relatório resumido de consumo energético pode ser visto na tabela IV. Em projetos de maior calibre podemos dizer que o report de potencia tem papel fundamental ao evoluir nos processos de fabricação, a descrição em hardware mostra possíveis padrões ajustáveis, além de num ponto avançado poder realizar mudanças de requisitos, tanto no que diz respeito a suporte energético da placa quanto das possibilidades físicas antes da fabricação. Aqui segue-se o mesmo raciocínio para a área consumida: tem-se mais blocos devido ao barramento de dados e, consequentemente, há um maior consumo quando comparado com outros processadores de 8-bits, por exemplo.

TABLE IV: Parâmetros de consumo de potência do circuito

Células	**Leakage Power(nW)**	**Dynamic Power (nW)**	**Total Power (nW)**
1025	$85,380$	$15083258,723$	$15083344,103$

Por fim, vale dizer, que não foram sintetizados os blocos de memória D e o bloco de memória I, pois a própria ferramenta providencia um gerador de memórias quando realizado o layout do circuito. Entretanto, para realizar a simulação, esses blocos foram integrados, mas não de maneira sintetizada.

Houve também uma solicitação do software para a inserção de um atraso de porta de saída. Nesse sentido, foi colocado no arquivo de constraint uma instrução para que seja atribuído um atraso às saídas do bloco processador_6_instruções, a saber: as saídas do PC e do I_rd. Este erro reportado durante a realização da síntese, chamou esforços para alguns pontos do projeto, como, atenção mais detalhada ao caminho de dados, remoção definitiva da memória D no processo de síntese, o ajuste feito, propõe um tempo de propagação na saída responsável por permitir que o dado esteja pronto.

C. *Simulação pós-síntese*

Na simulação pós-síntese, foi feita a mesma simulação da seção IV-A. Nesse sentido, foi necessário apenas observar o comportamento das formas de onda, que pode ser visto na Figura 22.

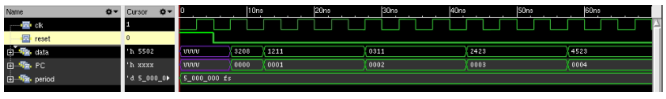

Fig. 22: Resultado da simulação pós-síntese lógica.

V. Conclusão e Discussão

O método de fluxo de projeto digital quando posto a prova, em comparação com outros métodos, se mostra bastante

eficiente, na velocidade de trabalho e facilidade de integração entre os projetistas e suas atribuições. Neste projeto foi utilizada a metodologia semi-custom, com base em arquiteturas exemplo do livro Design Digital por Frank VAHID, contraindo a partir do conceito geral módulos específicos para cada objetivo dentro do processador.

Aplicando as descrições Buttom up e Top down de forma mista conseguimos descrever os componentes que tornam mais simples a integração final, a forma como os dados caminham internamente também facilitam a síntese física realizada automaticamente.

VI. Apêndice

A. Digramas do projeto

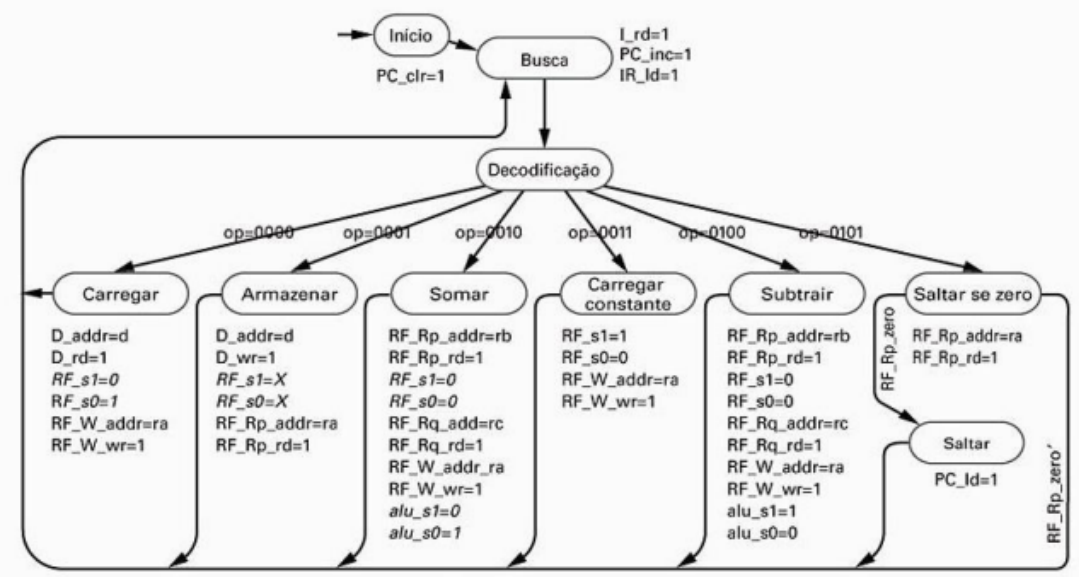

Fig. 23: Diagrama de estados da máquina

B. Testbench do bloco operacional