CI1210 – Projetos Digitais e Microprocessadores
Trabalho Semestral - Prof. Giovanni Venâncio
Implementação MICO x.1

Carlos Alberto Teixeira Mutzenberg , GRR: 20215520

INTRODUÇÃO:
O trabalho consiste na implementação do projeto de hardware do processador MICO X.1 e também a implementação do código Fibonacci na linguagem C, e em seguida em Assembly, para rodar no processador já projetado.

DESENVOLVIMENTO:
O processador é dividido em várias partes, portanto foi realiza-se a modularização do projeto, onde foi organizado e implementado em passos, observado a seguir:

1. **ULA** - Unidade Logica Aritmética (figura 1)

Primeiro módulo foi o desenvolvimento da ULA, onde novamente foi dividida em outros módulos: a)somador, b)subtrator, c)deslocador esquerdo e d)deslocador direito. As operações AND, OR, XOR e NOT foram usadas as já prontas da aplicação e a operação *eh_zero* é um AND com 32 entradas.

a) Para desenvolver o somador, foi implementado um somador de 4 bits *Carry Look Ahead* , ligados em série para transformar em 32 bits;
b) Para desenvolver o subtrator, foi utilizado a implementação do somador de 32 bits, utilizando a lógica de complemento de 2;
c, d) Para desenvolvimento dos deslocadores, foi utilizado a lógica proposta em sala de aula com a utilização de “mux’s”.

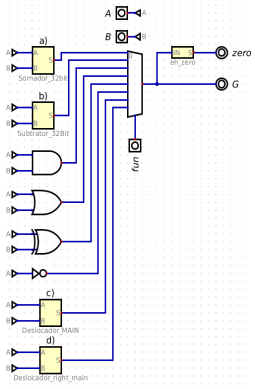

Figura 1 - ULA

2) **Banco de Registradores:** (figura 2)

No Banco de Registradores, foi inicialmente desenvolvido um Flip-Flop(D) que armazena 1 bit que, repicada 32 vezes para se tornar um registrador de 32bit’s, controlados por um MUX. Este, replicamos 15 vezes.
Na imagem podemos observar que o existem 16 saídas, mas 15 registradores, isto acontece, pois o primeiro fio é uma constante 0, que em seguida chama
remos de registrador zero.

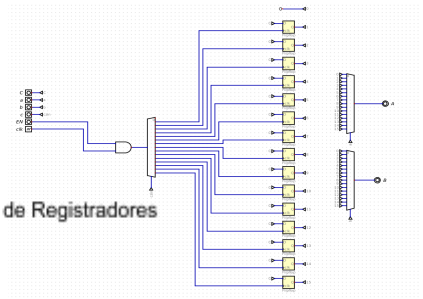
Figura 2 - Banco de Registradores

3)**Montagem Final**

Para finalizar o projeto do hardware(figura 3), utilizamos o enunciado do trabalho e conectamos os componentes restantes devidamente especificados : Memória de controle, Memória de Programa, PC + 4, Banco de Registradores, Unidade Lógica aritmética entre outro micro componentes que estruturam o projeto do processador que observamos a seguir, na figura 3.

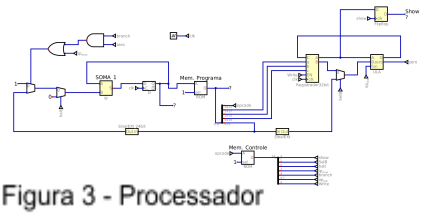
Figura 3 - Processador

Figura 4 - Tabela Me. Controle

Tabela de Memória CONTROLE

	Write	ula	branch	jump	halt	SelB	Show	Hex
NOP	0	000	0	0	0	0	0	0
ADD	1	000	0	0	0	0	0	100
SUB	1	001	0	0	0	0	0	120
AND	1	010	0	0	0	0	0	140
ORi	1	011	0	0	0	0	0	352
Xor	1	100	0	0	0	0	0	180
Not	1	101	0	0	0	0	0	1A0
Sll	1	110	0	0	0	0	0	1C0
Srl	1	111	0	0	0	0	0	1E0
Ori	1	011	0	0	0	1	0	162
Xori	1	100	0	0	0	1	0	182
Addi	1	000	0	0	0	1	0	102
Show	0	000	0	0	0	0	1	1
Jump	0	000	0	1	0	0	0	8
Branch	1	001	1	0	0	0	0	130
Halt	0	000	0	0	1	0	0	4

4)Tabelas finais

Neste passo foi o preenchido a tabela de Memória de Controle(figura 4), onde, seguimos a explicação descrita no

Moodle, que consiste em organizar os sinais de controle para cada operação.

O código Fibonacci é codificado em C(figura 5), iterativamente e em seguida transcrito para linguagem Assembly.

```c
int main()
{
    int N = 10;
    int first = 0;
    int next = 0;
    int second = 1;
    int i = 0;
    while (i < N)
    {
        printf("%d ", first);
        next = first + second;
        first = second;
        second = next;
        i++;
    }
    return 0;
}
```

Figura 5 - Código em C

Para isto, atribuí-se para cara variável um registrador (figura 6), onde posteriormente efetuamos o cálculo. Percebe-se que foi criado registradores a mais do que variáveis, isto ocorre, pois para incrementar a variável “i”(r5), foi preciso de mais dois registradores, dado que precisa de uma constante 1(r7), e também de outro registrador para armazenar a soma de r7 + r5, o registrador r6.

```
# r1 = N
# r2 = first
# r3 = next
# r4 = second
# r5 = i
# r6 = i_aux
# r7 = const 1
```

Figura 6 - Atribuição de Variáveis aos Registradores

Para o Código em Assembly, foi usado o Formato de instrução : OPCODE, Registrador a, Registrador b, Registrador c e Constante.

```
addi r1 r0 10 #Defina N como o número desejado de termos da sequência Fibonacci
addi r2 r0 0  #Inicializa first
addi r3 r0 0  #Inicializa next
addi r4 r0 1  #Inicializa second
addi r6 r0 0  #inicializa i
addi r7 r0 1  #inicializa i_aux
addi r8 r0 1  #inicializa const 1

branch r1 r5   #se r1 for igual a r5 pula
add r3 r2 r4   # next = first + second;
add r2 r0 r4   # first = second;
add r4 r0 r3   #second = next;
add r7 r5 r7   #i = i + 1;
add r5 r7 r5   #i_aux = i + 1;
jump r0 r0 -6  #linhas
```

Figura 7 - Assembly Code

O próximo passo foi Transcrever todo o código Assembly(figura 7) em linguagem binária para preencher a memória de programa, transcrevendo o endereço dos registradores, na tabela a seguir(figura 8):

OPCODE	Resgistrador a	Resgistrador b	Resgistrador c	CONST	Hex
1100	0000	0000	0000	0000000000000000	C0000000
1011	0000	0000	0001	0000000000001001	B0010009
1011	0000	0000	0010	0000000000000000	B0020000
1011	0000	0000	0011	0000000000000000	B0030000
1011	0000	0000	0100	0000000000000001	B0040001
1011	0000	0000	0101	0000000000000000	B0050000
1011	0000	0000	0111	0000000000000001	B0070001
1011	0000	0000	1000	0000000000000001	B0080001
1110	0001	0101	0000	0000000000000000	E1500000
0001	0010	0100	0011	0000000000000000	12430000
0001	0000	0100	0010	0000000000000000	10420000
0001	0000	0011	0100	0000000000000000	10340000
0001	1000	0101	0111	0000000000000000	18570000
0001	0000	0111	0101	0000000000000000	10750000
1100	0010	0000	0000	0000000000000000	C2000000
1101	0000	0000	0000	1111111111111001	D000FFF9
1111	0000	0000	0000	0000000000000000	F0000000

Figura 8 - Tabela Mem. de Programa

Transcrito em binário, converte-se toda linha de OPCODE até a CONST e transforma em Hexadecimal, para que logo em seguida seja carregado na memória e executado o programa. Na Inicialização do N, podemos observar que ele inicia com 9 e não com 10, isto ocorre porque o primeira impressão não foi calculada, mas somente impresso o número 0, portanto, ela precisa fazer somente 9 iterações.

REFERENCIAS:

2. Curso de Projetos Digitais e Microprocessadores. Laboratório HIPES. Youtube, 2023. Disponível em: Curso de Projetos Digitais e Microprocessadores - YouTube