Mico X.1 – Trabalho de Projetos Digitais e Microprocessadores (CI1210)

Caio Eduardo Ferreira de Miranda (GRR20232359) – UFPR

Este relatório descreve algumas tomadas de decisão em relação ao desenvolvimento do Mico X.1, um processador proposto nesta disciplina. Além disso, aqui também está o código pedido no enunciado em *C* e sua respectiva tradução em *assembly*. O simulador usado para a construção do circuito foi o Digital (https://github.com/hneemann/Digital).

ULA: a Unidade Lógica e Aritmética do Mico X.1 tem 8 operações e trabalha com valores de 32 bits, com um fio que indica se o valor da saída é zero (ZERO?), que é usado pela instrução de *branch*.

SLL e SRL: é importante ressaltar que os *shifts* da ULA foram implementados de uma maneira, digamos, *incomum*. O circuito realiza 32 *shifts* diferentes para um valor de B tal que $0 \leq B \leq 32$, e usa um multiplexador para selecionar qual o *shift* correto a se fazer para um valor qualquer de B no intervalo dito.

Memórias de programa e de controle: para ambas as memórias de programa e controle, foram usadas memórias ROM disponibilizadas no Digital, com valores binários convertidos para valores hexadecimais em cada célula. Foram seguidos os vídeos de auxílio do profº Daniel para criar as memórias. O Mico X.1 tem 16 instruções diferentes, todas carregadas na memória de controle, que tem como saída 9 bits para serem os sinais de controle do circuito.

Clock: o *clock* no simulador é de 10hz, para se ter uma execução relativamente rápida do programa.

Valores constantes: para valores constantes (como o 1 usado no complemento de 2 e o 1 que é somado ao IP), foram usados os fornecidos pelo Digital, por conveniência. É mais simples e intuitivo que usar conexões para a terra e para a fonte com barramentos.

Display: o *display* do circuito é um *latch* feito com um multiplexador. Na saída, é mostrado o último valor que passou para o *latch* enquanto ele estava ativado (ou seja, o último valor quando $D\text{-}EN = 1$).

Código para execução no Mico X.1: como exigido no enunciado, foram feitos códigos para exibir os n (no programa, são exibidos 15 números de Fibonacci, portanto, $n = 15$) números da sequência de Fibonacci (um em *C,* outro em *assembly* do Mico X.1). O código foi feito considerando que os casos base da sequência são 0 e 1, então os números exibidos no código são os n números de Fibonacci que ocorrem após os casos base (ex: 1 2 3 5 8 13 21 34 é a sequência para $n = 8$). A instrução de *branch* ocorre antes do *loop* começar, assumindo a forma de um *for loop*.

Código em *C:*

```
#include <stdio.h>
int main() {
int fib1 = 0;
int fib2 = 1;
int fib3;
int n;
printf("Escreva quantos números da seq. de Fibonacci você quer
mostrar (após 0 e 1): ");
scanf("%d", &n);
printf("%d %d ", fib1, fib2);
for (int i = 0; i < n; i++) {
      fib3 = fib1 + fib2;
      printf("%d ", fib3);
      fib1 = fib2;
      fib2 = fib3;
```

```
}
printf("\n");
return 0;
```

Código em *assembly* (Mico X.1):

```
nop # sem op. antes da primeira batida do clock
addi r1, r0, 0 # inicializando r1 com 0
addi r2, r0, 1 # inicializando r2 com 1
addi r3, r0, 15 # inicializando r3 com 15 ("n")
show r3 # mostra "n"
addi r4, r0, 0 # inicializa o contador em 0, e o guarda em r4
branch r3, r4, 7 # verifica se o contador é igual a "n" (condição
para
parar o loop)
add r5, r1, r2 # soma os valores em r1 e r2 e guarda a soma em r5
(soma
dos últimos termos)
show r5 # mostra a soma (número de Fibonacci)
add r1, r2, r0 # guarda o valor de r2 em r1
add r2, r5, r0 # guarda o valor de r5 (soma) em r2
addi r4, r4, 1 # incrementa o contador em r4
jump -6 # volta ao começo do loop
halt # para a execução do programa
```

Com tais instruções, é feito um programa que mostra $n = 15$ números de Fibonacci, mostrando n e cada um dos números da sequência, utilizando um *loop* simples com contador.