EA869 – Introdução a Sistemas de Computação Digital
Relatório do Exercício Computacional II

Prof. Levy Boccato – 1º semestre de 2019

Bryan Wolff **RA: 214095**
João Pedro Bizzi Velho **RA: 218711**

Item 1 -

Inicialmente armazenamos o valor de n no registrador 16 e definimos os dois termos iniciais da sequência de Fibonacci, 0 e 1, gravando um de cada vez no registrador 20 e colocando o valor deste registrador na pilha usando o comando PUSH. Para guardar o valor de n usamos um registrador e o comando LDI, para acompanhar o valor atual da sequência usamos o registrador 18 que terá o valor inicial 2 (pois já inserimos os dois primeiros valores da sequência na pilha).

Após realizada tal configuração inicial partimos para o loop principal, que consiste na chamada de uma subrotina, a qual usará o topo da pilha para guardar o endereço de retorno (portanto devemos nos atentar à utilização da pilha para não sobrescrever o valor de retorno da subrotina), portanto, inicialmente retiramos o valor do topo da pilha e guardamos no registrador 31 usando o comando o POP, agora com o mesmo comando tiramos o valor abaixo da pilha, que é o endereço de retorno e colocamos no registrador 20, por fim temos os dois valores que serão utilizados na sequência, portanto retiramos os dois dados da pilha e os colocamos em registradores 21 e 22 (valores mais alto e mais baixo respectivamente), para que possamos guardar o valor mais alto após a realização da soma, copiamos o valor mais alto da sequência para o registrador 17 e com este realizamos a soma.

Retirados os valores da pilha, devemos realizar a soma de r17 e r22 e em seguida recolocar todos os valores na pilha, atentando-nos à sequência em que foram retirados, portanto, o primeiro à ser colocado é o menor número da sequência, em seguida o segundo menor valor e depois o valor mais alto, o quarto valor a ser colocado é o endereço de retorno da subrotina e por fim o valor que foi retirado do topo da pilha(isto é, r21-r22-r17-r20-r31), após reinserir os valores na pilha incrementamos 1 o valor do registrador de contagem r18 e para finalizar usamos o comando RET para retornar à chamada da função.

Depois da chamada da subrotina e seu retorno temos que realizar a checagem do valor atual da sequência, usando o comando CP comparamos o valor que contém o índice atual da sequência r18 e o valor que contém o valor de n r16, se forem diferentes, ou seja, ainda não atingimos o índice desejado, o comando brne irá fazer o retorno para o início do loop, chamando a subrotina novamente, mas se forem iguais significa que a contagem chegou ao fim e o programa irá terminar executando break..

Item 2 -

Para começar, foi definido que entrará na memória de programa N bytes, e sairá na memória de dados 8N bytes. A técnica de codificação AMI consiste em analisar um bit, e criar um byte associado a este bit. Esse byte será 0 se o bit for 0. Se esse bit for 1, ele será codificado para os bytes com valores +1 ou -1 dependendo do último bit 1 codificado. Se o último bit 1 codificado teve como byte -1, o bit 1 atual terá o valor de seu byte como +1. No caso contrário, se o último bit 1 codificado teve como byte +1, o bit 1 atual terá o valor de seu byte como -1. Logo, os bytes possíveis são 0 (0x00), 1 (0x01) e -1 (0xFF). O valor -1, será o complemento de 2 do valor 1. Como para codificar um

bit 1 atual depende do último bit 1 codificado, para o primeiro bit 1 a ser codificados adotamos a convenção onde iremos codificar o primeiro bit como o byte +1 , como se o anterior tivesse sido -1.

Antes de tudo, definimos na memória de programa a quantidade (N) de bytes a serem codificados (3) e inserimos os seus respectivos bytes (0x86, 0x73, 0xa4). Retiramos da memória, com ajuda do ponteiro Z, o valor de N (número de bytes) colocando-o no r17, e o r18 assumirá o valor de cada byte que foi inserido na memória. Definimos também no r27 o local onde será armazenado o resultado codificado na memória de dados, no caso 0x0300, que é um registrador do ponteiro X.

Vamos guardar sempre o valor do byte associado ao bit no r21, e no final de cada análise r21 sempre será inserido na memória de dados com ajuda do ponteiro X, no local de memória que já foi estabelecido. Utilizando os comandos ROL e BST e BLT, vamos sempre isolar um bit do byte de r18 e jogar para o r19. Ou seja, r19 sempre vai percorrer bit por bit para ser analisado. Sabemos que para codificar temos que saber o que foi codificado antes. No caso mais simples, se o bit analisado for 0, o byte associado será 0, logo r21 será 0. Mas se o bit analisado for 1, utilizamos outro registrador como referência (r20) para definir se a codificação deverá ter como resultado +1 ou -1. Portanto, se r20 possuir o valor 0, quer dizer que a codificação anterior foi +1, e se possuir 1, quer dizer que a codificação anterior foi -1.

Como adotamos uma convenção no início da análise do primeiro byte, definimos r20 igual a 0. Logo, se r19 for 0, r21 assumirá 0 e será armazenado na memória. Se r19 for 1 e r20 for 0, r21 assumira -1 e r20 será atualizado para 1. Se r19 for 1 e r20 for 1, r21 assumirá +1 e r20 será atualizado para 0 para a próxima análise. Assim que terminar a análise dos 8 bits associado ao primeiro byte, r18 assumirá o próximo byte, e a analise se repete. O r22 será usado para contagem para sabermos qual bit do byte atual está sendo analisado, e sempre no final da análise de cada byte o r22 será restaurado ao seu valor inicial. Como o r17 carrega o valor de N, ele será usado como contagem e será decrementado a cada análise de byte, e quando for analisado todos os bytes o programa será encerrado. Após finalizar o programa, na posição 0x0300 da memória de dados estará os bytes da codificação dos valores exemplares colocado no programa (0x86, 0x73, 0xa4). Conforme a técnica de codificação utilizada, o bit por bit de cada byte, 0x86, 0x73 e 0xa4 será codificado para os seus respectivos bytes 01 00 00 00 00 ff 01 00 00 ff 01 ff 00 00 01 ff 01 00 ff 00 00 01 00 00, e estes valores estarão na memória de dados ao encerrar o programa.