Exercício Programa: Compilador Assembly Customizado
Organização de Computadores Digitais

Arthur Prince de Almeida	nº: 10782990
João Víctor Ferreira Araujo	nº: 10856758
José Luiz Oliveira Assumpção	nº: 10687576
Mauricio Mori Dantas Santana	nº: 7991170
Thor Cayres Fernandez	nº: 9844922

São Paulo – Brasil 02 de Julho de 2019

Sumário:

1 - Compilação.....3
2 - Execução.....3
3 - Funções.....3
4 - Codificação.....4
    4.1 - Classes.....4
5 - Opcodes.....5
6 - Arquitetura.....5
7 - Micro-instruções.....6
8 - Sinais de Controle.....6
9 - Problemas.....6

1 - Compilação:

Como ressalva para a compilação do código, é válido lembrar que o mesmo já virá compilado, ou seja, já terá os arquivos "*.class*" inclusos no arquivo de envio, contudo, se for necessária a verificação, basta compilar apenas a classe "*Teste.java*", na própria linha de comando.

2 - Execução:

A execução é feita na própria linha de comando, dando início à classe "*Teste.java*", após a escrita do código na "*entrada.txt*".

3 - Funções:

As funções disponíveis no conjunto do trabalho são:

Funções MOV:

mov r1,r2
mov r1,c
mov [r1],r2
mov [r1],c
mov [c],c
mov [c],r1
mov r1,[c]
mov r1,[r2]

Funções da ULA:

add r1,r2
add r1,c
sub r1,r2
sub r1,c
mul r1,r2
mul r1,c

div r1,r2
div r1,c
mod r1,r2
mod r1,c
cmp r1,r2
cmp r1,c

Funções JUMPS:

jmp c
je c
jne c
jg c
jge c
jl c
jle c

4 - Compilação:

O código em si vem acompanhado de comentários auto-explicativos, ainda sim, as especificações de cada função se encontram neste link:

https://docs.google.com/document/d/1d-qUNRQ0-CUtub2CU2I_DSt6QoUXkQD1Xli_AyTM3gY/edit?usp=sharing

4.1 - Classes:

Classe "Teste" - Contém o método "*main*". Basicamente aciona as funções das demais classes.
Classe "Ula" - Contém todas as funções aritméticas e as "*flags*".
Classe "Memoria" - Contém o compilador do código do arquivo "*entrada.txt*" e os métodos "*read*" e "*write*", além de ter uma memória que armazena o código compilado.
Classe "MemoriaDaUC" - Contém os micro-programas a serem lidos pela classe "*UnidadeDeControle*".
Classe "UnidadeDeControle" - Contém o método de execução das micro--instrucões, o método de "*jump*" (que serve para mudar o "*car*"), e o método que decodifica o "*IR*".
Classe "Registradores" – Apenas possui os registradores ("*ax", "bx", "cx", "dx*") e os métodos "*get*" e "*set*" de cada um deles.

5 - Opcodes:

Os “*opcodes*” foram escolhidos pensando que, há menos de 16 funções e menos de 16 tipos de parâmetros.

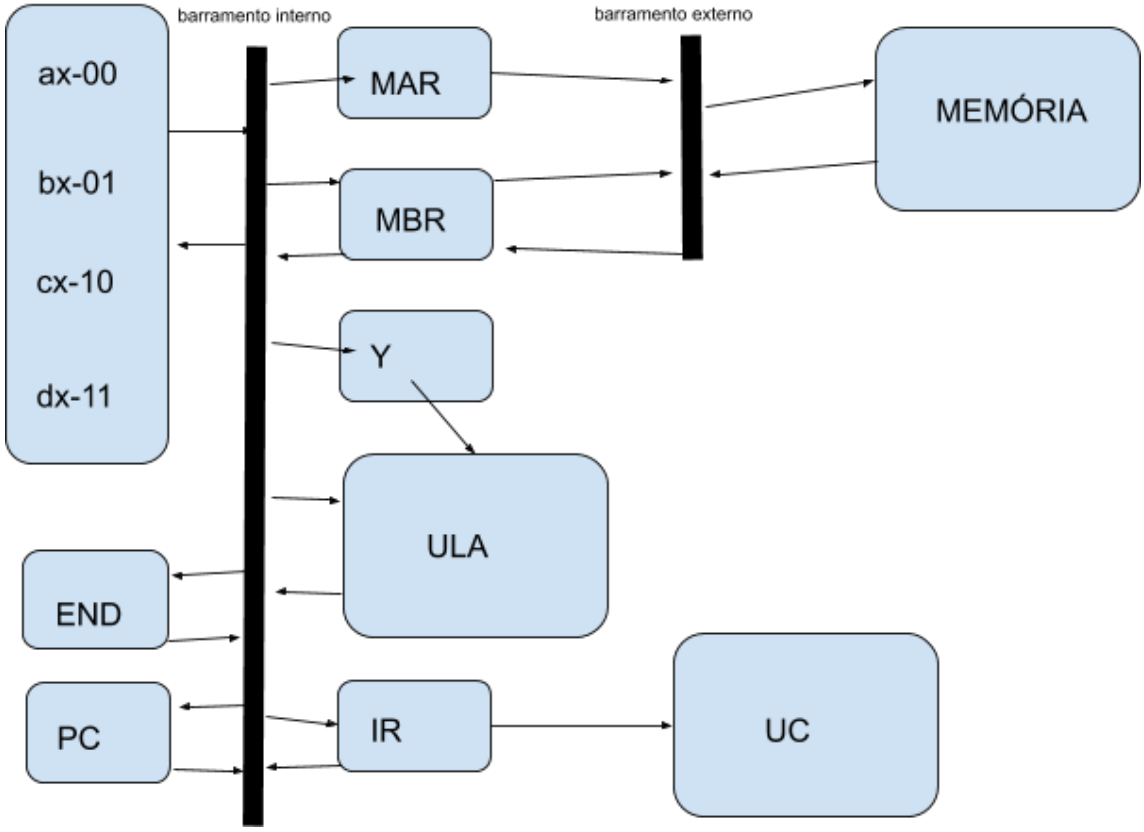
Como funciona:

Na terceira casa hexadecimal do “*opcode*” se encontra qual função foi selecionada *(“mov”, “add”, “jump”...),* na segunda casa, encontra-se o primeiro dos parâmetros *(“ax”, “AAA”, “[BBB]”)* e na primeira casa, o segundo.

Por essa razão, muitos tipos de códigos absurdos podem vir a compilar (por exemplo “*jump ax,bx*”), no entanto, nessas ocasiões será mostrado um erro na decodificação do “*IR*”.

Além disso, sempre que houver uma constante no código, ela se encontrará na linha seguinte do mesmo (inclusive os “*jumps*”). Isso deve ser considerado caso queira-se realizar um “*jump*” no programa.

6 – Arquitetura:



7 – Micro-Instruções:

Projetadas de forma horizontal.

As micro-instruções são formadas a partir da união dos micro-programas da “*MemoriaDaUC*” e o “*IR*” decodificado.

O “*IR*” decodificado contém o endereço do micro-programa a ser executado e os registradores, que serão usados no micro-programa.

Já a “*MemoriaDaUC*” contém os sinais de controle da CPU, de funções (especificado no link que explica o programa) e sinais de controle que se comunicam com o “*IR*” decodificado. Ou seja, é como se o “*IR*” decodificado contivesse o que varia e, a memória da UC, o que é constante, além de possuir um método de solicitação ao que está dentro do “*IR*” decodificado.

8 – Sinais de Controle:

Os sinais de controle que estão envolvidos com o barramento externo não realizam nada, pois não há um registrador na memória, além do registrador “*END*” ter sido criado com o objetivo de solucionar a função “*mov [c],c*”. Não apenas por isso como também pelo fato do sinal “*ULA++*” só servir para incrementar o “*PC*”.

Os demais sinais de controle estão especificados no artigo anexado a este (Link anteriormente exposto).

9 - Problemas:

Quando se faz um “*MOV*” para um endereço muito grande, adiciona-se “0” nos espaços entre o tamanho da memória anterior (todo o espaço que vez antes do endereço) e o endereço selecionado, o que acaba prejudicando na busca. A solução aqui seria usar um “*ArrayList*” ao invés de uma “*LinkedList*” (a “*LinkedList*” foi utilizada devido a sua orientação).

O segundo e último problema não é exatamente um defeito do programa, contudo uma particularidade que vale a pena ser aqui frisada, no caso, consiste nos números negativos aparecerem como “complemento de 2”. A solução aqui mais conveniente seria fazer um método que convertesse isso.