**Nome:** Giovanne Prestes Dias **RA** 171029

**Título:** Projeto M1 – Calculadora.

1. **Introdução**;

Criada na década de 50, o Assembly foi das primeiras linguagens de programação a aparecer. Ela usa uma sintaxe complicada e "exageradamente" difícil, isto porque, antes da década de 50 os programadores de máquinas tinham que escrever instruções em código binário, qualquer coisa como: 0110010110011011010110011010111010110101... Para escrever uma instrução. Na verdade, o Assembly foi criado para facilitar o uso dessa tarefa, mas é considerado uma linguagem de baixo nível, pois tudo o que o processador interpreta tem que ser descrito pelo programador. Assim o código acima seria "add EAX" em Assembly. Bastava apenas, depois de estar concluída a escrita do código, rodar o compilador e tínhamos o programa.

Cada arquitetura de computador tem seu próprio código de máquina, e cada montador gera códigos para uma arquitetura específica. Cada um desses montadores tem sua própria versão de código Assembly, que pode diferir ao uso de registradores, representação de números, ou até mesmo instruções mnemónicas. E isso pode dificultar um pouco na portabilidade do código, tendo em vista que o mesmo precisaria ser reescrito para poder ser montado para outra arquitetura.

Vantagens: programas extremamente rápidos e pequenos

Desvantagens: tempo de desenvolvimento lento e sujeito a erros; código preso a uma arquitetura.

Primeiramente, tivemos que codificar a lógica através do software Cpu Board Studio, o código em assembly e utilizando:

* Registrador de endereço (MAR) de 12 bits.
* Registrador de dados de (MBR) de 8 bits.
* Contador de programa (PC) de 12 bits.
* Registrador de instruções (IR) de 16 bits.
* Registrador acumulador (AC) de 8 bits.
* Registrador com flags de controle (SF, OF, ZF, GF e LF).

E com o seguinte conjunto de instruções:

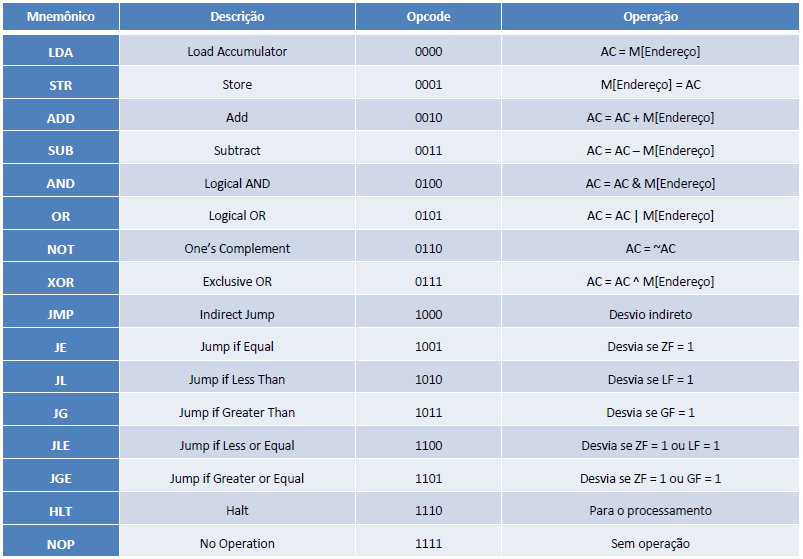


Figura 1: Conjunto de instruções.

E o seguinte formato de instrução:

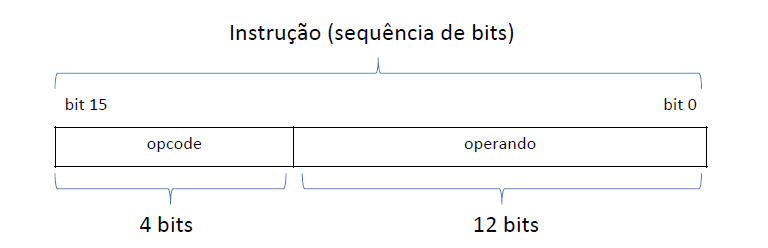


Figura 2: Formato de instrução.

1. Análise dos resultados (acrescentar comentários sobre o funcionamento do projeto);

Com as instruções informadas acima e o desafio proposto, chegamos na seguinte lógica:

ram 4000 0 *//zero*

ram 4001 1 *//soma*

ram 4002 2 *//subtração*

ram 4003 3 *//multiplicação*

ram 4010 4 *//divisão*

ram 4050 0 *//aux Div*

*//Inicio*

cseg

org Inicio

lda ac, 4094 *//pegar operando*

str 4004, ac *//M[4004] -> operando (+,-,\*,/)*

str 4093, ac

lda ac, 4094 *//pegar operador 1*

str 4005, ac *//M[4005] -> operador 1*

str 4092, ac

lda ac, 4094 *//pegar operador 2*

str 4006, ac *//M[4006] -> operador 2*

str 4093, ac

lda ac, 4004

sub ac, 4001

je Soma

lda ac, 4004

sub ac, 4002

je Sub

lda ac, 4004

sub ac, 4003

je Mult

lda ac, 4004

sub ac, 4010

je DivTesteZero

jmp Inicio

*//Soma*

org Soma

lda ac, 4005

add ac, 4006

jl Overflow

jmp Fim

*//Subtração*

org Sub

lda ac, 4005

sub ac, 4006

jmp Fim

*//Multiplicação*

org Mult

lda ac, 4005 *// lendo x*

str 4100, ac *// aux = x*

*//Inicio verificação num = 0 (Multiplicação)*

lda ac, 4005

sub ac, 4000

je Fim

lda ac, 4006

sub ac, 4000

je Fim

*//Inicio verificação operador 2 é menor que zero*

org InicioMult

lda ac, 4006 *//AC = y*

sub ac, 4001 *//AC = y-1*

str 4006, ac *//y = AC*

je FimMult *//Jump se y=1*

lda ac, 4005 *//AC = x*

add ac, 4100 *//AC = x + aux*

str 4005, ac *//x = AC*

jmp InicioMult

org FimMult

lda ac, 4005 *//AC = x*

jmp Fim

*//Divisao*

org DivTesteZero

lda ac, 4006

sub ac, 4000

je DivZero

jmp Div

org Div

lda ac, 4005 *//AC = x*

sub ac, 4006 *//AC = x-y*

jge AuxDiv *//if ac>=y vai p fim*

lda ac, 4050

jmp Fim

org AuxDiv

str 4005, ac

lda ac, 4050

add ac, 4001

str 4050, ac

jmp Div

*//Fim*

*//Display de cima = resultavo*

*//Display de baixo = 0*

org Fim

str 4092, ac

lda ac, 4000

str 4093, ac

hlt

*//Erros*

*//Overflow*

*//Display de baixo = 1*

org Overflow

lda ac, 4000

str 4092, ac

lda ac, 4001

str 4093, ac

hlt

*//Divisão por 0*

*//Display de baixo = 2*

org DivZero

str 4092, ac

lda ac, 4002

str 4093, ac

hlt

1. Análise dos resultados;

Analisando o código programado acima, é possível executar as operações matemáticas: soma, subtração, multiplicação e divisão, utilizando apenas as instruções apresentadas na Figura 1. Por este motivo, utilizamos apenas a soma (add) e subtração (sub), independente da operação matemática executada.

É possível verificar a validação de overflow na operação de soma, onde o segundo display mostra 1, e também, verificar a divisão por zero, onde o segundo display mostra 1. Caso contrário mostra o resultado da operação matemática executada no primeiro display, e no segundo display mostra o valor 0.

1. Conclusão.

Ao final deste experimento, obtivemos um conhecimento básico sobre assembly, em arquitetura de computadores, me familiarizei com os módulos CPU Board Studio e CPU Board Lab, conseguindo assim obter o conhecimento básico de manipulação de alguns periféricos conforme o proposto neste experimento.

Por fim, concluímos que o objetivo de fazer uma calculadora em assembly utilizando apenas as instruções que o CPU Board disponibiliza, mostrado na Figura 1. Mesmo faltando algumas validações, como overflow em subtração, multiplicação e divisão e o carry em todas as operações, a calculadora programada realiza as operações conforme o solicitado.