UNIVERSIDADE FEDERAL DE SANTA MARIA
ORGANIZAÇÃO DE COMPUTADORES
SISTEMAS DE INFORMAÇÃO

ISABELLA SAKIS
RHAUANI WEBER AITA FAZUL

- ARITMÉTICA COMPUTACIONAL -
- O PROCESSADOR: O CAMINHO DE DADOS E O CAMINHO DE CONTROLE -

Santa Maria, RS, Brasil
Julho de 2016

SUMÁRIO

1. INTRODUÇÃO

O segundo trabalho proposto da cadeira de Organização de Computadores, conta com quatro questões acerca dos seguintes temas: Aritmética Computacional, Caminho de Dados e Caminho de Controle do Processador, sendo que para a realização das duas primeiras questões, foi necessária a implementação de programas em linguagem de montagem para o processador MIPS.

A questão um consiste na elaboração de um programa em *Assembly* que realiza a multiplicação de dois números utilizando o segundo algoritmo de multiplicação apresentado no livro texto da disciplina, esse algoritmo faz uso de um mesmo campo para o multiplicador e para o produto, o qual possui, também, seu respectivo bit de *carry*. Já a questão dois consiste em um programa que encontra o seno de um ângulo, sendo esse um número em ponto flutuante e de precisão dupla, com isso, os registradores que devem ser utilizados são os do coprocessador um do software *MARS*, assim como as instruções serão em ponto flutuante.

Na questão três é solicitada a tradução das instruções *load word (lw), subtraction (sub), branch if equal (beq)* e *jump unconditionally (j)* para instruções em linguagem de máquina, sendo essas também representadas em hexadecimal, para tradução são utilizados os formatos de instrução implementados pelo hardware MIPS real. Por fim, a última questão faz uso do Diagrama de blocos do processador MIPS para representar, detalhadamente, o processamento das instruções *addition (add), load word (lw), store word (sw)* e *branch if equal (beq)*.

O trabalho tem foco nos capítulos 3 e 4 do livro *Organização e projeto de computadores*, além de abordar questões anteriormente estudadas no mesmo. O presente relatório irá relatar a resolução desse trabalho, dando o suporte necessário aos leitores através da revisão bibliográfica, a qual abordará os temas envolvidos nas questões. Na metodologia serão descritos com detalhes os processos envolvidos nas resoluções das quatro questões e no experimento serão mostradas tais resoluções. Por fim, através da análise dos resultados e dos levantamentos necessários para discussão, serão feitas avaliações sobre as dificuldades encontradas na realização do trabalho, assim como os benefícios obtidos, os quais serão salientados nas conclusões.

2. OBJETIVOS

2.1 OBJETIVO GERAL

Pretende-se, com a elaboração deste trabalho, aprimorar os conhecimentos dos alunos envolvidos, auxiliando na absorção da matéria passada em sala de aula e, possibilitando, de fato, entender como o computador realiza operações dado certas instruções, além de aprimorar a lógica de programação e conhecimentos acerca da linguagem de montagem *Assembly* que foi utilizada para escrita dos programas necessários no desenvolvimento de algumas das questões.

2.2 OBJETIVOS ESPECÍFICOS

- Elaborar um programa em *Assembly* que realiza a multiplicação de dois números utilizando o segundo algoritmo de multiplicação apresentado no livro texto da disciplina;
- Elaborar um programa que encontra o seno de um ângulo, sendo esse ângulo um número em ponto flutuante e de precisão dupla;
- Analisar, qual a melhor alternativa de implementação das questões que forem julgadas necessárias;
- Traduzir um código em linguagem de montagem do processador MIPS para a linguagem de máquina, representando-a em binário e em hexadecimal;
- Explicar detalhadamente como são processadas algumas instruções (*addition (add), load word (lw), store word (sw)* e *branch if equal (beq))*, utilizado o Diagrama de blocos do processador MIPS.

3. REVISÃO BIBLIOGRÁFICA

Um computador é uma máquina capaz de processar dados que são necessários para nós, usuários, pelos mais diversos motivos. Esse processamento de dados é realizado através da coleta e manipulação dos mesmos, para depois fornecer as informações, que são os resultados dessa manipulação, para o usuário. O computador processa os dados através da execução de instruções em linguagem de máquina, as quais o processador tem a capacidade de executar.

Os primeiros programadores se comunicavam com os computadores em números binários, tarefa maçante que logo foi substituída por novas notações simbólicas mais parecidas com a maneira como os humanos pensam, as quais passaram a ser traduzidas para binário através de programas auxiliares. O primeiro desses programas foi chamado de montador (*assembler*), utilizando a linguagem chamada de *Assembly*.

O programador hoje em dia, elabora primeiramente o algoritmo, composto por uma sequência de passos ou ações que determinam a solução de algum problema computacional, em seguida realiza a codificação do mesmo em uma linguagem de alto nível. Esse código é traduzido para um código correspondente em linguagem *Assembly*, através da interpretação ou da compilação do programa fonte. Nessa etapa, cada instrução da linguagem de alto nível é interpretada para executar a instrução correspondente, através do *hardware* do computador e, então, finalmente o programa será executado.

As palavras de um computador são compostas por bits e podem representar números armazenados na memória. Estes números podem ter diferentes significados, como inteiros (ponto fixo) ou reais (ponto flutuante), serem positivos ou negativos. A manipulação dos números inclui operações de soma, subtração, multiplicação e divisão. A aritmética computacional engloba as operações com tais bits, sendo que tais operações podem possuir uma velocidade de processamento mais alta ou mais baixa.

Dentre as operações aritméticas computacionais, destaca-se, nesse relatório, a de multiplicação, baseando-se na multiplicação de números decimais que aprendemos na escola. É fácil deduzir um algoritmo funcional para realizar a multiplicação entre dois registradores de 32 bits, um exemplo do primeiro algoritmo da multiplicação apresentado no livro texto da disciplina pode ser visto na Figura 1. Porém, o algoritmo usado na resolução de uma das questões desse trabalho é o segundo algoritmo da multiplicação, uma versão mais refinada do primeiro, o qual pode apresentar melhor desempenho e pode ser visto na Figura 2.

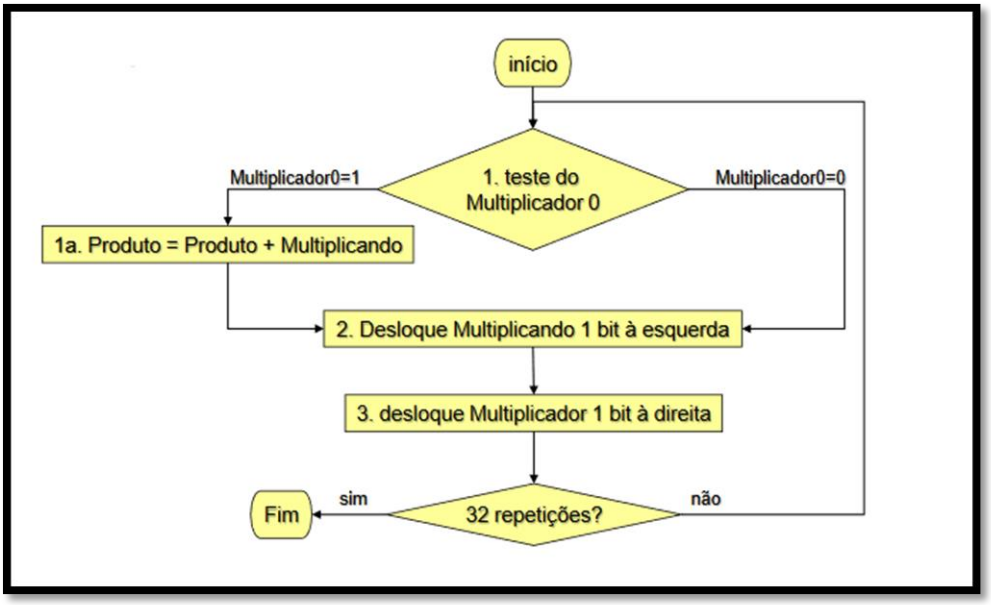

Figura 1 – Primeiro algoritmo da multiplicação.
Fonte*: http://www.inf.pucrs.br/~emoreno/undergraduate/EC/arqi/class_files/Aula08.pdf.*

A figura acima ilustra uma possibilidade de implementação de um algoritmo que realiza a multiplicação de dois números de 32 bits.

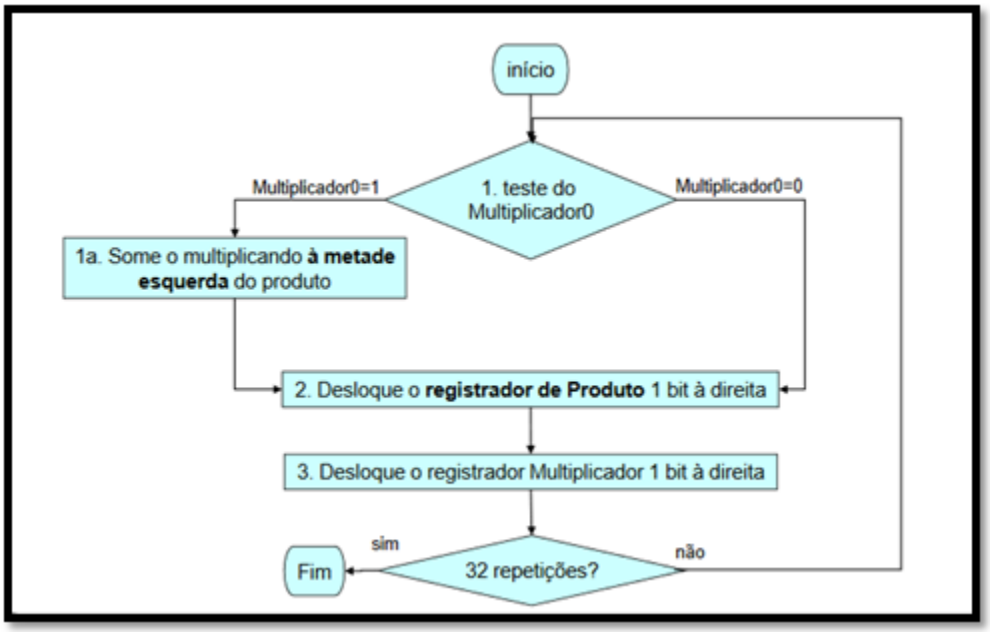

Figura 2 – Segundo Algoritmo da multiplicação.
Fonte*: http://www.inf.pucrs.br/~emoreno/undergraduate/EC/arqi/class_files/Aula08.pdf.*

A figura acima ilustra os passos que o programa em linguagem de montagem da questão um desse trabalho realiza, tal algoritmo pode ser considerado mais eficiente que o anterior.

O desempenho de um computador é principalmente determinado por três fatores: contagem de instruções, tempo de ciclo de *clock* e *CPI* (ciclos de *clock* por instrução). A implementação do processador do computador é o que determina o tempo de *clock* e o número de ciclos de *clock* por instrução. A implementação do processador é dada pelo Caminho e Dados e pelo Caminho de Controle do processador.

Para executar qualquer instrução, precisamos começar buscando a instrução na memória, para prepará-la para a execução da próxima instrução. É necessário também incrementar o contador de programa de modo que aponte para a próxima instrução, quatro bits depois. Desse modo, a implementação da maioria das instruções do processador *MIPS* é igual, principalmente as duas primeiras etapas, as quais são idênticas para todas instruções de cada classe de instrução (referência à memória, lógica e aritmética e desvios). Essas duas primeiras instruções são as seguintes:

- Enviar o contador de programa (PC) à memória que contém o código e buscar a instrução dessa memória.
- Ler um ou mais registradores, usando campos da instrução para selecionar os registradores a serem lidos.

Os registradores de uso geral de 32 bits do processador são armazenados em um banco de registradores, que é uma coleção de registradores em que qualquer registrador pode ser lido ou escrito especificando o número do registrador no banco.

Após essas duas etapas, as ações necessárias para completar a instrução dependem da classe da instrução. A simplicidade e a regularidade do conjunto de instruções simplificam a implementação tornando semelhantes às execuções de muitas das classes de instrução. O caminho de dados mais simples pode tentar executar todas as instruções em um único ciclo de *clock.*

4. METODOLOGIA

As questões um e dois foram implementadas em linguagem de montagem *Assembly*. O conjunto de instruções escolhido para a implementação vem da *MIPS Technology*, que é um exemplo elegante dos conjuntos de instruções criados desde a década de 1980, baseado em registradores, ou seja, a *CPU* usa apenas registradores para realizar operações aritméticas e lógicas. Esse conjunto de instruções é considerado bastante didático, pela sua elegância e simplicidade, e por isso é utilizado com frequência no ensino em faculdades.

O *software* usado para essa implementação foi o *M.A.R.S.* (*MIPS Assembler and Runtime Simulator)*, um ambiente de desenvolvimento interativo leve (*IDE*) para a programação em linguagem *Assembly* do *MIPS*.

As questões três e quatro foram realizadas utilizando os conteúdos vistos em sala de aula, além de se basear em conceitos e explicações do livro texto da disciplina. Outra ferramenta muito importante para o desenvolvimento do trabalho foi a *MIPS X-Ray* presente no software *MARS*, a qual permite uma visualização de como as instruções passam por dentro do processador, sendo uma maneira ilustrativa para melhor compreender o Caminho de dados e de Controle do processador.

Questão 1.

O programa em linguagem de montagem simula a multiplicação de dois números de acordo com o segundo algoritmo de multiplicação apresentado durante as aulas, que faz uso de um multiplicador (32 bits) e um multiplicando (32 bits) para apresentar como resultado o produto dos dois números (2 x 32 = 64 bits).

O método ‘Lápis e Papel’ desse algoritmo consiste em uma tabela com quatro colunas, sendo estas destinadas a:

(a) Iteração -> Inicio em 0 (irá até 32);
(b) Passo -> Descrição do que irá ser realizado;
(c) Multiplicando -> Escrito com 32 bits;
(d) Produto -> Sendo iniciado com o bit de *carry* (vai-um) igual a 0 mais 32 bits iguais a zero e os 32 bits do multiplicador.

Após a iteração inicial, em cada uma das mesmas será verificado o bit menos

significativo do produto e se este for igual a 1, será realizada a soma da metade esquerda do produto - parte *HI* - (considerando o bit de *carry*) com o multiplicando. Em seguida é feito um deslocamento lógico para a direita- do produto (bit vai-um incluído).

Após serem realizadas todas as iterações, o resultado da multiplicação dos dois números estará contido na coluna referente ao produto.

Questão 2.

Computadores antigos usavam tabelas armazenadas na memória, isto é, para um determinado ângulo existia um valor pré-determinado para uma função trigonométrica associada ao mesmo. Nos dias de hoje os computadores usam outra técnica. Matematicamente pode ser mostrado que o seno de um ângulo, por exemplo, é a soma de uma série infinita (chamada de série de Taylor).

O código em linguagem de montagem escrito nessa questão utiliza da série de Taylor (fórmula mostrada na sessão experimento do item dessa mesma questão) para encontrar o seno de um ângulo digitado pelo usuário.

Obviamente nós não podemos computar a soma de uma série infinita, mas podemos fazer que a soma termine quando uma determinada condição for ou não satisfeita, como por exemplo um número máximo de iterações ou se os termos da série apresentarem valores menores que algum valor pré-definido (chamado de erro). Nesse trabalho, apresentaremos o código fazendo uso das duas condições de paradas citadas – erro e iteração máxima.

Questão 3.

Instruções são mantidas no computador como uma série de sinais eletrônicos altos e baixos e podem ser representadas como números. Cada parte da instrução pode ser considerada como um número individual e a colocação desses números lado a lado forma a instrução.

Nessa questão utilizaremos o formato de instrução (uma forma de representação de instruções com campos de números binários) do *MIPS* para traduzir instruções em linguagem de montagem para linguagem de máquina (representando tanto em hexadecimal quanto em binário) utilizando o formato de instrução, anteriormente citado, com seus respectivos tipos e campos

para representar as instruções necessárias.

Questão 4.

Utilizando como referência a teoria encontrada no livro texto da disciplina e os exemplos mostrados em aula, usaremos o diagrama de blocos do processador *MIPS* para explicar e analisar detalhadamente o fluxo pelo caminho de dados das instruções solicitadas.

Para tal, também precisaremos conhecer a tradução das instruções em linguagem de máquina (vide questão três), além de utilizarmos as saídas de algumas das tabelas verdades apresentadas no livro texto, as quais auxiliam na definição das linhas de controle determinadas pelo *OpCode* da instrução e na forma como os bits de controle da unidade aritmética serão definidos dependentes dos bits de controle da *OpALU*, que serão necessárias para á analise do caminho das instruções pelo diagrama de blocos.

Uma ferramenta muito útil, como citado anteriormente, foi a *MIPS X-Ray* que auxiliou na visualização de como as instruções passam por dentro do processador.

5. EXPERIMENTO

Nessa seção serão apresentadas as resoluções das questões do trabalho.

As telas das questões um e dois são as principais telas exibidas para o usuário em tempo de execução, para um bom funcionamento do programa das mesmas é necessário abrir o arquivo *main.asm* referente a cada questão no *software MARS*, marcar a opção '*Assembly all files in directory*' na aba '*Settings*' e em seguida clicar em Assemble (ou clicar na tecla '*F3*'), as telas seguir apresentadas, serão mostradas na aba '*Run I/O*'.

Questão 1. Escreva um programa, em linguagem de montagem para o processador MIPS, que realiza a multiplicação de dois números de 32 bits. A multiplicação deve ser realizada com o segundo algoritmo da multiplicação (figura 3.6) do livro do Patterson, com as somas e deslocamentos. Não utilize instruções em ponto flutuante e instruções de multiplicação.

O programa que realiza a multiplicação de dois números com o segundo algoritmo da multiplicação se encontra na pasta de nome '*questao1*'.

Exemplos do funcionamento do programa:

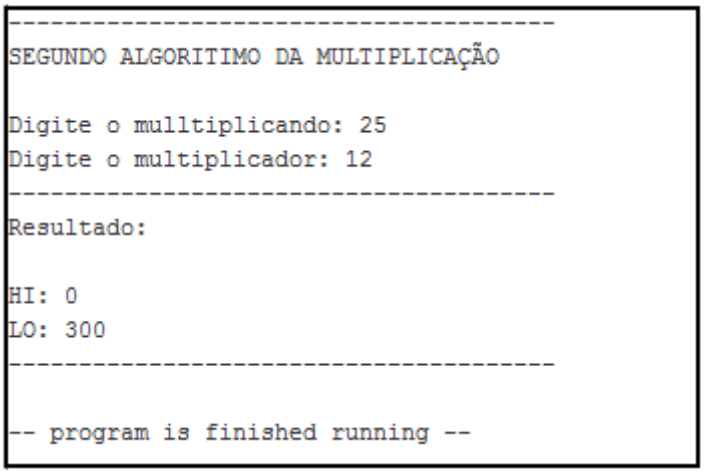

Figura 3 – Multiplicação de dois números.
Fonte: Elaborado pelos autores.

A multiplicação de 25 por 12 tem como resultado 300, o qual está na parte *LO* do resultado.

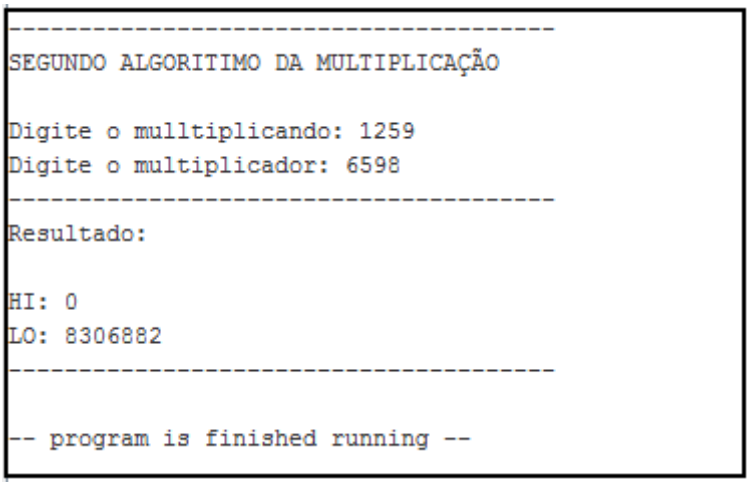

Figura 4 - Multiplicação de dois números.
Fonte: Elaborado pelos autores.

A multiplicação de 1.259 por 6.598 tem como resultado 8.306.882, o qual está na parte *LO* do resultado.

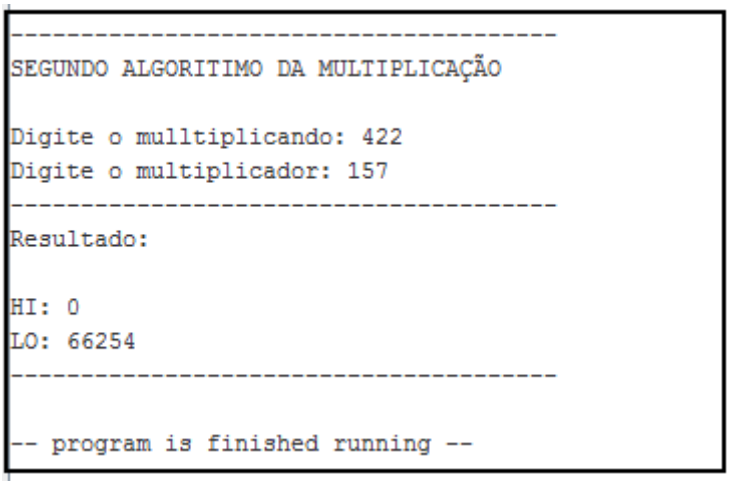

Figura 5 – Multiplicação de dois números.
Fonte: Elaborado pelos autores.

A multiplicação de 422 por 157 tem como resultado 66.254, o qual está na parte *LO* do resultado.

Como pode-se perceber, ao ler o multiplicando e o multiplicador do usuário o resultado da multiplicação apresentado é o correto, logo, o programa está calculando a multiplicação corretamente.

Questão 2. Escreva um programa em linguagem de montagem para o processador MIPS, para encontrar o seno de um ângulo x (um número em ponto flutuante em precisão dupla), usando a seguinte fórmula:

$$\boldsymbol{sin}(\mathbf{x}) = \sum_{n=0} -1^{n} \frac{x^{2.n+1}}{2.n+1} \;=\; x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} - \cdots$$

Essa questão terá seus resultados divididos em duas partes, na primeira, o ponto de parada de execução da fórmula é quando o valor absoluto do resultado da iteração X(n) subtraído da iteração X(n+1) resulta em algo menor ou igual a um erro de forma 1e-9, ou seja, $|X(n+1) - X(n)| \leq 1e\text{-}9$.

Como a fórmula utilizada para o cálculo do seno possui um sinal oscilando a cada iteração, tal condição de parada ser verdadeira significa que estão sendo realizadas operações muito pequenas, tendo em vista que a diferença entre duas iterações consecutivas é menor que o número descrito como erro, o qual, por sua vez, já é muito pequeno.

O ângulo lido do usuário será convertido para assim apresenta-lo em um valor correspondente ao círculo trigonométrico, que é, em geral, o resultado do seno mostrado nas calculadoras convencionais.

Na segunda parte, serão apresentados os resultados do cálculo do seno de um ângulo utilizando como ponto de parada a vigésima iteração da fórmula apresentada na descrição dessa mesma questão e, assim como na primeira parte, o ângulo lido será convertido.

Como será visto nas imagens que seguem, os resultados, em geral, são muito próximos, mas é importante ressaltar que o resultado exato do seno de um ângulo é uma periódica, por esse motivo, quanto mais iterações de somas e subtrações forem feitas, mais próximo do resultado real se chegará.

Os programas em linguagem de montagem se encontram na pasta de nome '*questao2*', que possui duas subpastas, sendo uma contendo o código que utiliza o teste de erro como condição de parada e outra que utiliza a 20ª iteração citada anteriormente, além de podermos conferir o código fonte dessa questão no Anexo A.

Parte um: Utilizando o erro 1e-9 como condição de parada

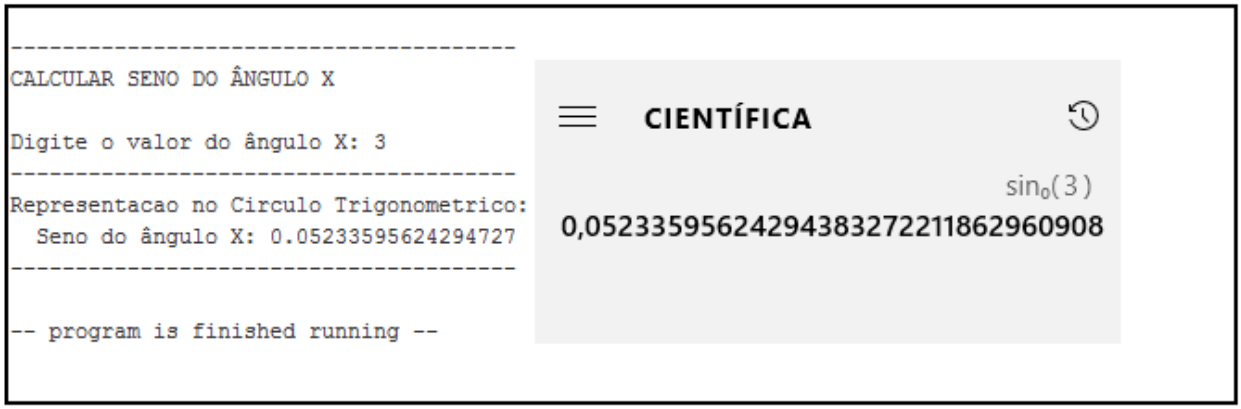

Figura 6 – Esquerda: Resultado do seno de 3 pelo programa. Direita: Resultado do seno de 3 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

A calculadora apresenta o resultado com mais precisão, porém ambos estão corretos.

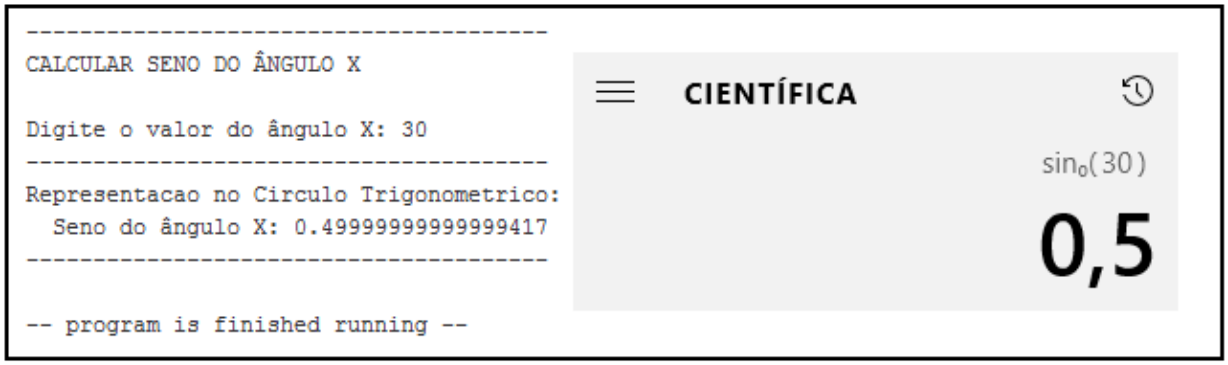

Figura 7 – Esquerda: Resultado do seno de 30 pelo programa. Direita: Resultado do seno de 30 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

Na figura 5 tanto o resultado obtido no programa quanto o resultado apresentado na calculadora são muito próximos.

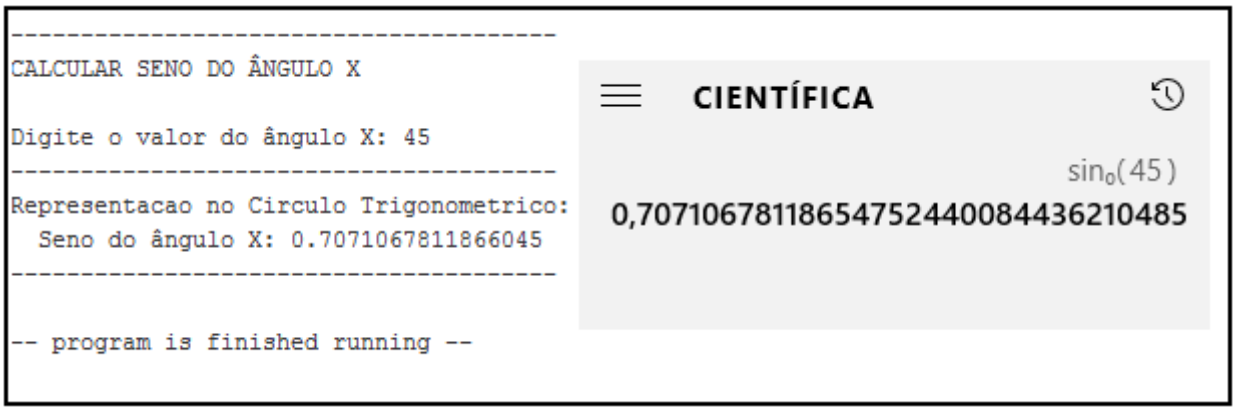

Figura 8 – Esquerda: Resultado do seno de 45 pelo programa. Direita: Resultado do seno de 45 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

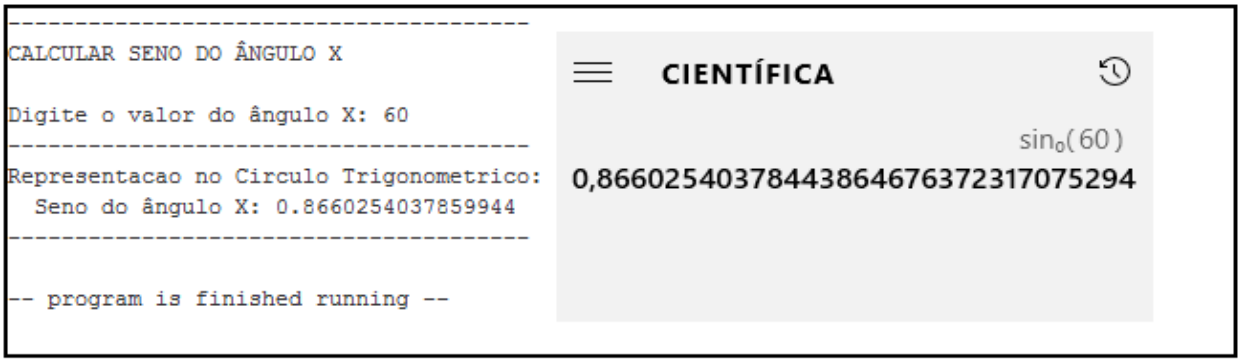

Figura 9 – Esquerda: Resultado do seno de 60 pelo programa. Direita: Resultado do seno de 60 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

Nas figuras 6 e 7 a calculadora apresenta o resultado com mais precisão, porém ambos estão corretos.

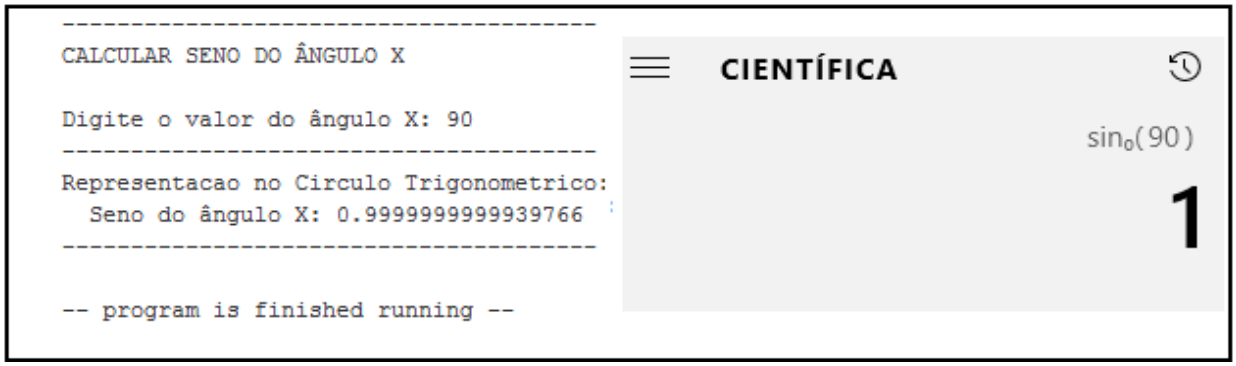

Figura 10 - Esquerda: Resultado do seno de 3 pelo programa. Direita: Resultado do seno de 3 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

Por fim, na figura 8, percebemos que ambos os resultados são muito próximos.

Parte dois: Utilizando a vigésima iteração como condição de parada.

Figura 11 – Esquerda: Resultado do seno de 3 pelo programa. Direita: Resultado do seno de 3 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

Nesse caso, a calculadora apresenta o resultado com mais precisão, porém ambos estão corretos.

Figura 12 – Esquerda: Resultado do seno de 30 pelo programa. Direita: Resultado do seno de 30 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

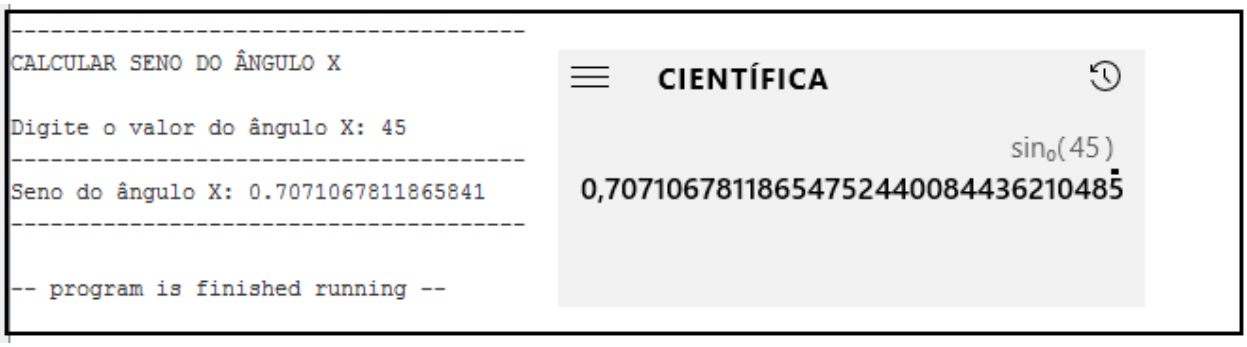

Figura 13 – Esquerda: Resultado do seno de 45 pelo programa. Direita: Resultado do seno de 45 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

Nas figuras 10 e 11 percebemos que tanto o resultado do programa quanto o resultado da calculadora são muito próximos.

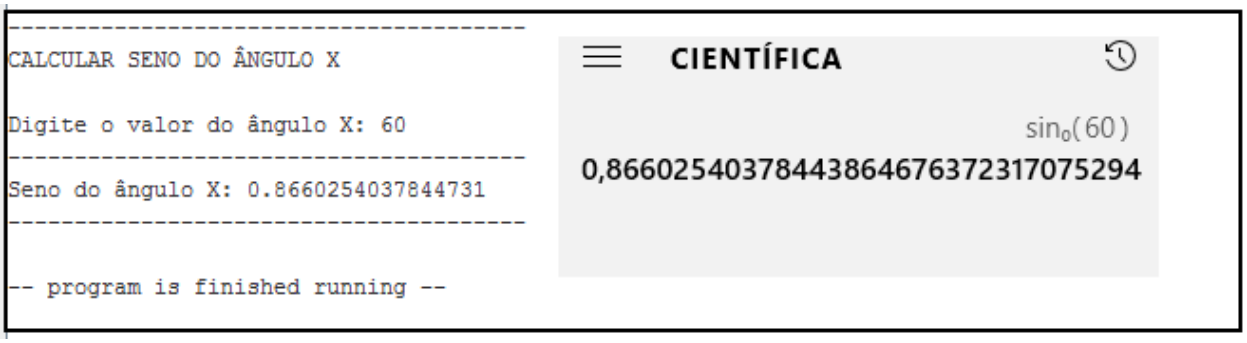

Figura 14 – Esquerda: Resultado do seno de 60 pelo programa. Direita: Resultado do seno de 60 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

Na figura 12 a calculadora apresenta o resultado com mais precisão, porém ambos estão corretos.

Figura 15 – Esquerda: Resultado do seno de 90 pelo programa. Direita: Resultado do seno de 90 pela calculadora.
Fonte: Esquerda: Elaborado pelos autores. Direita: Windows.

Temos exatamente o mesmo resultado com o programa e com a calculadora.

A análise dos resultados será realizada na sessão sete deste relatório, onde será discutido a eficácia de cada uma das maneiras anteriormente citadas e também serão analisados os resultados apresentados pelo programa sem realizar conversão do ângulo digitado pelo usuário, o que, deveria, a principio, resultar em um resultado em radianos.

Questão 3. Seja o seguinte código em linguagem de montagem para o processador MIPS:

```
lw $t0, 300($t1)
sub $t2, $t0, $t1
beq $t2, $t3, 1
j 0x00400014
```

a) Escreva este código em linguagem de máquina (represente em hexadecimal e em binário).

Instrução	Representação hexadecimal	Representação em binário
lw $t0, 300($t1)	0x8D28012C	$10001101001010000000000100101100_2$
sub $t2, $t0, $t1	0x01095022	$00000001000010010101000000100010_2$
beq $t2, $t3, 1	0x114B0001	$00010001010010110000000000000001_2$
j 0x00400014	0x08100005	$00001000000100000000000000000101_2$

b) Se o endereço da primeira instrução é 0x00400000, qual é o endereço da última instrução?

Instrução	Endereço
lw $t0, 300($t1)	0x00400000
sub $t2, $t0, $t1	0x00400004
beq $t2, $t3, 1	0x00400008
j 0x00400014	0x0040000c

c) Tome o código de máquina das instruções, encontre o tipo e desenhe os campos e seus valores.

1ª instrução: lw $t0, 300($t1) TIPO I

	Opcode		Rs ($t1)		Rt ($t0)		Endereço			
Quantia de bits	6 bits		5 bits		5 bits		16 bits			
Representação decimal	35		9		8		300			
Representação binária	1000	11	01	001	0	1000	0000	0001	0010	1100
Representação hexadecimal	8	D		2		8	0	1	2	C

2ª instrução: sub $t2, $t0, $t1 TIPO R

	Opcode		Rs ($t0)		Rt ($t1)		Rd ($t2)		Shamt		Funct	
Quantia de bits	6 bits		5 bits		5 bits		5 bits		5 bits		6 bits	
Representação decimal	0		8		9		10		0		34	
Representação binária	0000	00	01	000	0	1001	0101	0	000	00	10	0010
Representação hexadecimal	0	1		0		9	5		0		2	2

3ª instrução: beq $t2, $t3, 1 TIPO I

	Opcode		Rs ($t1)		Rt ($t0)		Endereço			
Quantia de bits	6 bits		5 bits		5 bits		16 bits			
Representação decimal	4		10		11		1			
Representação binária	0001	00	01	010	0	1011	0000	0000	0000	0001
Representação hexadecimal	1	1		4		B	0	0	0	1

4ª instrução: j 0x00400014 TIPO J

	Opcode		Endereço						
Quantia de bits	6 bits		26 bits						
Representação decimal	2		0x00400014 = 4194324/4 = 1048581						
Representação binária	0000	10	00	0001	0000	0000	0000	0000	0101
Representação hexadecimal	0	8		1	0	0	0	0	5

Questão 4. Usando o Diagrama de blocos do processador MIPS (cap. 4), explique detalhadamente como são processadas as seguintes instruções:

a) add $t1, $t2, $t3

	Opcode		Rs ($t0)		Rt ($t1)		Rd ($t2)		Shamt		Funct	
Quantia de bits	6 bits		5 bits		5 bits		5 bits		5 bits		6 bits	
Representação decimal	0		8		9		10		0		16	
Representação binária	0000	00	01	010	0	1011	0100	1	000	00	10	0000
Representação hexadecimal	0	1		4		B	4		8		2	0

Endereço em hexadecimal: 0x014B4820

Endereço em binário: 0000 0001 0100 1011 0100 1000 0010 0000

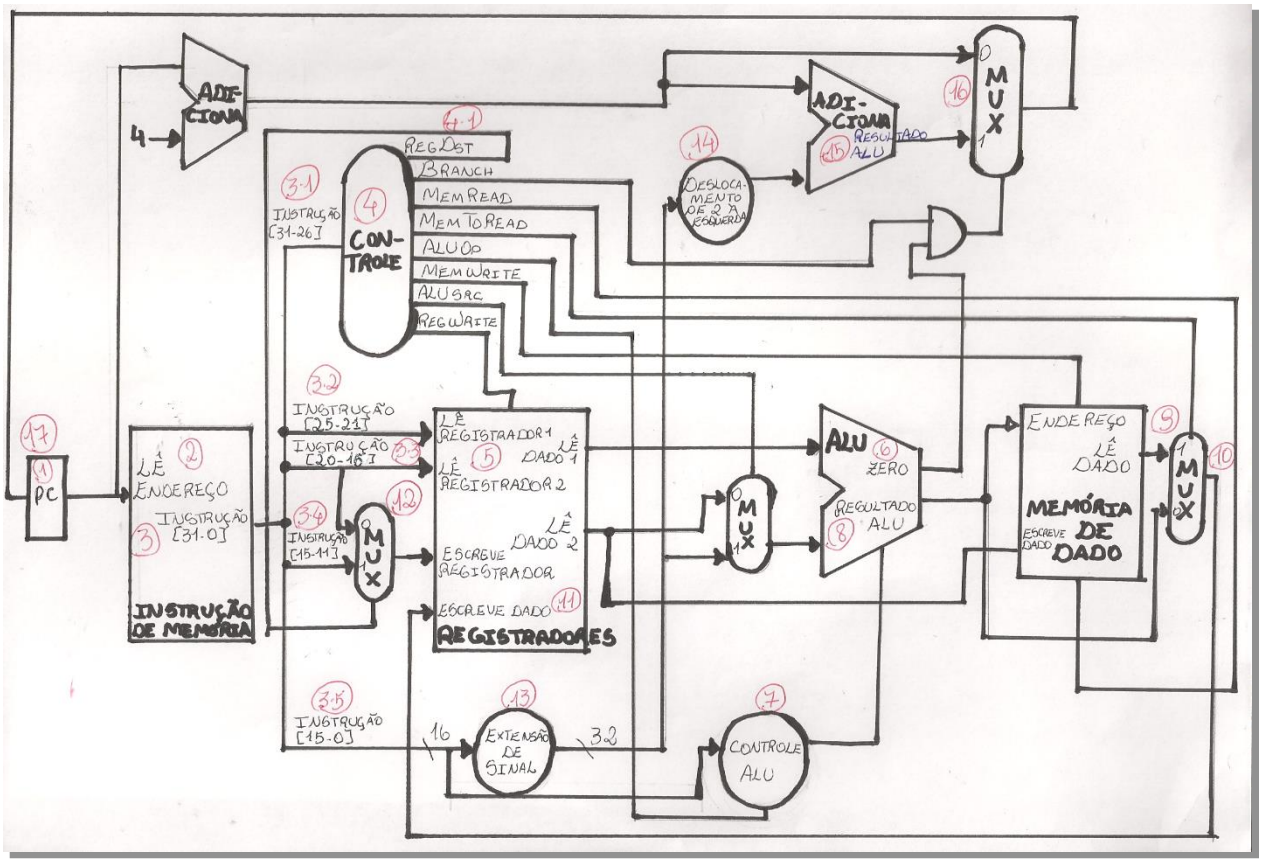

Figura 16 – Diagrama de blocos do MIPS para a instrução add $t1, $t2, $t3
Fonte: Elaborado pelos autores.

Sequência de passos do diagrama ilustrado:

1) Na primeira borda de subida o endereço 0x0040000 é carregado em PC.
2) A instrução é buscada da memória de instruções e o PC manda seu endereço para o bloco funcional ADD que posteriormente ira calcular seu novo endereço.
3) Na saída a instrução que contem 32 bits é dividida em conjuntos de bits.
   3.1) O primeiro grupo de bits compreende os primeiros 6 bits da instrução [31-26] correspondem ao Opcode.
   3.2) O segundo grupo de bits compreende os próximos 5 bits da instrução [25-21] correspondem ao primeiro campo registrador de origem (rs), nessa instrução, corresponde ao $t2
   3.3) O terceiro grupo de bits compreende os próximos 5 bits da instrução [20-16] correspondem ao segundo campo registrador de origem (rt), nessa instrução, corresponde ao $t3
   3.4) O quarto grupo de bits compreende os próximos 5 bits da instrução [15-11] correspondem ao registrador de destino (rd), nessa instrução, corresponde ao $t1
   3.5) O quinto grupo de bits compreende os 4 bits do grupo quatro mais os próximos 12 bits da instrução, totalizando 16 bits [15-0], corresponde aos campos rd, shamt e funct.
4) As linhas da unidade de controle principal são determinadas pelo campo OPcode (primeiro grupo) da instrução.

   4.1) Os valores da linha de controle correspondentes a instrução add (formato R) por padrão são:

   RegDst = 1, Branch = 0, MemRead = 0, MemtoReg = 0, ALUOP1 = 1, ALUOP0 = 0, MemWrite = 0, Alusrc = 0, RegWrite = 1.
5) O Registrador 1 ($t2) e o Registrador 2 ($t3) são lidos do banco de registrados, e a unidade de controle principal calcula a definição das linhas controle também durante essa etapa.
6) A ALU (Unidade Lógica Aritímetica) recebe os dois primeiros registradores, o Registrador 1 é recebido diretamente do banco de registradores e o Registrador 2 é selecionado pelo MUX (multiplexador) pois o seu sinal de controle (ALUSrc) é igual à 0, e o Registrador 2 está no bit 0 do MUX.

7) O controle ALU recebe os 5 bits do campo funct (00000) e o valor de ALUOP (10), com isso, realizando-se a tabela verdade, a operação resultado é 0010 (adição).
8) Com os dois registradores ($t2 e $t3) e a operação definida (add), é gerado o resultado ALU ($t2 + $t3).
9) Como o valor do MemtoReg é 0 o multiplexador indicado no número 9 no desenho vai selecionar o bit 0 o qual corresponde ao resultado da ALU. Como MemWrite e MemRead possuem valor 0, não são realizadas operações de leitura e escrita na Memória de dado.
10) Como o bit 0 do multiplixador foi o selecionado, o resultado da ALU ($t2 + $t3) chega ao banco de registradores.
11) Na próxima borda de subida, como o valor de RegWrite é 1, o resultado da operação ($t2 + $t3) é escrito em $t1, que foi selecionado no MUX indicado pelo número 12 no desenho.
12) O multiplexador recebe o valor de controle RegDst que é 1, logo o bit 1, que corresponde à rs ($t1) é escolhido.
13) O quinto grupo de bits da instrução [15-0] passa por uma extensão de sinal e passa a ter 32 bits.
14) É feito um deslocamento de 2 bits à esquerda com o resultado obtido no item 13, ou seja, o valor é multiplicado por 4.
15) O bloco funcional adiciona recebe o valor de PC + 4 e o resultado do deslocamento do item 14.
16) O resultado dessa adição do item 15 entra no bit 1 do MUX do desenho. E o bit 0 do MUX recebe PC + 4. O controle desse MUX é o resultado do AND indicado pelo 16.1 entre o valor do Branch (0) e o zero recebido da ALU.
    16.1) O valor resultado é 0.
    Com isso é escolhido o bit 0 do MUX, o qual é PC + 4.
17) Quando ocorrer a borda de subida, o valor referente ao item 11 será gravado no registrador referente ao item 12 e o valor de PC será incrementado.

b) lw $t1, 1000($t2)

	Opcode		Rs ($t2)		Rt ($t1)		Endereço			
Quantia de bits	6 bits		5 bits		5 bits		16 bits			
Representação decimal	35		10		9		1000			
Representação binária	1000	11	01	010	0	1001	0000	0011	1110	1000
Representação hexadecimal	8	D		4		9	0	3	E	8

Endereço em hexadecimal: 0x8D4903E8

Endereço em binário: 1000 1101 0100 1001 0000 0011 1110 1000

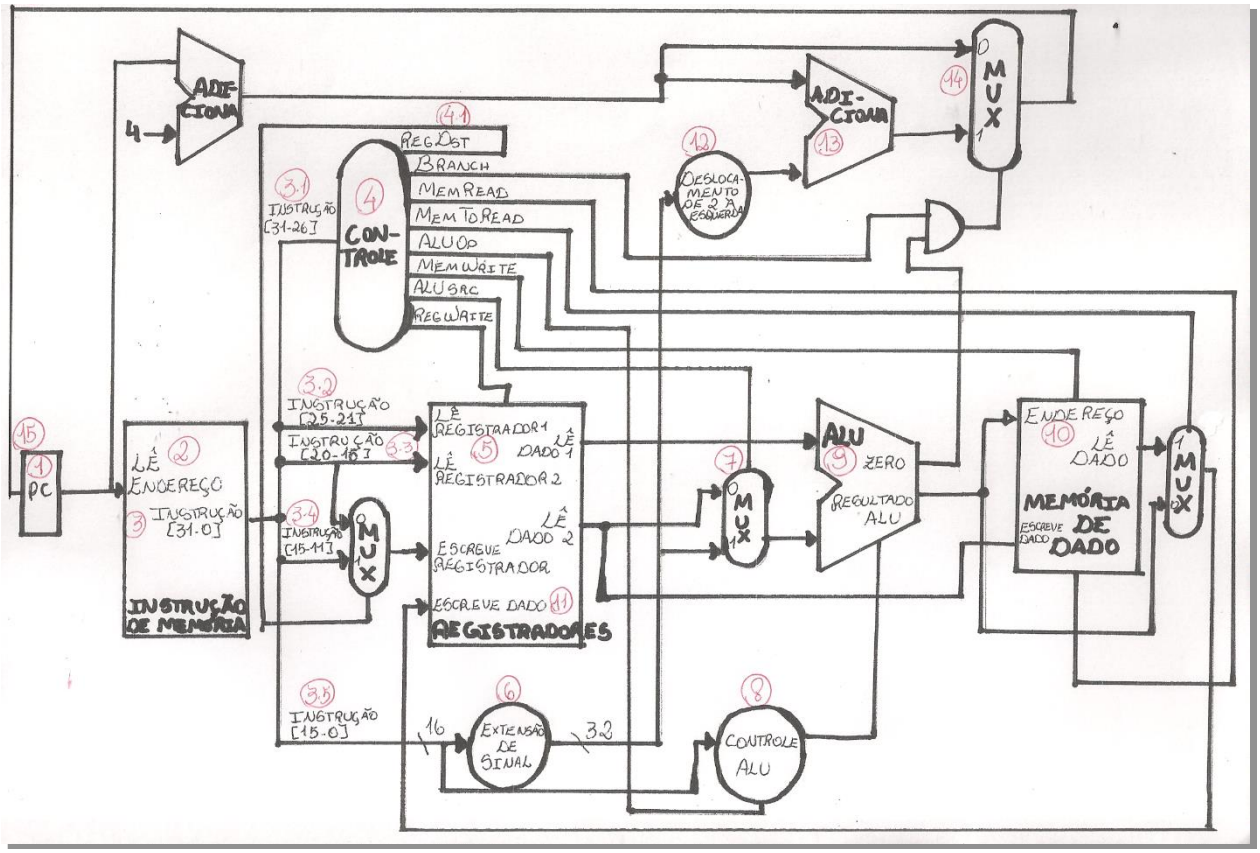

Figura 17 – Diagrama de blocos do MIPS para a instrução lw $t1, 1000($t2)
Fonte: Elaborado pelos autores.

Sequência de passos do diagrama ilustrado:

1) Na primeira borda de subida o endereço 0x0040000 é carregado em PC.
2) A instrução é buscada da memória de instruções e o PC manda seu endereço para o bloco funcional ADD que posteriormente ira calcular seu novo endereço.
3) Na saída a instrução que contem 32 bits é dividida em conjuntos de bits.
   3.1) O primeiro grupo de bits compreende os primeiros 6 bits da instrução [31-26] correspondem ao Opcode.
   3.2) O segundo grupo de bits compreende os próximos 5 bits da instrução [25-21] correspondem ao primeiro campo registrador de origem (rs), nessa instrução, corresponde ao $t2
   3.3) O terceiro grupo de bits compreende os próximos 5 bits da instrução [20-16] correspondem ao segundo campo registrador de origem (rt), nessa instrução, corresponde ao $t1
   3.4) O quarto grupo de bits compreende os próximos 5 bits da instrução [15-11], nessa instrução correspondem aos 5 bits mais significativos do campo de endereço (00000)
   3.5) O quinto grupo de bits compreende os 16 bits do campo de endereço [15-0].
4) As linhas da unidade de controle principal são determinadas pelo campo OPcode (primeiro grupo) da instrução.

   4.1) Os valores da linha de controle correspondentes a instrução lw (formato I) por padrão são:

   RegDst = 0, Branch = 0, MemRead = 1, MemtoReg = 1, ALUOP1 = 0, ALUOP0 = 0, MemWrite = 0, Alusrc = 1, RegWrite = 1.
5) O Registrador 1 ($t2) e o Registrador 2 ($t1) são lidos do banco e a unidade de controle principal calcula a definição das linhas controle também durante essa etapa.
6) O quinto grupo de bits da instrução [15-0] passa por uma extensão de sinal e passa a ter 32 bits.
7) Esse endereço resultante do item 6, com 32 bits, é selecionado pelo MUX (multiplexador) pois está no bit 1 do MUX, e o sinal de controle (ALUSrc) é igual 1.
8) O controle ALU recebe os 5 bits menos significativos do campo endereço (1000) e o valor de ALUOP (00), com isso, realizando-se a tabela verdade, a operação resultado é 0010 (adição).

9) A ALU (Unidade Lógica Aritímetica) recebe o Registrador 1 ($t2) e campo endereço com sinal extendido, como a operação definida é a add, é gerado o resultado ALU ($t2 + endereço).
10) Como MemRead possue valor 1, é realizada a leitura do dado da Memória. Como o valor do MemtoReg é 1 o multiplexador vai selecionar o bit 1, o qual corresponde ao dado lido da memória.
11) O dado da memória chega ao banco de registradores. Na próxima borda de subida, como o valor de RegWrite é 1, o resultado da operação é escrito.
12) É feito um deslocamento de 2 bits à esquerda com o resultado obtido no item 6, ou seja, o valor é multiplicado por 4.
13) O bloco funcional adiciona recebe o valor de PC + 4 e o resultado do deslocamento do item 12.
14) O resultado dessa adição do item 13 entra no bit 1 do MUX do desenho. E o bit 0 do MUX recebe PC + 4. O controle desse MUX é o resultado do AND indicado pelo 14.1 entre o valor do Branch (0) e o zero recebido da ALU.
    14.1) O valor resultado é 0.
    Com isso é escolhido o bit 0 do MUX, o qual é PC + 4.
15) Quando ocorrer a borda de subida, o valor referente ao item 11 será gravado em $t1 e o valor de PC será incrementado

c) sw $t1, 1000($t2)

	Opcode		Rs ($t2)		Rt ($t1)		Endereço			
Quantia de bits	6 bits		5 bits		5 bits		16 bits			
Representação decimal	43		10		9		1000			
Representação binária	1010	11	01	010	0	1001	0000	0011	1110	1000
Representação hexadecimal	A	D		4		9	0	3	E	8

Endereço em hexadecimal: 0xAD4903E8

Endereço em binário: 1010 1101 0100 1001 0000 0011 1110 1000

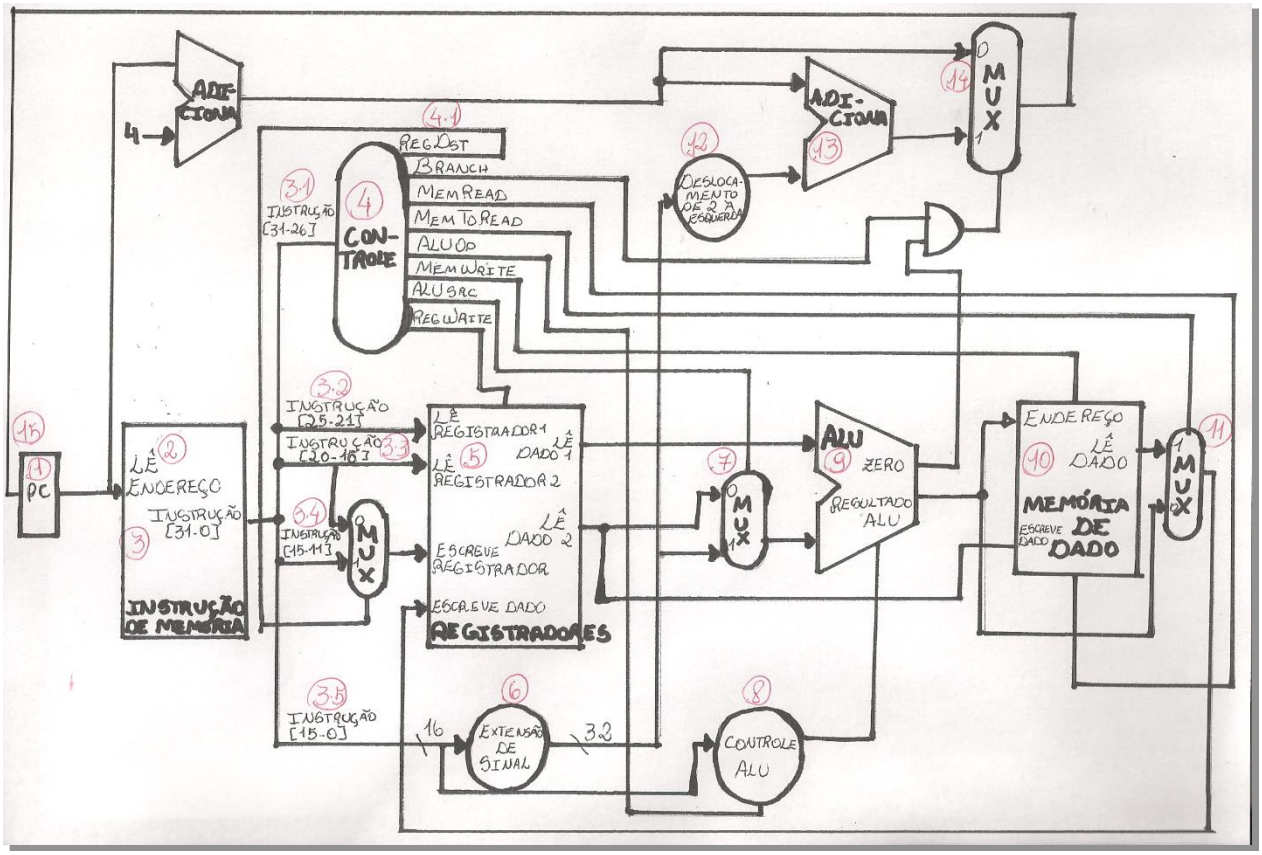

Figura 18 – Diagrama de blocos do MIPS para a instrução sw $t1, 1000($t2)
Fonte: Elaborado pelos autores.

Sequência de passos:

1) Na primeira borda de subida o endereço 0x0040000 é carregado em PC.
2) A instrução é buscada da memória de instruções e o PC manda seu endereço para o bloco funcional ADD que posteriormente ira calcular seu novo endereço.
3) Na saída a instrução que contem 32 bits é dividida em conjuntos de bits.
   3.1) O primeiro grupo de bits compreende os primeiros 6 bits da instrução [31-26] correspondem ao Opcode.
   3.2) O segundo grupo de bits compreende os próximos 5 bits da instrução [25-21] correspondem ao primeiro campo registrador de origem (rs), nessa instrução, corresponde ao $t2
   3.3) O terceiro grupo de bits compreende os próximos 5 bits da instrução [20-16] correspondem ao segundo campo registrador de origem (rt), nessa instrução, corresponde ao $t1
   3.4) O quarto grupo de bits compreende os próximos 5 bits da instrução [15-11], nessa instrução correspondem aos 5 bits mais significativos do campo de endereço (00000)
   3.5) O quinto grupo de bits compreende os 16 bits do campo de endereço [15-0].
4) As linhas da unidade de controle principal são determinadas pelo campo OPcode (primeiro grupo) da instrução.

   4.1) Os valores da linha de controle correspondentes a instrução sw (formato I) por padrão são:

   RegDst = X, Branch = 0, MemRead = 0, MemtoReg = X, ALUOP1 = 0, ALUOP0 = 0, MemWrite = 1, Alusrc = 1, RegWrite = 0.
5) O Registrador 1 ($t2) e o Registrador 2 ($t1) são lidos do banco e a unidade de controle principal calcula a definição das linhas controle também durante essa etapa.
6) O quinto grupo de bits da instrução [15-0] passa por uma extensão de sinal e passa a ter 32 bits.
7) Esse endereço resultante do item 6, com 32 bits, é selecionado pelo MUX (multiplexador) pois está no bit 1 do MUX, e o sinal de controle (ALUSrc) é igual 1.
8) O controle ALU recebe os 5 bits menos significativos do campo endereço (1000) e o valor de ALUOP (00), com isso, realizando-se a tabela verdade, a operação resultado é 0010 (adição).

9) A ALU (Unidade Lógica Aritímetica) recebe o Registrador 1 ($t2) e campo endereço com sinal extendido, como a operação definida é a add, é gerado o resultado ALU ($t2 + endereço).
10) Como MemWrite tem valor 1, o dado ($t1) é escrito na memória no endereço calculado.
11) O MUX possui como valor de controle MemtoReg, o qual é X (valor não importa). Então tanto o dado quanto o resultado ALU podem ser mandados para o Registrador.
12) É feito um deslocamento de 2 bits à esquerda com o resultado obtido no item 6, ou seja, o valor é multiplicado por 4.
13) O bloco funcional adiciona recebe o valor de PC + 4 e o resultado do deslocamento do item 12.
14) O resultado dessa adição do item 13 entra no bit 1 do MUX do desenho. E o bit 0 do MUX recebe PC + 4. O controle desse MUX é o resultado do AND indicado pelo 14.1 entre o valor do Branch (0) e o zero recebido da ALU.
    14.1) O valor resultado é 0.
    Com isso é escolhido o bit 0 do MUX, o qual é PC + 4.
15) Quando ocorrer a borda de subida o valor de PC será incrementado.

d) beq $t1, $t2, loop (considere que loop é um rótulo para a 10 instrução antes da instrução atual beq).

	Opcode		Rs ($t2)		Rt ($t1)		Endereço			
Quantia de bits	6 bits		5 bits		5 bits		16 bits			
Representação decimal	4		10		9		-11			
Representação binária	0001	00	01	001	0	1010	1111	1111	1111	0101
Representação hexadecimal	1	1		2		A	F	F	F	5

Endereço em hexadecimal: 0x112AFFF5

Endereço em binário: 0001 0001 0010 1010 1111 1111 1111 0101

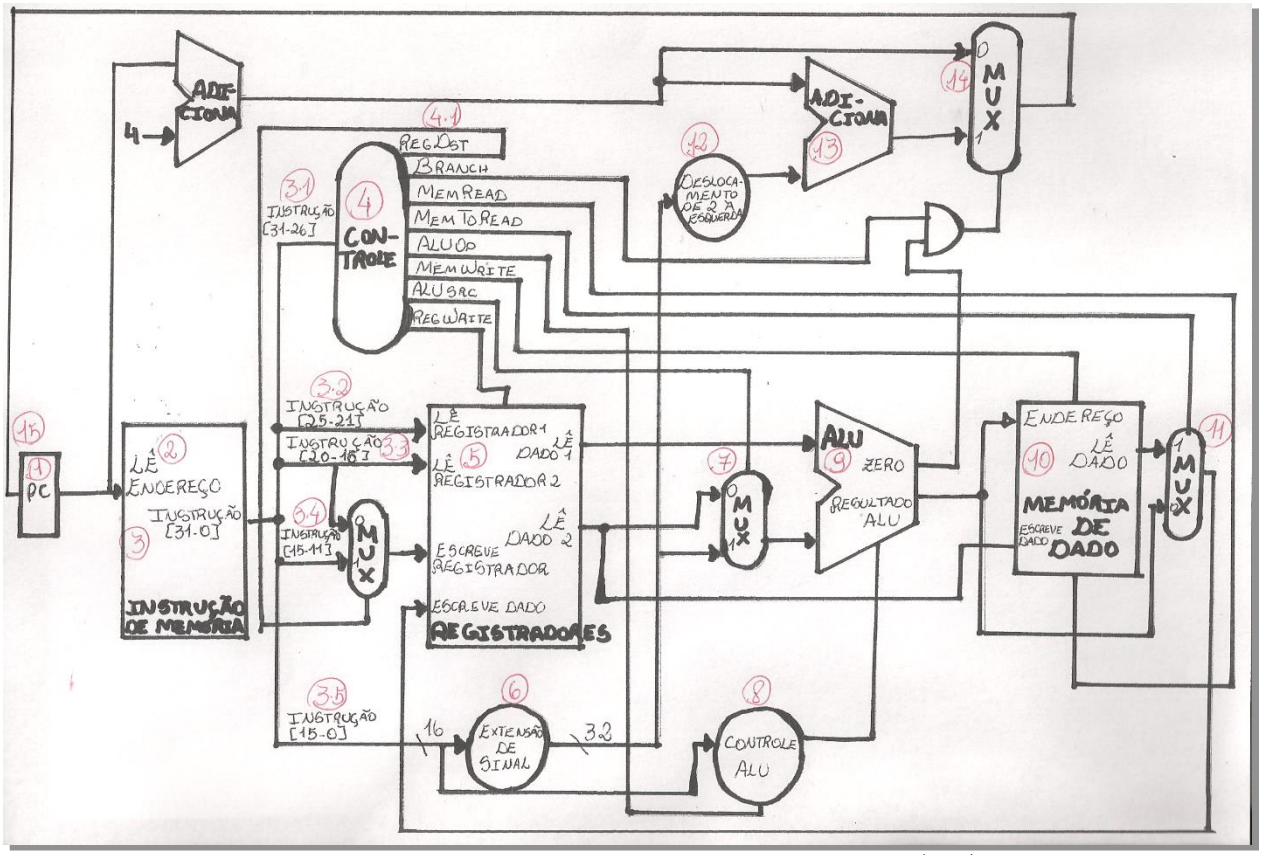

Figura 19 – Diagrama de blocos do MIPS para a instrução beq $t1, $t2, loop
Fonte: Elaborado pelos autores.

Sequência de passos:

1. Na primeira borda de subida o endereço 0x0040000 é carregado em PC.
2. A instrução é buscada da memória de instruções e o PC manda seu endereço para o bloco funcional adiciona que posteriormente ira calcular seu novo endereço.
3. Na saída a instrução que contem 32 bits é dividida em conjuntos de bits.

3.1 O primeiro grupo de bits compreende os primeiros 6 bits da instrução [31-26] correspondem ao Opcode.

3.2 O segundo grupo de bits compreende os próximos 5 bits da instrução [25-21] correspondem ao primeiro campo registrador de origem (rs), nessa instrução, corresponde ao $t2

3.3 O terceiro grupo de bits compreende os próximos 5 bits da instrução [20-16] correspondem ao segundo campo registrador de origem (rt), nessa instrução, corresponde ao $t1

3.4 O quarto grupo de bits compreende os próximos 5 bits da instrução [15-11], nessa instrução correspondem aos 5 bits mais significativos do campo de endereço (11111)

3.5 O quinto grupo de bits compreende os 16 bits do campo de endereço [15-0].

4.As linhas da unidade de controle principal são determinadas pelo campo OPcode (primeiro grupo) da instrução. Os valores da linha de controle correspondentes a instrução beq por padrão são:

4.1 RegDst = X, Branch = 1, MemRead = 0, MemtoReg = X, ALUOP1 = 0, ALUOP0 = 1, MemWrite = 0, Alusrc = 0, RegWrite = 0.

5) O Registrador 1 ($t2) e o Registrador 2 ($t1) são lidos do banco e a unidade de controle principal calcula a definição das linhas controle também durante essa etapa.

6) O quinto grupo de bits da instrução [15-0] passa por uma extensão de sinal e passa a ter 32 bits.

7) O dado 2 do banco de registradores é selecionado pelo MUX (multiplexador) pois está no bit 1 do MUX, e o sinal de controle (ALUSrc) é igual 0.

8) O controle ALU recebe os 5 bits menos significativos do campo endereço (11111) e o valor de ALUOP (01), com isso, realizando-se a tabela verdade, a operação resultado é 0110 (subtract).

9) A ALU (Unidade Lógica Aritímetica) recebe o Registrador 1 ($t2) e o Registrador 2 ($t1), como a operação definida é a sub, é gerado o resultado ALU ($t2 - $t1).

10) Como MemWrite e MemRead têm valor 0, não é feita a escrita e leitura na memória.

11) O MUX possui como valor de controle MemtoReg, o qual é X (valor não importa). Então tanto o dado quanto o resultado ALU podem ser mandados para o Registrador.

12) É feito um deslocamento de 2 bits à esquerda com o resultado obtido no item 6, ou seja, o valor é multiplicado por 4, ou seja o valor que entra no bloco Adiciona é 4*(-11) = -44.

13) O bloco funcional adiciona recebe o valor de PC + 4 e o resultado do deslocamento do item 12, resultando em PC - 40.

14) O resultado dessa adição do item 13 entra no bit 1 do MUX do desenho. E o bit 0 do MUX recebe PC + 4. O controle desse MUX é o resultado do AND indicado pelo item 14.1.

14.1) O AND recebe o valor do Branch (1) e o valor da ALU que, se os valores de $t2 e $t1 foram iguais, será 1, logo PC-40 será escolhido, ou se forem diferentes, o valor de ALU será 0, logo PC+4 será escolhido.

Quando ocorrer a borda de subida PC receberá seu novo valor calculado no item acima.

6. RESULTADOS

De acordo com os objetivos definidos nesse relatório, o resultado da resolução das questões foi o seguinte:

- O programa em *Assembly* que realiza a multiplicação de dois números utilizando o segundo algoritmo de multiplicação apresentado no livro texto da disciplina foi desenvolvido e tem um funcionamento correto;
- O programa que encontra o seno de um ângulo, sendo esse ângulo um número em ponto flutuante e de precisão dupla foi escrito e possui um bom funcionamento;
- Nas questões que necessitavam, foram analisadas as maneiras possíveis de resolução e, consequentemente, qual o resultado mais satisfatório;
- Os códigos em linguagem de montagem do processador MIPS foram traduzidos para a linguagem de máquina, representando-a em binário e em hexadecimal;
- Foi explicado, detalhadamente, como são processadas as instruções *addition (add), load word (lw), store word (sw)* e *branch if equal (beq*) pelo computador, utilizado o Diagrama de blocos do processador MIPS.

Por sua vez, tendo os objetivos iniciais cumpridos, acreditamos que a elaboração deste trabalho aprimorou os conhecimentos dos alunos envolvidos e, também, auxiliou na absorção da matéria passada em sala de aula, além de permitir uma melhor compreensão de como são realizadas, de fato, algumas das instruções pelo computador.

7. DISCUSSÃO

Questão 1.

Conforme visto nos resultados, o programa está realizando a multiplicação corretamente, para isso, são utilizados quatro registradores base, um contendo o multiplicando (32 bits) e dois contendo o produto, que é uma união tanto do resultado da multiplicação quanto do multiplicador, um deles possui a parte alta (*Hi*), com 32 bits e outro a parte baixa (*Lo*), também com 32 bits, além de um registrador que contém o bit de *carry* do produto.

Durante 32 etapas, caso o bit menos significativo do produto seja 1, somaremos os bits mais significativos (*Hi*: 32 bits) do produto com o multiplicando (32 bits), sinalizando no bit de *carry* (*MSB*) um possível *overflow* e após realizando um deslocamento para a direita tanto do bit de *MSB* quanto dos dois registradores do produto (*Hi* e *Lo*) e, caso o bit menos significativo do produto seja 0, não realizaremos a soma, apenas o deslocamento.

Um fato interessante é que as duas instruções de multiplicação do *MIPS* (*mult* e *multu*) ignoram o *overflow*, de modo que fica a critério do programa verificar se o produto é muito grande para caber nos 32 bits. Não existe overflow se *Hi* for 0 para *multu* ou o sinal replicado de *Lo* para *mult* A instrução *mfhi* pode ser usada para transferir *Hi* para um registrador a fim de testar o *overflow*.

No caso de nosso programa, utilizamos o algoritmo de detecção de *overflow* apresentado no livro texto da disciplina quando necessário.

Questão 2.

A seguir, serão analisadas as saídas geradas pelo programa apresentadas na sessão experimento, assim como outros resultados obtidos. Também analisaremos o comportamento do programa sem realizar a conversão para o círculo trigonométrico do ângulo lido pelo usuário.

A tabela abaixo ilustra tais resultados, os campos em verdes serão dados aos valores iguais ou muito próximos aos valores obtidos em uma calculadora convencional (primeira linha da tabela) e os campos em vermelho serão dados aos resultados que não estão corretos. A precisão da tabela foi de três casas decimais, os resultados que necessitarem de uma maior precisão para comparações serão retomados abaixo.

REULTADO ENCONTRADO	CONV.	ÂNGULO					
		3º	15º	30º	45º	60º	90º
Calculadora.	X	0,052	0,258	0,5	0,707	0,866	1
		0,141	0,650	-0,988	0,850	-0,304	0,893
Utilizando vigésima Iteração.	X	0,052	0,258	0,500	0,707	0,866	1,0
		0,141	0,655	3,702	9,788	1.630	3,297
Utilizando erro.	X	0,052	0,258	0,499	0,707	0,866	0,999
		0,141	0,650	-9,87	501,3	4,96	N/A

Tabela 1 – Comparações de resultados do seno de um ângulo, sendo este convertido ou não.
Fonte: Elaborado pelos autores.

Primeiro vamos analisar os campos correspondentes ao programa em que é realizada a conversão prévia do ângulo, indicado pelo campo CONV marcado com um X.

Percebemos que, ao realizar a conversão para o círculo trigonométrico do ângulo digitado pelo usuário, ambas as condições de parada apresentam resultados corretos (nesse caso os resultados foram comparados com os resultados obtidos na calculadora cientifica do Windows).

Outro ponto interessante a ser ressaltado é o fato de que em ângulos mais “exatos”, como é o caso de 30º e 90º (0,5 e 1, respectivamente), o programa apresenta um resultado aproximado, como pode ser visto na tabela a baixo.

Angulo	Resultado calculadora.	Utilizando erro como condição de parada.	Utilizando 20ª iteração como condição de parada.
30º	0,5	0,49999999999999417	0,5000000000000299
90º	1	0,9999999999939766	1,0

Tabela 2 – Resultados obtidos em ângulos específicos.
Fonte: Elaborado pelos autores.

Vale, é claro, ressaltar que a calculadora não está fazendo arredondamento do seno e, com isso, percebemos uma tendência na qual a fórmula utilizada pelo programa se aproxima do resultado por valores menores ou maiores que o mesmo:

- Utilizando o erro como condição de parada, temos um resultando minimamente menor do que o resultado apresentado pela calculadora (o qual estamos tomando como um valor absolutamente correto).

- Utilizando a vigésima iteração como condição de parada, percebemos que o resultado é minimamente maior ou igual ao resultado apresentado pela calculadora.

Nos demais casos, temos um resultado muito aproximado em suas casas decimais, tanto da calculadora como do programa (ver imagens apresentadas na sessão experimento).

Agora, partindo para os campos correspondentes ao programa que não realiza a conversão prévia do ângulo, indicado pelo campo CONV em que o X não está marcado, percebemos que os senos obtidos como resultado pelo programa deveriam ser em radianos, porém, talvez por não possuir uma condição de parada mais específica para o caso de se desejar obter o seno em radianos, não conseguimos resultados corretos a partir de certo ponto. É perceptível, no entanto, que com ângulos menores os resultados são certos, como é o caso dos ângulos 3 e 15.

Nessa análise não foram considerados ângulos acima de 90º, pois, como será visto na sessão correspondente a conclusão dessa questão, essa fórmula é válida para uma faixa específica de valores para o ângulo, sendo que ângulos com maior magnitude podem ser convertidos para ângulos entre 0 e 90 graus.

8. CONCLUSÕES E PERSPECTIVAS

O trabalho aqui relatado foi realizado com êxito, servindo ao propósito inicial do grupo, que era de se beneficiar da realização do trabalho para aprimorar os conhecimentos nos conteúdos da disciplina da cadeira de Organização de Computadores e também desenvolver melhor a lógica computacional.

Acerca da questão 1, foi percebido que realizar a multiplicação de dois números é um processo simples, basicamente realizado por somas e deslocamentos sendo que o hardware para tal também é igualmente simples, e que meramente precisará garantir que testará o bit da direita do produto para verificar uma possível soma e após realizar um deslocamento à direita.

No caso de nosso programa, o resultado apresentado para o usuário é divido em dois registradores correspondentes a parte *Hi* e a parte *Lo* do produto, quando é feita a multiplicação de números que podem ser representados em 32 bits a parte *Hi será* 0 (exceto se for feita tentativa de multiplicação de números com sinais) e quando a multiplicação for de números muito grandes, obviamente, teremos algum valor na parte *Hi*.

Uma possível melhoria do programa seria utilizar técnicas para números com sinais que viriam a melhorar o desempenho do programa, como o algoritmo de multiplicação de Booth, que foi citado em uma aula da disciplina.

Com base análise e discussão dos resultados da questão 2, algumas das discussões abordadas serão finalizadas a seguir, ressaltando para isso alguns tópicos interessantes:

- Melhor condição de parada

Para a maioria dos ângulos, ambas as condições podem ser igualmente satisfatórias e, além do mais, possuírem vantagens perante a outra.

Tomemos como exemplo a condição de parada como 20ª iteração, tal condição pode nos dar um resultado mais exato se vir a realizar mais operações de soma e subtrações que a condição de parada dependente do erro fará. Porém, ao mesmo tempo, podemos estar desperdiçando velocidade e eficácia, tendo em vista que a condição de erro resultar em verdadeira significa que estamos fazendo operações muito pequenas e que, no final, não afetarão significativamente o resultado.

Por esse motivo e, principalmente, por casos contrários, onde vinte iterações realizarão menos operações do que a condição de parada por erro, sendo que isso poderá resultar em valores finais equivocados, escolheremos como melhor condição de parada o teste por erro.

- Conversão

Como visto anteriormente na sessão de discussão, não converter o ângulo lido do usuário acarretará em resultados errados em qualquer ângulo superior a faixa dos 30º. Uma solução para se obter o resultado em radianos (resultado obtido pela forma sem converter o ângulo lido do usuário) seria, após o cálculo com o ângulo convertido, converte-lo novamente para radianos.

Porém no caso desse programa, aceitaremos como melhor opção o caso em que o ângulo lido do usuário é convertido, para assim, no fim, obtermos o seno do ângulo como um valor do circulo trigonométrico (*default* da maioria das calculadoras).

- Faixa de ângulos válida

Um modo de otimizar o programa caso o ângulo tenha magnitude muito grande pode ser utilizar um dos princípios matemáticos de cálculo de ângulos no círculo trigonométrico: Caso o ângulo não esteja representado no ciclo, e ele for maior que 360º, pode ser que seja um ângulo equivalente aos presentes no ciclo, então basta verificar o seno do equivalente que ele será válido para o ângulo em questão também.

O procedimento para verificar se um ângulo tem equivalente é bastante simples: dado um ângulo x qualquer, maior que 360º, fazemos x/360, pegando somente a parte inteira y do resultado. Multiplicamos esse y obtido por 360, e subtraímos o resultado do ângulo inicial x. Então é só verificar se o ângulo encontrado se encaixa com algum valor no ciclo trigonométrico. Exemplo:

- Ângulo 1200º: Dividimos *1200/360*, obtendo como resultado 3,333..., pegamos apenas a parte inteira (3) e multiplicamos por *360 (1080) e* subtraímos o resultado de *1200* (*1200 – 1080 = 120)*. Logo, o complementar desse ângulo é 120º, que significa dizer que ambos têm o mesmo seno.

Porém mesmo fazendo tal melhoria, ainda existem alguns ângulos em qual a fórmula utilizada não irá funcionar como, por exemplo, 180º e 360º e, por sua vez, todos seus equivalente (conforme visto anteriormente), e.g. 540, 720, etc.

Todavia, através da implementação e resolução relatadas nesse trabalho, foi possível colocar em prática o que foi aprendido sobre a linguagem *Assembly*, Aritmética Computacional, Caminho de dados e de Controle, sendo que tais assuntos são muito importantes para tentar compreender, de fato, como um computador consegue "entender" (processar) o que lhe é solicitado.

Portanto, considera-se que trabalhos desse escopo, que envolvem desde a implementação de algoritmos, tradução e processamento das instruções a escrita do relatório são fundamentais para que nós, alunos, consigamos compreender melhor, não só as instruções da linguagem, mas também a importância da mesma para a nossa vida como programadores e para a vida de todos que utilizam um computador.

REFERÊNCIAS BIBLIOGRÁFICAS

CRISTO, Fernando de. **Arquitetura de computadores** / Fernando de Cristo, Evandro Preuss, Roberto Franciscatto. – Frederico Westphalen: Universidade Federal de Santa Maria, Colégio Agrícola de Frederico Westphalen, 2013.

MONTEIRO, Mário A. **Introdução à organização de computadores.** 5. ed. Rio de Janeiro: LTC, 2007.

MORENO, E, **Arquitetura de computadores I**. Diponível em: *<http://www.inf.pucrs.br/~emoreno/undergraduate/EC/arqi/class_files/Aula08.pdf>. Acesso em: julho de 2016.*

PATTERSON, David A, HENNESSY, John L . **Organização e projeto de computadores: a interface hardware/software.** 3. ed. Rio de Janeiro: Elsevier, 2005.

PATTERSON, David A, HENNESSY, John L . **Organização e projeto de computadores: a interface hardware/software.** 4. ed. Rio de Janeiro: Elsevier, 2014.

VOLLMAR, Ken. MARS – MIPS Assembly and Runtime Simulator. Disponível em: <http://courses.missouristate.edu/kenvollmar/mars/>. Acesso em: julho de 2016.

APÊNDICES

APÊNDICE A – CÓDIGOS-FONTE

A seguir segue o código fonte das questões um e dois, sendo está última apresentada com as duas condições de parada de iteração da fórmula anteriormente analisada.

Questão 1.

- macros.asm:

```
#  Macros e equivalências usados por todos arquivos

#########################################################
#  Diretivas .eqv
.eqv programa_OK                0
.eqv servico_imprime_string     4
.eqv servico_imprime_inteiro    1
.eqv servico_le_inteiro         5
.eqv servico_encerra_programa   17
#########################################################

#  Diretivas .macro
.macro encerra_programa(%valor_fim_programa)
        li $a0, %valor_fim_programa
        li $v0, servico_encerra_programa
        syscall
.end_macro
#
.macro imprime_string(%string_para_impressao)
        la $a0, %string_para_impressao
        li $v0, servico_imprime_string
        syscall
.end_macro
#
.macro imprime_inteiro(%inteiro_para_impressao)
        move $a0, %inteiro_para_impressao
        li $v0, servico_imprime_inteiro
        syscall
.end_macro
#
.macro le_inteiro()
        li $v0, servico_le_inteiro
        syscall
        #  $v0 ira conter o inteiro lido
.end_macro
#
#########################################################
```

- main.asm:

```
#####################################################################
 .include "macros.asm"              #  Inclui arquivo com macros
#####################################################################
# Diretivas .eqv

.eqv NumBits              32
.eqv Deslocamento         1
.eqv MascaraBits          0x00000001          #  Para isolar ultimo bit
.eqv MascaraSetaMSB       0x80000000          #  Seta bit mais significativo
.eqv ValorInicial  0x00000000
#####################################################################

.text

.globl main
#
#  Procedimento main do programa que realiza a multiplicacao de dois numeros em binario utilizando
#  o segundo algoritimo de multiplicacao
#
#  historico:
#  20/06/2016      Primeira versao do procedimento
#  23/06/2016        Segunda versao do procedimento
#  24/06/2016        Terceira versao do procedimento
#  02/07/2016      Quarta versao do procedimento
#

main:
#
        ##############################################################################################
############################
#  prologo
                                                # Nao precisamos armazenar endereco de retorno ou
argumentos
#  corpo do procedimento
        imprime_string(separador)
        imprime_string(titulo)

        #  Lê multiplicando
        imprime_string(multiplicando)
        le_inteiro()                    #  $v0 contem o numero lido (multiplicando)
        move $t0, $v0                           #  $t0 <- multiplicando

        #  Lê multiplicador
        imprime_string(multiplicador)
        le_inteiro()                            #  $v0 contem o numero lido (multiplicador)

        #  Argumentos para funcao multiplica
        move $a1, $v0                           #  $a1 <- multiplicador (lido do usuario)
        move $a0, $t0                           #  $a0 <- multiplicando
```

```
        jal multiplica                          #  chama procedimento

        move $t0, $v0                           #  $t0 <- LO da multiplicacao
        move $t1, $v1                           #  $t1 <- HI da multiplicacao

        imprime_string(separador2)
        imprime_string(resultado)

        imprime_string(hi)
        imprime_inteiro($t1)                    #  imprime parte HI da multiplicacao

        imprime_string(lo)
        imprime_inteiro($t0)                    #  imprime parte LO da multiplicacao

        imprime_string(separador)
#  epilogo
        #  Nao existe quadro para ser destruido

        encerra_programa(programa_OK)
#

        ###############################################################################################
############################
#  Este procedimento multiplica dois numeros com o segundo algoritimo da multiplicacao
#
#  Parametro:
#  $a0 <- multiplicando
#  $a1 <- multiplicador
#
#  Valor de retorno:
#  $v0 <- parte LO do resultado da multiplicacao
#  $v1 <- parte HI do resultado da multiplicacao
#
#  hitorico:
#  20/06/2016      Primeira versao do procedimento
#  23/06/2016        Segunda versao do procedimento
#  24/06/2016        Terceira versao do procedimento
#  28/06/2016      Quarta versao do procedimento
#

##############################
# Quadro multiplica:            #
#         | $s0  |  $sp + 8    #
#         | $s1  |  $sp + 4    #
#         | $s2  |  $sp + 0    #
##############################

multiplica:
#  prologo
        addiu $sp, $sp, -12                     #  ajusta a pilha para 3 elementos
        # Salvando elementos na pilha para restauramos seus valores antes de voltar ao procedimento chamador
        sw $s0, 8($sp)                          #  guarda registradores $s0 - $s2
```

```
        sw $s1, 4($sp)                  #
        sw $s2, 0($sp)                  #
#  corpo do procedimento
        move $s0, $a0                   #  $s0 -> multiplicando
        li $s1, ValorInicial            #  $s1 -> HI produto
        move $s2, $a1                   #  $s2 -> LO produto
        li $t5, NumBits                 #  limite (iteraçoes do laço)
        li $t6,     ValorInicial        #  indice do laco
        li $t7, ValorInicial            #  $t7 -> bit de carry

LOOP:
        slt $t0, $t6, $t5           #  i deve ser menor que 3
        beq $t0, $zero, EndLoop             #  desvia para EndLoop se rodada >= 32

        and $t0, $s2, MascaraBits   #  verifica ultimo bit do produto
        beq $t0, $zero, NaoSoma     #  se 0 desvia para NaoSoma
Soma:                                       #  ultimo bit do produto = 1
#  verifica bit de carry (overflow)
        nor $t0, $s1, $zero                 #  $t0 <- NOT $s1
                                            #  (compl. a dois - 1: 2^32 - $s1 - 1)
        sltu $t0, $t0, $s0          #  (2^32 - $s1 - 1) < $s0
        beq $t0, $zero, SemOverflow         #  se (2^32 - 1 > $s1 + $s0) vai para SemOverflow

Overflow:                               #  houve overflow
        addu $s1, $s1, $s0              #  soma, sem interrupcao, o multiplicando com a parte HI do
produto
        and $t1, $s1, MascaraBits   #  $t1 <- 1 se ultimo bit parte HI = 1

        srl $s1, $s1, Deslocamento          #  desloca HI
        ori $s1, $s1, MascaraSetaMSB        #  Seta bit mais significativo HI

        beq $t1, $zero, NaoUmHI             #  desvia para NaoUmHI se ultimo bit parte HI = 0
        j UmHI                              #  se = 1 vai para UmHI

SemOverflow:                                #  nao houve overflow
        addu $s1, $s1, $s0                  #  soma, sem interrupcao, o multiplicando com a parte HI do
produto

        andi $t0, $s1, MascaraBits          #  $t0 <- 1 se ultimo bit da parte HI = 1
        srl $s1, $s1, Deslocamento          #  desloca HI
        beq $t0, $zero, NaoUmHI             #  desvia para NaoUmHI se ultimo bit HI = 0
        j UmHI                              #  desvia para UmHI se ultimo bit HI = 1

NaoSoma:
        andi $t0, $s1, MascaraBits          #  $t0 <- 1 se verifica ultimo bit da parte HI = 1
        srl $s1, $s1, Deslocamento          #  desloca HI
        beq $t0, $zero, NaoUmHI             #  desvia para NaoUmHI se ultimo bit da parte HI = 0
UmHI:                                       #  Ultimo bit do HI = 1
        srl $s2, $s2, Deslocamento          #  Desloca LO
        ori $s2, $s2, MascaraSetaMSB        #  Seta bit mais significativo LO
        j Ok
NaoUmHI:
```

```
        srl $s2, $s2, Deslocamento          #  desloca LO

Ok:                                         #  passo i foi completo
        addi $t6, $t6, 1                #  incrementa contador i ( $t6 = $t6 + 1 )
        j LOOP

EndLoop:                                    #  Multiplicacao completa
        mthi $s1                        #  move parte HI do produto para reg hi
        mtlo $s2                        #  move parte LO do produto para reg lo

        mflo $v0                            #  $v0 <- lo
        mfhi $v1                            #  $v1 <- hi

# epilogo
        #  Restauramos valores dos elementos para voltarmos ao procedimento chamador
        lw $s0, 8($sp)                      #  restaura $s0 - $s2
        lw $s1, 4($sp)                      #
        lw $s2, 0($sp)                      #
        addiu $sp, $sp, 12                  #  restaura a pilha

        jr $ra                              #  retorna ao procedimento chamador
#
        ##############################################################################################
############################
.data
titulo:         .asciiz "SEGUNDO ALGORITIMO DA MULTIPLICAÇÃO\n\n"
separador:      .asciiz "\n---------------------------------------\n"
separador2:     .asciiz "---------------------------------------\n"
multiplicando:  .asciiz "Digite o mulltiplicando: "
multiplicador:  .asciiz "Digite o multiplicador: "
resultado:      .asciiz "Resultado:\n"
hi:             .asciiz "\nHI: "
lo:             .asciiz "\nLO: "

        ##############################################################################################
############################
```

Questão 2.

- macros.asm:

```
#  Macros e equivalências usados por todos arquivos
#########################################################
#  Diretivas .eqv
.eqv programa_OK                    0
.eqv servico_imprime_string         4
.eqv servico_imprime_PF             3
.eqv servico_le_numeroPF            7
.eqv servico_encerra_programa       17
#########################################################
```

```
#  Diretivas .macro
.macro encerra_programa(%valor_fim_programa)
	li $a0, %valor_fim_programa
	li $v0, servico_encerra_programa
	syscall
.end_macro
#
.macro imprime_string(%string_para_impressao)
	la $a0, %string_para_impressao
	li $v0, servico_imprime_string
	syscall
.end_macro
#
.macro imprime_numeroPF(%numeroPF_para_impressao)
	mov.d $f12, %numeroPF_para_impressao
	li $v0, servico_imprime_PF
	syscall
.end_macro
#
.macro le_numeroPF()
	li $v0, servico_le_numeroPF
	syscall
	#  retorna em $f0 número em ponto flutuante, precisão dupla
.end_macro
#
#####################################################
```

- *main.asm* – Vigésima iteração como condição de parada:

```
#################################################################
 .include "macros.asm"		#  Inclui arquivo com macros
#################################################################
# Diretivas .eqv
.eqv iteracoes		20	#  iteracoes das somas e subtracoes (n) do calculo do seno do angulo x
#################################################################
.text

.globl main

#  Procedimento main do programa que calculo o seno de um ângulo x
#
#  historico:
#  25/06/2016	Primeira versao do procedimento
#  26/06/2016	  Segunda versao do procedimento
#
main:
#
	#############################################################################################
############################
#  prologo
                                                # Nao precisamos armazenar endereco de retorno ou
```

```
argumentos
#  corpo do procedimento
        imprime_string(separador)
        imprime_string(titulo)
        imprime_string(angulo)
        #  Argumento para funcao SenoAngulo
        le_numeroPF()                       #  $f0 contem o numero lido (multiplicando)

        mov.d $f12, $f0                     #  $f12 <- angulo x

        #  converte -> formula: x = 2.PI.rad / 360;
        l.d $f2, dois
        mul.d $f12, $f12, $f2
        l.d $f2, pi
        mul.d $f12, $f12, $f2
        l.d $f2, maxgrau
        div.d $f12, $f12, $f2

        jal SenoAngulo                      #  chama procedimento
        #  $f0 <- seno do angulo x

        imprime_string(separador2)
        imprime_string(resultado)


        imprime_numeroPF($f0)               #  imprime o seno do angulo x

        imprime_string(separador)

#  epilogo
        #  Nao existe quadro para ser destruido

        encerra_programa(programa_OK)
#

        ##########################################################################################
############################
#  Este procedimento calcula o seno de um angulo x
#
#  Parametro:
#  $f0 <- angulo x
#
#  Valor de retorno:
#  $f0 <- seno do angulo x
#
#  hitorico:
#  25/06/2016      Primeira versao do procedimento
#  26/06/2016        Segunda versao do procedimento
#

##############################
# Quadro SenoAngulo:          #
```

```
#         | $s1 |   $sp + 4    #
#         | $s0 |   $sp + 0    #
################################

SenoAngulo:
#
        ##############################################################################################
############################
#  prologo
        addiu $sp, $sp, -8                      #  ajusta a pilha para 2 elementos
        sw $s1, 4($sp)                          #  guarda registradores $s0 - $s1
        sw $s0, 0($sp)                          #
#  corpo do procedimento
        mov.d $f0, $f12                         #  $f0 (resultado) <- x
        li $s1, 0                       #  i =  0
        li $s0, 1                       #  $s0 <- negativo (oscilador de sinal)
        mov.d $f4, $f12                         #  $f4 <- numerador
        l.d $f6, incremento                     #  $f6 <- (1)
        mov.d $f8, $f6                          #  $f8 <- denominador
        mov.d $f10, $f6                         #  complemento fatorial

LOOP:
        beq $s1, iteracoes, END_LOOP            #  salta para END_LOOP se i == 20

#  calcula (x)^expoente
        #  (x)^y ja esta calculado, apenas multiplica x * x * x, assim evita de calcular tudo novamente para
encontrar (x)^y+2
        mul.d $f4, $f4, $f12
        mul.d $f4, $f4, $f12
#  calcula fatorial
        #  y! ja esta calculado, apenas multiplica (y+2) * (y+1) * y!, assim evita de calcular tudo novamente para
encontrar (y+2)!
        add.d $f10, $f10, $f6
        mul.d $f8, $f8, $f10
        add.d $f10, $f10, $f6
        mul.d $f8, $f8, $f10

        # $f16 <- (x^exp)/(exp!)
        div.d $f16, $f4, $f8

#  verifica se deve somar ou subtrair
        beq $s0, $zero, Soma                    #  desvia para Soma se controlador de sinal = 0
Subrai:  #  controlador = 1
        sub.d $f0, $f0, $f16
        j Ok
Soma:
        add.d $f0, $f0, $f16
Ok:

#  Oscila o sinal
        beq $s0, $zero, ViraNegativo            #  se sinal era 0 (positivo), vira negativo (1)
ViraPositivo:
```

```
        li $s0, 0
        j SinalOk
ViraNegativo:
        li $s0, 1
SinalOk:

        addi $s1, $s1, 1                # i = i + 1
        j LOOP
END_LOOP:

# epilogo
        #  Restauramos valores dos elementos para voltarmos ao procedimento chamador
        lw $s1, 4($sp)                  #  restura registradores $s0 - $s1
        lw $s0, 0($sp)                  #
        addiu $sp, $sp, 8               #  restura a pilha

        jr $ra
#
        ##############################################################################################
############################
.data
titulo:         .asciiz "CALCULAR SENO DO ÂNGULO X\n\n"
separador:      .asciiz "\n---------------------------------------\n"
separador2:     .asciiz "---------------------------------------\n"
angulo:         .asciiz "Digite o valor do ângulo X: "
resultado:      .asciiz "Seno do ângulo X: "
incremento:     .double 1.0
pi:             .double 3.14159265359
dois:           .double 2.0
maxgrau:        .double 360
        ##############################################################################################
############################
```

- *main.asm* – Erro como condição de parada:

```
#################################################################
 .include "macros.asm"              #  Inclui arquivo com macros
#################################################################
.text

.globl main

#  Procedimento main do programa que calculo o seno de um ângulo x
#
#  historico:
#  25/06/2016      Primeira versao do procedimento
#  26/06/2016        Segunda versao do procedimento
#  02/07/2016        Terceira versao do procedimento
#
main:
#
```

```
        ###############################################################################################
############################
#  prologo
                                                # Nao precisamos armazenar endereco de retorno ou
argumentos
#  corpo do procedimento
        imprime_string(separador)
        imprime_string(titulo)
        imprime_string(angulo)
        #  Argumento para funcao SenoAngulo
        le_numeroPF()                           #  $f0 contem o numero lido (multiplicando)

        mov.d $f12, $f0                         #  $f12 <- angulo x
        #  converte -> formula: x = 2.PI.rad / 360;
        l.d $f2, const2
        mul.d $f12, $f12, $f2
        l.d $f2, constPi
        mul.d $f12, $f12, $f2
        l.d $f2, const360
        div.d $f12, $f12, $f2


        jal SenoAngulo                          #  chama procedimento
        #  $f0 <- seno do angulo x

        imprime_string(separador2)
        imprime_string(resultado)


        imprime_numeroPF($f0)                   #  imprime o seno do angulo x

        imprime_string(separador)

#  epilogo
        #  Nao existe quadro para ser destruido

        encerra_programa(programa_OK)
#

        ###############################################################################################
############################
#  Este procedimento calcula o seno de um angulo x
#
#  Parametro:
#  $f0 <- angulo x
#
#  Valor de retorno:
#  $f0 <- seno do angulo x
#
#  hitorico:
#  25/06/2016       Primeira versao do procedimento
#  26/06/2016          Segunda versao do procedimento
```

```
#  29/07/2016          Terceira versao do procedimento
#  02/07/2016          Quarta versao do procedimento
#
##############################
# Quadro SenoAngulo:          #
#          | $s0 |   $sp + 0    #
##############################

SenoAngulo:
#
        ##########################################################################################
############################
#  prologo
        addiu $sp, $sp, -4                      #  ajusta a pilha para 1 elemento
        sw $s0, 0($sp)                          #  guarda $s0 na pilha
#  corpo do procedimento
        li $s0, 1                       #  $s0 <- negativo (oscilador de sinal)
        l.d $f6, incremento                     #  $f6 <- (1)
        l.d $f18, erro                          #  $f18 <- erro maximo iteracoes do metodo

        mov.d $f4, $f12                         #  $f4 <- numerador
        mov.d $f8, $f6                          #  $f8 <- denominador
        mov.d $f10, $f6                         #  complemento fatorial
        mov.d $f0, $f12                         #  $f0 (resultado) <- x

LOOP:
        mov.d $f16, $f0                         #  $f16 = x_n
#  calcula (x)^expoente
        #  (x)^y ja esta calculado, apenas multiplica x * x * x, assim evita de calcular tudo novamente para
encontrar (x)^y+2
        mul.d $f4, $f4, $f12
        mul.d $f4, $f4, $f12
#  calcula fatorial
        #  y! ja esta calculado, apenas multiplica (y+2) * (y+1) * y!, assim evita de calcular tudo novamente para
encontrar (y+2)!
        add.d $f10, $f10, $f6
        mul.d $f8, $f8, $f10
        add.d $f10, $f10, $f6
        mul.d $f8, $f8, $f10

        # $f2 <- (x^exp)/(exp!)
        div.d $f2, $f4, $f8

#  verifica se deve somar ou subtrair
        beq $s0, $zero, Soma                    #  desvia para Soma se controlador de sinal = 0
Subrai:  #  controlador = 1
        sub.d $f0, $f0, $f2
        j Ok
Soma:
        add.d $f0, $f0, $f2
Ok:
```

```
#  Oscila o sinal
        beq $s0, $zero, ViraNegativo            #  se sinal era 0 (positivo), vira negativo (1)
ViraPositivo:
        li $s0, 0
        j SinalOk
ViraNegativo:
        li $s0, 1
SinalOk:

        sub.d $f16, $f0, $f16                   #  $f16 = x_n+1 - x_n
        abs.d $f16, $f16                #  $f16 = |x_n+1 - x_n|
        c.le.d $f16, $f18               #  se falso nova iteracao
        bc1f LOOP

        #  $f0 ja possui o valor de retorno
# epilogo
        #  Restauramos valores dos elementos para voltarmos ao procedimento chamador
        lw $s0, 0($sp)                          #  restaura $s0
        addiu $sp, $sp, 4               #  restura a pilha

        jr $ra
#
        ##############################################################################################
############################
.data
titulo:         .asciiz "CALCULAR SENO DO ÂNGULO X\n\n"
separador:      .asciiz "\n---------------------------------------\n"
separador2:     .asciiz "---------------------------------------\n"
angulo:         .asciiz "Digite o valor do ângulo X: "
resultado:      .asciiz "Representacao no Circulo Trigonometrico:\n  Seno do ângulo X: "
incremento:     .double 1.0
constPi: .double 3.14159265359
const2:         .double 2.0
const360:       .double 360.0
erro:           .double 1e-9
        ##############################################################################################
############################
```