PROJETO FINAL 2019: PROJETO E SIMULAÇÃO DO PROCESSADOR *"F_MIPS"* E COPROCESSADOR *"F_POLI"*.

Introdução

O propósito deste projeto é a implementação, por meio de descrições comportamentais/estruturais em VHDL, do processador ***F_MIPS***. Este processador é uma versão reduzida e bastante simplificada da família de processadores MIPS (***F_MIPS***) e de seu coprocessador numérico (***F_Poli***). O coprocessador é capaz de realizar o cálculo das funções de multiplicação (algoritmo combinatório) e raiz quadrada.

A estrutura geral dos componentes deste projeto está mostrada na figura a seguir:

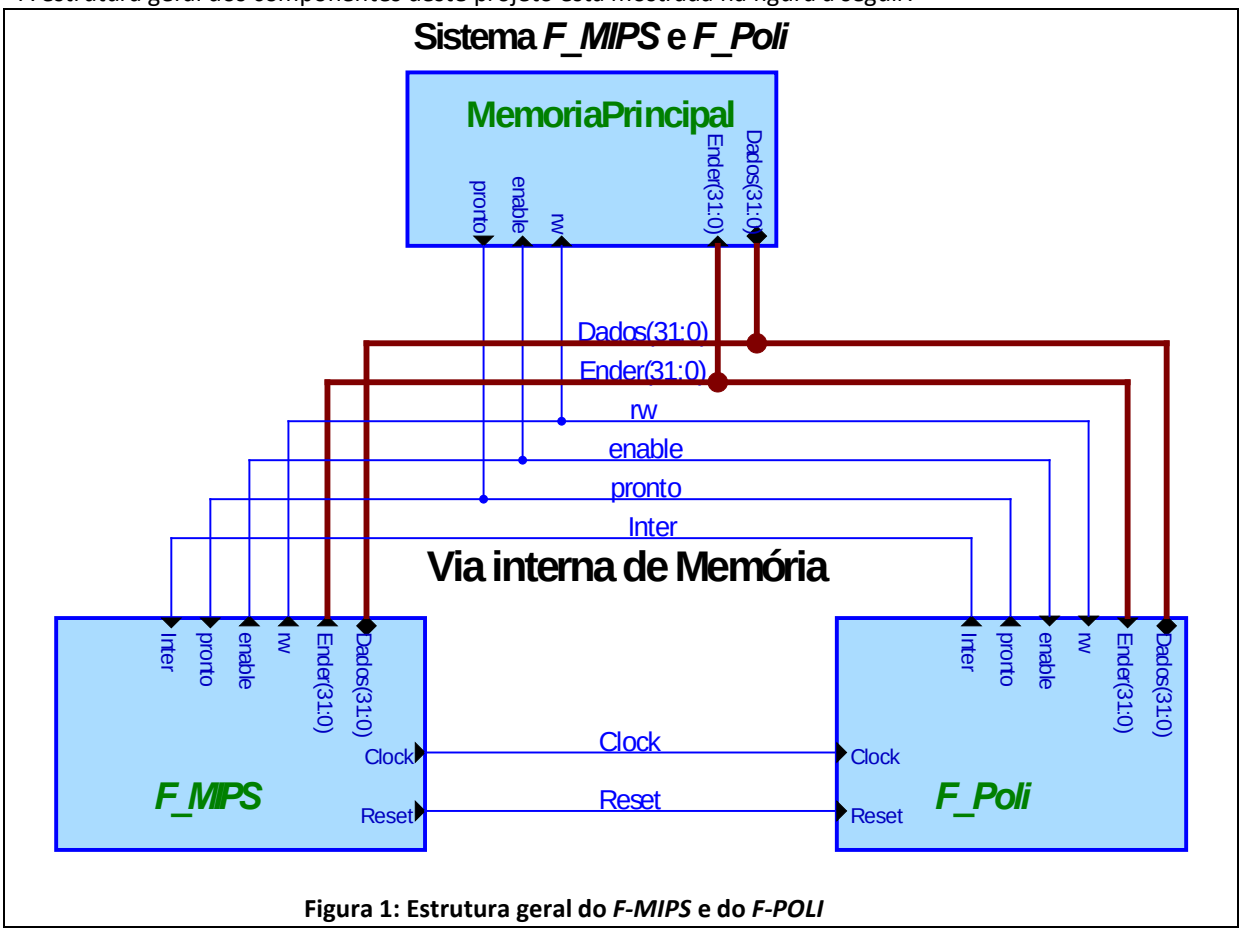


Figura 1: Estrutura geral do *F-MIPS* e do *F-POLI*

O projeto deve ser realizado em **grupos com até 6 alunos**, sendo recomendada a formação de grupos de trabalho de ao menos 5 alunos.

Além da implementação, é o objetivo deste projeto a caracterização do desempenho do processador, através da produção de relatórios detalhados de execução de trechos de programas selecionados. O projeto descrito em VHDL deve ser fiel aos tempos de resposta (atrasos) dos componentes do Fluxo de Dados (FD) e da Unidade de Controle (UC). Os alunos devem prover em seu projeto contadores e sensores para todas as métricas relevantes de desempenho.

Este documento se divide em quatro partes. Na primeira, é descrita a **Arquitetura do Conjunto de Instruções (ACI)** do ***F_MIPS***, a qual é uma versão simplificada da ACI MIPS32. Na segunda, são descritas a estrutura geral da MicroArquitetura (MA), bem como as especificações para a sua implementação. Na terceira parte são descritas as métricas, a carga de trabalho e as informações relevantes para produção do relatório de caracterização do desempenho do processador projetado.

Finalmente, na quarta parte descreve-se o coprocessador numérico que deve operar em conjunto com o ***F_MIPS***.

Arquitetura do Conjunto de Instruções (ACI).

CONJUNTO DE OPERAÇÕES:

Instruções de Acesso à Memória:

lw: Escreve o conteúdo de uma posição da Memória de Dados (MD) em um registrador.

sw: Escreve o conteúdo de um registrador em uma posição de memória.

Instruções Lógico-Aritméticas:

add: Soma dois operandos inteiros com sinal localizados em registradores

addi: Soma um valor fixo inteiro na instrução a um operando inteiro em registrador

addu: Soma dois operandos inteiros sem sinal em registradores

slt: Compara dois operandos, e assume valor "um" se um deles (sempre o mesmo) for menor que outro

slti: Compara dois operandos, um em registrador e outro imediato com sinal estendido, e assume valor "um" se o conteúdo do registrador for menor que o imediato.

sll: Deslocamento lógico à esquerda de um registrador pelo número de bits especificados no campo "shamt" da instrução.

Instruções de desvio:

beq: Compara o valor de dois operandos, e desvia o fluxo de execução para um endereço especificado na Memória de Instruções (MI) caso estes sejam iguais

bne: Compara o valor de dois operandos e desvia o fluxo de execução para um endereço especificado na MI caso estes sejam iguais.

j: Desvia o fluxo de execução para um endereço especificado na MI

jal: Guarda o endereço da próxima instrução em um registrador específico (número 31) e desvia o fluxo de execução para um endereço especificado na MI.

jr: Desvia o fluxo de execução para um endereço na MI especificado pelo conteúdo de um registrador.

Armazenamento Interno:

Operandos são referenciados de forma explícita, pela indicação de dois registradores que contém os valores dos operandos.

Operações com três (3) endereços (0 de MD) e 3 operandos.

Tipicamente designado "Registrador/Registrador" ou "Load/Store".

Codificação e Representação:

Representação de números inteiros em 32 bits (em complemento de 2)

Representação de números de ponto flutuante em 64 bits (IEEE 754)

Espaço de Endereçamento de Memória: 32 bits.

Embora o espaço lógico de memória possua 32 bits de endereço, neste projeto deve ser implementado um espaço com 16 bits de endereço, ou seja, somente 64 KBytes.

Menor unidade endereçável na Memória: 8 bits (1 byte).

Acessos à memória alinhados

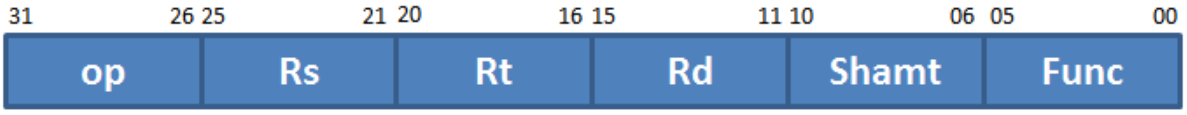
INSTRUÇÕES:

Tamanho fixo de 32 bits, com os seguintes campos:

- Código de operação (op): identifica uma instrução específica (6 bits)
- Registrador de origem (rs): primeiro operando (5 bits)
- Registrador alvo (rt): segundo operando (5 bits)
- Registrador de destino (rd): resultado de operações (5 bits)
- Magnitude de deslocamento ("shamt"): número de bits de deslocamento (8 bits)
- Código de função ("funct"): indica uma variante específica de determinada operação (6 bits)

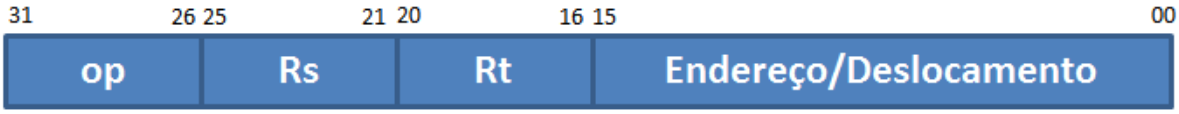
O diagrama abaixo representa a disposição dos bits numa instrução por tipo de instrução:

Instruções Tipo R (e.g., lógico-aritméticas):



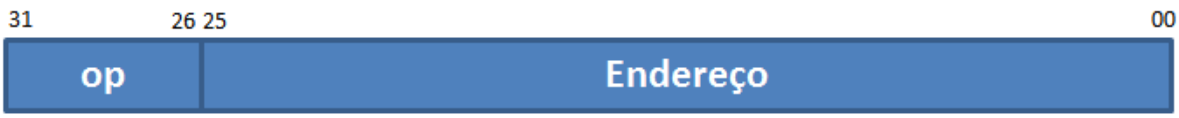
- Instrução **add**: op = 0 e funct = 32;
- Instrução **slt**: op = 0 e funct = 42;
- Instrução **jr**: op = 0 e funct = 08;
- Instrução **addu**: op = 0 e funct = 33;
- Instrução **sll**: op = 0 e funct = 00;

INSTRUÇÕES TIPO I (E.G., ACESSO À MEMÓRIA):



- Instrução **lw**: op=35;
- Instrução **sw**: op=43;
- Instrução **addi**: op=8;
- Instrução **beq**: op=4;
- Instrução **slti**: op=10;

Instruções Tipo J (e.g., salto incondicional):



- Instrução bne: op=5;
- Instrução j: op=2;

- Instrução jal: op=3;

Modos de Endereçamento:

- Imediato (16 bits);
- Indireto;
- Relativo.

Estado do processador:

32 REGISTRADORES DE PROPÓSITO GERAL INTEIROS DE 32 BITS:

Registrador	Nome	Função	Preservado
0	$zero	Constante 0	sim
1	$at	Reservado para Assembler	sim
2-3	$v0-$v1	Valores de retorno	não
4-7	$a0-$a3	Argumentos de procedimentos	não
8-15	$t0-#t7	Temporários	não
16-23	$s0-$s7	Preservados	sim
24-25	$t8-$t9	Temporários	não
26-27	$k0-$k1	Reservado para o SO	sim
28	$gp	Ponteiro Global	Sim
29	$sp	Ponteiro de Pilha	Sim
30	$fp	Ponteiro de Quadro	Sim
31	$ra	Endereço de Retorno	Sim

4 registradores de propósito específico de 32 bits:

- Contador de Instruções (CI) de 32 bits

- Registrador de Estado (RE) de 32 bits

- Registrador de Causa (RC) de 32 bits

- Registrador de Instrução em Exceção (EI) de 32 bits

Modelo de Execução

- Chamada de procedimento simples.
- Chamada de procedimento reentrante e recursivo.

Subdivisão do Espaço de Memória:

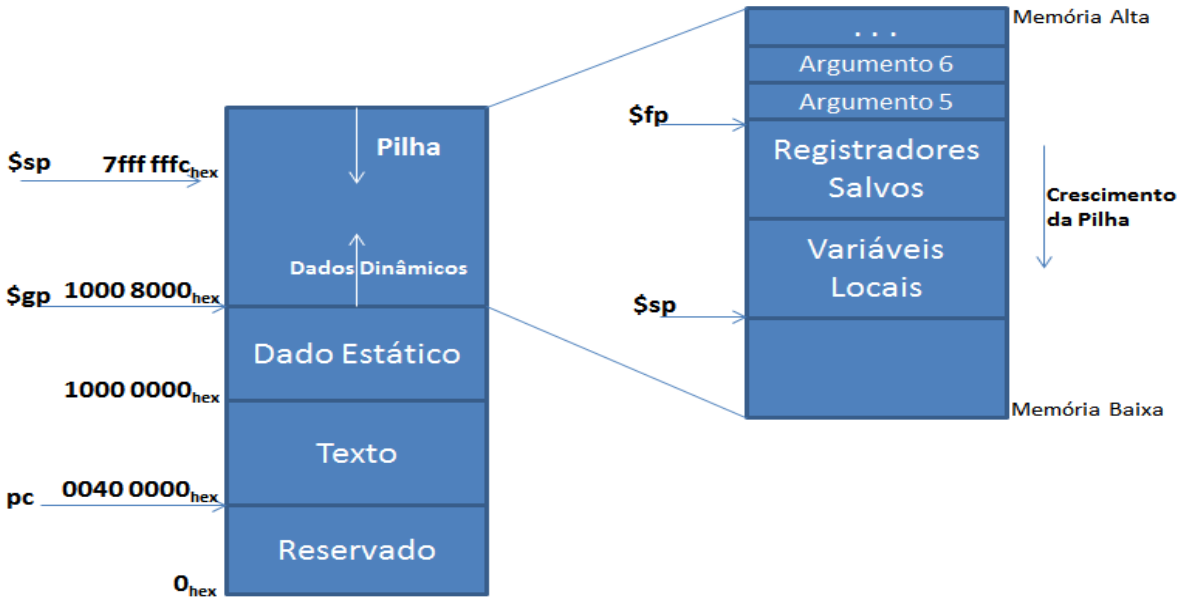


Figura 1: Estrutura de organização da memória do *F_MIPS*

Embora o mapeamento de endereços do MIPS seja o que se encontra mostrado na figura acima, neste projeto deve ser implementado o seguinte mapeamento:

pc = 0000_{hex};
Início da área de dados estáticos = 0100_{hex}
$gp = 0200_{hex}
$sp = $FFFF_{hex}$

Os sentidos de crescimento dessas áreas de memória são mantidos conforme esquema original do processador MIPS.

MICROARQUITETURA (MA)

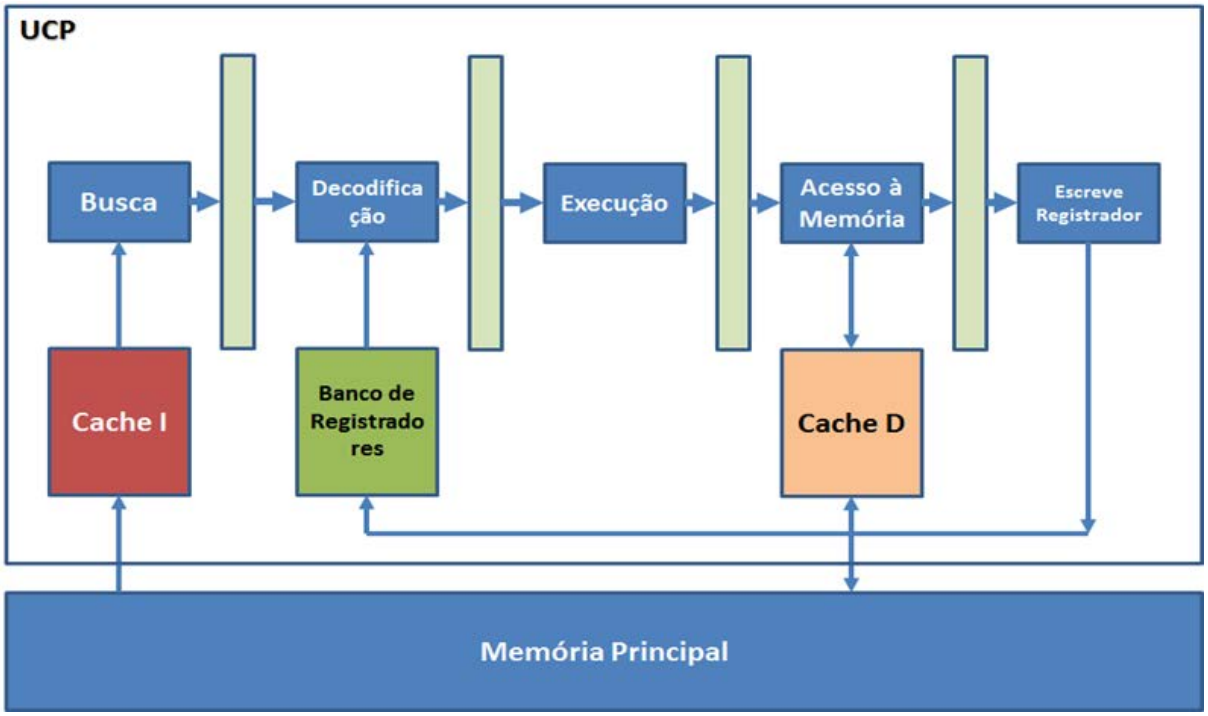
Pipeline

A MA deste processador deve ser organizada como um pipeline de 5 estágios. A descrição de cada estágio é a que segue:

- **Busca**: O conteúdo da posição de memória cujo endereço está armazenado no contador de instruções (PC) é lido e o PC é atualizado para o endereço da próxima instrução (PC+4).
- **Decodificação**: Parte do conteúdo da instrução lida é enviado para Unidade de Controle (UC) do processador e parte para o banco de registradores, a fim de permitir que a UC determine quais são as operações a serem executadas e os operandos especificados pela instrução sejam lidos do banco de registradores.

- **Execução**: As operações especificadas pela instrução são executadas sobre os operandos através da alimentação destes em uma ULA controlada pela UC. No caso da instrução lw (Load) o endereço efetivo é calculado neste estágio.
- **Acesso à Memória**: Caso a instrução processada exija acesso à memória (lw ou sw), este é executado neste estágio (leitura no caso de lw e escrita no caso de sw). Se a instrução sendo executada não necessita acesso à memória este estágio não realiza nenhuma operação. No caso da instrução de desvio, se ele tiver que ser realizado é neste estágio que o novo endereço é carregado no contador de instruções (pc).
- **Escreve de Volta**: O resultado da Execução ou do Acesso à Memória (lw) é armazenado no banco de registradores neste estágio.

O estágio de execução deve ser implementado com uma unidade lógica e aritmética tradicional (ULA). Mais detalhes sobre a operação do pipeline tradicional do processador MIPS podem ser encontrados no livro texto da disciplina, em seu capítulo 4.
O esquemático abaixo representa as principais interconexões entre cada estágio e a hierarquia de memória na MA do ***F_MIPS***.



Este pipeline deve observar as seguintes restrições e características operacionais:

- O Tempo de Operação (TO) nos estágios Decodificação, Execução e Escreve de Volta é sempre igual a 1 ciclo de relógio (10 ns).
- O TO dos estágios Busca e Acesso à Memória dependerá da latência da hierarquia de memória tratada a seguir.
- O Banco de Registradores (BR) deve ter uma latência de leitura e escrita igual à metade (5 n s) de um ciclo de relógio ou tempo de operação dos estágios do pipeline cada (i.e. deve ser possível **ler e escrever** no banco de registradores em um único ciclo).

- O BR opera de modo a garantir a prioridade da escrita sobre a leitura, de modo que uma requisição de escrita sempre "bloqueie" a requisição de leitura até que a primeira seja finalizada.
- O cálculo de endereço nas instruções de desvio condicional é realizado no estágio de Execução. O endereço efetivo é então propagado ao Contador de Instruções (pc), no estágio de Acesso à Memória.
- O pipeline deve possuir uma "Hazard Detection Unit", que permita ao mesmo detectar dependências de Dados ou de Controle no fluxo de instruções executadas e tomar as ações apropriadas. Entre as ações apropriadas estão a inserção de bolhas de pipeline ou a propagação de valores de registradores através da "leitura antecipada" "forwarding".

Hierarquia de Memória

- Memória Principal (MP)
  - Possui somente um módulo de memória contendo todos os blocos endereçáveis.
  - A MP possui 2^{30} palavras (no caso deste projeto a MP deverá possuir 2^{14} palavras) de 32 bits (4 bytes).
  - Latências:
    - 40 ns por ciclo de acesso (leitura ou escrita).
- **Cache I: Instruções**
  - Possui 16 KBytes ou 4096 palavras de 32 bits
  - 16 palavras por bloco
  - Mapeamento direto
  - Tempo de acesso = 5 ns (para leitura ou escrita)
- Cache D: Dados
  - 16KBytes ou 4096 palavras de 32 bits
  - 16 palavras por bloco
  - Algoritmo de mapeamento de bloco: Associativo por conjunto de 2 blocos
  - Algoritmo de substituição de bloco: LRU
  - Algoritmo de escrita de bloco: Write Back com um buffer de escrita de uma palavra
  - Tempo de acesso = 5 ns (para leitura ou escrita)

O esquemático abaixo representa a hierarquia de memória na MA especificada aqui.

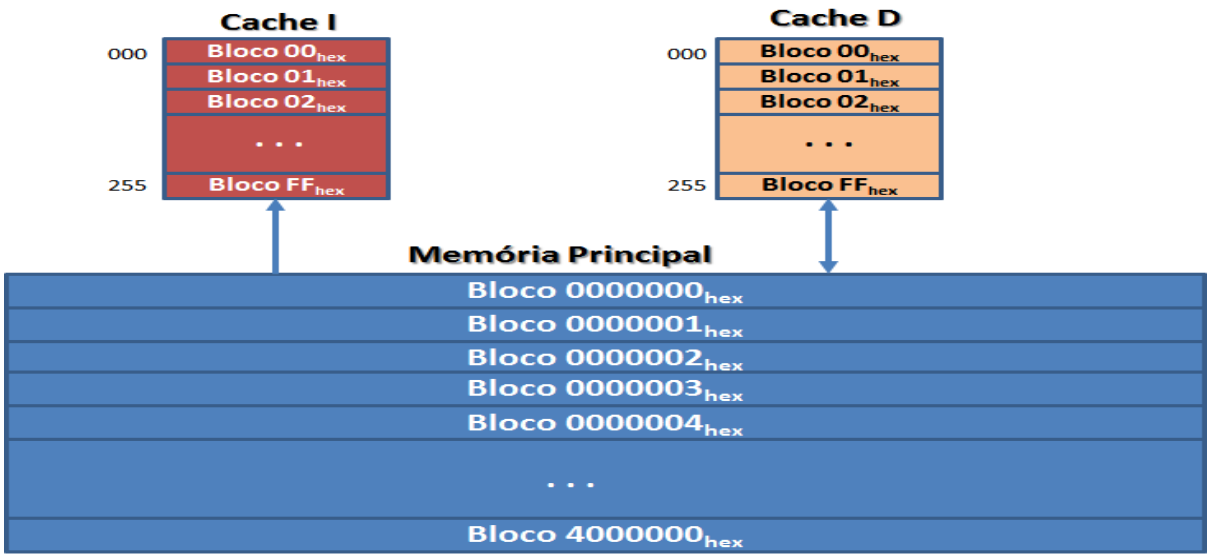


Figura 3: Estrutura da Hierarquia de Memória do *FemtoMIPS*

Interrupções

O tratamento de interrupções é uma função primordial do processador. O aluno deverá recorrer a descrição deste projeto no livro texto para definir e implementar o procedimento de interrupção.

ESTADO INICIAL

- Sempre que for "ligado" ou logo após um "reset" (que deve ser um sinal assíncrono externo, também modelado na descrição em VHDL), o processador deve retornar a um estado conhecido, especificado abaixo.
    - O pc (contador de instruções) deve conter o valor 0x0000
    - Os registradores 0 a 27 do BR (Banco de registradores) devem conter o valor 0x0000
    - Registrador 28 ($gp) deve conter o valor 0x0200
    - Registrador 29 ($sp) deve conter o valor 0xffff
    - Registrador 30 ($fp) deve conter o valor 0x7ffff
    - Registrador 31 ($ra) deve conter o valor 0x0000
    - Todos os caches devem ter os seus conteúdos igualados à zero.
    - A memória principal deverá ser carregada com o código do programa de teste.

Detalhes para codificação VHDL

A Memória Principal (MP) deve ser implementada como um arquivo texto (com a extensão .txt) de múltiplas duplas de linhas, sendo cada dupla apresentada no seguinte formato:

- <Endereço inicial>|<espaço>|<número de palavras a serem lidas>|<espaço>|comentário

- <Palavras a serem lidas separadas por espaço>|<espaço>|<comentários>

Os quatro campos de cada dupla de linha devem ser separados por pelo menos um espaço e têm o seguinte formato:

- O campo <endereço inicial> deverá conter uma cadeia de caracteres ASCII que representa um **endereço em hexadecimal** com o número de caracteres correspondentes ao número de bits de endereçamento da memória sendo carregada indicando o endereço a partir do qual as palavras devem ser carregadas.
- O campo <número de palavras> é um campo com o **número decimal inteiro** de palavras a serem lidas na próxima linha do arquivo.
- O campo <Palavras> também deverá conter uma cadeia de caracteres ASCII que representa um **valor hexadecimal** com o número de caracteres correspondentes ao número de bits da palavra de conteúdo da memória sendo carregada. Cada palavra nessa linha deve ser separada por um <espaço>.
- O campo <comentário> pode conter uma cadeia de caracteres ASCII alfanuméricos.

Este arquivo texto é esparso: apenas os endereços cujos valores são diferentes de zero devem estar representados por duplas de linhas individuais explicitando tais valores. Um conjunto de linhas que explicita o conteúdo da MP no início da operação do processador (i.e. logo após este ser ligado) deve fazer parte do arquivo texto em seu "estado inicial", o mesmo ao qual o processador deve retornar após um "reset".
Finalmente, é importante ressaltar que não há necessidade de qualquer ordenação no arquivo texto que implementa esta MP, dado que a localização da mesma se dará diretamente.

Um exemplo deste arquivo texto é o que segue:

00 4 – 4 palavras de 16 bits a partir do endereço 00, onde o número de bits de endereço da memória é 8 bits

A023 819C FF87 0034 – Palavras de 16 bits a serem carregadas a partir do endereço 0

0F 2 – 2 palavras de 16 bits a partir do endereço 0F

AB65 FE00 – duas palavras de 16 bits sendo carregadas a partir do endereço 0F

Caracterização

MÉTRICAS

Para a caracterização de desempenho do processador, deve-se coletar dados relevantes durante a execução de programas específicos. A coleta de dados deverá ser executada através da leitura de "sondas de desempenho", que assumem a forma de registradores de propósito especial que deverão fazer parte da descrição comportamental do processador codificada em VHDL. As "sondas de desempenho" devem ser as que seguem.

- Número de ciclos transcorridos.
- Número de instruções executadas.

- Cache "hit ratio" para os caches L1I e L1D.
- Número de acessos à memória (Cache ou Principal).
- Tempo de execução do programa de teste.
- Frequência de ocorrência das instruções na execução do programa de teste.
- Número médio de ciclos por instrução do programa de teste.

Cargas de Trabalho

PROGRAMAS SELECIONADOS

O programa utilizado para o teste do processador ***PicoMips*** é o programa SORT já desenvolvido pelos alunos numa tarefa anterior.
Um segundo programa utilizado para teste do processador é aquele que realiza uma multiplicação de matrizes cujos elementos são números inteiros com sinal.

CONJUNTOS DE DADOS

Os conjuntos de dados serão compostos por estruturas de dados (vetores e matrizes) cujos valores serão gerados aleatoriamente. Cada conjunto de dados será gerado com tamanhos variáveis.

SESSÃO DE CARACTERIZAÇÃO DO PROCESSADOR NO RELATÓRIO FINAL

Além do relatório de documentação do projeto que vem sendo construído ao longo da execução do projeto, os alunos devem incluir uma sessão de caracterização do processador projetado.
A caracterização será realizada pela execução da carga de trabalho, tendo estas como fator o tamanho do conjunto de dados e como níveis cada um dos tamanhos selecionados. O relatório de caracterização deverá então conter as seguintes informações.

- Caracterização de desempenho do processador, com ao menos os seguintes gráficos e/ou tabelas.
  - CPI x Carga de Trabalho X Tamanho do Conjunto de Dados
  - "Hit Ratio x Carga de Trabalho X Tamanho do Conjunto de Dados
  - "Acesso à Memória" x Carga de Trabalho X Tamanho do Conjunto de Dados
- Uma análise crítica do comportamento do processador na execução de várias cargas de trabalho e vários tamanhos dos conjuntos de dados.

Todos os dados devem ser adequadamente tratados do ponto de vista estatístico, incluindo o número de execuções, a média e o coeficiente de variação. O relatório e o código VHDL devem ser submetidos pelo site do TIDIA-Ae na ferramenta "Atividades".

Pergunta-se:

a) Faça o projeto lógico do ***F_MIPS*** incluindo o seu fluxo de dados e a Unidade de controle seguindo a metodologia de projeto estruturado com pelo menos 2 níveis diferentes de descrição seguindo as seguintes etapas:
   i. Faça o diagrama ASMD mnemônico do algoritmo de execução de instruções do ***F_MIPS***. Este diagrama ASMD deve seguir o conceito de máquina de Mealy **(1ª entrega)**;
   ii. Determine o número mínimo de registradores, unidades funcionais e barramentos que são necessários para sintetizar o fluxo de dados **(1ª entrega);**
   iii. Projeto o fluxo de dados definindo os seguintes itens: **(2ª entrega)**
      i. Estrutura de componentes, unidades funcionais e barramentos do fluxo de dados. Identifique todos os componentes do fluxo de dados anotando os valores de atrasos necessários para dimensionamento do circuito;
      ii. Identifique todos os sinais de controle e de condição que devem existir interconectando o fluxo de dado e a unidade de controle;
      iii. Calcule o ciclo do fluxo de dados projetado e identifique os sinais de relógio que serão necessários para a operação correta deste fluxo de dados;
      iv. Traduza o diagrama ASMD mnemônico que deu origem ao fluxo de dados projetado para uma versão onde em cada estado são acionados somente sinais de controle e no elemento de decisão sejam testado somente sinais de entrada e/ou de condição;
      v. Descreva em VHDL a estrutura do fluxo de dados projetado e codifique o diagrama ASMD, na versão de acionamento de sinais, e simule o sistema para verificar o seu correto funcionamento;
   iv. Dentre as técnicas de projeto da unidade de controle que utilizam memória ROM escolha aquela que resulta no menor número de bits da memória de controle e obtenha o circuito final do ***F_MIPS*** desenhando o diagrama lógico do circuito da unidade de controle; **(3ª entrega)**
   v. Descreva em VHDL a estrutura do circuito projetado da unidade de controle e interligue-o com o circuito do fluxo de dados. Simule o circuito resultante para verificar o seu correto funcionamento; **(3ª entrega)**
   vi. Determine o ciclo de máquina do ***F_MIPS*** dando o seu valor em unidades de tempo. **(3ª entrega)**

b) Execute o programa SORT e o de Multiplicação de Matrizes no ***F_MIPS*** para medir o tempo de execução das instruções projetadas. Essa medida deve ser feita em unidades de ciclo de relógio e convertidas em unidades de tempo usando o valor calculado do ciclo de relógio obtido no item anterior. Meça também o tempo de latência do sistema para responder a um pedido de interrupção (tempo entre o acionamento do sinal **Inter** até o início da execução da primeira instrução do programa de tratamento).

Faça uma tabela com as seguintes informações: **(3ª entrega)**

Instrução	Frequência de execução de instruções (SORT)	Número de instruções executadas (SORT)	Frequência de execução de instruções (Mult_Mat)	Número de instruções executadas (Mult_Mat)	Número de ciclos para executar a instrução	Tempo (ns)
Add						
Addi						
Sub						
Bne						
Slt						
J						
Jal						
Jr						
Lw						
Sw						
Iret						
Latência de Interrupção	-----	--------	--------	------	------	
CPI (número médio de ciclos por instrução)						

Coprocessador Numérico: Raiz Quadrada

MOTIVAÇÃO

No processamento gráfico uma série de operações envolvem a utilização de relações matemáticas da geometria analítica. Operações como cálculo de distância entre pontos e o cálculo de outras grandezas vetoriais são muito comuns. Nesse tipo de operação matemática é frequente o uso dos operadores de multiplicação, divisão, potenciação, potência ao quadrado e extração da raiz quadrada de números inteiros, positivos ou negativos.
Como é frequente o uso desses operadores, é importante calcula-los com rapidez. Por este motivo, os processadores de pequeno porte fazem uso de coprocessadores destinados ao cálculo específico dessas operações.
Neste projeto define-se um coprocessador, chamado ***F_POLI***, capaz de executar as operações de multiplicação (algoritmo combinatório) e extração de raiz quadrada de números inteiros representados em ponto fixo em complemento de 2.

SISTEMA DE NUMERAÇÃO EM PONTO FIXO COM 16 BITS

O ***F_POLI*** aceita como entrada números inteiros de 16 bits em representação ponderada de ponto fixo. Nesse sistema os números possuem 8 bits para a parte inteira e 8 bits para a parte fracionária. Nesta representação um número ***X*** é representado da seguinte forma:

$$x_7\, x_6\, x_5\, x_4\, x_3\, x_2\, x_1\, x_0\, .\, x_{-1}\, x_{-2}\, x_{-3}\, x_{-4}\, x_{-5}\, x_{-6}\, x_{-7}\, x_{-8}$$

cujo valor é $\mathbf{X/2^8}$, onde **X** é o número inteiro representado em 16 bits pela mesma sequência de dígitos ($x_7 \ldots x_1\, x_0\, x_{-1} \ldots x_{-8}$), ou seja, sem considerar o ponto decimal.
O maior e o menor valor representável neste sistema de numeração são determinados considerando-se o menor e o maior número inteiro representável com 16 bits em complemento de dois, ou seja: $x_{min} = -2^{15}\ e\ x_{max} = 2^{15} - 1$.
Para obter os valores mínimo e máximo dessa representação em ponto fixo com 16 bits (8 bits parte inteira e 8 bits parte fracionária) deve-se multiplicar cada um desses valores por $\mathbf{2^{-8}}$:

$$x_{max} * 2^{-8} = \frac{(2^{15}-1)}{2^8} = \frac{37767}{256} = 127.99609375\ ,\ e$$

$$x_{min} * 2^{-8} = \frac{-2^{15}}{2^8} = \frac{-37768}{256} = -128.00000000$$

Portanto os valores de **X** que podem ser representados neste sistema encontram-se entre os seguintes valores extremos:

$$-128 \leq X \leq 127.99609375$$

A menor unidade numérica representável neste sistema de numeração é:

$$ulp = 2^{-8} = \frac{1}{256} = 0.00390625.$$

Isto implica que os números que possuem representação exata neste esquema estão espaçados em intervalos iguais a uma ***ulp***. Os demais números fracionários representáveis apresentam erro de no máximo $ulp/2$, ou seja,

$$erro_{max} = \frac{\pm 0.00390625}{2} = \pm\, 0.00195312\ .$$

Por exemplo, vejamos como alguns números fracionários são representados neste sistema de numeração:

Valor Decimal	Valor Inteiro (sem ponto) (X)	Parte Inteira ($x_7 \ldots x_0$)	Parte Fracionária ($x_{-1} \ldots x_{-8}$)
25,0	int (25 * 256) = 6400	0001 1001	0000 0000
3,5	int (3,5 * 256) = 896	0000 0011	1000 0000
12,0625	int (12,0625 * 256) = 3088	0000 1100	0001 0000

237,3	**int (237,3 * 256) = 60748**	**1110 1101**	**0100 1100**
49,0	**int (49 * 256) = 12544**	**0011 0001**	**0000 0000**
135,3257	**int (135,3257 * 256) = 34643**	**1000 0111**	**0101 0011**
1,25	**int (1,25 * 256) = 320**	**0000 0001**	**0100 0000**
4,75	**int (4,75 * 256) = 1216**	**0000 0100**	**1100 0000**
4,375	**int (4,375 * 256) = 1120**	**0000 0100**	**0110 0000**

As operações aritméticas realizadas neste sistema de numeração são feitas com os valores inteiros de 16 bits. Dependendo da operação realizada alguma correção deve ser feita para se determinar a representação correta do resultado em ponto fixo.
Por exemplo, se for feita a soma de (**3,5 + 1,25 = 4,75**) a seguinte operação binária deve ser realizada:

0000 0011. 1000 0000 **896_{10}** **$3{,}50_{10}$**
0000 0001. 0100 0000 **+** **320_{10}** **+** **1.25_{10} +**

0000 0100. 1100 0000 **1216_{10}** **$4{,}75_{10}$**

Neste caso assim como no da subtração não é necessário se fazer nenhuma correção, porém no caso da multiplicação deve-se fazer um ajuste para obter o resultado correto.
Na seguinte multiplicação (**3,5*1,25 = 4,375)** deve-se corrigir o resultado da multiplicação inteira dividindo-o por $\mathbf{2^8 = 256}$ (que é equivalente a deslocar à direita o resultado de 8 posições) para retornar à representação com 8 bits na parte fracionária e 8 bits na parte inteira:

0000 0011. 1000 0000 **896_{10}**
0000 0001. 0100 0000 ***** **320_{10}** *****

0000 0100. 0110 0000 **((286720_{10})/256)= 1120**

Neste caso, como estamos trabalhando com inteiros de 16 bits é necessário obter o resultado da multiplicação com 32 bits se não pode ocorrer overflow e se chegar a um resultado errado.

FUNÇÕES DO COPROCESSADOR *F_POLI*

O coprocessador ***F-POLI*** deve realizar as seguintes operações aritméticas:

Nome	Código Rcmd(1:0)	Operação
Nop	**00**	**Nenhuma operação**
Multiplicação	**01**	**Rst = Ra * Rb ;**
Raiz Quadrada	**11**	**Rst =** $\sqrt{Ra}$;

onde **Rcmd** é o registrador de comando, **Ra** e **Rb** são os registradores que recebem os operandos de entrada e **Rst** é o registrador que contém o resultado da operação realizada. Os operandos de entrada são representados em ponto fixo com 16 bits, sendo 8 para a parte inteira e 8 para a parte fracionária.
Os registradores do coprocessador ***F_POLI*** são 5, a saber:

Registrador	Função	Tamanho / Endereço
Ra(31:0)	Recebe o 1º operando	32 bits / x"000000F8"
Rb(31:0)	Recebe o 2º operando	32 bits / x"000000F9"
Rst(31:0)	Recebe o resultado	32 bits / x"000000FA"
Rcmd(1:0)	Recebe o código da função	2 bits / x"000000FB"
Re(3:0)	Possui o estado do coprocessador	4 bits / x"000000FC"

O significado dos bits de estado do coprocessador encontra-se definido na tabela a seguir:

Registrador de Estado	Significado
Re(0)	Re(0) = '0' se o co processador está Livre; Re(0) = '1' se Ocupado
Re(1)	Se Rst = 0 então Re(1) = '1' se não Re(1) = '0'.
Re(2)	Se deu erro na operação (na multiplicação ocorrer overflow e na raiz quadrada o operando for negativo) então Re(1) = '1' se não Re(1) = '0'.
Re(3)	Se Re(3) = '1' o coprocessador está pedindo interrupção, se Re(3) = '0' o coprocessador não está pedindo interrupção.

MODO DE OPERAÇÃO

O coprocessador possui dois modos de operação: modo de espera e modo de execução. Quando ele inicia a sua operação ele entra no modo de espera. Nesse modo, o coprocessador fica esperando receber uma instrução de escrita ou de leitura em um de seus registradores. Nas instruções de escrita em **Ra** e **Rb**, o coprocessador simplesmente garante o recebimento dos dados enviados no respectivo registrador. Já no caso da escrita no registrador de comando **Rcmd** ele interpreta que uma ordem de execução foi emitida pelo processador. Neste caso, ele faz o bit **Re(0) = '1'** sinalizando que se encontra ocupado entrando no modo execução. Inicialmente ele faz a decodificação da operação prosseguindo com a execução da função solicitada.
Ao terminar o cálculo, ele carrega o resultado no registrador **Rst**, atualiza os bits **Re(2)** e **Re(3)** com os valores associados ao resultado obtido e sinaliza um pedido de interrupção ao processador fazendo o bit **Re(3) = '1',** retornando ao modo de espera.
A partir deste momento o coprocessador fica à espera de uma instrução de leitura no registrador **Rst**. Quando esta instrução tiver sido executada o coprocessador coloca o conteúdo do registrador **Rst** na via de dados permanecendo no modo de espera no estado livre pronto para receber um novo comando de execução.

CALCULANDO A RAIZ QUADRADA

O cálculo da raiz quadrada de um número deve ser feito usando-se o método de iteração de Newton-Raphson. Este método é utilizado para determinar a raiz de uma dada equação. Se essa equação for escolhida convenientemente ela pode ser utilizada para realizar o cálculo de determinadas funções.
Dada uma função f (x) pode-se calcular o valor de x tal que f (x) = 0 usando o método de Newton-Raphson que estabelece um processo iterativo partindo de um valor inicial $\mathbf{x_0}$ e usando a seguinte formula de iteração:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$, onde **f ' (x_i)** é a derivada de **f (x)** em relação à **x** calculada no ponto $\mathbf{x_i}$.

Este processo iterativo continua até que a diferença entre $\mathbf{x_{i+1}}$ e $\mathbf{x_i}$ seja menor ou igual a uma determinada precisão ∈. Neste caso $\mathbf{x_{i+1}}$ será o valor da raiz da equação **f (x) = 0**.

Para o cálculo da raiz quadrada escolhemos a função $f(x) = \frac{1}{x^2} - X$, quando se deseja calcular a raiz quadrada de ***X***. Pode-se observar que a raiz da equação escolhida é $x = \frac{1}{\sqrt{X}}$ **.** Obtendo a raiz dessa equação e multiplicando-se esse valor novamente por ***X*** chega-se ao valor de $\sqrt{X} = x_{i+1} * X$.
A função de recorrência da equação escolhida para operação do método Newton-Raphson é a seguinte:

$$x_{i+1} = \left(\frac{x_i}{2}\right) * (3 - X * {x_i}^2)$$

Para operar este processo iterativo deve-se iniciar com um valor $\mathbf{x_0}$ e uma precisão desejada ∈**.** O processo iterativo se encerra quando $x_{i+1} - x_i \leq \in$. Esta equação envolve a execução de 3 multiplicações, uma subtração e um deslocamento à direita de uma posição (divisão por 2), além das operações necessárias para se fazer a comparação de convergência do método.
Ao se obter o valor da raiz dessa equação deve-se ainda realizar mais uma multiplicação pelo valor de **X** para se chegar ao valor da **raiz quadrada de X.**

Pergunta-se:

a) Determine o algoritmo para se calcular o valor das funções executadas pelo coprocessador: **multiplicação (pesquise um algoritmo que possa ser sintetizado por um circuito combinatório)** e a da **raiz quadrada usando o método de Newton-Raphson**; **(1ª entrega)**

b) Execute manualmente o seu algoritmo para o caso da $\sqrt{5}$. Mostre os valores intermediários de seu cálculo usando a tabela a seguir: **X = 5** , $\in = 0.01\ (1\%)$**. (1ª entrega). Utilize a semente indicada na tabela.**

Iteração	x_i	$x_i/2$	x_i^2	$x_i^2 * X$	$3 - x_i^2 * X$	x_{i+1}
0	0.14453125					
	37					
1						
2						
3						
4						
5						
6						
7						
8						
9						
$\sqrt{5} =$ 2.2360	$x_{i+1} * X =$					

c) Escreva o algoritmo (ASMD) de controle do coprocessador que incorpora os seus dois modos de operação: espera e execução. **(1ª entrega)**

d) Escreva os algoritmos de cálculo das operações do coprocessador quando ele se encontra no modo de execução (multiplicação combinatória e raiz quadrada). Determine os ASMD's associados a esses algoritmos usando escalonamento **ASAP**. A operação de multiplicação deve ser sintetizada por um circuito interno do ***F_POLI*** usando somente deslocadores e somadores/Subtratores ou portas lógicas. Para cada uma das funções realizadas determine o número de unidades funcionais e o número de estados necessários; **(1ª entrega)**

e) Projete o fluxo de dados do coprocessador lembrando que os valores dos operandos de entrada devem ser carregados pelo ***F_MIPS*** na memória interna do ***F_POLI*** antes de ser dado o comando de cálculo da função solicitada. O programa no ***F_MIPS*** permanecerá aguardando um acionamento do **sinal de interrupção** do processador sinalizando que o resultado está pronto. O procedimento de interrupção do processador aciona uma rotina especial (que se encontra no endereço x"000000F0" de memória) de tratamento de interrupções que é responsável pela leitura do valor calculado no coprocessador através da execução de uma **instrução de leitura de dados (Lw)** de dados. Os valores calculados deverão ser armazenados nos registradores internos do processador. Determine o valor mínimo do ciclo do fluxo de dados, os sinais de relógio necessários à sua operação e o diagrama ASM que especifica a sua

unidade de controle. Descreva o fluxo de dados em VHDL e simule-o para verificar o seu correto funcionamento. **(2ª entrega)**

f) Baseado nos resultados do item anterior projete a unidade de controle do coprocessador. Na síntese da unidade de controle utilize um dos métodos de lógica padronizada baseado em EPROM. Escolha o método de endereçamento (trajetória ou par qualificador-estado) que utiliza a memória com o menor número de bits úteis. Determine o valor mínimo do ciclo de máquina (**Tciclom**) para que a unidade de controle e o fluxo de dados possam operar corretamente. **(3ª entrega)**

g) Integre o fluxo de dados e a unidade de controle e simule o sistema para verificar o seu comportamento. Meça os tempos de execução das operações do coprocessador completando a seguinte tabela: **(3ª entrega)**

Função	No. De Ciclos	Tempo de Execução (ns)
Multiplicação		
Raiz Quadrada		

h) **(3ª entrega)** escreva um programa no ***F_MIPS*** para calcular a distância geométrica entre dois pontos. Execute o programa que calcula a distância entre dois pontos no plano cartesiano para diversos valores de coordenadas. Esse programa deve utilizar as funções implementadas pelo coprocessador. Meça o número de instruções executadas (**Ninst**) numa dada rodada do programa. Determine o número de ciclos (**Nciclos**) de máquina e o tempo de execução (Te). Com esses dados calcule:

   i. O tempo médio de execução das instruções desse programa $\boldsymbol{Tinst_{medio}} = \frac{\boldsymbol{Te}}{\boldsymbol{Ninst}}$,

   ii. O número médio de ciclos por instrução (CPI) = $CPI = \frac{Nciclos}{Ninst}$,

Usando o valor do ciclo de máquina (**Tciclom**) obtido anteriormente, calcule o valor do seguinte produto: $\boldsymbol{Ninst} * \boldsymbol{CPI} * \boldsymbol{Tciclom}$. Compare o valor calculado com o tempo de execução medido do programa executado.

ANEXO I: ATRASO DE COMPONENTES

Os componentes utilizados no projeto devem ter o seu atraso retirado da tabela abaixo:

Tipo de Componente	Tipo do atraso	Valor do atraso
Registrador ou Flip-Flop	Tsetup	**0.25 ns**
	Thold	**0.25 ns**
	Tpropogação	**1.0 ns**
Multiplexador	Tseleção	**0.5 ns**
	Tdado	**0.25ns**
ULA	Tsoma	**1.0 ns**
	Tsubtração	**1.25 ns**
Registrador de deslocamento de número variável de bits	Tsetup	**0.25 ns**
	Tcarga	**1 ns**
	Tdeslocamento	**0.5 ns**
	Thold	**500 ps**
Atraso de porta lógica	Tpropagação	**0.25 ns**
Deslocadorcombinatório número fixo de bits	Tpropagação	**0 ns**
Register File	Tleitura	**4 ns**
	Tescrita	**4 ns**
RAM	Tleitura	**5 ns**
	Tescrita	**5 ns**
ROM	Tleitura	**5 ns**
Decodificador	Tpropagação	**0.5 ns**
Contador	Tcarga	**1 ns**
	Tsetup	**0.25 ns**
	Thold	**0.25 ns**
	Tcontagem	**1 ns**
Porta Terceiro estado	Tenable	**0.5 ns**
	Tpropagação	**0.25 ns**

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Deschamps, Jean-Pierre, G.J.A. Bioul and G. D. Sutter; "Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems, Willey-Interscience, 2006.
[2] Gajski, Daniel, "Principles of Digital Design", Prentice-Hall, 1997.
[3] Ruggiero, W; "Notas de Aula de PCS2307", EPUSP, 2011.
[4] MIPS