PROJETO FINAL: PROJETO E SIMULAÇÃO DO PROCESSADOR *"T_FIVE"*

Introdução

O propósito deste projeto é a implementação, por meio de descrições comportamentais/estruturais em VHDL, do processador ***T_FIVE (Tiny RISK V)***. Este processador é uma versão reduzida e bastante simplificada da família de processadores ***RISK-V*** (***RISK-V***). Esta implementação envolverá duas organizações diferentes. A primeira terá a característica de um processador ***serial Multi-ciclo (T-FIVE-MC)*** e a segunda terá a característica de um processador ***paralelo em Pipeline (T-FIVE-Pipe)***. Estas duas organizações serão avaliadas pelos mesmos programas e os alunos terão que coletar os indicadores de desempenho das duas implementações com o objetivo de compará-las.

A comparação será feita pela caracterização do desempenho das duas organizações 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) fornecidos na tabela de atrasos ao final deste texto. Os alunos devem prover em seu projeto contadores e sensores para todas as métricas relevantes de desempenho.

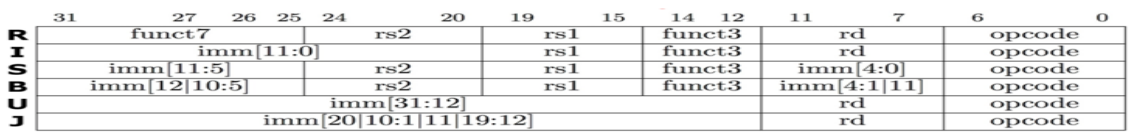
Este documento se divide em três partes. Na primeira, é descrita a **Arquitetura do Conjunto de Instruções (ACI)** do ***T_FIVE***, a qual é uma versão simplificada da ACI ***RISK-V 32*** bits. Na segunda, são descritas a estrutura geral das duas **MicroArquiteturas (MA)**, bem como as especificações para as suas implementações em VHDL. 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.

T-FIVE: Arquitetura do Conjunto de Instruções (ACI)

CONJUNTO DE INSTRUÇÕES DO *T-FIVE*:

O conjunto de instruções do ***T-FIVE*** é um subconjunto das instruções do ***RISK V 32I***. As instruções foram escolhidas de forma a tornar viável a execução dos programas de testes planejados para esta tarefa.

Existem diversos tipos e formatos de instruções nessa arquitetura os quais são mostrados a seguir:

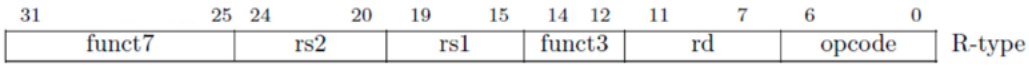


	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
B	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
J	imm[20|10:1|11|19:12]					rd	opcode

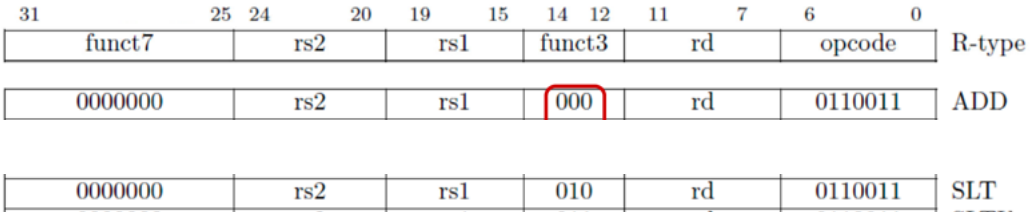
```
R: sll, srl, sra, add, sub, xor, or, and, slt, sltu
I: slli, srli, srai, addi, xori, ori, andi, slti,
sltiu, jalr, lw, lh, lb
U: lui, auipc
B: beq, bne, blt, bge, bltu, bgeu
J: jal
S: sw, sh, sb
```

A seguir são descritos os tipos de instruções e as instruções do processador ***T-FIVE***:

TIPO R: REGISTRADOR-A-REGISTRADOR:



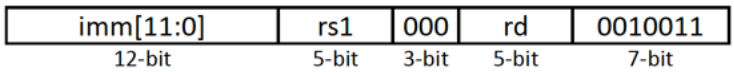
Os seguintes códigos estão associados as instruções deste tipo:



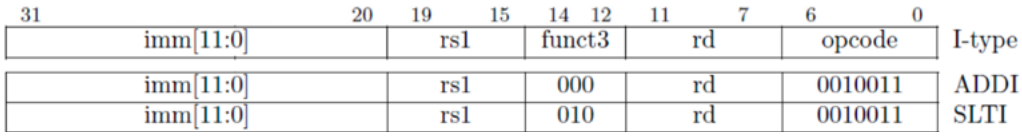
Em particular, neste tipo de instruções o ***T-FIVE*** implementa somente as seguintes instruções:

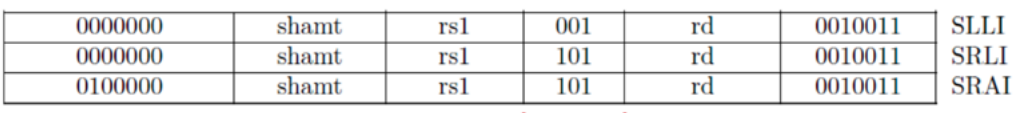
Soma: **ADD**: Add rd, rs1, rs2; GPR[rd] = GPR[rs1] + GPR[rs2];
Set Less Than: **SLT**: Slt rd, rs1, rs2; Se GPR[rs1] < GPR[rs2] então GPR[rd] = 1, se não GPR[rd] = 0;
Observação: GPR representa o conjunto de 32 registradores de propósito geral do ***T-FIVE***.

TIPO IMEDIATO:



O código das instruções neste formato está mostrado a seguir:



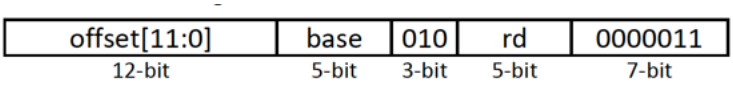


Neste formato o ***T-FIVE*** implementa as seguintes instruções: **ADDI, SLTI, SLLI, SRLI** e **SRAI**.

Soma imediato:	**ADDI** rd, rs2, sign-extended (Im12); GPR [rd] = GPR [rs1] + sign-extended (Imm12);
Set Less Than Imediato:	**SLTI** Se GPR [rs1] < Signextended (Imm12) então GPR [rd] = 1, se não GPR [rd] = 0;
Deslocamento lógico à esquerda imediato:	**SLLI** rs1, rd, shamt GPR [rd] = GPR[rs1] << shamt;
Deslocamento lógico à direita imediato:	**SRLI** rs1, rd, shamt GPR[rd] = GPR[rs1} >>shamt;
Deslocamento Aritmético à direita imediato:	**SRAI** rd, rs1, shamt; GPR [rd] = GPR [rs1] >> shamt, repetindo o bit mais significativo;

Observação: << representa um deslocamento à esquerda e >> o deslocamento à direita.

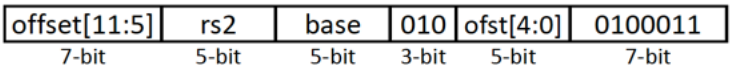
TIPO LOAD:



Base = rs1; offset = deslocamento com sinal estendido.

LOAD: Carrega posição de memória em registrador: **Lw rd, offset(rs1);**
GPR[rd]=M[GPR[rs1]+ offset];
Nesta instrução o offset deve ter o seu sinal estendido.

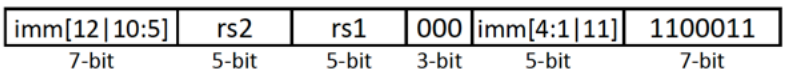
TIPO STORE:



Base = rs1; offset = deslocamento com sinal estendido;

STORE: Armazena registrador em posição de memória
Sw rd, offset(rs1); M[GPR[rs1] + offset] = GPR[rd];
Nesta instrução o offset deve ter o seu sinal estendido.

TIPO BRANCH:



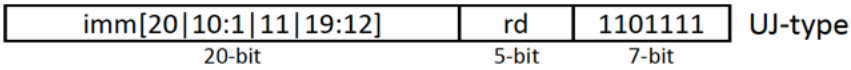
Instruções neste tipo: **Beq, Bne, Blt**:

Salto condicional: **Beq** rd, rs1, Imm13; Se GPR[rs1] = GPR[rd] então salta para target, se não PC = PC + 4;

- target = PC + sign-extend(imm_{13})
- if GPR[rs1]==GPR[rs2] then PC ← target
  else PC ← PC + 4

Observação: O bit menos significativo do target deve ser sempre igual a zero, por esse motivo o campo de imediato é concatenado com o bit zero em sua posição menos significativa.
No caso de **Bne** o teste é de diferença e no caso de **Blt** o teste é de menor ou igual.

TIPO BRANCH AND LINK:



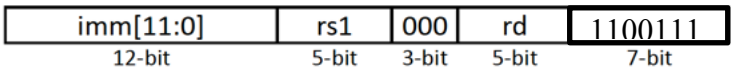
Instruções neste tipo: **Jal rd, target; Jalr rd, rs1, Imm;**

No caso da instrução **Jal** tem-se o seguinte significado:

- target = PC + sign-extend(imm_{21})
- GPR[rd] ← PC + 4
- PC ← target

Observação: O bit menos significativo do target deve ser sempre igual a zero, por esse motivo o campo de imediato é concatenado com o bit zero em sua posição menos significativa.

Já no caso da instrução **Jalr**, utiliza-se um outro formato, o mesmo das instruções imediatas, com o seguinte significado:
Jalr rd, rs1, Imm;



- GPR[rd] = PC + 4;
- PC = GPR[rs1] + Imm;

Neste caso não se insere o bit zero na posição menos significativa do endereço alvo (target).

PSEUDO INSTRUÇÕES:

Algumas instruções existentes no Assembler do ***T-FIVE*** podem ser obtidas por casos particulares das instruções já definidas acima.
Alguns exemplos dessas instruções são:

Li: **carrega imediato**: **Li rd, Imm12 ➔ ADDI rd, x0, Imm12**; carrega uma constante em um dos 32 registradores do ***T-FIVE***. Esta instrução pode ser codificada por meio da instrução ADDI, da seguinte forma: ADDI rd, x0, Imm12; Neste caso a constante representada pelo campo Imm12, será somada com o valor zero (registrador x0) e o resultado carregado no registrador rd, ou seja: GPR[rd] = 0 + Imm12; O campo Imm2 terá o seu sinal estendido.

NOP: **Nenhuma Operação**: **NOP**; esta instrução pode ser codificada por meio da instrução **SllI x0, x0, 0**; desloca o registrador zero à esquerda de zero posições e armazena no registrador zero.

J: **Pula incondicional** para um endereço: esta instrução pode ser obtida pela codificação adequada da instrução **Jal x0, target**; ou seja, o registrador para armazenar o endereço de retorno é o **x0**, portanto nenhum endereço é salvo e simplesmente a instrução pula para o endereço **target**, relativo ao contador de instruções.

Jr: pula para um endereço em registrador: esta instrução pode ser codificada usando a instrução **Jalr**, ou seja**, Jalr x0, rs1, Imm=0;** O valor do endereço de retorno não é salvo no registrador zero e o valor do registrador **rs1** é somado ao imediato, que possui o valor zero, e carregado ao contador de Instruções PC:
PC = GPR[rs1] + 0; Esta instrução é utilizada para fazer o retorno de sub-rotinas.

Mv: **Mv rd, rs1**; mover o conteúdo de um registrador para outro: esta instrução pode ser codificada pela seguinte instrução de máquina: **ADDI rd, rs1, 0**; assim o conteúdo do registrador especificado pelo campo rs1 será movido para o registrado especificado no campo rd, uma vez que o campo de imediato possui valor zero.

Estado do processador:

32 REGISTRADORES DE PROPÓSITO GERAL INTEIROS DE 32 BITS:

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5	t0	Temporary/alternate link register	Caller
x6	t1	Temporaries	Caller
x7	t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10	a0	Function arguments/return values	Caller
x11	a1	Function arguments/return values	Caller
x12	a2	Function arguments	Caller
x13	a3	Function arguments	Caller
x14	a4	Function arguments	Caller
x15	a5	Function arguments	Caller
x16	a6	Function arguments	Caller
x17	a7	Function arguments	Caller
x18	s2	Saved registers	Callee
x19	s3	Saved registers	Callee
x20	s4	Saved registers	Callee
x21	s5	Saved registers	Callee

x22	s6	Saved registers	Callee
x23	s7	Saved registers	Callee
x24	s8	Saved registers	Callee
x25	s9	Saved registers	Callee
x26	s10	Saved registers	Callee
x27	s11	Saved registers	Callee
x28	t3	Temporaries	Caller
x29	t4	Temporaries	Caller
x30	t5	Temporaries	Caller
x31	t6	Temporaries	Caller

RISC-V Register Usage Convention

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller

1 registrador de propósito específico de 32 bits:

- Contador de Instruções (CI- ou Program Counter – PC) de 32 bits

Modelo de Execução

- Chamada de procedimento simples.
- Chamada de procedimento reentrante e recursivo (com ajuda de software)

Subdivisão do Espaço de Memória:

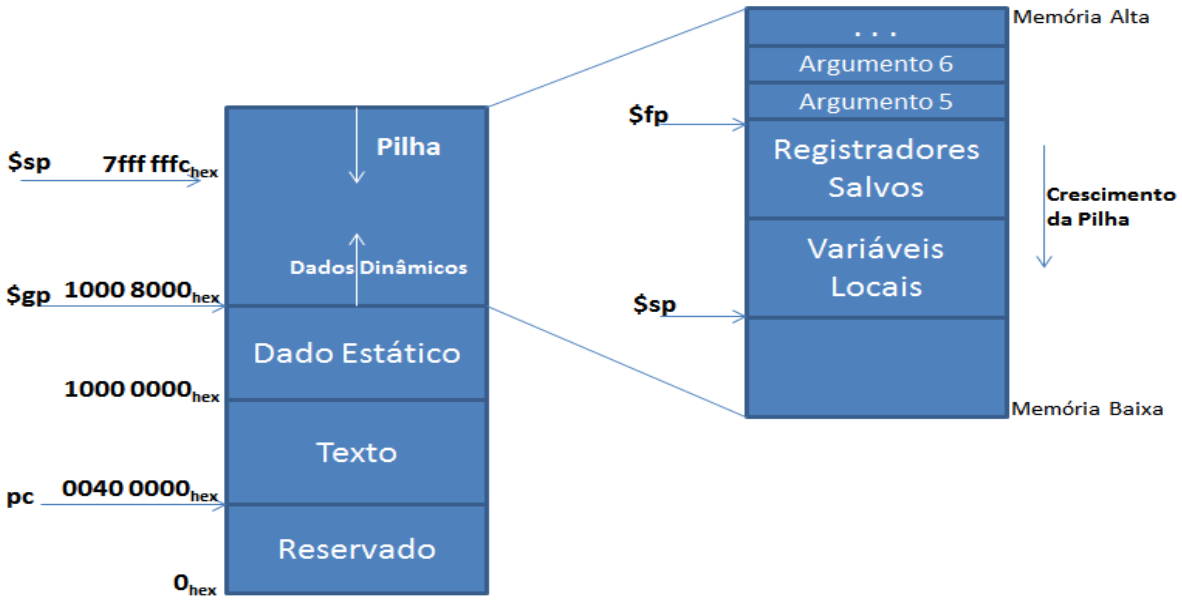


Figura 1: Estrutura de organização da memória do *T-FIVE*

Embora o mapeamento de endereços do ***RISK V*** 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 ***RISK V***.

T-FIVE: MICROARQUITETURA (MA)

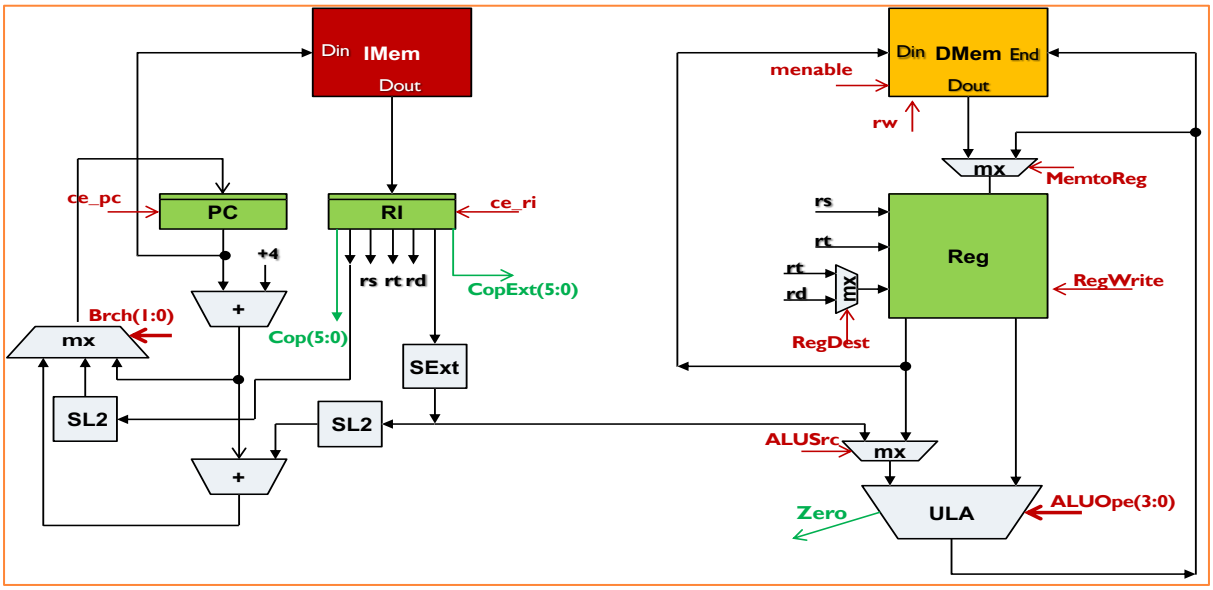
O projeto do processador ***T-FIVE*** deve ser feito em duas versões: Multi Ciclo (***T-FIVE-MC***) e outra Pipeline (***T-FIVE-Pipe***). No caso da organização Multi Ciclo os alunos devem se basear nas sugestões apresentadas nos slides desta disciplina. No caso da organização em Pipeline, os alunos devem ser basear no livro texto da disciplina (Patersson e Henessy).

Modelo Multi Ciclo – T-FIVE-MC

A MA do processador Multi Ciclo deve seguir a orientação que é dada nos slides de aula onde é apresentada uma organização do MIPS Multi Ciclo que deve servir de base para o projeto dos alunos.

Na organização Multi Ciclo, as memórias Cache, de instrução e de Dados, devem ser utilizadas para acesso de leitura e escrita sem se implementar o esquema de Hierarquia de memória. A **unidade de memória principal não faz parte deste projeto**, pois acessaremos diretamente as memórias cache (MI e MD). No caso real, o conjunto de memória Principal e os dois caches formam a hierarquia de memória.

A figura a seguir ilustra essa organização:



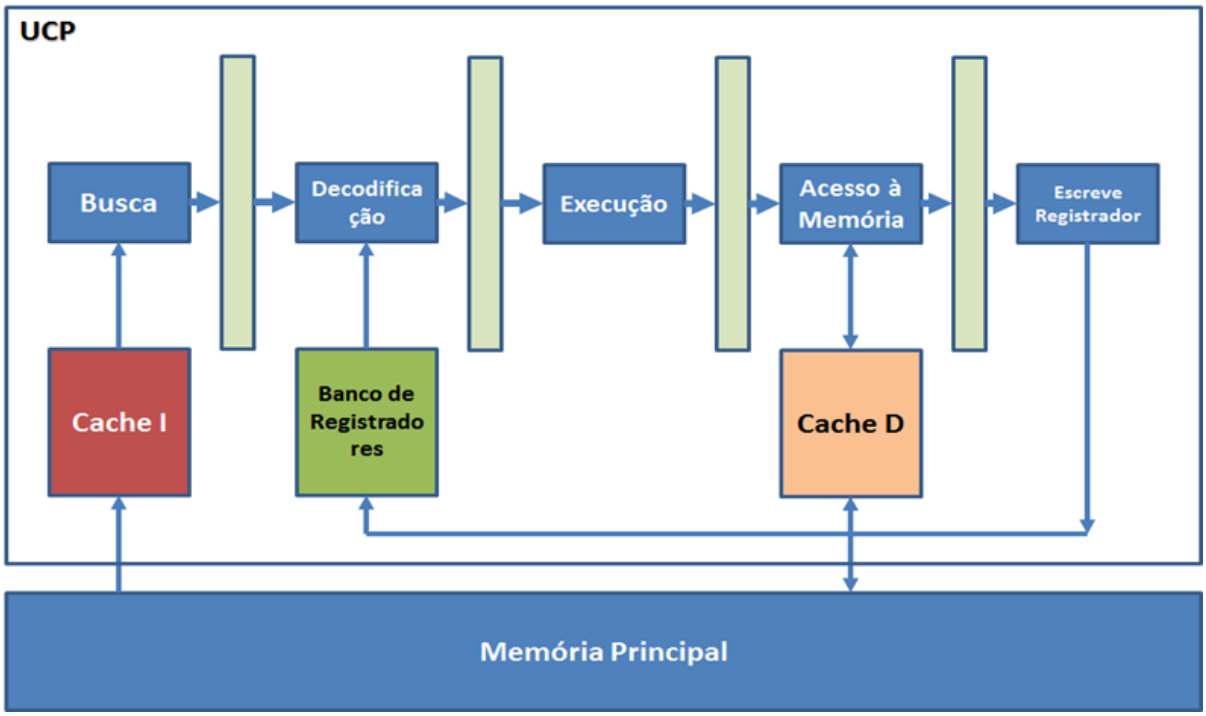
Pipeline – T-FIVE-Pipe

A MA em Pipeline deste processador deve ser organizada com 5 estágios, similar ao processador MIPS 32. A descrição de cada estágio é a que segue e mais referências a ela encontram-se no livro texto capítulo 4:

- **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 e sw (Load e Store) 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 no 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 processador ***MIPS*** que deve servir de exemplo para o projeto do ***T-FIVE-Pipe***.



A **unidade de memória principal não faz parte deste projeto**, pois acessaremos diretamente as memórias cache (MI e MD). No caso real, o conjunto de memória Principal e os dois caches formam a hierarquia de memória.

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**).
- Nos estágios de Busca e Acesso à Memória a latência da memória cache (Instruções e Dados deve ter um tempo de acesso igual a 5 ns.
- O Banco de Registradores (BR) deve ter uma latência de leitura e escrita igual à 5 ns. Deve ser possível **ler e escrever** no banco de registradores em um único ciclo do pipeline.
- O cálculo de endereço nas instruções de desvio condicional é realizado no estágio de Execução. O endereço efetivo é então carregado no Contador de Instruções ((PC), no estágio de Acesso à Memória.
- O pipeline deve possuir uma "Hazard Detection Unit", que permita detectar conflitos ocasionados por dependências de Dados e 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".

T-FIVE: Memórias Cache

- **Cache I - MI: Instruções**
  - Possui 16 KBytes ou 4096 palavras de 32 bits
  - 16 palavras por bloco

- Tempo de acesso = 5 ns (para leitura ou escrita)

- Cache D - MD: Dados
  - 16KBytes ou 4096 palavras de 32 bits
  - 16 palavras por bloco
  - Tempo de acesso = 5 ns (para leitura ou escrita)

O esquemático abaixo representa a hierarquia de memória na MA. **O projeto da Hierarquia de Memória não faz parte desta tarefa**, somente as memórias Cache de instruções (**MI**) e de dados (**MD**) é que devem fazer parte das organizações a serem projetadas.
Nas memórias cache, devem ser implementadas somente os acessos de leitura e escrita. Os procedimentos operacionais de gestão das memórias Cache na Hierarquia de Memória não fazem parte deste projeto.

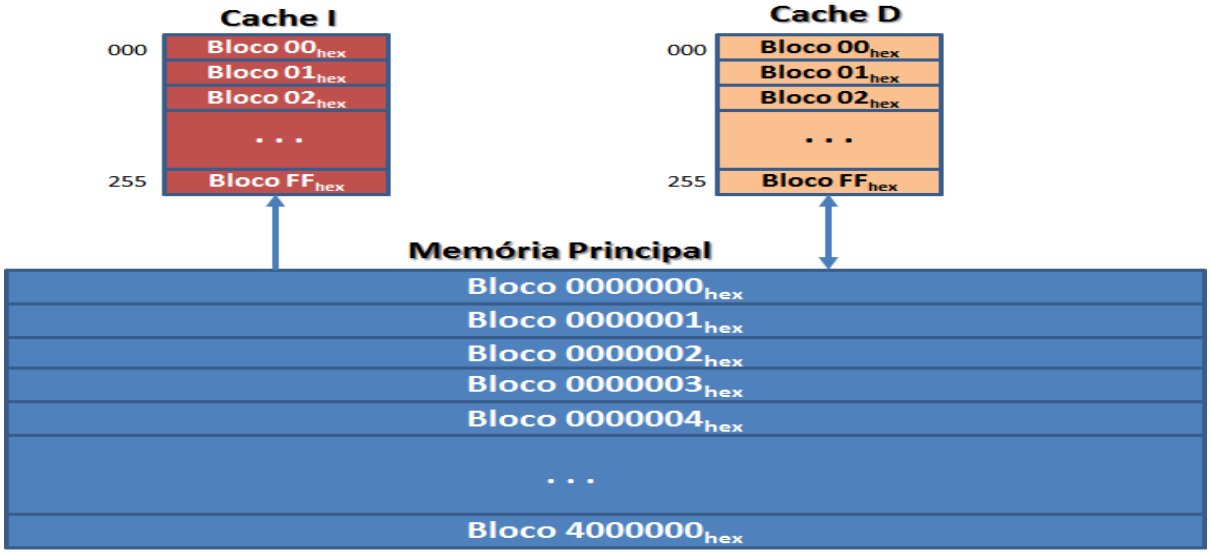


Figura 3: Estrutura da Hierarquia de Memória do *T-FIVE* (somente a título de ilustração).

T-FIVE: 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 31 do BR (Banco de registradores) devem conter o valor 0x0000
  - Todos os caches devem ter os seus conteúdos igualados à zero.
  - A memória Cache de instruções deverá ser carregada com o código do programa de teste e a memória Cache de dados deve ser carregada com os dados associados ao programa de teste.

T-FIVE: Detalhes para codificação VHDL

Na biblioteca de componentes fornecida na disciplina existem componentes para implementar as memórias Cache de instruções e de Dados. Esses componentes possuem a funcionalidade de carregarem o conteúdo dessas memórias por meio da leitura de um arquivo .txt com um formato específico.

A Memória Cache de instruções e de dados podem ser carregadas com 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 memória 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 memória, dado que a localização dela 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

T-FIVE: 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.
- Número de acessos à memória (Cache de Instruções e de Dados).
- 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 ***T-FIVE*** é o programa SORT cujo código fonte para o processador MIPS 32 encontra-se no livro texto.
O aluno deverá converter esse código do Assembler do MIPS 32 para o código do Assembler do ***RISK-V***, que é o mesmo do ***T-FIVE***.

CONJUNTOS DE DADOS

Os conjuntos de dados serão compostos por estruturas de dados (vetores) 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.
  - Número de instruções executadas pelo programa de teste x Carga de Trabalho x Conjunto de Dados;
  - CPI (Ciclos médios por instrução) x Carga de Trabalho X Tamanho do Conjunto de Dados
  - "Número de Acessos à Memória Cache" x Carga de Trabalho X Tamanho do

Conjunto de Dados

- T_e - Tempo de execução médio do programa de testes x Carga de trabalho x Conjunto 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:

Para organização Multi Ciclo as seguintes etapas devem ser percorrida pelos alunos:

a) Faça o projeto lógico do ***T-FIVE-MC*** 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. As etapas a serem seguida no projeto lógico estão detalhadas a seguir:
   i. Faça o diagrama ASMD mnemônico do algoritmo de execução de instruções do ***T-FIVE***. Este diagrama ASMD deve seguir o conceito de máquina de Mealy. Inclua nesse diagrama os estados necessários para o tratamento de interrupções (**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:
      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; (**2ª. Entrega**)
      ii. Identifique todos os sinais de controle e de condição que devem existir interconectando o fluxo de dado e a unidade de controle;(**2ª. Entrega**)
      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; (**2ª. Entrega**)
      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 testados somente sinais de entrada e/ou de condição; (**2ª. Entrega**)
      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;(**2ª. Entrega**)
   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 ***T-FIVE-MC*** 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 ***T-FIVE-MC*** dando o seu valor em unidades de tempo. (**3ª. Entrega**)

b) Escreva o programa **SORT,** que inclui a sub-rotina **SWAP**, em Assembler do ***T-FIVE (RISK V)***; (**1ª. Entrega**)

c) Execute e teste no processador projetado o seguinte trecho de programa:

```
Addi x7, x0, 3;
Addi x8, x0, 4;
Add  x9, x7, x8; (3a. Entrega).
```

d) Execute a sub-rotina **SWAP** no processador projetado (**3ª. Entrega**);

e) Execute o programa **SORT** no ***T-FIVE*** 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. (**3ª. Entrega**).

f) 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). (**3ª. Entrega**)

No caso da versão Pipeline do projeto do ***T-FIVE-Pipe***, os alunos devem seguir os passos ilustrados no livro texto da disciplina, capítulo 4. Neste caso as seguintes etapas devem ser realizadas:

1. Identifique todas as informações e sinais que devem ser passados de um estágio a outro em função das instruções de máquina que devem ser implementadas no processador ***T-FIVE-Pipe***;(**1ª. Entrega**)
2. Baseado nas informações do item anterior faça o projeto lógico de cada um dos 5 estágios do pipeline. Para cada estágio defina também a sua bancada de teste para realizar testes individuais em cada um dos estágios antes de interligá-los para formar o pipeline completo; (**1ª. Entrega – Estágio de Busca (IF) e Registradores -)ID)) 2ª. Entrega – os demais estágios – EX, MD e WB**)
3. Numa primeira versão faça o pipeline sem as unidades de detecção de conflitos ("Hazard detection") e antecipação de valores ("Forwarding"); (**2ª. Entrega**)
4. Após serem testados individualmente todos os estágios do pipeline, faça uma primeira integração sem as unidades de detecção de conflitos e antecipação de valores. Teste essa integração com a execução da instrução Load (**Lw rd, 0(rs1)**); (**2ª. Entrega**)
5. Faça o projeto lógico das unidades de detecção de conflito ("**Hazard Detection**") e antecipação de valores ("**Forwarding**") e adicione ao pipeline projetado e testado no item anterior; (**3ª. Entrega**)
6. Com o novo pipeline integrado, faça o teste novamente da instrução de load, para confirmar que ela continua funcionando. Faça mais um teste com o seguinte trecho de programa:

   ```
   Addi x7, x0, 3;
   Addi x8, x0, 4;
   Add  x9, x7, x8;
   ```

   Estes testes são para serem entregues na **3ª. Entrega**.
7. Escolha outras situações de conflito de dados, recursos e controle e teste o seu pipeline para verificar o correto funcionamento mesmo em situações que demandam resolução de conflitos. (**3ª. Entrega**)
8. Teste a sub-rotina **SWAP** na organização Pipeline do processador; (**3ª. Entrega**)
9. Execute o programa **SORT** no ***T-FIVE-Pipe*** 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. (**3ª. Entrega**)
10. Meça o tempo de latência do processador 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). (**3ª. Entrega**)
11. Faça a comparação de desempenho entre as duas organizações usando como base o tempo de execução do programa SORT e dos demais programas testados, em múltiplos conjuntos de dados com tamanhos diferentes sendo executados em ambas as organizações. (**3ª. Entrega**)
12. Registre toda o projeto do grupo em um relatório contendo informações suficiente para entender o projeto realizado. Este relatório deve ser submetido à correção juntamente com todos os arquivos do projeto ("Workspace), com evidências de seu funcionamento correto (testes realizados e suas formas de ondas obtidas e resultados conseguidos). (**3ª. Entrega**)

As entregas do projeto serão divididas em três partes. Em cada uma das entregas os alunos devem submeter os itens assinalados na lista mostrada acima. As datas de cada uma das três entregas serão as seguintes:

1ª. Entrega: **09/05/2022**

2a. Entrega: **13/06/2022**

3ª. Entrega (Final): **18/07/2022**

Faça uma tabela para cada organização (**MC e Pipe**) com as seguintes informações: **(3ª entrega)**

Instrução	Frequência de execução de instruções (SORT)	Número de instruções executadas (SORT)	Número de ciclos para executar a instrução	Tempo (ns)
Add				
Addi				
Beq				
Bne				
Slt				
Slti				
Jal				
Jalr				
Lw				
Sw				
Latência de Interrupção	-----	--------	------	
CPI (número médio de ciclos por instrução)				

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	**0.5 ns**
Atraso de porta lógica	Tpropagação	**0.25 ns**
Deslocadorcombinatório número fixo de bits	Tpropagação	**0 ns**
Register File	Tleitura	**5 ns**
	Tescrita	**5 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] **Patterson, David and John Hennessy**, "Computer Organization and Design - The Hardware/Software Interface", Morgan Kaufmann, 4 edição, 2009;
[2] Gajski, Daniel, "Principles of Digital Design", Prentice-Hall, 1997.
[3] Ruggiero, W; "Notas de Aula de PCS2307", EPUSP, 2011.
[4] MIPS