Universidade Federal de Pernambuco
Centro de Informática

ESPECIFICAÇÕES DO PROJETO DE UM SUBCONJUNTO DE INSTRUÇÕES DO PROCESSADOR RISC V

IF674 - Infraestrutura de Hardware

Monitoria de Infraestrutura de Hardware - EC

Recife, Março de 2019
Professora: Edna Natividade da Silva Barros

Sumário

1 Apresentação

Neste documento é apresentada a especificação de projeto da disciplina Infraestrutura de Hardware. A elaboração de tal documento tem como base a experiência adquirida durante os períodos que os escritores desempenharam suas atividades como monitor.

Este documento é composto por três capítulos. No primeiro é apresentada a especificação e as várias etapas de apresentação do projeto. No segundo é dada uma visão geral da CPU que será implementada. Neste capítulo também está presente o repertório de instruções que o processador desenvolvido deve conter e a descrição dos componentes fornecidos pela equipe de monitores da disciplina. No terceiro capítulo é apresentada a especificação do relatório de projeto. Ainda está presente um anexo que ensina como configurar e carregar a memória que irá compor o projeto.

Deve-se ressaltar que é dever de cada equipe ler e entender o que aqui está descrito. Caso existam dúvidas, os monitores deverão ser consultados, pois estes estão aptos e a disposição para esclarecer qualquer dúvida que possa existir durante a execução deste trabalho. É importante destacar que ao final de cada capítulo existem algumas observações que devem ser atentamente seguidas pelas equipes. Caso alguma dessas observações não seja seguida, a nota no projeto será coerente com a falha cometida.

A correção do projeto será totalmente baseada neste documento, portanto a equipe que desenvolver o projeto seguindo as recomendações e exigências aqui contidas e conseguir implementar corretamente todas as instruções do repertório do processador não terá problemas com relação à nota que será atribuída ao seu trabalho. Não serão aceitas reclamações que contrariem as regras estabelecidas neste documento.

É importante que o aluno tenha profissionalismo ao fazer seus trabalhos e respeite as regras que estão definidas. Ressalta-se ainda que os monitores não são autorizados a relevar qualquer descumprimento das regras em qualquer parte do trabalho e, caso isso venha a ocorrer, a nota do trabalho entregue refletirá tal descumprimento. É importante lembrar que as regras servem para facilitar o trabalho de alunos e corretores e, caso elas sejam cumpridas a risca, não haverão problemas com relação à correção.

Bom trabalho!

2 Especificações da CPU

O projeto a ser entregue consiste na implementação de um processador, a qual poderá ser feita com base na implementação descrita nos slides das aulas da professora e na figura abaixo apresentada. O processador deverá incluir três blocos básicos: unidade de processamento, unidade de controle e memórias. A seguir é dada uma descrição sucinta do processador a ser projetado.

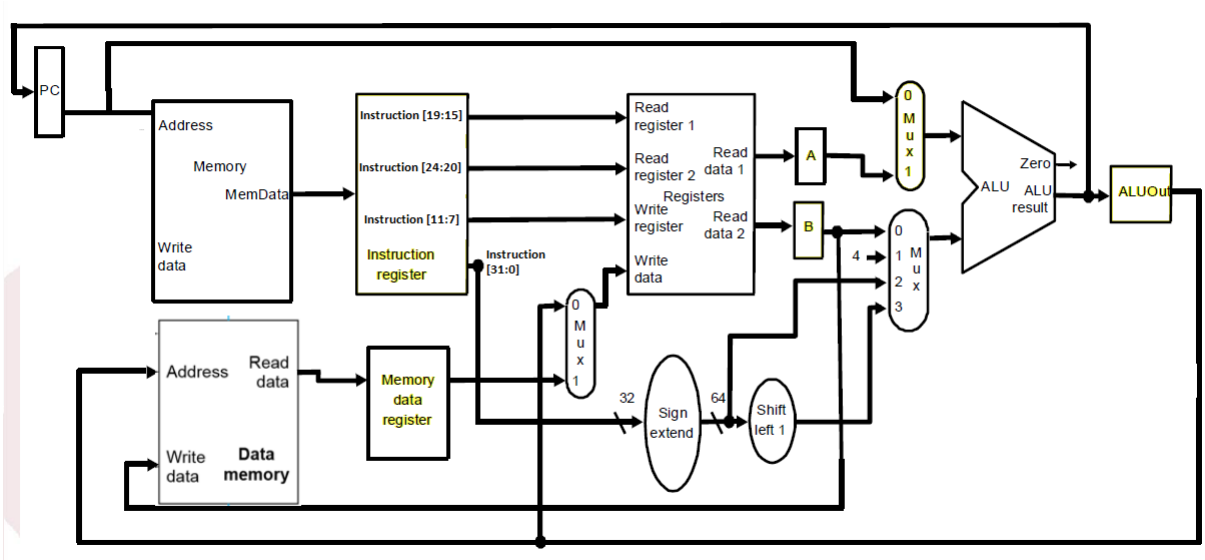


Figura 1 – Circuito base para a construção do projeto.

Inicialmente será projetada a unidade de processamento e a seguir a unidade de controle. A unidade de processamento será desenvolvida a partir da utilização dos seguintes componentes já descritos em VHDL/Verilog e fornecidos através da página da disciplina:

1. **ALU com operandos de 64 bits**
2. **Registrador de Deslocamento**
3. **Registradores de propósito geral de 64 bits**
4. **Registrador de Instruções**
5. **Banco de Registradores com 32 Registradores de 64 bits**
6. **Memória de Instruções de 32 bits**
7. **Memória de Dados de 64 bits**

Após o projeto da unidade de processamento com a definição de todos os seus componentes e suas respectivas portas de entrada e saída, deverá ser projetada a unidade de controle, que será implementada como uma máquina de estados finita (FSM) que controlará os sinais dos módulos usados a fim de implementar as instruções pedidas.

O processador RISC-V possui 32 registradores de propósito geral, cada um de 64 bits. Um subconjunto das instruções do RISC-V deverá ser implementado, de acordo com esta especificação. As instruções estão descritas no capítulo 3 nas Tabelas 2 a 7 e agrupadas por formato, cujo resumo está na Tabela 1.

3 Conjunto de instruções

	7 bits	5 bits	5 bits	3bits	5 bits	7 bits	
Tipo	[31..25]	[24..20]	[19..15]	[14..12]	[11..7]	[6..0]	Comentários
R	funct7	rs2	rs1	funct3	rd	**opcode**	Formato de Instruções Aritméticas
I	immediate[11:0]		rs1	funct3	rd	**opcode**	Loads e constantes aritméticas
S	immed[11:5]	rs2	rs1	funct3	immed[4:0]	**opcode**	Stores
SB	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	**opcode**	Formato de branch condicional
UJ	immediate[20,10:1,11,19:12]				rd	**opcode**	Formato de Jump Incondicional
U	immediate[31:12]				rd	**opcode**	Formato do Immediate mais significativos

Tabela 1 – Formatos de instruções do RISC - V

Tipo R

Instrução	Assembly	funct7	rs2	rs1	funct3	rd	opcode	Comportamento
add	rd,rs1,rs2	0000000	rs2	rs1	000	rd	0110011	rd = rs1 + rs2
sub	rd,rs1,rs2	0100000	rs2	rs1	000	rd	0110011	rd = rs1 - rs2
and	rd,rs1,rs2	0000000	rs2	rs1	111	rd	0110011	rd = rs1 AND rs2
slt	rd,rs1,rs2	0000000	rs2	rs1	010	rd	0110011	rd = (rs1 < rs2) ? 1:0

Tabela 2 – Instruções tipo R

Tipo I

Instrução	Assembly	immediate[11:0]	rs1	funct3	rd	opcode	Comportamento
addi	rd,rs1,imm	imm[11:0]	rs1	000	rd	0010011	rd = rs1 + imm *
slti	rd,rs1,imm	imm[11:0]	rs1	010	rd	0010011	rd = (rs1 < imm) ? 1:0 *
jalr	rd,rs1,imm	imm[11:0]	rs1	000	rd	1100111	rd = PC; PC = (rs1 + imm)*
lb	rd,imm(rs1)	imm[11:0]	rs1	000	rd	0000011	rd [7:0] = MEM[rs1 + imm]*
lh	rd,imm(rs1)	imm[11:0]	rs1	001	rd	0000011	rd[15:0] = MEM[rs1 + imm]*
lw	rd,imm(rs1)	imm[11:0]	rs1	010	rd	0000011	rd [31:0] = MEM[rs1 + imm]*
ld	rd,imm(rs1)	imm[11:0]	rs1	011	rd	0000011	rd = MEM[rs1 + imm]
lbu	rd,imm(rs1)	imm[11:0]	rs1	100	rd	0000011	rd [7:0] = MEM[rs1 + imm]**
lhu	rd,imm(rs1)	imm[11:0]	rs1	101	rd	0000011	rd[15:0] = MEM[rs1 + imm]**
lwu	rd,imm(rs1)	imm[11:0]	rs1	110	rd	0000011	rd [31:0] = MEM[rs1 + imm]**
nop	----	000000000000	00000	000	00000	0010011	No Operation
break	-----	000000000001	00000	000	00000	1110011	Para a execução
* **Extensão do Sinal**							
** **Completar com zeros**							

Tipo I (Shifts)

Instrução	Assembly	Funct6	shamt	rs1	funct3	rd	opcode	Comportamento
srli	rd,rs1,shamt	000000	shamt	rs1	101	rd	0010011	rd = rs1 >> shamt (lógico)
srai	rd,rs1,shamt	010000	shamt	rs1	101	rd	0010011	rd = rs1 >> shamt (aritm.)
slli	rd,rs1,shamt	000000	shamt	rs1	001	rd	0010011	rd = rs1 << shamt (lógico)

Tabela 3 - Instruções tipo I

Tipo S

Instrução	Assembly	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Comportamento
sd	rs2,imm(rs1)	imm	rs2	rs1	111	imm	0100011	MEM[rs1 +imm] = rs2
sw	rs2,imm(rs1)	imm	rs2	rs1	010	imm	0100011	MEM[rs1 +imm] = rs2[31:0]
sh	rs2,imm(rs1)	imm	rs2	rs1	001	imm	0100011	MEM[rs1 +imm] = rs2[15:0]
sb	rs2,imm(rs1)	imm	rs2	rs1	000	imm	0100011	MEM[rs1 +imm] = rs2[7:0]

Tabela 4 - Instruções tipo S

Tipo SB

Instrução	Assembly	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Comportamento
beq	rs1,rs2,imm	imm	rs2	rs1	000	imm	1100011	PC = PC + desl se rs1 == rs2
bne	rs1,rs2,imm	imm	rs2	rs1	001	imm	1100111	PC = PC + desl se rs1 != rs2
bge	rs1,rs2,imm	imm	rs2	rs1	101	imm	1100111	PC = PC + desl se rs1 >= rs2
blt	rs1,rs2,imm	imm	rs2	rs1	100	imm	1100111	PC = PC + desl se rs1 < rs2
Ver Próxima Seção (Tipo SB)								

Tabela 5 - Instruções tipo SB

Tipo U

Tipo - U	Assembly	immediate[31:12]	rd	opcode	Comportamento
lui	rd,imm	immediate[31:12]	rd	0110111	----------
Ver Próxima Seção (Tipo U)					

Tabela 6 - Instruções tipo U

Tipo UJ

Tipo - UJ	Assembly	immediate[20,10:1,11,19:12]	rd	opcode	Comportamento
jal	rd,imm	immediate[20,10:1,11,19:12]	rd	1101111	rd = PC; PC = PC + imm
Ver Próxima Seção (Tipo UJ)					

Tabela 7 - Instruções tipo UJ

4 Campo immediate nos tipos de instruções

O campo immediate, que representa uma constante que é passada na própria instrução, é disposta de diferentes maneiras e utilizada de diferentes maneiras dependendo do tipo da instrução. Aqui listaremos como estarão dispostos o campo immediate e como devem ser usados dependendo do tipo da instrução.

4.1 Tipo I

Nas instruções do Tipo I o immediate tem 12 bits e está disposto nos campos [31:20] da instrução. Em quase todos os casos do tipo I, exceto algumas instruções de deslocamento, o immediate deve ser estendido para 64 bits com extensão de sinal ou completando com zeros. A extensão de sinal é feita completando o restante dos bits dependendo do bit mais significativo, o bit de sinal. Se esse bit é 1 a extensão deve ser feita com 1's, já se o primeiro bit é 0 a extensão deve ser feita com 0's.

Apenas a instrução de **lbu (load byte unsigned)** que a extensão é feita com 0's independente

do bit de sinal, uma vez que essa instrução assume que o número que é passado no campo immediate já é um número sem sinal.

Nas instruções de deslocamento há uma pequena alteração no campo immediate. Ele passa a ser dividido em dois campos, o **Funct6** de 6 bits e o **shamt** que também é de 6 bits, que vai ser referente a quantidade de vezes que o valor presente no registrador rs1 deve ser deslocado.

4.2 Tipo S

As instruções do tipo S são as instruções de Store, que escrevem um dado na Memória de Dados.

Aqui o campo immediate tem 12 bits e está dividido em duas partes na instrução. A primeira parte contém 7 bits e está presente no campo [31:25] da instrução, a segunda parte contém 5 bits e está presente no campo [11:7] da instrução, onde o immediate final é dado pela concatenação desses dois campos que devem ser estendidos para 64 bits com extensão de sinal independente da instrução. Esse valor de 64 bits obtido será somado com o valor de rs1 para se obter o endereço onde será escrito o dado na Memória de Dados.

4.3 Tipo SB

O Tipo SB contém as instruções que fazem desvio condicional, onde dependendo de alguma condição o desvio é executado ou não.

Aqui o campo immediate também tem 12 bits mas é disposto embaralhado nos campos [31:25] e [11:7] da instrução conforme é mostrado na Tabela 5. Este número deve ser ordenado de acordo com a ordem que é mostrada na Tabela 5, e **antes de ser estendido para 64 bits**, deve ter um **0 (zero) concatenado a direita e deve ser feito um shit para a esquerda**, para então fazer a extensão de sinal para 64 bits, cujo valor resultante será somado ao PC a fim de se encontrar o endereço de desvio.

4.4 Tipo UJ

O Tipo UJ contém apenas a instrução **jal** que desvia sem a necessidade da verificação de alguma condição e salva o valor que está em PC em no registrador rd.

O campo immediate aqui contém 20 bits e está disposto no campo [31:12] da instrução de forma embaralhada. Este número deve ser ordenado de acordo com a ordem que é mostrada na Tabela 7, e **antes de ser estendido para 64 bits**, deve ter um **0 (zero) concatenado a direita e deve ser feito um shit para a esquerda**, para então fazer a extensão de sinal para 64 bits, cujo valor resultante será somado ao PC a fim de se encontrar o endereço de desvio.

O RISC-V usa o endereçamento relativo a PC para o cálculo do endereço de desvio nos desvios condicionais e incondicionais (Tipos SB e UJ).

$$PC = PC + Offset\ de\ Desvio$$

4.5 Tipo U

O Tipo U contém apenas a instrução **lui**. Nela o immediate está no campo [31:12] da instrução com a ordem correta.

A diferença no lui está em como deve ser salvo esse valor no registrador rd. O immediate de [31:12] é salvo na posição [31:12] do registrador rd. Agora, nos bits menos significativos [11:0] são colocados 0's, enquanto nos bits [63:32] é feita uma extensão de sinal, ou seja, eles terão uma cópia do bit 31. Este valor resultante é o que será escrito em rd.

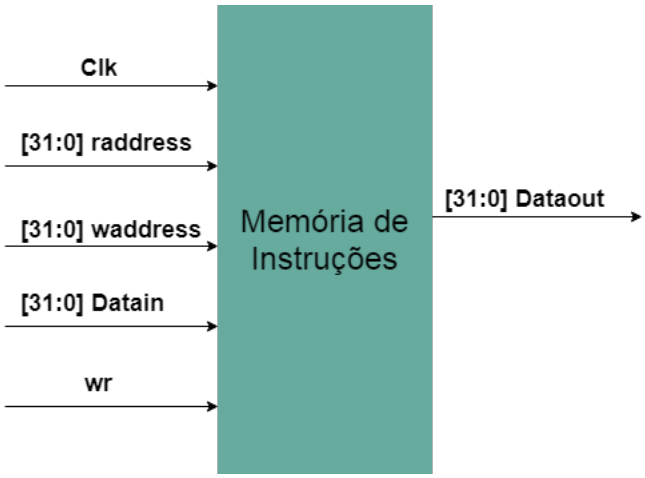
5 Componentes Fornecidos

Nesta seção são descritos os componentes do projeto que são fornecidos pela equipe de monitores da disciplina. Tais componentes poderão ser baixados a partir da página da disciplina.

No projeto, usaremos duas memórias. A primeira será a memória de instruções, onde cada instrução ocupa 32 bits e a segunda, a memória de dados, onde cada dado ocupa 64 bits.

5.1 Memória de Instrução

A memória usada no projeto possuirá palavras de 32 bits, ou seja, cada instrução contém 32 bits ou 4 bytes, com endereçamento por byte e tem o tamanho de 65536 bytes. As entradas e saídas da memória são:



1. **raddress: endereço de 32 bits**
   Endereço de leitura da memória (32 bits).
2. **waddress: endereço de 32 bits**
   Endereço de escrita da memória (32 bits).
3. **Datain: entrada de dados de 32 bits**
   Entrada de Dados da memória (32 bits).
4. **Dataout: saída de dados 32 bits**
   A saída de Dados da memória (32 bits).
5. **Clk: entrada de 1 bit**
   Bit de clock comum a todo o "computador".
6. **Wr: entrada de 1 bit**
   O bit que diz se vai ler ou escrever, onde 1 é para escrita e 0 para leitura.

Peculiaridades da memória:

1. Enquanto o bit "wr" estiver com o valor 0 (zero) a memória estará sendo lida. Já quando wr estiver com o valor 1 (um) a memória estará sendo escrita.
2. Ao se fazer uma requisição de leitura, o valor só estará pronto após um ciclo de clock.
3. A escrita também é concluída apenas após um ciclo.

4. Instruções e dados serão armazenados na memória usando a estratégia *little-endian*.

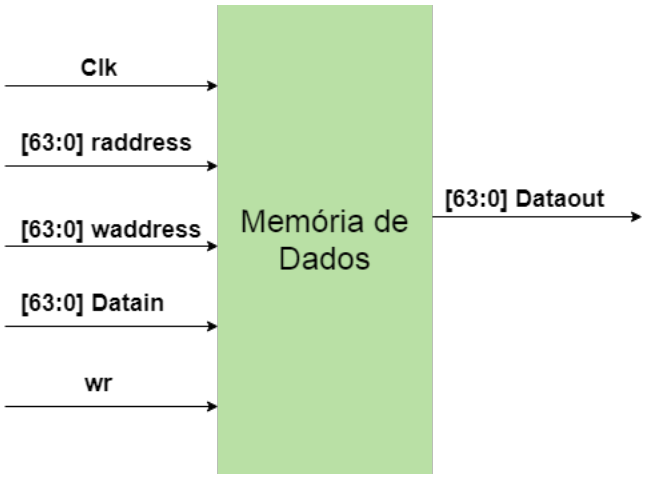
5. A memória de instruções, obviamente, só conterá instruções, logo, nela ocorrerão apenas leituras das instruções. O que se pode acontecer é a mudança da ordem de leitura das instruções com instruções como os branchs, mas não escritas de dados ! **Atenção a isso !!**

6. Ambas as memórias leem de um arquivo .mif as instruções ou dados.

7. Na **memória de instruções** o nome do arquivo mif é **instructions.mif**

5.2 Memória de Dados

A memória de dados tem um funcionamento parecido com a memória de instruções. Porém, ela recebe dados de 64 bits como entrada e retorna dados de 64 bits.



1. **raddress: endereço de 64 bits**
   Endereço da leitura (64 bits)

2. **waddress: endereço de 64 bits**
   Endereço de escrita (64 bits)

3. **Datain: Entrada de dados**
   Dados que serão escritos na memória (64 bits)

4. **Dataout: Saída de dados**
   Dados que serão lidos da memória (64 bits)

5. **Clk: entrada de 1 bit**
   Bit de clock comum a todo o "computador".

6. **Wr: entrada de 1 bit**
   O bit que diz se vai ler ou escrever, onde 1 é para escrita e 0 para leitura.

A **memória de dados** lê os dados de um arquivo mif chamado **dados.mif**

5.3 Registrador de 64 bits

Registradores são usados em algumas partes do circuito do processador. Este módulo consiste de um registrador de propósito geral de 64 bits. Por exemplo, o PC e o ALUOut são registradores de propósito geral de 64 bits.
Obs: o dado que entra em Data_in só passa para Data_out quando o registrador passa 1 ciclo de clock com o sinal ***Load*** *ativo (Load = 1).*

1. **clk**
   Sinal de clock comum a todo processador.

2. **DadoIn**
   Entrada de **64 bits** que contém o dado a ser escrito no registrador.

3. **regWrite**
   Sinal que controla a escrita no Registrador

4. **reset**
   Quando o sinal é ativado é feito o Reset do registrador, apagando os valores salvo nele.

5. **DadoOut**
   Mostra o dado de **64 bits** que está escrito no Registrador.

5.4 Banco de Registradores

O banco de registradores é composto por 32 registradores de 64 bits cada. Dois registradores podem ser visíveis simultaneamente nas saídas *Read Data 1* e *Read Data 2*.

1. **regreader1**
   Número, de **5 bits**, do registrador 1 a ser lido do banco.

2. **regreader2**
   Número, de **5 bits**, do registrador 2 a ser lido do banco.

3. **regwriteaddress**
   Número, de **5 bits**, do registrador que vai receber o dado presente em **datain**.

4. **datain**
   Dado, de **64 bits** que vai ser escrito em **regwriteaddress**.

5. **write**
   Sinal que controla a escrita de **datain** em **regwriteaddress**.

6. **reset**
   Limpa todos os registradores

7. **dataout1**
   Valor salvo no registrador **regreader1**.

8. **dataout2**
   Valor salvo no registrador **regreader2**.

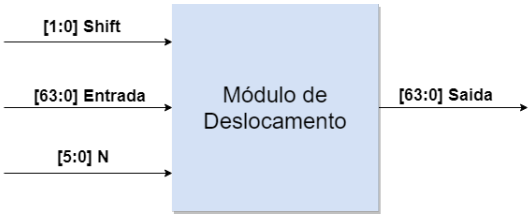
A leitura dos registradores é combinacional, isto é, se os valores nas entradas **regreader1** ou **regreader2** forem alterados, os valores nas saídas ou **ReadData2** podem ser alterados. O registrador a ser escrito é selecionado pela entrada **regwriteaddress** e quando a entrada **write** é ativada (igual a 1) o registrador cujo número está em **regwriteaddress** recebe o conteúdo da entrada **datain**. O sinal de **reset** Limpa todos os registradores e é assíncrono.

5.5 Módulo de Deslocamento

O módulo de deslocamento é capaz de deslocar um número inteiro de 64 bits para esquerda e para a direita. No deslocamento a direita o sinal pode ser preservado ou não, dependo da instrução fazer um shift aritmético ou lógico.

O número de deslocamentos pode variar entre 0 e 64 e é especificado na entrada **N** (6 bits) do módulo de deslocamento. A funcionalidade desejada do módulo de deslocamento é especificada na entrada Shift, de dois bits, conforme figura abaixo. O Módulo de Deslocamento é combinacional, ou seja, basta colocar um valor na entrada e alterar as entradas N e Shift que o valor deslocado fica pronto em Saida no mesmo ciclo.

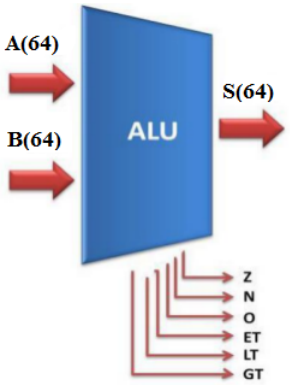
Shift	Descrição
00	Shift a esquerda lógico de N vezes
01	Shift a direita lógico de N vezes
10	Shift a direita aritmético de N vezes
11	Sem operação, coloca a entrada na saída apenas



5.6 Unidade Lógica e Aritmética (ALU)

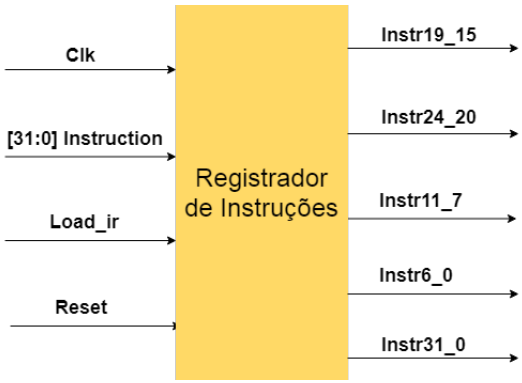
A Unidade Lógica e Aritmética (ALU) é um circuito combinacional que permite a operação com números de 64 bits na notação complemento a dois. A funcionalidade é especificada pela entrada **F**. A ALU também tem algumas flags que podem ser ativadas dependendo da operação. Segue descrição abaixo:

Função	Operação	Descrição	Flags
000	S = A	Carrega A	Z, N
001	S = A + B	Soma	Z, N, O
010	S = A – B	Subtração	Z, N, O
011	S = A and B	And lógico	Z
100	S = A + 1	Incremento de A	Z, N, O
101	S = not A	Negação de A	Z
110	S = A xor B	OU exclusivo	Z
111	S = A comp B	Comparação	EG, GT, LT



5.7 Registrador de Instruções

O registrador de Instruções fica após a memória de instruções e guarda a instrução lida da memória até o fim da execução da mesma.



1. **Instruction [31:0]**
   Entrada de 32 bits do Registrador de Instruções, que nesse caso é a instrução lida da memória.

2. **Load_ir**
   Sinal que controla a escrita no Registrador de Instruções.

3. **Instr19_15**
   Campo referente ao rs1, usado como o registrador fonte 1 em alguns tipos de instruções.

4. **Instr24_20**
   Campos referente ao rs2, usado como o registrador fonte 2 em alguns tipos de instruções.

5. **Instr11_7**
   Campo referente ao rd, usado como o registrador de destino em alguns tipos de instruções.

6. **Instr6_0**
   Campo referente ao opcode das instruções.

7. **Instr31_0**
   Esta saída retorna toda a instrução lida da memória e é necessária, por exemplo, em instruções do tipo *S,SB,U,UJ* onde o Immediate não está presente nas outras saídas e é necessário ter toda a instrução para obtê-lo, ou para obter os campos funct das instruções.

8. **Clk**
   Sinal de Clock comum a todo o processador.

9. **Reset**
   Limpa os valores presentes no Registrador de Instruções.

6 Tratamento de exceções

O processador deve incluir tratamento de dois tipos de exceções: opcode inexistente (ou instrução indefinida) e overflow. Na ocorrência de uma exceção, o endereço da instrução que causou a exceção deverá ser salvo no registrador **EPC** e a causa da exceção **0 para opcode inexistente e 1 para overflow**, também deverão ser salvos em novo registrador chamado de **registrador de causa** a ser inserido na unidade de processamento. O PC será carregado, posteriormente, com o valor do endereço da rotina de tratamento da exceção correspondente.

A rotina de tratamento, que estará armazenada na memória, armazenará o valor 1 no registrador 30 e para a execução de um programa no caso de um opcode inexistente. No caso de um overflow, o valor 2 deverá ser armazenado no registrador 30 e a execução do programa deve continuar. **Não é necessário implementar a rotina de tratamento, mas apenas colocar em PC o endereço da rotina de tratamento.**

Basicamente os passos que projeto deve ter implementados após ocorrer uma exceção são os seguintes:

1. Ao ser detectada uma exceção o endereço da instrução que a causou deve ser salvo no registrador EPC e a causa correspondente no Registrador de Causa.

2. O byte localizado na posição de memória 254 ou 255 (dependendo da exceção) deverá ser lido e estendido para 32 bits. Essa extensão é feita apenas com zeros.

3. O número de 32 bits resultante do passo anterior deverá ser o novo valor de PC. Após completar tais passos a rotina de tratamento de exceção que está armazenada na memória será executada, fazendo assim por software a tarefa de armazenar o valor adequado no registrador 30.

7 Metodologia de Projeto e Cronograma de Avaliação

Neste capítulo são apresentados os procedimentos que devem ser seguidos para o desenvolvimento das várias etapas do projeto do processador descrito no capítulo anterior desta especificação. Para cada uma das etapas os grupos devem apresentar o projeto da unidade de processamento como diagrama de blocos, o projeto da unidade de controle como diagrama de estados e a implementação em SystemVerilog funcionando. Os diagramas de bloco e de estado podem ser feitos em cartolinas ou no computador.

7.0.1 Descrição da unidade de processamento

Os circuitos que irão compor o projeto devem ser desenhados na cartolina ou em alguma ferramenta no computador como a draw.io tomando como base a figura 1 do Capítulo 2 desta especificação. A quantidade e quais elementos serão incrementados ao circuito base irá depender da forma como cada grupo decidir implementar o conjunto de instruções que é pedido neste trabalho. Apesar dos grupos terem a liberdade de escolher quantas unidades irão compor o circuito do processador, a arquitetura desenvolvida deve dar sentido a cada um dos componentes que a forma. Além disso, é recomendável

que o projeto dê ênfase ao melhor aproveitamento de todos os seus componentes, ou seja, não adianta fazer um circuito com várias unidades que façam pouca ou nenhuma atividade. Deve-se sempre pensar em economia de hardware e rapidez de execução das instruções.

Todas as ligações que forem relevantes ao entendimento do circuito sendo projetado deverão estar presentes no desenho da cartolina, sendo que todos os sinais que partem ou chegam à unidade de controle deverão **obrigatoriamente** ser apresentados. Devem estar presentes também os sinais que interligam as demais unidades.

Observação: Não é necessário desenhar os sinais de clock e reset que estarão presentes na maioria das unidades, pois a presença destes sinais é intuitivamente percebida. Os sinais que saem da unidade de controle deverão estar nomeados, pois isso facilita no entendimento geral do circuito.

7.0.2 Descrição da Unidade de Controle

As equipes deverão apresentar o diagrama de estados da máquina de estados da unidade de controle do processador. Tal diagrama deverá ser desenhado em uma nova cartolina, ou novamente no draw.io e terá que conter os estados que a unidade de controle está enquanto o processador executa suas instruções. Cada estado deverá conter os valores e os nomes dos sinais de saída da unidade de controle. Não é necessário especificar todos os sinais da unidade em cada estado, porém é necessária a presença de todos os sinais que foram alterados em cada estado específico.

7.1 Primeira Etapa: Projetando a Busca da Instrução + Pequeno Subconjunto de instruções

Nesta etapa os grupos apresentarão a implementação da etapa de busca da instrução na memória e a execução de algumas instruções. Os alunos deverão apresentar a unidade de processamento que faz a busca e a unidade de controle em cartolina ou em algum diagrama no computador. A implementação em SystemVerilog deverá ser apresentada e simulada de forma a ser possível visualizar o valor de PC, a informação lida da memória o registrador de instruções, o conteúdo dos registradores lidos, o conteúdo a ser escrito nos registradores, o conteúdo a ser lido ou escrito da memória de dados e os estados da máquina de estados e as saídas que seja possível visualizar que uma instrução está sendo executada corretamente.

Na data disponível no cronograma da cadeira referente a Entrega do Subconjunto 1, deverão ser entregues as seguintes instruções:

Tipo R

Instrução	Assembly	funct7	rs2	rs1	funct3	rd	opcode	Comportamento
add	rd,rs1,rs2	0000000	rs2	rs1	000	rd	0110011	rd = rs1 + rs2
sub	rd,rs1,rs2	0100000	rs2	rs1	000	rd	0110011	rd = rs1 - rs2

Tipo I

Instrução	Assembly	immediate[11:0]	rs1	funct3	rd	opcode	Comportamento
addi	rd,rs1,imm	imm[11:0]	rs1	000	rd	0010011	rd = rs1 + imm *
ld	rd,imm(rs1)	imm[11:0]	rs1	011	rd	0000011	rd = MEM[rs1 + imm] *
* **Extensão do Sinal**							
** **Completar com zeros**							

Tipo S

Instrução	Assembly	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Comportamento
sd	rs2,imm(rs1)	imm	rs2	rs1	111	imm	0100011	MEM[rs1 +imm] = rs2

Tipo SB

Instrução	Assembly	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Comportamento
beq	rs1,rs2,imm	imm	rs2	rs1	000	imm	1100011	PC = PC + desl se rs1 == rs2
bne	rs1,rs2,imm	imm	rs2	rs1	001	imm	1100111	PC = PC + desl se rs1 != rs2

Tipo U

Tipo - U	Assembly	immediate[31:12]	rd	opcode	Comportamento
lui	rd,imm	immediate[31:12]	rd	0110111	----------

Tabela das Instruções do Subconjunto 1

7.2 Segunda Etapa: Implementando um conjunto maior de Instruções do RISC-V

Na segunda etapa do projeto, com a data também disponível no cronograma da cadeira, os grupos deverão apresentar a busca e execução de todo o conjunto de instruções do RISC-V descrito no capítulo 3 **(subconjunto 1 + subconjunto 2)**, incluindo o tratamento de exceções de opcode inexistente e overflow descritos no capítulo 6. Os alunos deverão apresentar a unidade de processamento e a unidade de controle que realizam a busca de instruções, as instruções do subconjunto 1 e as instruções do subconjunto 2 descrito nas tabelas abaixo.

Tipo R

Instrução	Assembly	funct7	rs2	rs1	funct3	rd	opcode	Comportamento
add	rd,rs1,rs2	0000000	rs2	rs1	000	rd	0110011	rd = rs1 + rs2
sub	rd,rs1,rs2	0100000	rs2	rs1	000	rd	0110011	rd = rs1 - rs2
and	rd,rs1,rs2	0000000	rs2	rs1	111	rd	0110011	rd = rs1 AND rs2
slt	rd,rs1,rs2	0000000	rs2	rs1	010	rd	0110011	rd = (rs1 < rs2) ? 1:0

Tipo I

Instrução	Assembly	immediate[11:0]	rs1	funct3	rd	opcode	Comportamento
addi	rd,rs1,imm	imm[11:0]	rs1	000	rd	0010011	rd = rs1 + imm *
slti	rd,rs1,imm	imm[11:0]	rs1	010	rd	0010011	rd = (rs1 < imm) ? 1:0 *
jalr	rd,rs1,imm	imm[11:0]	rs1	000	rd	1100111	rd = PC; PC = (rs1 + imm)*
lb	rd,imm(rs1)	imm[11:0]	rs1	000	rd	0000011	rd [7:0] = MEM[rs1 + imm]*
lh	rd,imm(rs1)	imm[11:0]	rs1	001	rd	0000011	rd[15:0] = MEM[rs1 + imm]*
lw	rd,imm(rs1)	imm[11:0]	rs1	010	rd	0000011	rd [31:0] = MEM[rs1 + imm]*
ld	rd,imm(rs1)	imm[11:0]	rs1	011	rd	0000011	rd = MEM[rs1 + imm]
lbu	rd,imm(rs1)	imm[11:0]	rs1	100	rd	0000011	rd [7:0] = MEM[rs1 + imm]**
lhu	rd,imm(rs1)	imm[11:0]	rs1	101	rd	0000011	rd[15:0] = MEM[rs1 + imm]**
lwu	rd,imm(rs1)	imm[11:0]	rs1	110	rd	0000011	rd [31:0] = MEM[rs1 + imm]**
nop	----	000000000000	00000	000	00000	0010011	No Operation
break	-----	000000000001	00000	000	00000	1110011	Para a execução
* Extensão do Sinal							
** Completar com zeros							

Tabela das Instruções do Subconjunto 2

Tipo I (Shifts)

Instrução	Assembly	Funct6	shamt	rs1	funct3	rd	opcode	Comportamento
srli	rd,rs1,shamt	000000	shamt	rs1	101	rd	0010011	rd = rs1 >> shamt (lógico)
srai	rd,rs1,shamt	010000	shamt	rs1	101	rd	0010011	rd = rs1 >> shamt (aritm.)
slli	rd,rs1,shamt	000000	shamt	rs1	001	rd	0010011	rd = rs1 << shamt (lógico)

Tipo S

Instrução	Assembly	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Comportamento
sw	rs2,imm(rs1)	imm	rs2	rs1	010	imm	0100011	MEM[rs1 +imm] = rs2[31:0]
sh	rs2,imm(rs1)	imm	rs2	rs1	001	imm	0100011	MEM[rs1 +imm] = rs2[15:0]
sb	rs2,imm(rs1)	imm	rs2	rs1	000	imm	0100011	MEM[rs1 +imm] = rs2[7:0]

Tipo SB

Instrução	Assembly	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Comportamento
bge	rs1,rs2,imm	imm	rs2	rs1	101	imm	1100111	PC = PC + desl se rs1 >= rs2
blt	rs1,rs2,imm	imm	rs2	rs1	100	imm	1100111	PC = PC + desl se rs1 < rs2

Tipo UJ

Tipo - UJ	Assembly	immediate[20,10:1,11,19:12]	rd	opcode	Comportamento
jal	rd,imm	immediate[20,10:1,11,19:12]	rd	1101111	rd = PC; PC = PC + imm

Tabela das Instruções do Subconjunto 2

Observações:

Durante o desenvolvimento destes trabalhos os alunos terão algumas aulas de laboratório para o acompanhamento de seus projetos. Caso ainda existam dúvidas, os alunos poderão procurar os monitores ou enviar e-mail para o grupo da disciplina: infradehardwareec@googlegroups.com .

Após o término de cada etapa do projeto os alunos deverão guardar as suas descrições da arquitetura, pois elas serão muito úteis no desenvolvimento da próxima etapa.

7.3 Cronograma das apresentações

O cronograma da cadeira pode mudar, porém, as datas mais atualizadas estarão sempre no cronograma da cadeira disponível no site da disciplina.

8 Formato do relatório

O relatório deverá conter as seguintes partes:

1. Capa;
2. Índice;
3. Descrição dos módulos;
4. Descrição das operações;
5. Descrição dos estados de controle;
6. Máquina de estados da unidade de processamento;

É obrigatória a presença de todas as partes citadas. Uma descrição de cada uma delas é feita na seção seguinte.

8.1 Descrição das Partes do Relatório

8.1.1 Capa

A capa deve conter um cabeçalho com o nome da universidade e do centro onde a disciplina é ensinada. Deve também apresentar o título: Relatório de Projeto (título único que deve estar presente em todos os relatórios). O nome dos alunos que compõem a equipe deve vir logo em seguida, juntamente com seus logins, e por último deve vir a data da entrega do projeto.

Não será necessária nenhuma imagem na capa que tenha como simples objetivo embelezar o trabalho. A capa padrão é definida no molde já descrito, portanto não faça além do que foi pedido.

Ex.:

Universidade Federal de Pernambuco
Centro de Informática

RELATÓRIO DE PROJETO

Eduardo Barreto Brito, ebb2
Lucas Pontes de Albuquerque, lpa
Miguel Luiz Pessoa da Cruz Silva, mlpcs

Recife, 4 de Novembro de 2016
Professora: Edna Natividade da Silva Barros

8.1.2 Índice

Um índice simples deve compor o relatório para que a correção seja facilitada.

8.1.3 Descrição dos módulos

Cada dos módulos que forem implementados* pelas equipes deve ser descrito em detalhes. O objetivo almejado ao construir tais módulos deve estar claramente apresentado, bem como sua funcionalidade dentro do escopo do projeto. Os sinais de entrada e saída devem ser especificados assim como o conjunto de atividades executadas pelo algoritmo interno de cada um para prover os resultados esperados.

Exemplo: Usamos como exemplo a descrição do registrador de 64 bits que é fornecido juntamente à especificação do projeto.
Módulo: Registrador de 64 bits.
Entradas:
Clk (1 bit): representa o clock do sistema.
Reset (1 bit): sinal que, quando ativado zera o conteúdo do registrador.
Load (1 bit): sinal que ativa o carregamento de um valor de entrada no registrador. Entrada (64 bits): vetor de bits que será armazenado no registrador.
Saida (64 bits): vetor de bits que contém o valor armazenado no registrador.
Objetivo:
Módulo criado para armazenamento temporário de valores, que na implementação multiciclo devem ser preservados durante à execução de cada estágio das instruções.
Algoritmo:
Quando valor de Load está ativado e ocorre a ativação do sinal Clk o vetor de bits Entrada é disponibilizado como o sinal de Saida até que o sinal de load esteja novamente ativado na subida de Clk. Caso o sinal Clear esteja ativado o valor da saída passa a ser o vetor zero.

*Multiplexadores não devem ser descritos, pois seu funcionamento é trivial, nem também os módulos já cedidos.

8.1.4 Descrição das operações

Nesta parte deverão ficar todos os passos que compõem a execução de cada uma das instruções exigidas no projeto destacando todas os módulos envolvidos na sua execução. Cada instrução deverá ser descrita separadamente.
Exemplo:
Instrução add rd, rs1, rs2
Após identificar a instrução add no estágio de decodificação os valores dos registradores rs1 e rs2 são carregados a partir do banco de registradores e a operação de soma é realizada na ULA de acordo com o sinal de controle que vem da unidade de controle, posteriormente o resultado é gravado no registrador de destino rd e uma nova instrução é buscada na memória.

8.1.5 Descrição dos estados de controle

O objetivo aqui é que as equipes apresentem a descrição de cada um dos estados de controle do processador desenvolvido. Não será necessário que o aluno especifique o valor que cada sinal deve receber dentro do estado, pois isso já estará claro no código do projeto, porém é imprescindível que seja descrito o que cada estado deve fazer, ou seja, o que a equipe objetivava ao construir cada um.
Exemplo:

Estado: Decodifica
Neste estado o opcode e o funct (quando necessário) são analisados. Isso define qual o novo estado que a máquina deve assumir para continuar o processamento.

8.1.6 Máquina de estados da unidade de processamento

Deverá ser apresentado também o diagrama de estados de controle em forma de figura

Neste segmento do relatório, deve estar presente a imagem da máquina de estados da unidade de processamento, que deve estar dentro desta seção do relatório ou em anexo ao relatório, caso seja feito no draw.io ou caso seja feito na cartolina, ela deve ser entregue. É proibido usar a imagem gerada automaticamente pelo Software Quartus !!

Observações:
As regras descritas nesse capítulo devem ser seguidas a risca na elaboração do relatório, sendo que reclamações que não estejam de acordo com o que foi exigido nessa especificação não serão consideradas válidas.
Os monitores serão extremamente rígidos com relação a tais regras, portanto os alunos devem ler atentamente a especificação, entender o que foi pedido para, posteriormente, fazer o relatório.
O relatório deve ser entregue no formato PDF (digital) na página do Google Drive que será criada pela monitoria para cada grupo submeter os relatórios e o projeto.
O relatório do grupo deve ser entregue até a data especificada neste documento, então não serão aceitos documentos atrasados, fazendo com que o grupo leve uma penalidade na nota.

Boa sorte e bom trabalho!