PCS3225 - Sistemas Digitais II
Projeto 5 - Unidade de Controle do PoliLEG

Prof. Edson S. Gomi rev. Prof. Sergio R. M. Canovas

10/11/2021

Nos projetos anteriores, você projetou a ROM (`Instruction Memory`), a RAM (`Data Memory`), o banco de registradores (`Registers`) e a Unidade Lógico-Aritmética (`ALU`). Neste projeto você construirá outra unidade funcional, o `Sign-extend`. Você também fará uma modificação na `ULA` que foi construída no projeto anterior, com o objetivo de adequá-la às instruções do PoliLEG, e implementará o `ALUControl`. Finalmente, você construirá a Unidade de Controle (`Control`) do processador.

Introdução

Conforme foi explicado no projeto anterior, o processador PoliLEG usa o paradigma de Unidade de Controle e Fluxo de Dados. Na Figura 1 podemos observar o diagrama de blocos, com as interligações entre a unidade de controle e o fluxo de dados. É importante lembrar que as memórias de instruções e de dados são componentes externos ao processador.

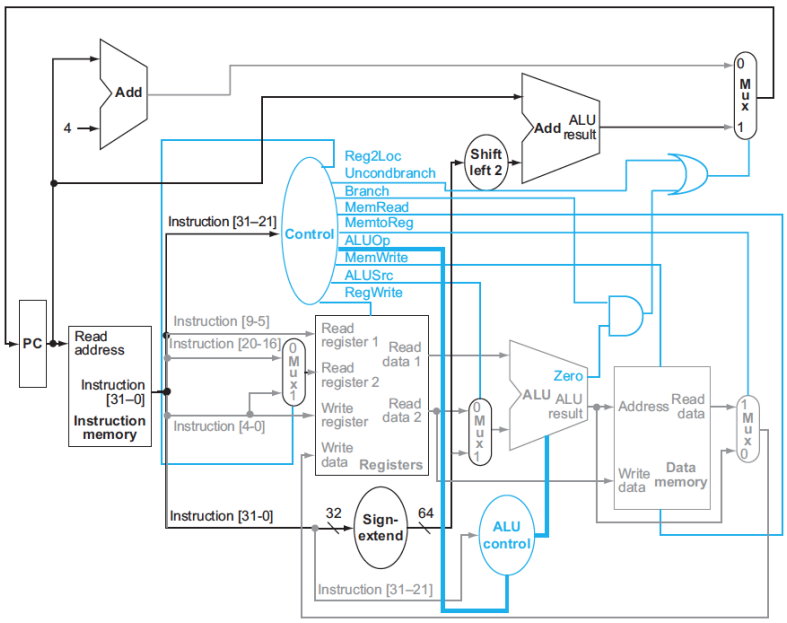

Figura 1: Diagrama do PoliLEG

Para facilitar o projeto dos circuitos deste projeto, será bom rever o funcionamento e o formato das instruções do PoliLEG. A leitura recomendada são os Capítulos 2 (Seções 2.1 a 2.10), 4 (Seções 4.1 a 4.4) e o Apêndice A5 do livro texto da disciplina [1]. As 8 instruções do PoliLEG a serem implementadas são apresentadas na Tabela 1. Nesta tabela você encontrará a sintaxe em Linguagem Assembly e a descrição da ação executada em cada instrução, além do tipo de formato e o opcode.

[1] D.A. Patterson and J.L. Hennessy. *Computer Organization and Design ARM Edition: The Hardware Software Interface*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2016

Instrução	Formato	opcode	Sintaxe/Descrição
LDUR	D	11111000010	LDUR Rt, [Rn + address] Rt = Memory[Rn + address]
STUR	D	11111000000	STUR Rt, [Rn + address] Memory[Rn + address] = Rt
CBZ	CB	10110100	CBZ Rt, address if (Rt == 0) goto (PC + address)
B	B	000101	B address goto (PC + address))
ADD	R	10001011000	ADD Rd, Rn, Rm Rd = Rn + Rm
SUB	R	11001011000	SUB Rd, Rn, Rm Rd = Rn - Rm
AND	R	10001010000	AND Rd, Rn, Rm Rd = Rn AND Rm (bit a bit)
ORR	R	10101010000	ORR Rd, Rn, Rm Rd = Rn OR Rm (bit a bit)

Tabela 1: Instruções do PoliLEG

A composição dos bits das instruções depende de cada formato. O formato das instruções do tipo R (ADD, SUB, AND e ORR) é apresentado na Tabela 2, enquanto que os formatos das instruções do tipo D (LDUR e STUR), do tipo CB (CBZ) e da instrução B estão nas Tabelas 3, 4 e 5, respectivamente. Note que cada formato tem campos diferentes, mas todos têm o mesmo comprimento de 32 bits.

Campo	opcode	Rm	shamt	Rn	Rd
Posições (bit)	31-21	20-16	5-10	9-5	4-0

Tabela 2: Formato R

Campo	opcode	address	0	Rn	Rt
Posições (bit)	31-21	20-12	11-10	9-5	4-0

Tabela 3: Formato D

Campo	opcode	address	Rt
Posições (bit)	31-24	23-5	4-0

Tabela 4: Formato CBZ

Campo	opcode	address
Posições (bit)	31-26	25-0

Tabela 5: Formato B

Atividades

Projeto 5, Atividade 1

P5A1 (3 pontos) Implemente um componente em VHDL correspondente à unidade funcional `Sign-extend`, respeitando a entidade da Figura 2. Você deverá submeter ao Juiz o arquivo VHDL da descrição (entidade e arquitetura) do `signExtend`, incluindo eventuais componentes adicionais que utilizar.

A unidade funcional `Sign-extend` é usada nas instruções de salto condicional (CBZ) e incondicional (B), além de transferência entre a memória de dados e o banco de registradores (LDUR e STUR). Esta

```
entity signExtend is
  port(
    i: in        bit_vector(31 downto 0); -- input
    o: out       bit_vector(63 downto 0) -- output
    );
end signExtend;
```

Figura 2: Entidade para Sign-extend

unidade funcional recebe a instrução de 32 bits (entrada i) e converte o respectivo campo `address` num inteiro de 64 bits (saída o), mantendo o sinal de `address`. Tanto `address` como a extensão para 64 bits são representados em Complemento de 2. Por exemplo, na instrução CBZ o campo `address` tem 19 bits de comprimento e, se `address =` 40003_{16}, então `Sign-extend(address) =` $FFFFFFFFFFC0003_{16}$.

Recomendação: ao fazer o testbench para o `Sign-extend`, prepare testes para as 4 instruções que usam o `Sign-extend`: LDUR, STUR, CBZ e B.

Projeto 5, Atividade 2

P5A2 (1 ponto) Esta atividade tem como objetivo substituir a operação "SLT", que foi implementada na ULA do Projeto 4, pela operação "pass B". Para isso, modifique a ULA de 1 bit e a ULA de tamanho genérico de *size* bits, onde *size* era um `generic` na entidade `alu`, para que tenham as operações descritas nas Tabelas 6 e 7, respectivamente. A operação "pass B" significa simplesmente que a entrada b deve ser copiada para a saída da ULA, sem ser alterada por nenhuma operação lógica ou aritmética. Projete as ULAs de 1 e de *size* bits, cujos sinais de entrada e de saída devem ser de acordo com a descrição das entidades mostradas nas Figuras 3 e 4. Você deverá submeter ao Juiz o arquivo VHDL da descrição (entidade e arquitetura) da `alu1bit` e da `alu`, ambas no mesmo arquivo, incluindo eventuais componentes adicionais que utilizar, também no mesmo arquivo. Lembre-se de que o projeto da ULA de tamanho genérico *size* deve utilizar a ULA de 1 bit, atualizada com a operação "pass B", como componente em sua descrição VHDL.

operation	Descrição
00	a AND b
01	a OR b
10	sum
11	b

Tabela 6: operation para ULA de 1 bit

op	Descrição
0000	$A\&B$, bit a bit
0001	$A\|B$, bit a bit
0010	$A + B$
0110	$A - B$
0111	$PassB$
1100	$\overline{A\|B}$, bit a bit

Tabela 7: operation para ULA de *size* bits

```
entity alu1bit is
  port (
    a, b, less, cin: in bit;
    result, cout, set, overflow: out bit;
    ainvert, binvert: in bit;
    operation: in bit_vector(1 downto 0)
  );
end entity;
```

Figura 3: Entidade da ULA de 1 bit

```
entity alu is
  generic (
    size : natural := 64
  );
  port (
    A, B: in bit_vector(size-1 downto 0); -- inputs
    F: out bit_vector(size-1 downto 0); -- output
    S: in  bit_vector(3 downto 0); -- op selection
    Z: out bit; -- zero flag
    Ov: out bit; -- overflow flag
    Co: out bit -- carry out
    );
end entity alu;
```

Figura 4: Entidade da ULA de *size* bits

Projeto 5, Atividade 3

P5A3 (2 pontos) O objetivo desta atividade é projetar o componente da Unidade de Controle denominado `ALU Control`. A descrição da entidade referente a `ALU Control` está na Figura 5. Este componente tem a função de configurar a operação a ser feita na `ULA` de tamanho genérico (que no caso do PoliLEG será de 64 bits), conforme descrito na tabela da Figura 6. Observe que o `ALU Control` recebe 2 entradas: o sinal de 2 bits `aluop` e o campo de 11 bits `opcode` das instruções do Tipo R. No caso das instruções LDUR, STUR e CBZ o campo `opcode` será ignorado pelo `ALU Control`. A saída do `ALU Control` é o sinal de 4 bits `aluCtrl`. A instrução B não utiliza o `ALU Control` e, por isso, a saída `aluCtrl` será arbitrária neste caso. Você deverá submeter ao Juiz o arquivo VHDL da descrição (entidade e arquitetura) do `alucontrol`, incluindo eventuais componentes adicionais que utilizar.

```
entity alucontrol is
  port (
    aluop: in bit_vector(1 downto 0);
    opcode: in bit_vector(10 downto 0);
    aluCtrl: out bit_vector(3 downto 0)
  );
end entity;
```

Figura 5: Entidade ALU Control

Note que a última coluna da Figura 6 se chama "ALU control input" porque esse sinal servirá como uma das entradas de controle da ULA, mas do ponto de vista da entidade `alucontrol`, é a **saída** correspondente à porta `aluCtrl`.

Instruction	ALUOp	Instruction operation	Opcode field	Desired ALU action	ALU control input
LDUR	00	load register	XXXXXXXXXXX	add	0010
STUR	00	store register	XXXXXXXXXXX	add	0010
CBZ	01	compare and branch on zero	XXXXXXXXXXX	pass input b	0111
R-type	10	ADD	10001011000	add	0010
R-type	10	SUB	11001011000	subtract	0110
R-type	10	AND	10001010000	AND	0000
R-type	10	ORR	10101010000	OR	0001

Figura 6: Sinais de entrada e de saída do ALU Control

P5A4 (4 pontos) Implemente a Unidade de Controle do processador monociclo PoliLEG, seguindo a descrição da entidade apresentada na Figura 7. Você deverá submeter ao Juiz o arquivo VHDL da descrição (entidade e arquitetura) do `controlunit`, incluindo no mesmo arquivo eventuais componentes adicionais que utilizar.

Projeto 5, Atividade 4

```
entity controlunit is
  port (
    -- To Datapath
    reg2loc : out bit;
    uncondBranch : out bit;
    branch: out bit;
    memRead: out bit;
    memToReg: out bit;
    aluOp: out bit_vector(1 downto 0);
    memWrite: out bit;
    aluSrc: out bit;
    regWrite: out bit;
    -- From Datapath
    opcode: in bit_vector(10 downto 0)
  );
end entity;
```

Figura 7: Entidade Control Unit

Como vimos em exemplos anteriores na disciplina, quando estamos projetamos uma Unidade de Controle de um sistema digital, tipicamente pensamos em uma máquina de estados finita. Porém, nesse caso, pelo fato de ser um processador monociclo, a Unidade de Controle (UC) do PoliLEG se reduz a um circuito combinatório, que recebe como entrada o `opcode` de 11 bits e fornece na saída os 9 sinais de controle apresentados na Tabela 8. Isso facilita o projeto e, por essa razão, não precisamos especificar a UC do PoliLEG como um diagrama ASM ou equivalente.

A tabela está preenchida apenas com os valores correspondentes à instrução LDUR. Caberá a você preencher os valores dos sinais de controle para as demais instruções. Com a tabela completa será possível fazer o projeto do circuito combinatório da UC. Nos casos das instruções CBZ e B, cujos campos de `opcode` tem menos que 11 bits, o circuito da UC deverá ignorar os bits em excesso.

Controle	LDUR	STUR	CBZ	B	Tipo R
reg2loc	x				
uncondBranch	0				
branch	0				
memRead	1				
memToReg	1				
aluOp	00				
memWrite	0				
aluSrc	1				
regWrite	1				

Tabela 8: Tabela dos sinais da UC

Recomendação: ao fazer o testbench para a Unidade de Controle, prepare testes para todas as 8 instruções do PoliLEG.

Instruções para Entrega

Há um link específico no e-Disciplinas para a submissão de cada atividade deste projeto. Acesse-o somente quando estiver confortável para enviar sua solução. Em cada atividade, você pode enviar apenas um único arquivo codificado em UTF-8. O nome do arquivo não importa, mas sim a descrição VHDL que está dentro. A entidade da síntese (implementação) que você submeterá precisa estar, obrigatoriamente, de acordo com o que foi definido/especificado no enunciado e deve ser idêntica na sua solução ou o juiz não irá processar seu arquivo.

Qualquer editor de código moderno suporta UTF-8 (e.g. Atom, Sublime, Notepad++, etc)

O juiz corrigirá imediatamente sua submissão e retornará com a nota. Caso não esteja satisfeito com a nota, você pode enviar novamente e somente a maior nota para aquela atividade será válida. A nota para este projeto é composta pela soma ponderada das notas dadas pelo juiz para cada atividade. Faça seu *testbench* e utilize um simulador de VHDL para validar sua solução antes de postá-la para o juiz.

A quantidade de submissões para essas atividades foi limitada a 5 por atividade.

Não use a biblioteca std_logic_1164, a textio e qualquer outra biblioteca não padronizada, para minimizar possível fonte de problemas. Também se certifique que não imprime nada na saída da simulação.

Atenção: não atualize a página de envio e não envie a partir de conexões instáveis (e.g. móveis) para evitar que seu arquivo chegue corrompido no juiz.

Referências

[1] D.A. Patterson and J.L. Hennessy. *Computer Organization and Design ARM Edition: The Hardware Software Interface*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2016.