Processador RISC

Projeto e implementação em SystemC

Vitor Greati [1] Vinicius Campos[1] Artur Curinga[1]

[1]Instituto Metrópole Digital
UFRN

Dezembro, 2016

Conteúdo

Palavra de instrução

A programação consiste na listagem de instruções no formato:

OPCODE	OD	F1	F2
4 bits	9 bits	9 bits	9 bits

Onde:

- OPCODE Código da operação
- OD Operando Destino (D)
- F1 Operando Fonte 1 (F1)
- F2 Operando Fonte 2 (F2)

Conjunto de instruções

Instrução	Ação	Exemplo
LRI	$R[D] \leftarrow F1$	LRI 1 27
AND	$D \leftarrow F1 \& F2$	AND 1 2 3
OR	$D \leftarrow F1 \mid F2$	OR 1 2 3
XOR	$D \leftarrow F1 \wedge F2$	XOR 1 2 3
NOT	$D \leftarrow \bar{F1}$	NOT 1 2
CMP	$Z = 1$, se $F1 == F2$; $N = 1$, se $F1 < F2$; $R[OD] = 0$, se $F1 < F2$; $R[OD] = 1$, se $F1 == F2$, $R[OD] = 2$, se $F1 > F2$	CMP 1 2 3
ADD	$R[D] \leftarrow F1 + F2$	ADD 1 2 3
SUB	$R[D] \leftarrow F1 - F2$	SUB 1 2 3
LD	$R[D] \leftarrow MEM[F1]$	LD 1 2
ST	$MEM[D] \leftarrow R[F1]$	ST 1 2
J	$CP \leftarrow F1$	J 23
JN	$CP \leftarrow F1$ if $N == 1$	JN 23
JZ	$CP \leftarrow F1$ if $Z == 1$	JZ 23

Tabela: Conjunto de instruções.

Modos de endereçamento

Permitem-se três modos de endereçamento:

Direto

É fornecido o endereço da memória de dados que se deseja manipular. Apenas instruções **LD** e **ST** podem referenciar diretamente a memória.

Registrador direto

É fornecido o endereço do registrador com que se deseja trabalhar.

Registrador imediato

É fornecido o endereço do registrador e um valor imediato a ser inserido nele.

Memórias

Registradores

Há 32 registradores disponíveis para palavras de 32 bits.

Memória de instruções

256 palavras de 32 bits.

Memória de dados

512 palavras de 32 bits.

Barramentos

Controle

Transmite os sinais de controle para a parte operativa.

Dados

Transmite as palavras de dados de 32 bits.

Endereços

Transmite os endereços utilizados nas leituras e escritas em memória, com largura de 8 bits.

Pipeline

- Apenas dois estágios: entre a busca e a execução;

Pipeline

- Apenas dois estágios: entre a busca e a execução;
- Um registrador guarda a palavra de instrução decodificada;

Pipeline

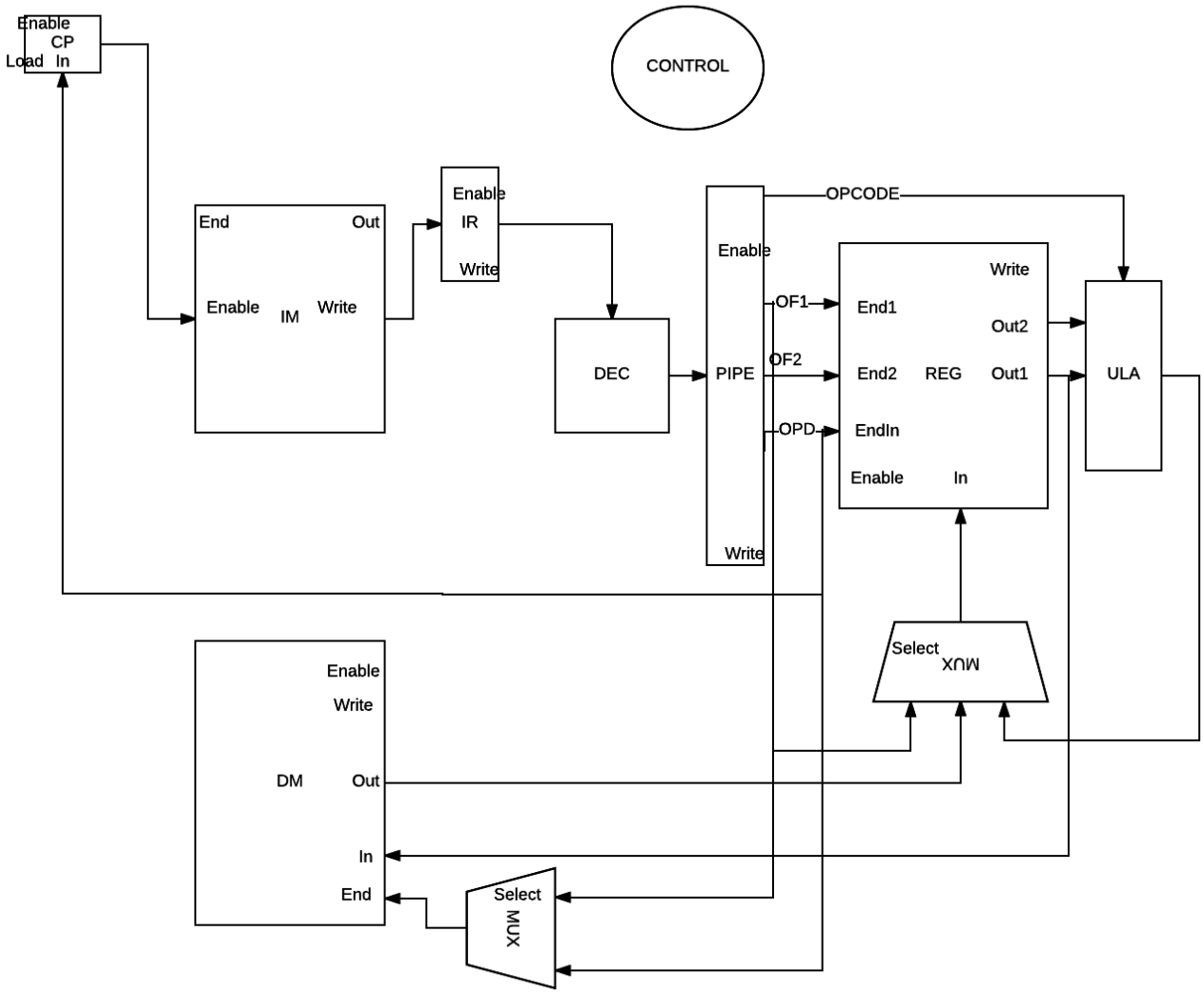
- Apenas dois estágios: entre a busca e a execução;
- Um registrador guarda a palavra de instrução decodificada;
- É pessimista: sempre que ocorre uma intrução de *jump*, desconsidera a instrução pré-carregada;

Pipeline

- Apenas dois estágios: entre a busca e a execução;
- Um registrador guarda a palavra de instrução decodificada;
- É pessimista: sempre que ocorre uma intrução de *jump*, desconsidera a instrução pré-carregada;
- Ainda que simples, demonstrou redução visível nos ciclos, como mostrado nos testes mais adiante.

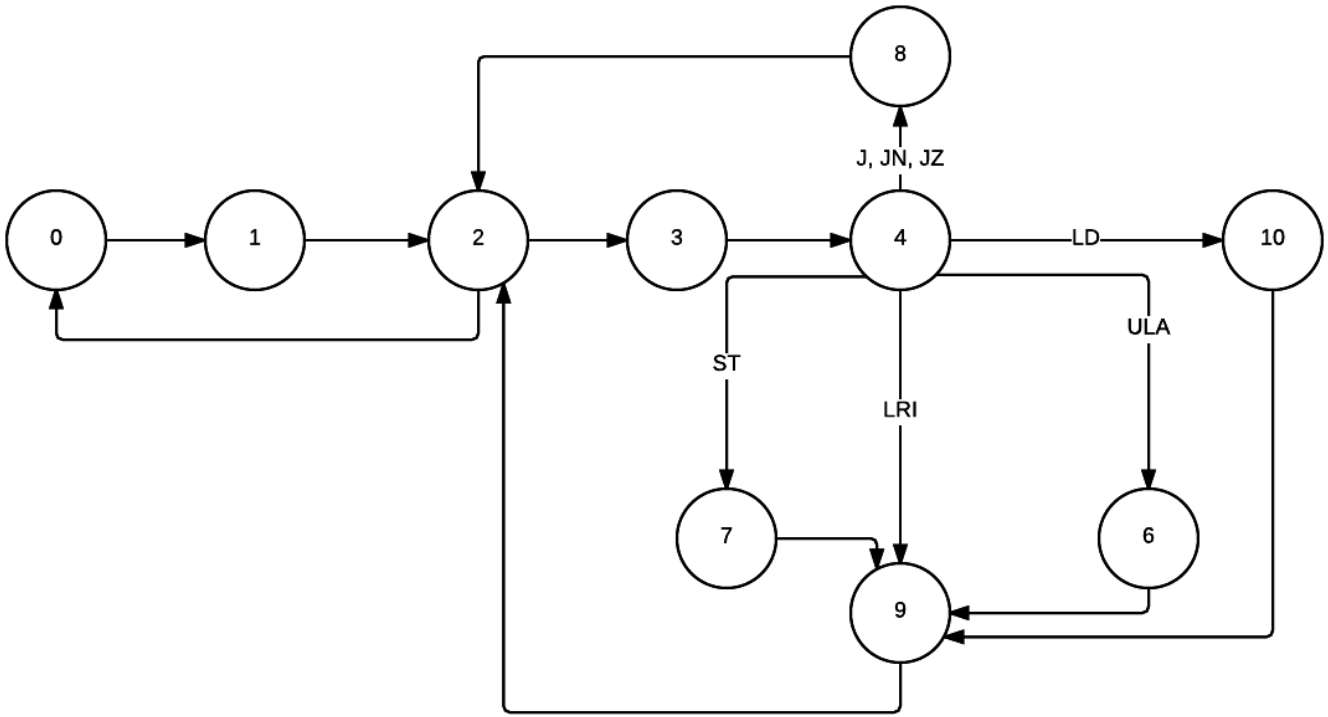
Conteúdo

Parte operativa



Parte de controle

A parte de controle consiste em um bloco que recebe a palavra de instrução decodificada e utiliza uma máquina de estados para gerar os sinais de controle adequados a cada microinstrução.



Estados

A tabela abaixo descreve o que ocorre em cada estado:

Estado	Ações
0	Prepara para escrever a instrução no barramento.
1	Prepara para escrever a instrução no RI.
2	Caso o pipeline não seja reiniciado, prepara para escrever no registrador de pipeline. Caso seja, envia para o estado 0.
3	Desabilita escrita no pipeline, prepara nova instrução para o barramento do RI (pipelining).
4	Prepara a execução da instrução de fato e a escrita no RI da próxima instrução (pipelining).
6	Desabilita sinais após a execucação da ULA.
7	Desabilita sinais após a escrita na memória.
8	Desabilita escrita no CP e envia para o estado 2, para reiniciar o processo.
9	Desabilita sinais após guardar os resultados no banco de registradores.
10	Prepara execução da operação de LD.

Conteúdo

Implementação

- Biblioteca SystemC 2.3.1;

Implementação

- Biblioteca SystemC 2.3.1;
- Cada módulo implementado separadamente, mantendo boa organização;

Implementação

- Biblioteca SystemC 2.3.1;
- Cada módulo implementado separadamente, mantendo boa organização;
- O programa recebe um arquivo com um algoritmo escrito segundo as instruções da arquitetura;

Implementação

- Biblioteca SystemC 2.3.1;
- Cada módulo implementado separadamente, mantendo boa organização;
- O programa recebe um arquivo com um algoritmo escrito segundo as instruções da arquitetura;
- Para executar: `./processador_run instrucoes.txt`

Conteúdo

Execuções individuais

Primeiro, testamos cada instrução separadamente, obtendo o número de ciclos que cada uma toma, **considerando a identificação de parada**:

Instrução	Ciclos
AND	9
OR	9
XOR	9
NOT	9
CMP	9
ADD	9
SUB	9
LD	9
ST	9
J	11
JN	11
JZ	11
LRI	9

Tabela: Ciclos para cada instrução.

Algoritmos para testes

Para testar o funcionamento do processador e do pipeline, os seguintes algoritmos foram utilizados:

3x6

```
LRI 1 3
LRI 2 6
LRI 3 1
LRI 4 0
LRI 5 0
CMP 9 4 1
JZ 10
ADD 5 5 2
SUB 1 1 3
J 5
ST 1 5
```

Algoritmos para testes

Computa o décimo elemento da sequência de Fibonacci e o armazena na memória:

Fib(10)

```
LRI 3 1
LRI 6 0
LRI 7 1
LRI 1 0
LRI 2 1
LRI 4 1
LRI 5 10
SUB 5 5 7
CMP 9 5 4
JN 15
ADD 3 1 2
ADD 1 6 2
ADD 2 6 3
ADD 4 7 4
J 8
ST 1 3
```

Algoritmos para testes

Realiza todas as operações lógicas possíveis com o processador em operandos advindos da memória de dados:

Logic

```
LRI 0 2
LRI 1 3
ST 0 0
ST 1 1
LD 2 0
LD 3 1
ADD 4 3 2
SUB 5 4 3
AND 6 1 2
OR 7 1 2
XOR 8 1 2
NOT 9 1
```

Resultados dos algoritmos

Algoritmo	Ciclos sem pipeline	Ciclos com pipeline
3x6	149	115
Fib(10)	492	374
Logic	82	58

Tabela: Resultados de execução dos algoritmos de teste.

Conteúdo

Conclusões

- Implementou-se um processador simples, mas funcional, unindo conceitos da disciplina de OAC e de Circuitos Lógicos;

Conclusões

- Implementou-se um processador simples, mas funcional, unindo conceitos da disciplina de OAC e de Circuitos Lógicos;
- Ainda com um *pipeline* simplificado, foi visível a redução no ciclo de instruções, validando o poder dessa técnica;

Conclusões

- Implementou-se um processador simples, mas funcional, unindo conceitos da disciplina de OAC e de Circuitos Lógicos;
- Ainda com um *pipeline* simplificado, foi visível a redução no ciclo de instruções, validando o poder dessa técnica;
- O projeto consolidou os conceitos aprendidos nas aulas teóricas, além de mostrar como se pode descrever *hardware* de forma ainda mais abstrata.