

Wilson Cazarré Sousa

**Desenvolvimento e implementação de um
processador compatível com a Arquitetura 6502
em FPGA**

São José dos Campos - Brasil

Abril de 2024

Wilson Cazarré Sousa

Desenvolvimento e implementação de um processador compatível com a Arquitetura 6502 em FPGA

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Laboratório de Sistemas Computacionais: Arquitetura e Organização de Computadores.

Docente: Prof. Dr. Tiago de Oliveira

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Abril de 2024

Resumo

Esse trabalho irá apresentar o desenvolvimento de uma microprocessador capaz de executar um subconjunto das instruções do microprocessador 6502. O 6502 foi um microprocessador lançado em 1975 e redefiniu o que um computador pessoal podia fazer, sendo usado em muitos dispositivos populares da época como o NES e o Apple I. O desenvolvimento do mesmo será realizado em VHDL e implementado em um FPGA. O trabalho também apresenta os detalhes da arquitetura implementada, bem como seu *datapath*, modos de endereçamento e ciclos de execução. **Palavras-chaves:** 6502. NES. FPGA. Verilog.

Lista de ilustrações

Figura 1 – Arquitetura de von Neumann	9
Figura 2 – Endereçamento imediato	12
Figura 3 – Endereçamento absoluto. Note que o primeiro byte na memória é o menos significativo	13
Figura 4 – Endereçamento absoluto - Deslocado em X (ou Y)	14
Figura 5 – Endereçamento <i>Zero-Page</i>	15
Figura 6 – Endereçamento relativo	16
Figura 7 – Endereçamento indireto	17
Figura 8 – Datapath do 6502 implementado	22

Lista de tabelas

Tabela 1 – Tamanho da instrução por modo de endereçamento	15
Tabela 2 – Conjunto de instruções	19

Sumário

1	INTRODUÇÃO	7
1.1	Motivação	7
1.2	Metodologia	7
1.3	Objetivos	7
2	FUNDAMENTAÇÃO TEÓRICA	9
2.1	Visão geral de um sistema computacional	9
2.1.1	Arquitetura de von Neumann	9
2.2	Microprocessador 6502	10
2.2.1	Arquitetura Original	10
2.2.2	Registrador de Status (SR)	10
2.2.3	Contador de Programa (PC)	11
2.2.4	<i>Stack Pointer</i> (SP)	11
2.2.5	Modos de endereçamento	11
2.2.5.1	Endereçamento imediato	11
2.2.5.2	Endereçamento absoluto	12
2.2.5.3	Endereçamento absoluto - Deslocado em X (ou Y)	12
2.2.5.4	Endereçamento <i>Zero-Page</i>	12
2.2.5.5	Endereçamento relativo	12
2.2.5.6	Endereçamento indireto	13
2.2.6	Endereçamento <i>Zero-Page</i> , deslocado em X (ou Y)	13
2.3	RTL - <i>Register-Transfer Logic</i>	13
3	DESENVOLVIMENTO	19
3.1	Conjunto de instruções	19
3.2	O <i>datapath</i> da implementação	20
3.3	Unidades funcionais	21
3.3.1	Registradores de propósito geral (AC, X, Y)	21
3.3.2	Contador de Programa (PCH e PCL)	21
3.3.3	Registrador de Dados (DR)	21
3.3.4	Registrador de Status (P)	21
3.3.5	Unidade de controle	22
3.3.6	Geração de <i>Clock</i> (CLK)	23
3.3.7	Registrador do barramento de endereço (ABL e ABH)	23
3.3.8	Unidade Lógica aritmética (ALU)	23

4	RESULTADOS OBTIDOS E DISCUSSÕES	25
5	CONSIDERAÇÕES FINAIS	27
	REFERÊNCIAS	29
	APÊNDICES	31

1 Introdução

Durante a disciplina de Laboratório de Sistemas Computacionais: Arquitetura e Organização de Computadores, ofertada no Instituto de Ciência e Tecnologia da UNIFESP, é proposto que os discentes escolham uma arquitetura de processador para realizar sua implementação em um dispositivo FPGA. Esse relatório irá apresentar a fundamentação, bem como todo o processo de desenvolvimento de um sistema computacional baseado no microprocessador 6502.

1.1 Motivação

mandaram eu fazer esse trabalho ou eu seria reprovado na matéria

1.2 Metodologia

O trabalho apresentado será desenvolvido no *software* Quartus©Prime da Intel e implementado na linguagem de descrição de *hardware* VHDL. O circuito será implementado usando a abstração de *Register-Transfer Level* onde o fluxo de dados no circuito é representado como registradores e as unidades de lógica combinacional que determinam seus estados. O projeto então será testado em bancada onde deverá ser capaz de executar qualquer tipo de lógica definida como "computável"(ou seja, ter a mesma funcionalidade de uma Máquina de Turing).

1.3 Objetivos

Geral

Desenvolver uma CPU capaz de executar um subconjunto das instruções da família MCS650X. A implementação deverá ser feita em VHDL e sintetizada pelo *software* Quartus©Prime da Intel.

Específico

- Definir o subconjunto de instruções;
- Desenvolver uma unidade lógica e aritmética;
- Desenvolver os registradores do processador;

- Desenvolver a unidade de controle;
- Integrar os registradores, a unidade de controle e a unidade lógica e aritmética;
- Desenvolver casos testes para o processador;
- Testar o processador.

2 Fundamentação Teórica

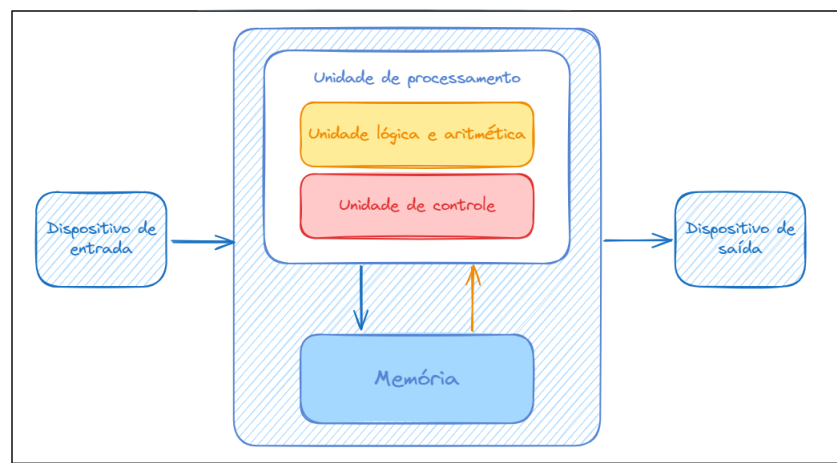
2.1 Visão geral de um sistema computacional

2.1.1 Arquitetura de von Neumann

A Arquitetura de von Neumann foi proposta por um grupo de engenheiros liderados por **Jonh von Neumann** em 1945 (1). O design descrito pelo documento tem como objetivo de ser um caso generalizado para um computador digital e é composto dos seguintes componentes (Figura 1):

1. Uma unidade capaz de executar operações aritméticas ;
2. Uma unidade lógica de controle;
3. Uma memória de "tamanho considerável". Essa memória guarda as instruções e dados do programa.
4. Dispositivos de entrada e saída.

Figura 1 – Arquitetura de von Neumann



Fonte: Autoria própria

Nesse tipo de arquitetura, o processador pode ler apenas uma instrução OU dado por vez. Isso porque ambas as leituras ocorrem por meio do mesmo barramento.

A arquitetura apresentada na seção 2.2 segue exatamente os mesmos princípios apresentados aqui: um único barramento de endereços no qual o processador pode comunicar

qual o endereço da informação que está tentando acessar, e um barramento de dados por onde a informação se propaga.

É importante também destacar que ainda que seja possível fisicamente separar as memórias de dados e de programa (o que de fato é algo que será feito) durante esse trabalho, do ponto de vista do processador essa não é uma diferença efetiva. O processador apenas consegue "enxergar" um único barramento de dados e de endereço, não importa quais dispositivos estejam conectados diretamente.

2.2 Microprocessador 6502

O microprocessador 6502 é o segundo membro da família MCS650X. Essa família de microprocessadores de 8 bits foi lançada em 1975 pela *MOS Technology*. Os processadores dessa família apresentam o mesmo conjunto de instruções e modos de endereçamento, com pequenas diferenças em recursos e sua utilização (2). Por conta de sua eficácia e baixo custo, o microprocessador se popularizou rapidamente ao ser usado em diversos sistemas da época como *O Nintendo Entertainment System (NES)*, *Apple II*, *Commodore 64* e muitos outros.

2.2.1 Arquitetura Original

O microprocessador conta com um Barramento de Dados de 8 bits e um Barramento de endereços 16-bits. Qualquer operação que o processador precisa executar normalmente é iniciada colocando o endereço de acesso no Barramento de endereços e posteriormente lendo (ou escrevendo) um valor de 8-bits no Barramento de dados.

Internamente, 3 registradores de propósito geral podem ser usados.

- **Acumulador (A)**: Usado também para armazenar o resultado das operações lógicas e aritméticas;
- **Index X e Y**: Ambos os registradores podem ser usados para operações com modos de endereçamento especiais, que serão abordados mais a frente no relatório.

Além dos 3 registradores que podem ser acessados diretamente, o 6502 também possui alguns registradores usados por funções específicas do processador.

2.2.2 Registrador de Status (SR)

O **registrador de status** é responsável por armazenar *flags* usadas para o controle do fluxo de programa do processador. Elas normalmente são atualizadas durante operações lógicas, aritméticas e de transferência de dados.

- **Carry (C)**: Indica se a operação gerou um *carry*;
- **Negativo (N)**: Indica se a operação gerou um valor com o bit mais significativo ativo;
- **Overflow (V)**: Indica se a operação gerou um...;
- **Zero (Z)**: Indica se a operação gerou o valor zero;
- **Decimal (D)**: Indica se o processador está em modo aritmético decimal BCD;
- **Bloqueio de interrupções (I)**: Indica se o processador está ignorando as requisição de interrupções;
- **Break (B)**: Indica se a interrupção atual foi disparada via *software* pela instrução BRK, ao invés de uma interrupção via *hardware*.

2.2.3 Contador de Programa (PC)

O único registrador de 16-bits definido pela arquitetura. Esse registrador é responsável por manter o endereço de memória atualmente acessado pelo processador.

2.2.4 Stack Pointer (SP)

O *stack* é uma região de memória destinada para rápido acesso e escrita. A eficácia nessas operações vem do fato de que o processador utiliza o endereço no SP para saber exatamente onde a próxima leitura e escrita vai ocorrer. O registrador é incrementado ou decrementado de acordo após cada operação. O 6502 também utiliza o *stack* para armazenar os endereços de retorno quando subrotinas ou interrupções são executadas.

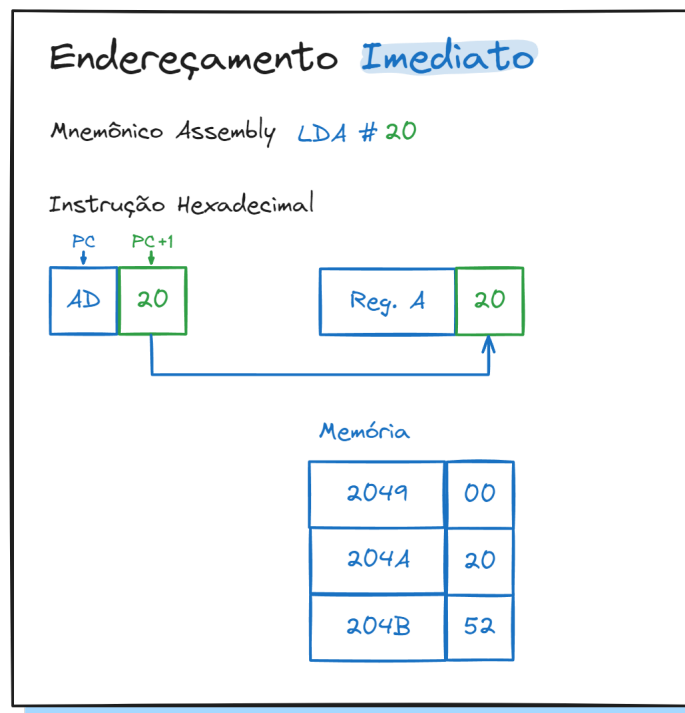
2.2.5 Modos de endereçamento

O 6502 é capaz de endereçar 65.536 bytes de memória. Qualquer operação ou estrutura de dados dentro do processador compartilham esse mesmo espaço de memória. O processador também providencia 13 diferentes métodos de calcular o endereço efetivo de memória na qual a operação vai ser executada (3). Na computação, chamamos esses métodos de **modos de endereçamento** e aqueles disponíveis no 6502 serão descritos aqui.

2.2.5.1 Endereçamento imediato

Nesse tipo de instrução o operando é usado imediatamente após a instrução ter sido lida. Nenhum acesso a memória ou cálculo é realizado (Figura 2). Essa tipo de instrução utiliza 2 bytes de memória.

Figura 2 – Endereçamento imediato



Fonte: Autoria própria

2.2.5.2 Endereçamento absoluto

Nesse tipo de instrução dois bytes são passados além do opcode. O processador usa esses bytes como um endereço de acesso a memória (Figura 3).

2.2.5.3 Endereçamento absoluto - Deslocado em X (ou Y)

Esse modo é uma variação do endereçamento absoluto: dois bytes são buscados da memória e usados como endereço de acesso. A diferença está no fato de que o valor do registrador (X ou Y) é somado ao endereço de acesso. (Figura 4).

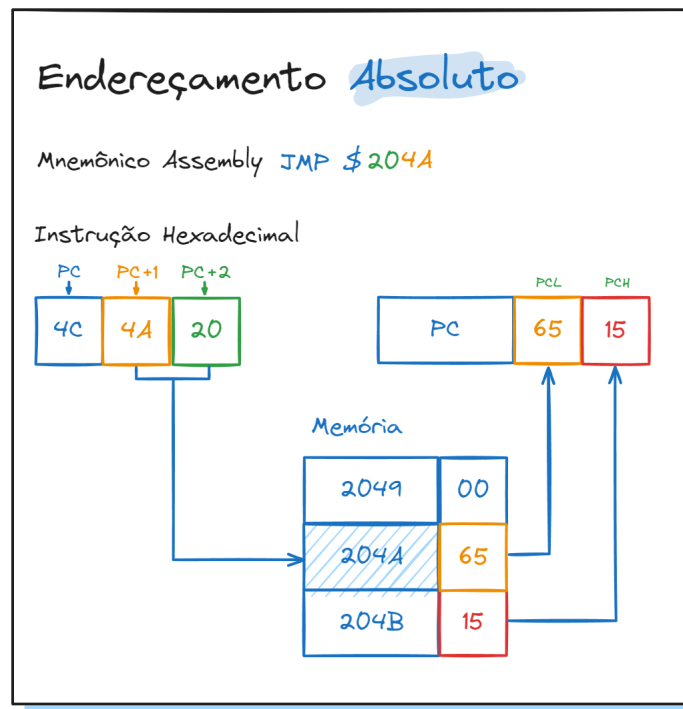
2.2.5.4 Endereçamento Zero-Page

Idêntico ao endereçamento absoluto, exceto que apenas um byte é lido da memória (o byte menos significativo). O byte mais significativo é inferido como 0 (Figura 5). Logo esse modo de endereçamento sempre retorna um dado localizado na primeira "página" da memória (os primeiros 256 bytes).

2.2.5.5 Endereçamento relativo

O endereçamento relativo é usado especificamente para instruções de *branch*. Nele, o operando contém um valor que será somado ao valor atual do Contador de Programa. Esse valor é então colocado de volta no Contador de programa para que a execução possa

Figura 3 – Endereçamento absoluto. Note que o primeiro byte na memória é o menos significativo



Fonte: Autoria própria

continuar a partir daí. É importante destacar que o byte de deslocamento passado nessa instrução pode possuir sinal positivo ou negativo. Isso significa pular para um endereço posterior ou anterior contando que o valor de deslocamento esteja entre -128 e 127.

2.2.5.6 Endereçamento indireto

Nesse tipo de endereçamento, o processador busca o endereço efetivo no endereço que foi passado pelo operando (Figura 7). Figura 7

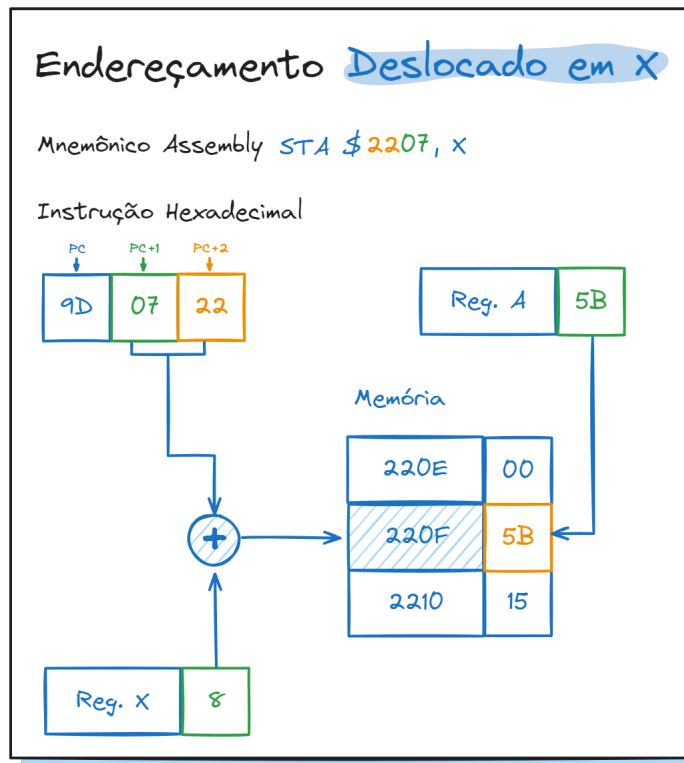
2.2.6 Endereçamento Zero-Page, deslocado em X (ou Y)

Idêntico ao Endereçamento Absoluto deslocado em X (ou Y), exceto que o endereço de acesso está sempre nos primeiros 256 bytes do espaço de memória (Figura 4).

2.3 RTL - Register-Transfer Logic

Quando tratamos do design de circuitos digitais complexos, é comum abstrairmos diferentes níveis do design com a intenção de tornar esses problemas mais simples de serem resolvidos.

Figura 4 – Endereçamento absoluto - Deslocado em X (ou Y)



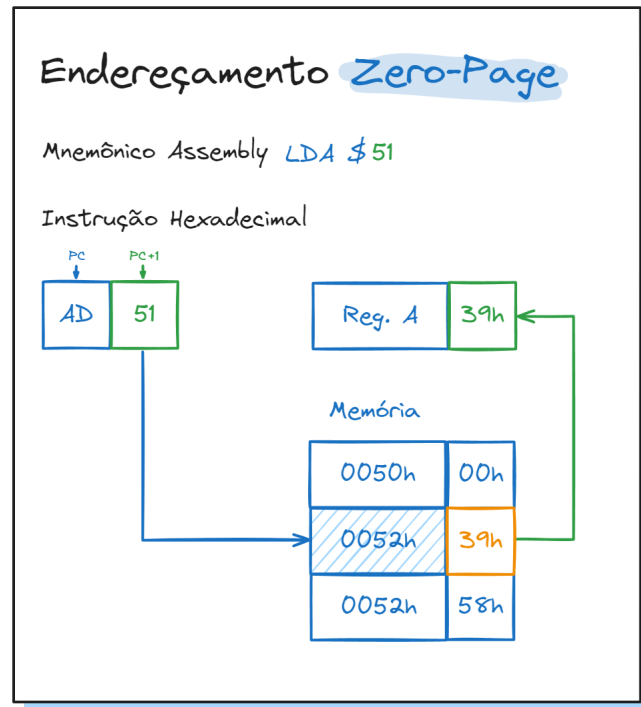
Fonte: Autoria própria

3 níveis diferentes de abstração são definidos por 4 na construção de circuitos digitais:

1. **Transistor Level:** Conectar transistores para construir componentes lógicos.
2. **Logic Level:** Utilizar-se de Portas Lógicas como bloco principal de construção para desenvolver circuitos combinacionais.
3. **Register-transfer Level:** Conectar uma rede de registradores e construir blocos que definem a lógica de transferência de estado entre esses registradores.

De maneira geral, no *Register-Transfer Level Design* (ou Design RTL) cada bloco do design deve desempenhar uma (e apenas uma) de duas possíveis funções:

1. **Lógica Combinacional:** São os blocos responsáveis pela computação do próximo estado. De maneira geral, esses blocos devem ser determinísticos e sempre apresentar a mesma saída para uma determinada entrada.
2. **Lógica Sequencial:** São blocos responsáveis por guardar e propagar o estado computado pelos blocos combinacionais de maneira síncrona.

Figura 5 – Endereçamento *Zero-Page*

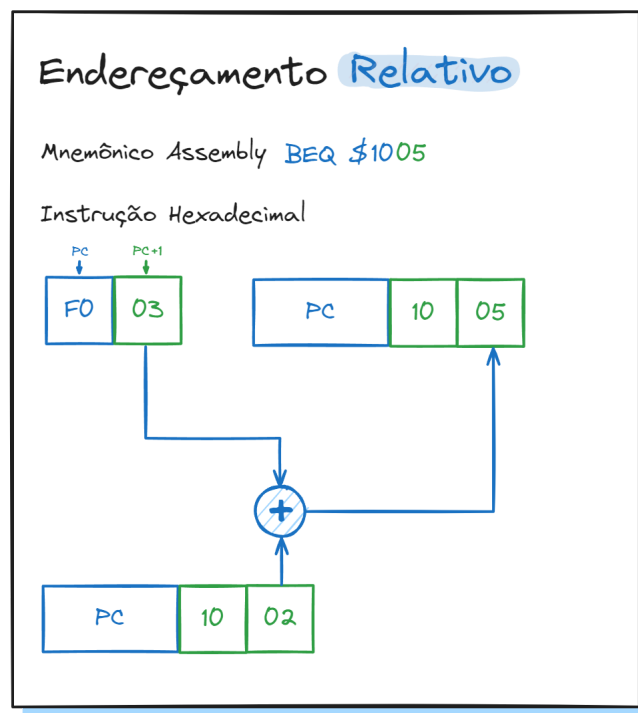
Fonte: Autoria própria

Tabela 1 – Tamanho da instrução por modo de endereçamento

Modo de endereçamento	Tamanho em bytes
Acumulador (A)	1
Absoluto (abs)	3
Absoluto, deslocado em X (abs, x)	3
Absoluto, deslocado em Y (abs, y)	3
Imediato (#)	2
Implícito (impl)	1
Indireto (ind)	3
Indireto, deslocado em X (X, ind)	2
Indireto, deslocado em Y (ind, Y)	2
Relativo (rel)	2
Zero-Page (zpg)	2
Zero-Page, deslocado em X (zpg, x)	2
Zero-Page, deslocado em Y (zpg, y)	2

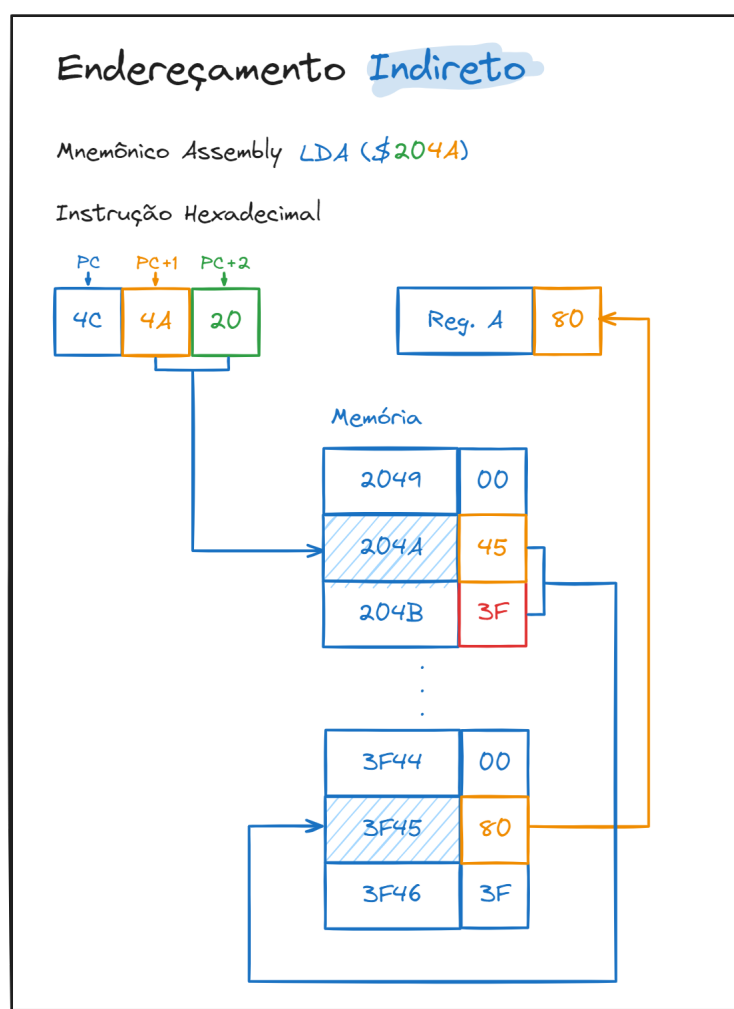
Fonte: Autoria Própria

Figura 6 – Endereçamento relativo



Fonte: Autoria própria

Figura 7 – Endereçamento indireto



Fonte: Autoria própria

3 Desenvolvimento

O desenvolvimento da CPU se deu em algumas etapas. Primeiramente o conjunto de instrução foi definido como um subconjunto da família MCS650X original. Depois disso foram escolhidos alguns modos de endereçamento e um *datapath* foi definido.

3.1 Conjunto de instruções

O conjunto de instruções apresentado na [Tabela 2](#) é apenas um subconjunto da família MCS650X original. Os mesmos *opcodes* da arquitetura original serão mantidos aqui (5).

Tabela 2 – Conjunto de instruções

Instruções de transferência				
Instrução	Opcode	Mod. End	Assembly	Operação
LDA	2f	imm	LDA imm	RegAC \leftarrow imm
	2f	abs	LDA addr	RegAC \leftarrow Mem[addr]
	2f	(abs, x)	LDA addr, x	RegAC \leftarrow Mem[addr + x]
LDX	2f	imm	LDX imm	RegX \leftarrow imm
	2f	abs	LDX addr	RegX \leftarrow Mem[addr]
	2f	(abs, x)	LDX addr, x	RegX \leftarrow Mem[addr + x]
LDY	2f	imm	LDY imm	RegY \leftarrow imm
	2f	abs	LDY addr	RegY \leftarrow Mem[addr]
	2f	(abs, x)	LDY addr, x	RegY \leftarrow Mem[addr + x]
STA	2f	abs	STA addr	Mem[addr] \leftarrow RegAC
	2f	(abs, x)	STA addr, x	Mem[addr + x] \leftarrow RegAC
STX	2f	abs	STX addr	Mem[addr] \leftarrow RegX
	2f	(abs, x)	STX addr, x	Mem[addr + x] \leftarrow RegX
STY	2f	abs	STY addr	Mem[addr] \leftarrow RegY
	2f	(abs, x)	STY addr, x	Mem[addr + x] \leftarrow RegY
Instruções lógicas e aritméticas				
Instrução	Opcode	Mod. End	Assembly	Operação
ADC	2f	imm	ADC imm	RegAC \leftarrow RegAC + imm + C
	2f	abs	ADC addr	RegAC \leftarrow RegAC + Mem[addr] + C
	2f	(abs, x)	ADC addr, x	RegAC \leftarrow RegAC + Mem[addr + x] + C
SBC	2f	imm	SBC imm	RegAC \leftarrow RegAC - imm - C
	2f	abs	SBC addr	RegAC \leftarrow RegAC - Mem[addr] - C
	2f	(abs, x)	SBC addr, x	RegAC \leftarrow RegAC - Mem[addr + x] - C
AND	2f	imm	AND imm	RegAC \leftarrow RegAC AND imm
	2f	abs	AND addr	RegAC \leftarrow RegAC AND Mem[addr]
	2f	(abs, x)	AND addr, x	RegAC \leftarrow RegAC AND Mem[addr + x]
EOR	2f	imm	EOR imm	RegAC \leftarrow RegAC XOR imm
	2f	abs	EOR addr	RegAC \leftarrow RegAC XOR Mem[addr]
	2f	(abs, x)	EOR addr, x	RegAC \leftarrow RegAC XOR Mem[addr + x]

ORA	2f	imm	ORA imm	RegAC <= RegAC OR imm
ORA	2f	abs	ORA addr	RegAC <= RegAC OR Mem[addr]
	2f	(abs, x)	ORA addr, x	RegAC <= RegAC OR Mem[addr + x]
Comparação				
Instrução	Opcode	Mod. End	Assembly	Operação
CMP	2f	imm	CMP imm	C, N, V, Z <= ACC - imm
	2f	abs	CMP addr	C, N, V, Z <= ACC - Mem[addr]
	2f	(abs, x)	CMP addr, x	C, N, V, Z <= ACC - Mem[addr - x]
Flags				
Instrução	Opcode	Mod. End	Assembly	Operação
SEC	2f	impl	SEC	C <= 1
CLC	2f	impl	CLC	C <= 0
Instruções de <i>branch</i>				
Instrução	Opcode	Mod. End	Assembly	Operação
BCC	2f	rel	BCC	branch on C = 0
BCS	2f	rel	BCS	branch on C = 1
BEQ	2f	rel	BEQ	branch on Z = 1
BMI	2f	rel	BMI	branch on N = 1
BNE	2f	rel	BNE	branch on Z = 0
BPL	2f	rel	BPL	branch on N = 0
BVC	2f	rel	BVC	branch on V = 0
BVS	2f	rel	BVS	branch on V = 1
Instruções de controle				
JMP	2f	abs	JMP HLL	PCL <= LL PCH <= HH
NOP	2f	impl	NOP	
HLT	2f	impl	HLT	

Fonte: Autoria Própria

3.2 O *datapath* da implementação

A Figura 8 mostra o *datapath* que será implementado durante esse trabalho.

O processador possui 3 registradores de propósito geral e é capaz de manipular 8-bits por ciclo de clock, por consequência toda instrução no 6502 leva mais de 1 ciclo para ser executada, considerando que o opcode consiste sempre de 8-bits.

A Figura 8 também divide os componentes em dois grupos principais:

- **Registradores externos:** São os registradores que o programador tem consciência que estão lá e pode, por meio do conjunto de instruções, interagir com eles de maneira direta ou indireta.

- **Microarquitetura interna:** São os componentes internos que não são diretamente definidos pela arquitetura MCS650X, são invisíveis do ponto de vista do programador mas são vitais para o funcionamento do processador.

O processador possui um Barramento de Endereços de 16-bits, isso significa que ele se comunica com até 65,536 diferentes endereços. Esses diferentes endereços serão mencionados daqui em diante como o Espaço de Memória (EM) de 6502.

3.3 Unidades funcionais

Essa seção apresenta uma breve descrição de cada componente no *datapath*.

3.3.1 Registradores de propósito geral (AC, X, Y)

Esses registradores de 8-bits que podem ser acessados diretamente utilizando suas respectivas instruções de *Load* e *Store*. Além disso, eles desempenham funções específicas no processador:

- **Acumulador (AC):** Toda operação lógica e aritmética tem como base o valor armazenado nesse registrador, além disso o resultado dessas operações também é diretamente armazenado aqui.
- **X e Y:** Esses registradores armazenam o valor de deslocamento das instruções com os modos de endereçamento deslocados.

3.3.2 Contador de Programa (PCH e PCL)

O contador de programa é usado para endereçar o espaço de 16-bits de memória disponível para o processador. Por conta do processador conseguir manipular apenas 8-bits por vez, utiliza-se 2 registradores de 8 bits para armazenar o endereço completo. O programador pode manipular seu valor por meio das instruções de *jump* e *branch*.

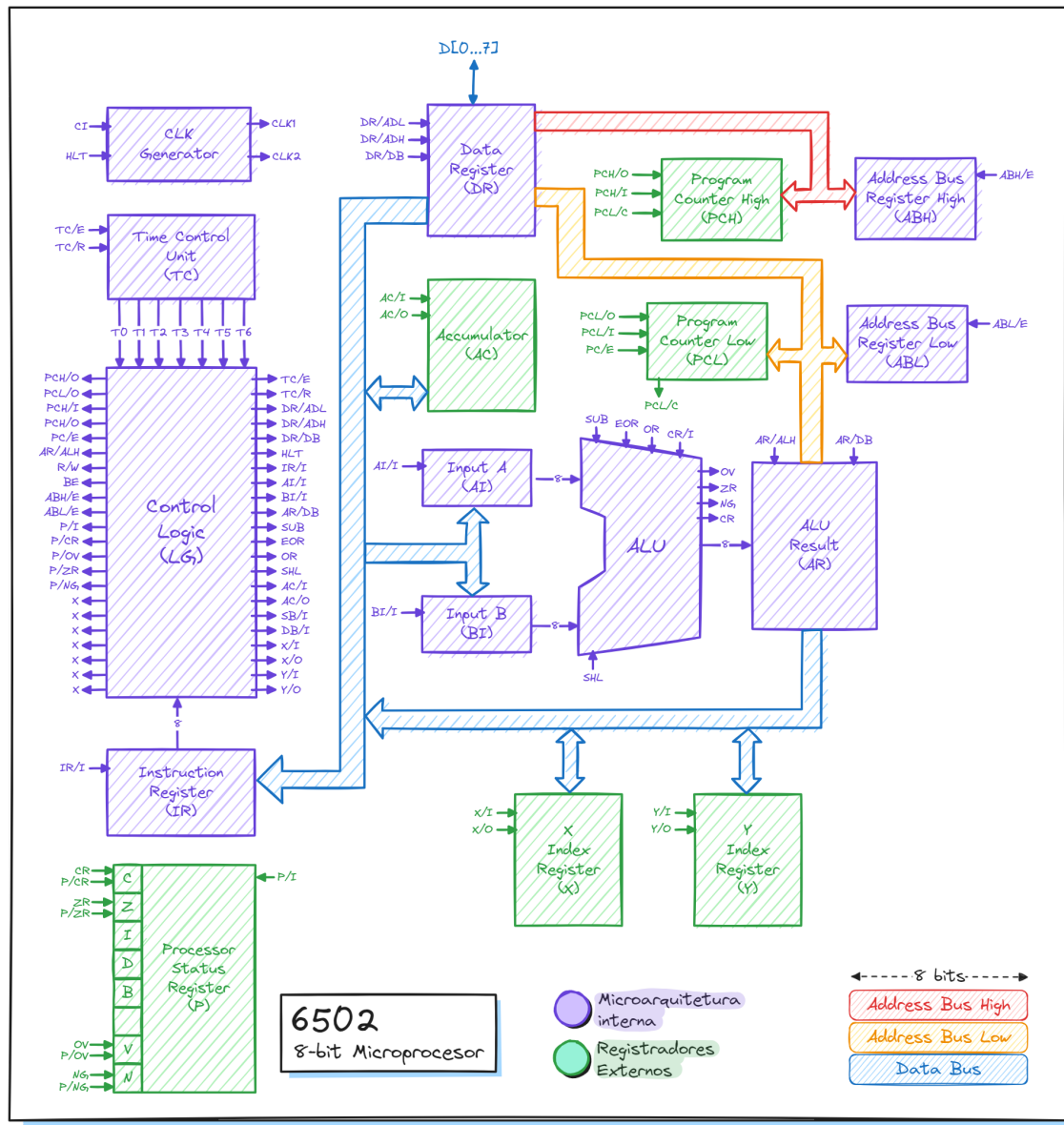
3.3.3 Registrador de Dados (DR)

O registrador de dados é responsável por controlar a entrada de informações no processador e distribuir para um dos 3 barramentos disponíveis.

3.3.4 Registrador de Status (P)

Essa implementação difere da apresentada em 2.2.2. As *flags* I, D e B não serão implementadas. Os outros valores serão controlados pelo resultado de operações aritméticas, *loads* e *stores*.

Figura 8 – Datapath do 6502 implementado



Fonte: Autoria própria

3.3.5 Unidade de controle

A unidade de controle é responsável por enviar todos os sinais de controle necessários para a execução de uma instrução em particular. Ela é composta por 3 partes:

1. **Registrador de instrução (IR):** Armazena o *opcode* da instrução atualmente sendo executada.
2. **Unidade de tempo (TCU):** No começo de toda instrução, seu valor é definido como T0, na **descida** de cada ciclo de clock seu valor é incrementado (T1, T2, T3, etc).

3. **Lógica de Controle (LG):** Responsável por definir todos os sinais de controle baseado na instrução que está sendo executado atualmente (armazenada no IR) e qual ciclo o processador se encontra dentro dessa instrução (armazenado no TCU).

3.3.6 Geração de *Clock* (CLK)

Essa unidade é responsável por gerar o sinal de clock do processador. O clock de entrada (CI) é dividido em 2 clocks espelhados. Diferentes componentes podem executar operações em um dos dois ciclos de *clocks*.

3.3.7 Registrador do barramento de endereço (ABL e ABH)

O endereço que está sendo acessado atualmente pelo processador ficará armazenado nesse registrador. Como o endereço é de 16-bits, 2 registradores de 8-bits serão usados.

3.3.8 Unidade Lógica aritmética (ALU)

A ALU é o circuito combinacional responsável por executar todas as operações lógicas e aritméticas do processador. Além disso, registradores auxiliares são acoplados a unidade: AI e BI armazenam os valores de entrada e AR armazena o valor de saída.

4 Resultados Obtidos e Discussões

5 Considerações Finais

Referências

- 1 NEUMANN, J. von. *Introduction to “The First Draft Report on the EDVAC”*. 1945. Citado na página 9.
- 2 TECHNOLOGY, I. M. *MCS6500 Microcomputer Family Hardware Manual*. 2nd edition. ed. Norristown, PA: MOS Technology, INC, 1976. Citado na página 10.
- 3 CENTER, I. T. W. D. *W65C02S 8-bit Microprocessor Datasheet*. Vienna, Austria, 2018. Citado na página 11.
- 4 VAHID, F. *Digital Design with RTL Design, VHDL, and Verilog*. 2nd edition. ed. Waltham/MA, EUA: John Wiley & Sons, Inc, 2011. Citado na página 14.
- 5 LANDSTEINER, M. N. *6502 Instruction Set*. Disponível em: <https://www.masswerk.at/6502/6502_instruction_set.html>. Citado na página 19.

Apêndices

