Universidade Tecnológica Federal do Paraná
Câmpus Apucarana
Departamento Acadêmico de Física

PROJETO MINI SIMULADOR

PROCESSADOR 6507

ACADÊMICO(S):	RA:
HUGO MASSARO	2383918
RAFAEL KENDY	2478544

PROFESSOR:

ANDRÉ D'AMATO

APUCARANA

JULHO, 2023

OBJETIVO:

Esse projeto visa exercitar e fixar os conhecimentos adquiridos sobre linguagem Assembly e arquitetura de computadores. Reforçar o conhecimento sobre decisões de projeto e funcionamento de hardware. Exercitar conceitos sobre integração software e hardware por meio de simulação.

SUMÁRIO

1 - INTRODUÇÃO

O 6507 é um microprocessador de 8 bits desenvolvido pela MOS TECHNOLOGY e foi amplamente utilizado em alguns sistemas de videogame na época como o Atari 2600. Sendo uma variante simplificada do processador MOS Technology 6502, que também foi usado em outros sistemas populares da época, como o Apple II e o NES. O 6507 possui um barramento de endereços de 13 bits, o que permite acessar até 8 KB de memória, no entanto, devido a algumas limitações de design, o Atari 2600 originalmente vinha com apenas 4 KB de memória. O processador 6507 opera a uma frequência de clock de aproximadamente 1,19 MHz e é capaz de executar instruções básicas de manipulação de dados, lógica e controle de fluxo. Embora considerado um processador relativamente simples, o 6507 foi capaz de oferecer uma ótima experiência, tornando o Atari um clássico.

A sua história começa em meados da década de 70, onde a Atari começou a desenvolver um novo console que usaria cartuchos, havia equipes que usavam chips Intel, Motorola e MOS destes apenas o último alcançava o preço preterido pela Atari, porém seu principal ativo, o MOS 6502, que custava 25 dólares por produção, ainda era considerado muito caro para se tornar a CPU do Atari 2600, eles precisavam de algo que custasse 12 dólares ou menos. Por estar perdendo muitos clientes para concorrentes a MOS precisava desesperadamente fazer negócios, foi então que eles se propuseram a ajustar o processador 6502 para o Atari. Assim, a MOS pegou a CPU 6502, que possuia 40 pinos, removeu as linhas de endereço, de 16 para 13 bits, e as interrupções para encaixá-la em um pacote menor e mais barato de 28 pinos. A CPU reduzida resultante só poderia endereçar 8K de memória, mas ainda assim era muita memória. Apesar dos “downgrades” o 6507 agia exatamente igual ao 6502, utilizando do mesmo conjunto de instruções e etapas de execução.

2 - FUNCIONAMENTO

O 6507 possui alguns registradores importantes como: o acumulador (A), que é usado para operações aritméticas e lógicas; o registrador de índice X, usado para endereçamento e iteração de loops; o registrador de índice Y, também usado para

endereçamento e cálculos; o registrador de status (P; sendo: Negativo(N), Overflow(V), Decimal(D), Interrupt Disable(I), Zero(Z), Carry(C)) e o contador de programa (PC). O seu barramento de dados é bidirecional de 8 bits, permitindo a transferência de dados entre o processador e outros dispositivos, como a memória RAM, já o barramento de endereço possui 13 bits, endereçando 8 KB de memória e serve para selecionar a localização da memória para leitura ou gravação de dados. Sobre sua unidade de controle (ULA), ela controla a sequência de operações realizadas pelo processador, interpretando as instruções de memória para ativar os circuitos correspondentes fazendo a execução da operação desejada.

O pipeline é diferente do que vimos no MIPS, sendo considerado um pipeline “mínimo” e nesse caso possuindo apenas duas etapas: busca e execução. Na busca (Opcode Fetch), o processador procura o opcode, representação numérica, da instrução na memória de programa. Na etapa de execução, a instrução é executada com base no opcode anteriormente obtido. Um detalhe interessante é que as instruções de pipeline no 6507 permitiam uma pré-busca (prefetch), ou seja, sobrepor a busca da próxima instrução com a execução atual sempre que possível, ou seja, enquanto uma instrução era executada, ao mesmo tempo buscava-se a próxima instrução, isso permitia um aumento de desempenho significativo, minimizando o tempo ocioso e maximizando o uso dos clock’s

Fonte: https://upload.wikimedia.org/wikipedia/commons/thumb/5/5f/KL_UMC_UM6507.jpg/220px-KL_UMC_UM6507.jpg

3 - COMPILAÇÃO E EXECUÇÃO

Para compilar e executar programas para o MOS 6507 é necessário o uso de um emulador de Atari 2600, atualmente o mais utilizado é o Stella. Os passos básicos são: escrever o código na linguagem Assembly 6502, traduzir o código Assembly em linguagem de máquina gerando então um arquivo binário, carregar e executar o arquivo binário no emulador Stella.

4 - INTRUÇÕES IMPLEMENTADAS

A = acumulador;

X e Y = registradores de índice;

Load & Store:

- LDA, LDX, LDY (Load): carrega para A, X, Y;
- STA, STX, STY (Store): salva para A, X, Y

Operações Aritméticas:

- ADC (Add with carry): adição com Carry;
- SBC (Subtract with carry): subtração com Carry;

Incremento e Decremento:

- INX, INY (Increment Memory Location): incrementa X, Y em 1;;
- DEX (Decrement Memory Location): decrementa X em 1;

Transferência de Registrador:

- TXA (Transfer accumulator): transferência de valor de X, para A;

SHIFT:

- ASL (Arithmetic Shift Left): dá um shift para esquerda e manda o bit "perdido" para C;
- LSR (Logical Shift Right): dá um shift para direita e manda o bit "perdido" para C;
- ROL (Rotate Left): recupera e salva o bit "perdido" pós shift;

Comparação de Bits:

- CMP (Compare accumulator): compara com algo;
- CPX (Compare X): compara X com algo;

Jump & Branch:

- JMP (Jump): pula para outra localização;
- BCC (Branch if carry flag clear): se o carry for zero, pula;
- BNE (Branch if zero flag clear): se Z = 0, pula;
- BEQ (Branch if zero flag set): se Z=1, pula;

Status Flag:

- CLC (Clear Carry Flag)

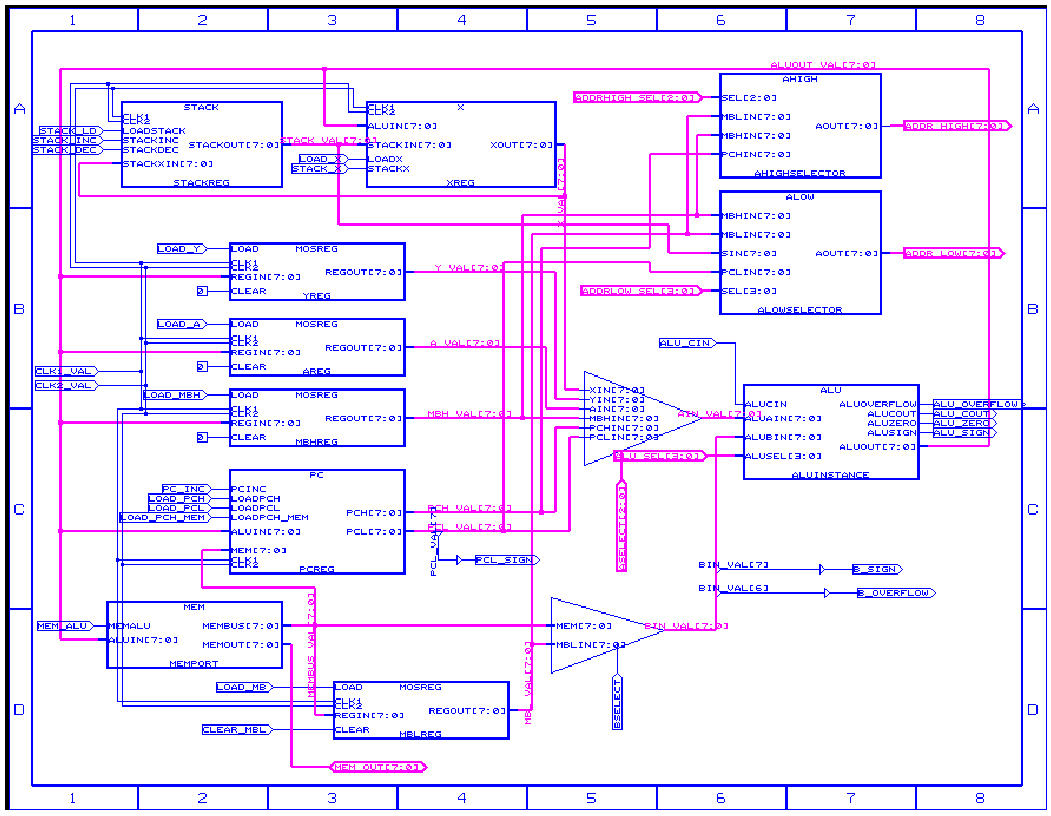
Sub-rotina & Interromper:

- BRK (Break): finaliza o programa;

5 – DATAPATH

O Datapath consiste em: seis registradores de 8 bits que podem armazenar dados temporariamente para serem processados ou manipulados pelas instruções do processador. Um montador de programa de 13 bits responsável por rastrear a próxima instrução a ser buscada e executada na memória, possuindo funções separadas de leitura e gravação para os bytes altos e baixos. Uma ALU principal, que suporta todas as principais operações matemáticas como subtração, adição e soma, também pode

ser um portão não operacional para transferência de valores de um registrador para outro. Incrementos e decrementos separados para o contador de programa e o ponteiro de pilha, pois a ALU principal não pode fazer essas operações, porque algumas instruções (como operações de pilha) requerem um decremento de pilha ao usar a ALU. Uma interface de memória que pode ler ou controlar o barramento de memória para leitura e gravação, sendo utilizada para transferir dados entre o processador e a memória externa. Travas separadas para enviar, corretamente direcionados, os diferentes endereços dos registradores do caminho de dados para cada tipo diferente de endereçamento exigido pela instrução.



Fonte: http://graphics.stanford.edu/~ianbuck/proj/Nintendo/img8.gif

6 – CISC

A arquitetura escolhida é Cisc, apesar de ser altamente confundida com Risc por ser rápida e ter instruções simples se comparadas com outras CPU's. Porém uma arquitetura Risc possui poucos modos de endereçamento, um opcode minimalista, uma pipeline completa, vários registradores de propósito geral e normalmente utilizam de 3 a 4 ciclos de clock por instrução, além de originalmente exigir registradores de 32 bits (posteriormente abandonaram essa ideia), já o processador 6507 possui modos de endereçamento complexos, pilha fixa, operações diretas na memória e o ciclo de clock é 1 por instrução. O único recurso que o 6507 possui de uma arquitetura Risc é a pré-busca (prefetch), mas isso não o torna Risc.

O processador 6507 possui no total 13 modos de endereçamento que afetam em como as instruções serão acessas, sendo eles:

- Acumulador: a operação é feita no acumulador, nenhum dado adicional é necessário. Exemplo:

```
ASL
LSR
ROL
ROR
```

- Absoluto: os dados são acessados usando os 13 bits de endereço especificados como constantes. Exemplo:

```
LDA $06d3
STX $0200
JMP $5913
```

- Absoluto, X: os dados são acessados usando os 13 bits de endereçamento, ao qual o valor do registrador X é adicionado, com carry. Exemplo:

```
LDA $8000,x
STA $8000,x
```

- Absoluto, Y: os dados são acessados usando os bits de endereçamento, ao qual o valor do registrador Y é adicionado, com carry. Exemplo:

```
LDA $8000,y
STA $8000,y
```

- Imediato: os dados são pegos do byte seguindo o opcode. Exemplo:

```
LDA #$05
ORA #$80
```

- Implícito: os dados estão implícitos na operação. Exemplo:

```
CLC ; clears the Carry flag - está implicito que
 opera no registro de status
RTS ; return from subroutine - está implícito que
o endereço de retorno será retirado da pilha
```

- Indireto: os dados são acessados usando um ponteiro o endereço de 16 bits do ponteiro é fornecido nos dois bytes que seguem o opcode. Exemplo:

```
JMP ($9000) ; Pula para localização
apontada pelos endereços $9000 (low) e $9001 (high)
```

- X, Indireto: um endereço de página zero de 8 bits e o registrador X são adicionados, sem carry, sendo o endereço resultante utilizado como ponteiros para os dados que estão sendo acessados. Como os ponteiros têm 2 bytes de

comprimento, o registrador X deve ser um número par ao acessar a lista de ponteiros, caso contrário será obtido metade de um ponteiro e metade de outro. Exemplo:

```
LDA ($05,x); se x=4, então o ponteiro em $09 (e $0a) será usado,
e o acumulador será carregado a partir do endereço indicado pelo ponteiro
```

- Indireto, Y: Um endereço de 8 bits identifica um ponteiro, o valor do registrador Y é adicionado ao endereço contido no ponteiro. Efetivamente o ponteiro é o endereço base e o registrador Y um índice através desse endereço de base. Exemplo:

```
LDA ($10),y ; se y=4, e o ponteiro em $10 (e $11)
contém o valor $FF00, então o acumulador é carregado
do endereço ($FF00+$04)=$FF04
```

- Relativo: um deslocamento de 8 bits é feito, esse valor é adicionado ao contador de programa (PC) para encontrar o endereço efetivo. Exemplo:

```
BNE $ 0600 ; O valor "$0600" é montado em um
deslocamento. O destino deve estar no intervalo
de (-128:127) bytes do valor atual do PC.
```

- Página Zero: um endereço de 8 bits é fornecido na página zero, é como um endereço absoluto, mas como o argumento é de um byte, a CPU não gasta um ciclo adicional para buscar o byte high. Exemplo:

```
LDX $13
AND $07
```

- Página Zero, X: um endereço de 8 bits é fornecido, ao qual o registrador X é adicionado, sem carry. Se o endereço estourar o endereço volta para a página zero. Exemplo:

```
STA $00,x
LDA $00,x
```

- Página Zero, Y: um endereço de 8 bits é fornecido, ao qual o registrador Y é adicionado, sem carry. Se o endereço estourar o endereço volta para a página zero. Exemplo:

```
STA $00,y
LDA $00,y
```

7 – RESULTADOS OBTIDOS

Para a elaboração do emulador referente ao processador 6507, começamos criando um código em assembly 6507, optamos por começar com um simples código de adição e subtração, que também faz o salvamento de valores na memória. A partir disso foi feita a tradução do código para hexadecimal e então para binário, nessa etapa usamos o site HexEdit para gerar o arquivo binário. Tendo este arquivo em mãos, iniciamos a implementação do emulador propriamente dito, usando a linguagem C para este trabalho.

O emulador funciona da seguinte forma, primeiramente é feito a leitura de todas as instruções do arquivo binário, as instruções utilizadas possuem 1 ou 2 bytes, por causa disso dividimos o código em seções com um byte cada, que ficam armazenadas dentro de um vetor “ram”. Para navegar pelas instruções usamos os contadores PC, que aumenta a cada instrução, e o IC, que nesse caso é usado como um contador de bytes, devido a isso eles nem sempre possuem o mesmo valor, já que PC será sempre incrementado de um em um, enquanto isso o IC pode ser incrementado em dois ou um, dependendo do tamanho da instrução.

Usamos então IC como índice para navegar dentro de ram e um switch com um case para cada comando utilizado, naqueles que possuem dois bytes, o sistema reconhece que o próximo byte ainda faz parte do mesmo comando, nesses casos foi necessária a inclusão de um switch interno ao case correspondente, para que a variável correta seja usada.

Após efetuarmos os testes e correções necessárias para o funcionamento desse primeiro código assembly, passamos para a criação dos outros dois códigos solicitados, seguindo essa mesma linha de raciocínio, fazendo alterações quando preciso.

REFERÊNCIAS:

https://dfarq.homeip.net/atari-2600-cpu/

https://en.wikipedia.org/wiki/MOS_Technology_6502

https://en.wikipedia.org/wiki/MOS_Technology_6507

http://graphics.stanford.edu/~ianbuck/proj/Nintendo/node14.html

http://www.visual6502.org/

https://www.masswerk.at/6502/6502_instruction_set.html

https://research.swtch.com/6502

https://www.lb2.com.br/blog/processadores-cisc-x-risc-qual-a-diferenca-entre-essas-duas-arquiteturas

http://www.emulator101.com/6502-addressing-modes.html

https://wiki.cdot.senecacollege.ca/wiki/6502_Addressing_Modes