UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

Relatório

Nome do projeto: Simulação em Arduino de um Pipeline com cinco estágios

Alunos: Felipe Barbalho Rocha
Raul Silveira Silva

Dezembro/2015

1. METODOLOGIA

A simulação consiste no uso de 4 Arduinos UNO conectados uns aos outros pela interface decomunicação Serial (pinos RX/TX) simulando 4 estágios de um Pipeline, o *Instruction Fetch, Instruction Decoder, Executor* e *Data Memory*. O 5º estágio (*Write Back*) foi simplificado para uma escrita no Arduino que guarda o banco de registradores. A simulação também possui 4 displays de LCD 16x2 para facilitar a visualização de qual instrução está sendo processada em cada estágio.

Para a parte de controle de saltos foi utilizado um 5º Arduino para simular o *Controller* que faz uma alteração no contador de programa para saltar para a próxima instrução. Nas próximas seções será exposto o funcionamento de cada Arduino/estágio, o conjunto de instruções que foram utilizadas e alguns aprimoramentos que podem ser pensados para uma próxima versão.

Para estabelecer uma comunicação padronizada e de fácil manipulação, foi definido um protocolo de 4 Bytes como mostra a Figura 1.

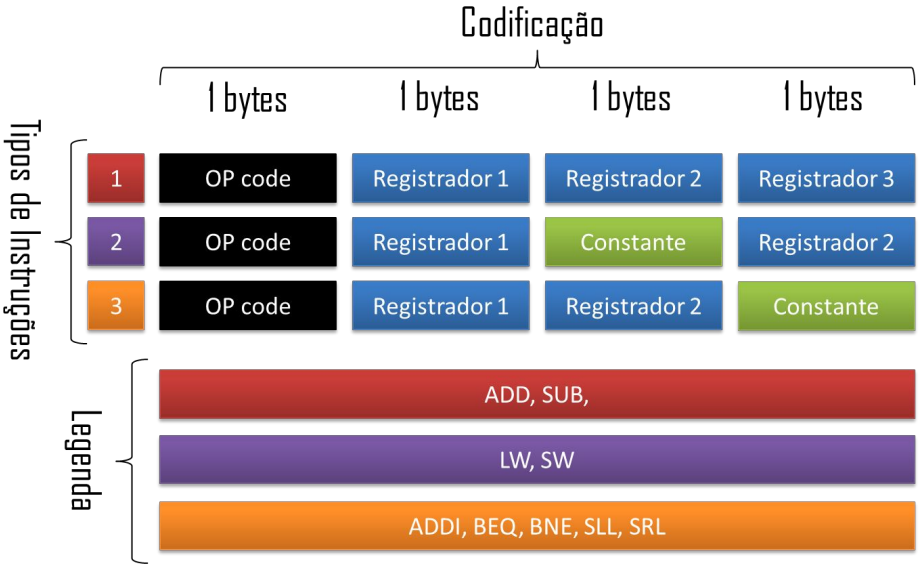

Figura 1 - Codificação Utilizada para as instruções.

2. INSTRUCTION FETCH

Para a memória de instruções foi utilizado um Ethernet + SD shield para armazenar as instruções do programa a ser executado. Um arquivo com o nome “code.txt” armazenado na raíz de um cartão micro SD contem todo o código em Assembly MIPS do programa a ser executado. O Arduino lê esse arquivo atravéz do SD *card reader* do Ethernet Shield e envia cada instrução, separadamente com um

delay de 3 segundos entre cada instrução, pela Serial (pino 1 - TX) para o *Instruction Decoder*. O envio de cada OP code, cada identificador de registrador e constantes são baseadas em um protocolo descrito pela equipe de desenvolvimento, onde são enviados 4 Bytes por instrução.

As operações implementadas nessa simulação do Pipeline estão listadas abaixo com seu respectivo byte de identificação:

- ADD -> 0b00000000.
- SUB -> 0b00000001.
- ADDI -> 0b00000010.
- LW -> 0b00000011.
- SW -> 0b00000100.
- SLL -> 0b00000101.
- SRL -> 0b00000110.
- BEQ -> 0b00000111.
- BNE -> 0b00001000.

Cada registrador também possui seu byte de identificação para facilitar na hora de transmitir cada instrução e de indentificá-lo no banco de registradores. Abaixo estão listados os registradores implementados nesse Pipeline e seu respectivo byte:

- $0 -> 0b00000000.
- $s0 -> 0b00000001.
- $s1 -> 0b00000010.
- $s2 -> 0b00000011.
- $s3 -> 0b00000100.
- $s4 -> 0b00000101.
- $s5 -> 0b00000110.
- $s6 -> 0b00000111.
- $s7 -> 0b00001000.

As constantes necessárias em algumas operações possui um limite máximo de um byte, e são estritamente valores positivos, ou seja, podem variar de 0 à 255.

Caso ocorra alguma instrução de salto, uma *flag* é acionada com base na informação que foi recebida pela Serial (pino 0 - RX) e salta a quantidade de linhas que o *Controller* informou.

3. INSTRUCTION DECODER

Na decodificação da instrução foi usada a EEPROM do Arduino para armazenar o banco de registradores. Como a EEPROM é um tipo de memória não volátil, é recomendado rodar um programa para zerar toda a EEPROM antes de iniciar o programa, caso contrário podemos encontrar valores de operações de outros programas.

Após feita a leitura dos registradores na EEPROM os mesmos são enviados pela Serial (pino 1 - TX) para o *Executor* junto com o OP code, o byte do registrador a ser modificado e a constante, se necessário.

Em caso de salto, o *Controller* informa o estágio através da SoftwareSerial, que simula uma segunda porta de comunicação Serial, e o mesmo bloqueia a instrução lida pelo estágio anterior, depois da instrução de salto, e espera a próxima instrução a ser executada.

Quando houver um *Write Back* os dados do registrador e do valor também são recebidos pela Software Serial e é realizada a escrita na EEPROM, passando o byte do registrador a ser escrito e seu novo valor.

4. EXECUTOR

No 3º estágio do Pipeline são recebido pela Serial (pino 0 - RX) o OP code, o registrador de escrita e os valores a serem operados, exceto no caso de instruções como LW e SW que são executadas no 4º estágio. Após identificadas e realizadas as operações temos dois casos de saida:

- Caso 1 - Salto: A informação vai direta para o *Controller* que decide mudar a *Flag* de salto baseando-se no que foi recebido. Tal informação é enviada através de uma SoftwareSerial do 3º estágio.
- Caso 2 - Outras operações: Simplesmente são passadas adiante em 3 Bytes, um para o OP code, outro para o registrador de escrita e outro para o valor a ser escrito.

Para as operações de *Load* e *Store*, são enviados os 4 Bytes recebidos pelo 2º estágio.

5. DATA MEMORY

O 4° estágio é dedicado as instruções de escrita (*Store Word* - SW) e leitura (*Load Word* - LW) na memória. As demais instruções são simplesmente encaminhada para Write Back.

Nesta etapa utilizamos um array de bytes para simular a memória, com capacidade para 511 bytes, que corresponde ao valor de endereço máximo armazenado no registrador (255) somado com o valor máximo da constante de alinhamento da memória (255). Além disso um arquivo “MEM.txt” armazenado em um cartão SD é responsável por registrar as operações na memória e possibilita o acompanhamento dos dados que foram salvos na memória, como pode ser visualizado na Figura 2.

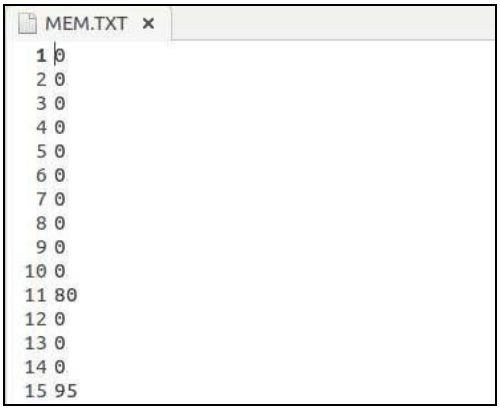

Figura 2 - Arquivo de registro da memória.

6. CONTROLLER

Na simulação do Pipeline implementada o papel do *Controller* é identificar um instrução de salto e notificar o 1º e 2º estágio de uma necessidade de pular uma certa quantidade de linhas do programa. Essa notificação é feita ao mesmo tempo para ambos estágios usando uma comunicação Serial (pino 1 - TX) comum aos dois primeiros Arduinos.

7. TRABALHOS FUTUROS

A atual implementação do Pipeline possui algumas limitações como não realizar o tratamento de bolhas (*STALL*) e os saltos somente podem ser feitos para frente, ou seja, programas com laços de repetições não podem ser descritos. Além disso como o

clock não está sincronizado a primeira execução do programa pode não ser bem executada, para resolução de tal problema, faz-se necessário que a alimentação dos Arduinos sejam feitas na seguinte ordem:

1. *Controller.*
2. *Data Memory.*
3. *Executor.*
4. *Instruction Decoder.*
5. *Instruction Fetch.*

Após essa ligação deve ser feito um reset em todos os Arduinos. Caso seja necessário aumentar o periodo do Clock, basta alterar a constante “CLOCK” em cada estágio do Pipeline.

Para uma segunda versão do simulador, a proposta é de implementar o tratamento de bolhas, melhorar o sincronismo do Clock e fazer modificações para que sejam executados programas com laços de repetição.

8. CONCLUSÃO

Foram implementadas na simulação do Pipeline algumas instruções Assembly MIPS e foi feito o tratamento de saltos. A Figura 3 representa o esquemático da simulação em Arduino e a Figura 4 representa o projeto já implementado usando os 5 Arduinos.

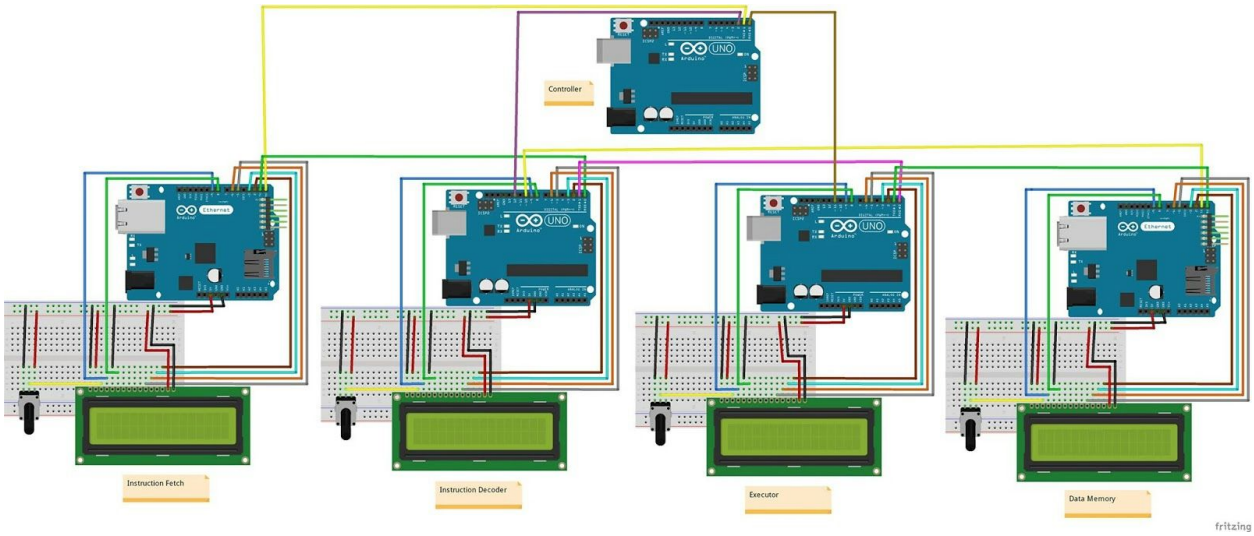

Figura 3 - Esquemático.

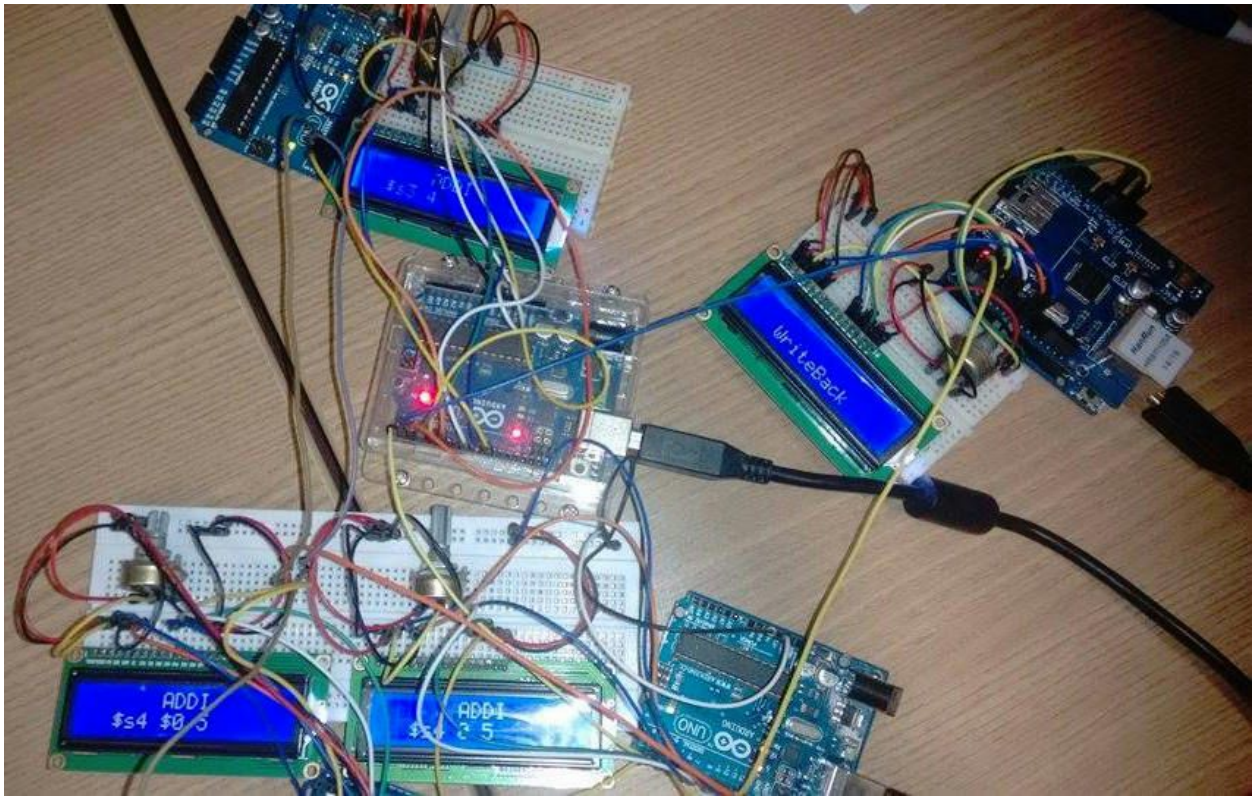

Figura 4 - Simulação do Pipeline.