DESCRIÇÃO DE COMPUTADOR WR80

Neste documento, permite analisarmos tudo que rodeia este novo microprocessador, sua modelagem e arquitetura, seus procedimentos e instruções, além de possibilidades programáveis. Começaremos com uma introdução aos elementos iniciais deste processador.

INTRODUÇÃO

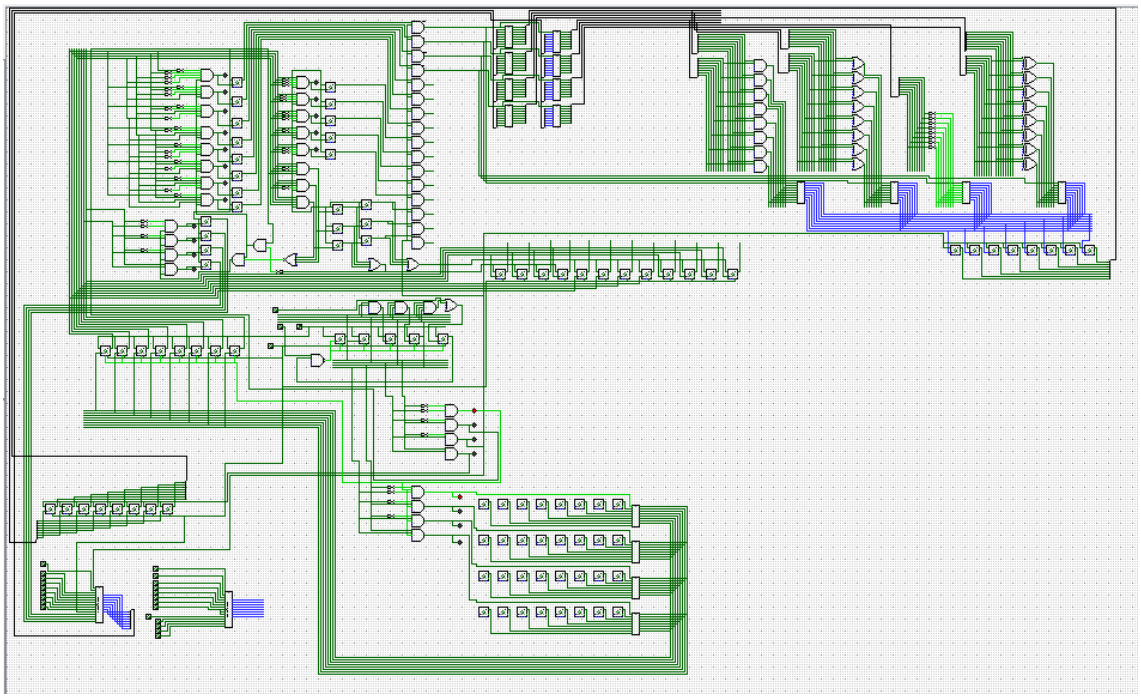
Este aqui é o circuito que estou trabalhando com o Renilson, o meu cliente das aulas. Tivemos uma aula mais recente de 4 hrs, e aí pra completar a sessão de 6 hrs, vamos concluir este circuito em 2 hrs de aula. Aí eu pedi o circuito dele da aula mais recente e implementei as saídas das 4 operações lógicas da ULA (AND, OR, NOT e XOR). Este é parte do processador que será chamado de WR80.

As letras iniciais vêm de W de Wenderson e R de Renilson e 80 segue o padrão de nomes da Intel, pelo fato de ser um processador de 8 bits. No lado esquerdo superior pode ver que existe o decodificador de instrução, separado em decodificador de Opcodes e decodificador de endereços. Na parte esquerda inferior é o circuito de controle, para contabilizar a leitura de instruções da memória pelo contador de programa e a seleção de estágios da CPU (busca, decodificação, execução e escrita).

Abaixo deste circuito de controle, temos os registradores, que é o acumulador de resultados (DR - Data Register) e os registradores de usuário. Os registradores de usuário são: R0, R1, R2 e R3. Cada um de 8 bits. No canto superior direito são as operações da ULA (Unidade Lógico-Aritmética) e ainda contém as 4 operações lógicas.

Elas são operações bit-a-bit (Os chamados "bitwise" como AND bitwise, OR bitwise, etc.). Estas linhas pretas em cima são os barramentos internos do microprocessador, que ler dados dos registradores (de acumulador e/ou usuário) e escreve dados para o acumulador.

COMO FUNCIONA INICIALMENTE O WR80?



Este circuito funciona assim: Um contador de 4 bits vai contabilizar de 0 a 15, os seus 2 bits menos significativos serão multiplexados por um multiplexador que vai selecionar/ativar cada um dos 4 estágios. A cada vez que os 4 estágios são selecionados em sequência, +1 é somado nos 2 bits mais significativos, estes 2 bits serão chamados de "Contador de Programa" (PC - Program Counter). Quando a 1ª ativação do seletor de estágio é feita, o estágio de busca se inicia, onde irá ativar o multiplexador de endereços da memória RAM, que até então tem 4 bytes de endereçamento, já que 2 elevados a 2 bits = 4 bytes de RAM. Quando este multiplexador é ativado, ele vai ler um agrupamento de células (de 8 bits) no endereço que o contador de programa está apontando e vai enviar estes 8 bits de células para o registrador IR (Instruction Register ou Registro de Instrução).

O registrador IR está próximo ao circuito de controle onde tem o contador de programa. Quando o tempo de busca é finalizada, ativando a decodificação, a instrução de 8 bits recém lida da memória será escrita e imediatamente enviada ao decodificador de instrução no estágio de decodificação (2ª linha de ativação do seletor de estágios). O decodificador vai separar a instrução de 8 bits em 2 conjuntos de 4 bits: Os 4 bits mais altos será o "Opcode" (Código de Operação) e os 4 bits mais baixos poderá ser tanto um endereço de um registrador (de R0 a R3 ou de P0 a P3), quanto um número literal (de 0 a 15). O opcode é decodificado pelo multiplexador de opcodes e o endereço é decodificado pelo multiplexador de endereços. Ambos, ativam uma linha específica, onde o opcode

poderá ativar até 16 linhas específicas, enquanto que o endereço ativará (por enquanto) 4 linhas específicas.

Em uma destas 16 linhas, aquela que for ativada será armazenada em um respectivo Flip-Flop tipo D, assim a gente não perde a ativação quando o estágio de decodificação ser desativado. Da mesma forma é o multiplexador de endereços, a sua linha ativada será armazenada em um Flip-Flop tipo D específico. Do Flip-Flop da linha ativada, será enviada a um Habilitador de 8 bits no estágio de execução. Para cada linha de ativação, teremos 2 habilitadores: Um habilitador receberá a entrada lida do registrador DR e o outro habilitador receberá a entrada lida de qualquer um dos registradores R0 a R3. O habilitador que tiver ativado, vai habilitar sua saída para um circuito lógico específico, que poderá ser uma das 4 operações: AND bitwise, OR bitwise, NOT bitwise e XOR bitwise. Neste mesmo instante, a mesma linha que ativou o habilitador, também vai ativar o alternador de saída de uma destas operações.

O alternador que tiver ativado, vai enviar sua saída para um registro temporário, que será um registro de 8 bits que sempre vai sobrepor a cada novo resultado. Quando o seletor de estágios ativar a linha de "Escrita", o resultado da operação das 2 entradas passadas pelo habilitador, vai ser copiada do registro temporário para o registro DR, que é o acumulador. Enquanto que durante a execução, as entradas do habilitador dos registros de usuário, só será possível se o multiplexador de endereços "selecionar" o endereço do Registrador, ou seja, se for 0001 por exemplo, o registrador R1 será selecionado, então sua saída é lida na linha do estágio de execução, enviando pro habilitador e por sua vez operando na operação que o habilitador ativou.

Quando o estágio de escrita for finalizado e o resultado de qualquer uma das operações for escrita, o ciclo se reinicia voltando ao estágio de busca e lendo a próxima instrução da memória, até que o contador de programa chegue ao final da memória (11b), quando chegar, ele zera o contador e volta pro topo da memória. Nessa, se um bit específico do contador tiver ativo, o contador para de contabilizar e a leitura trava, se não, se o bit tiver limpo, o contador continua contando, reiniciando a leitura de memória (Ou seja, o programa ficará em loop).

COMPONENTES FUNDAMENTAIS

Neste circuito estamos usando distribuidores, que ajuda a otimizar o espaço do LogiSIM. Os distribuidores podem ser redirecionados para duas posições: Leste e Oeste. Quando ele ta configurado para Leste, a entrada de uma linha de barramento se espalha para várias trilhas. Esta quantidade de trilhas deve ser configurada aumentando ou diminuindo a largura de entrada e saída. Quando ele ta configurado para Oeste, é o inverso, várias trilhas se reduzem à apenas 1 barramento. Por isso você ver estas linhas pretas no circuito, este é um barramento de 8 bits que contém 8 trilhas internas. Já os habilitadores são CIs (Circuitos Integrados feitos no LogiSIM usando sub-

circuitos) que é um conjunto de portas AND conectadas a cada entrada. Já que portas AND podem utilizadas para habilitar circuitos, elas são frequentemente utilizadas tanto nos multiplexadores (como, por exemplo, nas decodificações), quanto nos seletores (Que pode tanto selecionar um estágio da CPU ou um endereço de memória), logo, operações como habilitação e seleção é utilizado portas AND. Então o habilitador encapsula esta operação e aí, ele recebe as entradas de 8 bits e recebe +1 entrada adicional que é para ativar o habilitador. Quando o habilitador está ativado, sua entrada é enviada para a saída, mas quando está desativado, o habilitador "barra" a entrada com a saída, bloqueando o envio de dados.

O alternador tem a mesma função que o habilitador, no entanto, no alternador a gente considera usar um método de circuito chamado "Tri-state", que permite você ter 3 estados diferentes: 0 lógico, 1 lógico e estado desabilitado. Por mais que 0 lógico já pareça a própria "desabilitação", ele não é literalmente esta desabilitação, então o estado desabilitado serve para você ter isso de forma literal e permite interligar inúmeras conexões em um só conjunto de conexões sem dar conflitos, justamente porque uma das conexões são desabilitadas, logo, ela aceitará outras conexões habilitadas.

O alternador possibilita isso porque ele usa uma espécie de "Buffer controlado", que é um chip parecido com a porta AND, só que com um pino a mais pra ativar ou desativar a entrada com a saída. Já no caso do habilitador, isso não seria possível, pois portas AND só contém 2 estados: 1 ou 0. Logo, os habilitadores devem ser usados na entrada para "forçar" um estado existente, como 0 ou 1, devido a estados desabilitados de alternadores não ser compatíveis com entradas de portas lógicas, como exemplo: O NOT (Pois daria conflito).

Já no caso de Saídas, podemos usar alternadores pois não contém portas lógicas conectadas na saída, apenas envios diretos para Flip-flop pré-prontos do LogiSIM. Os Flip-flop como você sabe são células de memória, usadas em registradores internos, memória cache, ou outros tipos de memória, no geral, dos tipos SRAM (Static RAM). Eles podem ser feitos do zero usando portas NOR e AND, ou podem ser utilizados os dispositivos prontos do LogiSIM. Este circuito do WR80 considera utilizar as duas formas.

OPERAÇÕES POSSÍVEIS DA ULA

Por enquanto, nós temos as 4 operações lógicas de Bitwise (operações bit-a-bit), porém iremos adicionar o circuito Full-Adder para soma aritmética. O full-adder contém uma otimização, que é um bit adicional Carry-IN que podemos utilizá-lo para ativar o circuito de subtração (Subtractor), ou seja, transformamos um simples full-adder em um Full-Adder_Subtractor apenas implementando uma porta XOR em cada bit. Logo podemos executar a instrução ADD e SUB só usando este circuito. As instruções de salto vão utilizar a soma aritmética do Full-Adder para somar o contador de programa PC + O registro de Offsets, que é um

registro de 12 bits para endereçar desvios da memória RAM. Esta soma vai alterar a leitura sequencial de memória, pois PC estará sendo alterado, então isso caracteriza um desvio ou salto.

Porém, podemos ter tanto desvios condicionais como os não-condicionais. Os condicionais vão depender de 2 bits: Bit C (Carry) ou Bit Z (Zero). Estes 2 bits serão os bits menos significativos de um registro chamado SR (STATUS Register), que terá 8 bits. Toda operação lógica da ULA pode alterar o bit Z para 0 ou 1, mas toda operação lógico-aritmética pode tanto que alterar o bit Z, como o bit C de 0 para 1, exemplo: Estouro de resultado na adição muda bit C para 1, se não, permanece em 0. Estouro de subtração (resultados negativos) limpa o bit C (muda para 0), se não, permanece em 1. Resultados de operações lógico-aritméticas que darão 0, altera o bit Z para 1 (Como 5 - 5 = 0), se não, permanece em 0.

As instruções de salto condicionais só vão somar PC + Offset caso um destes bits sejam iguais a 1 como no caso das instruções JC e JZ. Já a instrução JP é um salto incondicional, então não depende dos bits C e Z, ele soma independentemente. A instrução de comparação BT (Bit Test) vai realizar uma subtração, porém não vai escrever o resultado da subtração como a instrução SUB, ela simplesmente vai alterar os bits Z e C. Isto permite verificar se valores são menores que, maiores que, iguais ou diferentes, etc. (Operações relacionais).

As instruções ST (STORE) e LD (LOAD), serão instruções de movimento, portanto, elas não terão nenhuma operação lógico ou aritmética a não ser apenas enviar dados de um ponto para outro, ou seja, dos habilitadores para o registro temporário e do registro temporário para DR ou Rx (Onde 'x' vai de 0 a 3). No entanto, STA e LDA vão controlar a própria memória RAM, então eles vão depender das portas P0, P1 e P2 para ler um dado em um endereço de memória P0:P1 da porta P2 para DR (Ou pegando de DR para P2, enviando dados ao endereço P0:P1, o inverso de antes).

Cada instrução dessa leva 1 ciclo de CPU (4 estágios), no entanto, apenas as instruções JC, JZ e JP levarão 2 ciclos (8 estágios), porque o endereçamento é de 12 bits (Endereça até 4096 bytes de RAM), porém a instrução só tem 8 bits, então 12 + 4 do opcode = 16, logo, os 16 bits devem ser divididos em 2 leituras de 8 bits separadas da memória, ou seja, 2 ciclos. Okay, esta é a arquitetura do WR80 criado durante em aula. Quando ele tiver pronto, vamos encapsular tudo em um sub-circuito e conectar em um monitor TTY (para imprimir dados de programas Assembly) e uma memória RAM de 4096 bytes (dispositivo pronto) para enviar instruções para o processador. E daí, criarei programas testes, assim como um mini-sistema operacional ou uma mini-BIOS Assembly simulada para esta arquitetura. Me esqueci do Clock, frequência máxima de 4.1KHz e o Clock alimenta o circuito de controle, o que permite as contagens e seleções de estágios.

PROGRAMAS MONTADOS OU COMPILADOS

Antes eu falei da Mini-BIOS, okay! primeiro será os programas testes codificados diretamente em código hexadecimal das nossas instruções. Este programa será criado um arquivo e carregado no dispositivo de Memória RAM do LogiSIM. Depois vou criar um montador pro nosso Assembly WR80, e aí ficará mais fácil criar programas mais customizáveis, assim poderei criar uma mini-bios. Na verdade, seria só um sistema que simula a inicialização de um computador, exibindo Strings na tela e carregamento de um código de uma memória ROM para a RAM (Como se fosse carregamento de sistema operacional do HD para a RAM).

Tendo este montador, será possível até criar um compilador básico com estrutura de linguagem alto-nível, para gerar o código Assembly WR80. Posso até atribuir uma versão alternativa e paralela da linguagem Plax para gerar este Assembly e criar programas melhores pro WR80, e também penso em desenvolver novos comandos declarativos para gerar XML dos circuitos (Arquivo de projeto dos circuitos lógicos), seria como se fosse uma VHDL mais básica (Linguagem de Descrição de Hardware Virtual).

Mas aí penso em dar um toque especial, utilizar a linguagem Plax para VHDL usando sintaxes estruturadas e declarativas, com IF, ELSE e tudo mais, além de operações matemáticas. Ou seja, o Plax vai poder tanto que recriar ou otimizar esse computador WR80, quanto que ele poderá criar os próprios programas pra rodar nesse computador, além dos programas que já rodam no Windows.

O Plax será uma linguagem multiparadigma e ajudará muito em desenvolvimento de circuitos lógicos, softwares e sistemas operacionais, utilizando tanto que declarações lógicas e/ou funcionais, quanto que estruturas e objetos com conceitos de outros paradigmas.