Projeto 1 - Entrega Intermediária

Henry Rocha e Vitor Eller

1. Arquitetura do Processador

Utilizaremos a arquitetura Registrador Memória para o projeto. Essa escolha se deu a partir de uma análise do funcionamento do projeto. Pensamos na estrutura de dados e nas operações que serão feitas pelo relógio, tais quais: o armazenamento do horário atual na memória e a necessidade de se incrementar o valor guardado na memória repetidas vezes.

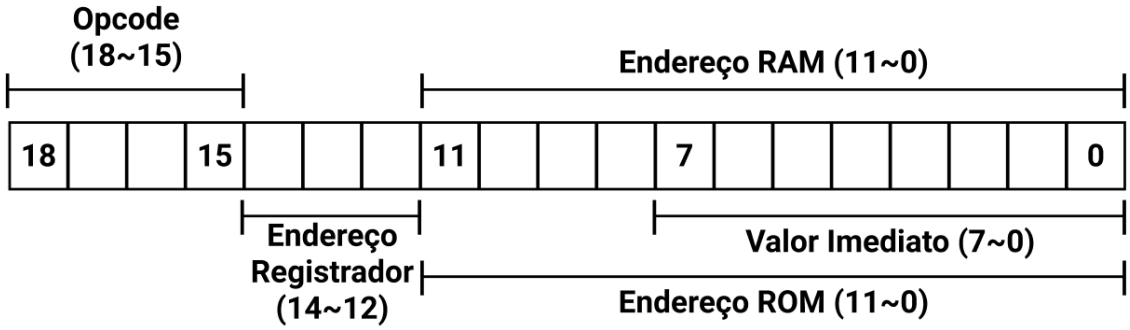
Percebemos que, caso fosse utilizada outra arquitetura, precisaríamos utilizar muitas instruções de carregamento memória -> registrador. Como a arquitetura Registrador Memória possibilita aritmética entre memória e registrador, optamos por utilizá-la.

2. Instruções utilizadas

1. **ADD**
    1. **Descrição:** Soma valor em memória a um registrador e guarda o resultado no próprio registrador
    2. **Sintaxe:** ADD Reg, [Mem];
2. **ADDI**
    1. **Descrição:** Soma valor imediato a um registrador e guarda o resultado no próprio registrador
    2. **Sintaxe:** ADDI Reg, 0x01;
3. **SUB**
    1. **Descrição:** Subtrai valor em memória de um registrador e guarda o resultado no próprio registrador
    2. **Sintaxe:** SUB Reg, [Mem];
4. **SUBI**
    1. **Descrição:** Subtrai valor imediato de um registrador e guarda o resultado no próprio registrador
    2. **Sintaxe:** SUB Reg, 0x01;
5. **MOVM**
    1. **Descrição:** Move valor em memória para um registrador.
    2. **Sintaxe:** MOV Reg, [MEM];
6. **MOVR**
    1. **Descrição:** Move valor em registrador para um endereço de memória.
    2. **Sintaxe:** MOV [MEM], Reg;

7. **LEA**
    1. **Descrição:** Carrega valor imediato em registrador
    2. **Sintaxe:** LEA Reg, 0x01;
8. **CMP**
    1. **Descrição:** Compara um valor imediato com o registrador, isso é feito subtraindo os dois.
    2. **Sintaxe:** CMP Reg, 0x01;
9. **JE**
    1. **Descrição:** Desvia para o endereço imediato caso a flag que indica se a saída da ULA foi zero está ativa.
    2. **Sintaxe:** JE 0x0F; JE [LABEL]
10. **JNE**
    1. **Descrição:** Desvia para o endereço imediato caso a flag que indica se a saída da ULA foi zero não está ativa.
    2. **Sintaxe:** JNE 0x0C; JNE [LABEL]
11. **JMP**
    1. **Descrição:** Pula para o endereço imediato
    2. **Sintaxe:** JMP 0xAA; JMP [LABEL]
12. **OR**
    1. **Descrição:** Operador lógico OR entre um registrador e o valor guardado em um endereço de memória, o resultado é guardado no registrador dado.
    2. **Sintaxe:** OR Reg, [MEM];
13. **AND**
    1. **Descrição:** Operador lógico AND entre um registrador e o valor guardado em um endereço de memória, o resultado é guardado no registrador dado.
    2. **Sintaxe:** AND Reg, [MEM];
14. **CMP**
    1. **Descrição:** Compara se o valor imediato passado é o mesmo presente no registrador indicado. Utilizado para acionar a flagZero e possibilitar o JE e o JNE
    2. **Sintaxe:** CMP Reg, 0x01;

3. Formato das Instruções



Como a quantidade de instruções que o processador possui é 13, o número mínimo de *bits* necessário para representar todas elas é 4, sendo assim o tamanho do *Opcode* é de 4 *bits*.

O processador conta com 8 registradores e para que seja possível endereçar todos eles são necessários 3 *bits* para representar o endereço dos registradores.

De modo a simplificar o endereçamento da *RAM* e *ROM*, os dois periféricos possuem o mesmo tamanho de endereço, 12 *bits*.

Visto que é um requisito funcional que o processador trabalhe com 8 *bits*, o valor imediato também tem de seguir essa norma, sendo assim, o valor imediato tem 8 *bits*.

4. Fluxo de dados para o processador

- **Opcode (Bits [18-15])**
    - Passados à unidade de controle, onde será decodificado. Essa decodificação resultará na mudança dos pontos de controles (mostrados abaixo), responsáveis por possibilitar a execução da instrução passada.
- **Endereço do Registrador (Bits [14-12])**
    - Indica o endereço do registrador que será utilizado na operação. A depender do ponto de controle utilizado ele será usado para leitura e/ou escrita.
- **Endereço RAM (Bits [11-0])**
    - Indica qual endereço de memória será usado na operação. Assim como o registrador, o uso depende dos pontos de controle utilizado. Porém, nesse caso, pode-se apenas ler ou escrever, não ocorrendo ambos na mesma instrução.
- **Endereço ROM (Bits [11-0])**
    - Indica o endereço da ROM para onde o programa será desviado em casos de a instrução passada ser de desvio.
- **Valor Imediato (Bits [7-0])**
    - Valor utilizado em operações com constantes.

Na arquitetura registrador-memória, as operações são feitas entre um registrador e um valor vindo do barramento de dados de entrada ou um valor imediato, contido na própria instrução. Desse modo não é possível realizar uma operação diretamente com um valor em memória. Nesse caso é necessário primeiramente carregar esse valor para um registrador usando a instrução MOVM, e posteriormente realizar a operação desejada.

Para exemplificar essa situação, caso desejássemos incrementar um valor contido em memória, seria necessário primeiramente executar a instrução MOVM R1, 0xXX para armazenar o valor em um registrador e em seguida executar ADD R1, 0x01 para incrementar o valor dentro de R1 e então podemos guardar esse valor de volta na memória usando MOVR 0xXX, R1.

Do mesmo modo, não é possível executar um desvio condicional com apenas uma instrução. Isso introduz a necessidade de uma instrução de comparação (CMP). Sendo assim, para realizar um desvio condicional é necessário primeiro realizar a comparação, usando CMP R1, 0x01, e depois executar um pulo condicional usando JE 0xXX. Tal programa pula para o endereço XX caso o registrador R1 seja igual a 1.

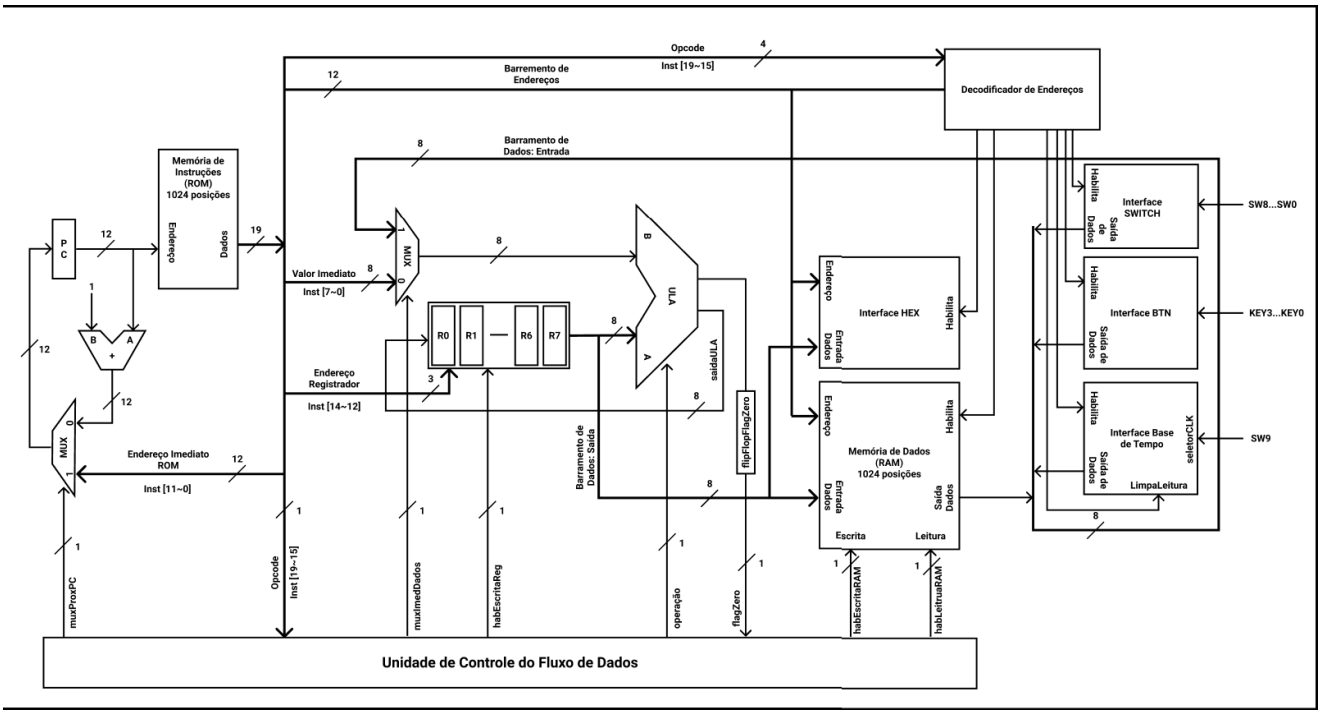
Em mais detalhes, essa situação funciona por primeiramente subtraindo o valor contido em R1 pelo valor imediato 0x01, tal resultado ativa a *flagZero* da *ULA* caso o resultado dessa operação seja 0. A partir dessa flag a Unidade de Controle determina, na próxima instrução, se deve ser realizado o pulo ou não, com base na *flagZero*.

5. Listagem dos Pontos de Controle

muxProxPC	Indica ao *Program Counter* se deve ser realizado um pulo para o endereço imediato.

muxImedDados	Decide se a entrada B da *ULA* será o valor imediato ou um valor do barramento de dados de entrada.
habEscritaReg	Indica ao banco de registradores se deve ser escrito um novo valor no registrador endereçado na instrução.
operação	Indica a *ULA* qual operação deve ser realizada.
habEscritaRAM	Indica a *RAM* se deve ser escrito um novo valor no endereço indicado.
habLeituraRAM	Indica a *RAM* se deve ser lido o valor no endereço indicado.

6. Rascunho do diagrama de conexão do processador com os periféricos



7. Mapa de Memória

A memória será dividida em 4 blocos e os periféricos agrupados dentro desses blocos. A interface *SWITCH* e a interface *BTN* estarão localizadas no bloco 0. O bloco 0 é responsável pela faixa de endereços entre 0 e 1023. Já a interface *Base de Tempo* se encontra no bloco 1, responsável pela faixa de endereços de 1024 até 2047. A interface *HEX* está contida no bloco 2, que é responsável pela faixa de 2048 até 3071. Por último, a memória de dados (*RAM*) se localiza no bloco 3, que existe entre a faixa de 3072 e 4095.

Essa organização dos periféricos pode ser resumida na tabela a seguir:

Bloco	Faixa	Endereço	Interface
0	0 - 1023	0	SWITCH [0]
		...	...
		8	SWITCH [8]
		10	BTN [0]
		...	...
		13	BTN [3]
1	1024 - 2047	0	LER_BT
		1	RST_BT
2	2048 - 3071	0	HEX0
		1	HEX1
		2	HEX2
		3	HEX3
		4	HEX4
		5	HEX5
3	3072 - 4095	0	RAM [0]
		...	...
		1023	RAM [1023]

Obs.: todos os blocos contêm 1024 endereços, mas apenas os endereços que estão em uso foram representados, com exceção do bloco 3, que está totalmente preenchido pela RAM.

8. Controles fixos

A chave SW9 é responsável por controlar a base de tempo do processador. Quando ela está ligada, a base de tempo conta 1.000.000 clocks antes de ser habilitada. Já quando ela está desligada, a base de tempo conta 50.000.000 de clocks antes de ser habilitada.

9. Controles do programa relógio

A chave SW1 permite a entrada no modo de edição de horário. O modo de edição de horário paralisa o relógio, e libera o acionamento dos botões para correção de horário. Para corrigir o horário utilize os botões KEY0, KEY1 e KEY2. O botão KEY0 acrescenta 1 no display do segundo, o botão KEY1 acrescenta 1 no display do minuto e o botão KEY2 acrescenta 1 no display das horas. Para todos as keys é possível segurar ou apertar uma vez. O aperto acrescenta apenas uma unidade, enquanto segurar acrescenta unidades até o momento em que a key é liberada.

10. Utilização do assembler

Regras básicas:

- Use '$' quando for fazer referência à um registrador (i.e. '$1' representa o registrador 1)

- Use a representação hexadecimal para indicar endereços e valores imediatos (i.e. '0x1' representa 1, '0x1a' representa 26).
- Para comentários no arquivo, inicie a linha com ';'
- Para rodar, utilize o comando:python3 assembler.py -i <input_file> -o <output_file>
- Se necessário, observe o arquivo de entrada 'example.nasm'.

11. Dificuldades durante o projeto

Tivemos grandes dificuldades em utilizar a base de tempo disponibilizada pelo professor, sendo que o sinal de saída do componente era sempre 1. Tentamos inúmeras vezes analisar o waveform para tentar entender o que causava essa saída, porém sem sucesso. Com isso, optamos por criar um novo componente, baseado na lógica da base de tempo já criada pelo professor. Nesse novo componente, apenas contamos os clocks até que cheguem no valor de 50.000.000, ou 1.000.000 no caso de acelerarmos o relógio, indicando que se passou 1 segundo.

12. Link do Repositório

https://github.com/HenryRocha/computer-design-clock