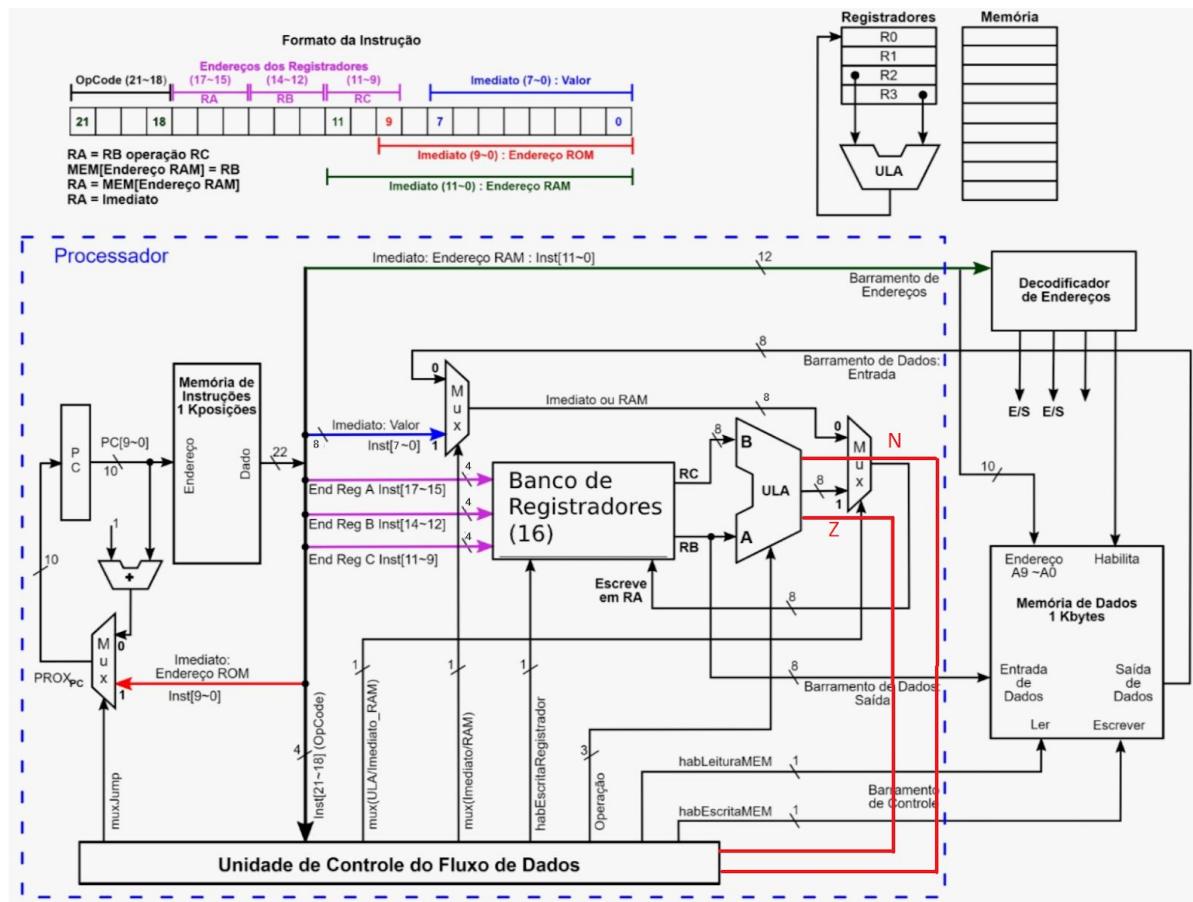


Projeto Relógio - Entrega Final

Giovanna Cabral, Maria Eduarda Bicalho e Mayra Peter

Github do projeto: <https://github.com/mayrapeter/RelogioDesign.git>

Fluxo de dados da arquitetura



Fonte : https://insper.blackboard.com/bbcswebdav/pid-740452-dt-content-rid-6136417_2/courses/202062.GRENGCOM_201561_0004.DESIGNCOMP_6ENGCOMPA/Atividades/desComp.html

Imagem sofreu alterações pela equipe

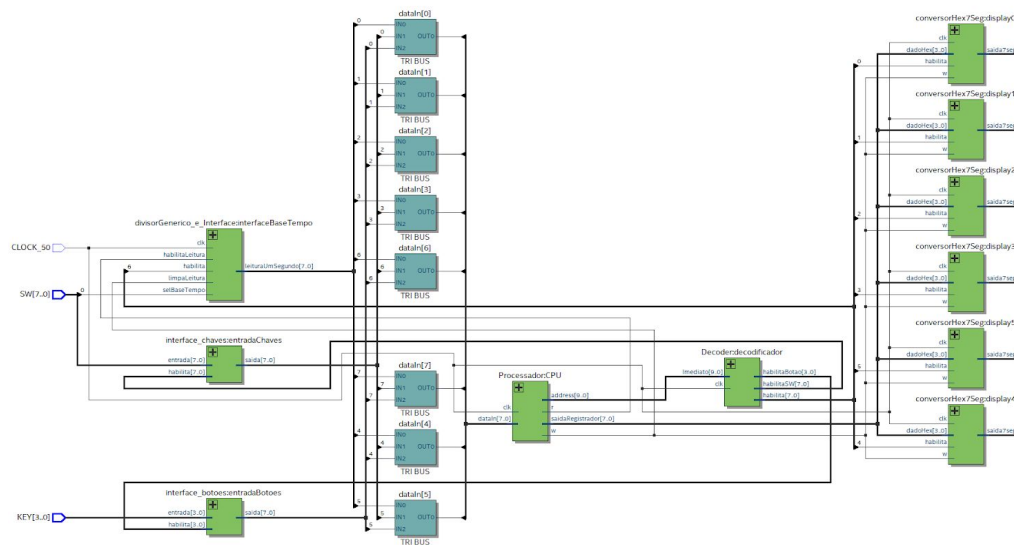


Imagem 2: Diagrama de Blocos do circuito

Explicação:

A memória de instruções passará as instruções, que no nosso caso contém 26 bits. Os primeiros 4 bits da instrução são o Opcode, ele informa qual Instrução deverá ser efetuada pelos pontos de controle na Unidade de Controle do Fluxo de dados. Na tabela presente no tópico de instruções gerais é possível ver essa relação. Quando o Opcode está em 0001, por exemplo, a instrução add deve ser acionada, então o ponto de controle “operação” estará 00 e o ponto de controle “habEscritaRegistrador” receberá 1. Para realizar jumps condicionais (JE e JL), serão usadas duas flags, Z e N, que saem da ULA e entram na Unidade de Controle do Fluxo de Dados. Quando Z é 1, o JE pode ser realizado, e quando N é 1, o JL pode ser realizado.

O assembler traduz o código assembly do relógio para a ROM. O program counter lê a próxima instrução que deve ser passada para o circuito, podendo somar mais 1 para a próxima linha da instrução ou pulando para a instrução que o jmp pede. As instruções ativam os pontos de controle para o funcionamento do circuito. A partir do opcode presente na instrução do PC sabe-se qual é a instrução, e na Unidade de Controle são relacionados os pontos que devem ser ativados. O decoder recebe o imediato presente na instrução, que sinaliza qual será o endereço do periférico que deve ser utilizado. no circuito existem os periféricos: Base de Tempo, Chaves e LCD. Chaves para leitura, LCD para escrita e a Base de tempo que utiliza do clock da placa para “contar” um segundo, lemos desse periférico se um segundo já passou ou não.

Instruções para cada funcionalidade do projeto:

- **SW0:** Essa chave permite a escolha da base de tempo. Quando ela está acionada (valor 1), a base de tempo escolhida é a mais rápida, e quando o valor está 0, a base é a mais devagar.
- **SW1:** Essa chave permite pausar o relógio, assim é possível acertar o horário. Quando ela está acionada (valor 1), o relógio pausa, caso contrário ele continua funcionando.
- **SW2:** Essa chave permite escolher o formato do relógio, AM/PM ou 24 horas. Quando a chave está acionada (valor 1), o formato escolhido é AM/PM. O led indica se é AM ou PM, quando o led está aceso é PM.
- **SW3:** Essa chave permite começar o temporizador. Quando só essa chave está ativada o temporizador começa a contagem regressiva com a base de tempo normal. Se quiser ver a contagem regressiva com a base de tempo rápida tem que acionar a SW0 também.
- **KEY0:** Quando a função está na função de pause, ou seja a SW0 está acionada, esse botão quando pressionado zera o relógio inteiro.
- **KEY1:** Quando a função está na função de pause, ou seja a SW0 está acionada, esse botão quando pressionado adiciona 1 nos minutos.
- **KEY2:** Quando a função está na função de pause, ou seja a SW0 está acionada, esse botão quando pressionado adiciona 1 nas horas.

Requisitos:

- C

- [X] Indica horas, minutos e segundos:
 - [X] O horário deverá ser mostrado através do display de sete segmentos
- [X] Possui algum sistema para acertar o horário
- [X] Possui seleção da base de tempo

- Extras

- [X] Indicação do horário com base em 12 horas -AM/PM
- [X] Temporizador com contagem regressiva
- [X] Montador (assembler) para o processador criado:
 - [X] Deve ser em Python

Total de instruções e sua sintaxe

Instruções gerais:

	Instrução	Sintaxe	Tradução	Comando
1	ADD	add reg1 reg2 reg3	reg3 = reg2 + reg1	0001
2	SUB	sub reg1 reg2, reg3	reg3 = reg2 - reg1	0010
3	MOVR	movr reg1 reg2	reg2 = reg1	0011
4	MOVC	movc \$c reg1	reg1 = \$c	0100
5	STORE	store reg2 M[0x1FF]	M[0x1FF] = reg2	0101
6	LOAD	load M[0x1FF] reg1	reg1 = M[0x1FF]	0110
8	JMP	jmp end	goto end	1000
9	JE	je reg1 reg2, end	if (reg1 == reg2) goto end	1001
10	JL	jl reg1 reg2 end	if (reg1 < reg2) goto end	1010
11	CMP	cmp reg1 reg2	reg3 = reg1/reg2	1011

Pontos de controle

- **Opcode:** são os primeiros 4 bits da instrução. Ele representa qual será a operação realizada pelo processador. Por exemplo, se o Opcode for 0001, o processador realizará uma soma.
- **muxJump:** se o valor for 0, o PC aumenta em 1, ou seja, a próxima linha do código é rodada. Se o valor for 1, o PC assume o valor do imediato e é feito um jump para a linha do valor do imediato.
- **mux(ULA/Imediato_RAM):** se o valor for 0 o mux seleciona o imediato ou RAM e se o valor for 1 o mux seleciona o resultado que sai da ULA.
- **mux(Imediato/RAM):** se o valor for 0 o mux seleciona a RAM e se o valor for 1 o mux seleciona o imediato.
- **habEscritaRegistrador :** se o valor for 1 é permitida a escrita no registrador e se for 0, não é possível.
- **Operação:** Se o valor 01, ULA fará operação de soma. Se o valor for 10, fará operação de subtração.
- **habLeituraMEM:** se o valor for 1, é permitida a leitura da memória e se for 0, não é possível ler os valores da memória.
- **habEscritaMEM:** se o valor for 1, é permitido escrever na memória. Se o valor for 0, não é possível alterar valores na memória.
- **reset Zero:** Quando o rstz está 1, o flip flop da saída flag zero da ULA é restartado, quando 0 não.
- **reset Negativo** Quando o rstz está 1, o flip flop da saída flag negativo da ULA é restartado, quando 0 não.