PROJETO PROCESSADOR-RELÓGIO

Design de Computadores - Engenharia da Computação 6 Semestre - INSPER

Gabriel Moura, Guilherme Aliperti e Nicolas Stegmann.

1) **Código em assembly do pseudocódigo do relógio:**

```
        loadim %0x0
        store $UnSeg
        display Useg
        store $DezSeg
        display Dseg
        store $UnMin
        display Umin
        loadim %0x0
        store $DezMin
        display Dmin
        loadim %0x0
        store $UnHora
        display Uhora
        loadim %0x0
        store $DezHora
        display Dhora
LABEL MEIO
        Loadmem $DezSeg
        Display Dseg
        loadbt %0xE0
        cmpne %0x80
        jc MEIO
        reset %0x0
        loadmem $UnSeg
        add %0x1
        store $UnSeg
        display Useg
        cmpne %0xA
        jc MEIO
        loadim %0x0
        store $UnSeg
        display Useg
        loadmem $DezSeg
        add %0x1
        store $DezSeg
        display Dseg
        cmpne %0x6
```

```
    jc MEIO
    loadim %0x0
    store $DezSeg
    display Dseg
    loadmem $UnMin
    add %0x1
    store $UnMin
    display Umin
    cmpne %0xA
    jc MEIO
    loadim %0x0
    store $UnMin
    display Umin
    loadmem $DezMin
    add %0x1
    store $DezMin
    display Dmin
    cmpne %0x6
    jc MEIO
    loadim %0x0
    store $DezMin
    display Dmin
    loadmem $UnHora
    add %0x1
    store $UnHora
    display Uhora
    cmpe %0x4
    jc ZERA
    LABEL UNHORA
    loadmem $UnHora
    cmpne %0xA
    jc MEIO
    loadim %0x0
    store $UnHora
    display Uhora
    loadmem $DezHora
    add %0x1
    store $DezHora
    display Dhora
    jmp MEIO
  LABEL ZERA
    loadmem $DezHora
    cmpne %0x2
    jc UNHORA
    loadim %0x0
    store $DezHora
    store $UnHora
    store $DezMin
    store $UnMin
    store $DezSeg
    store $UnSeg
    display Dhora
    display Uhora
    jmp MEIO
```

PSEUDO-CÓDIGO DO RELÓGIO EM C:

```c
void relogio (int dh, int uh, int dm, int um, int ds, int us){
    while (1) {
        us = us + 1;
        if (us == 10){
            us = 0;
            ds = ds + 1;
        }
        if (ds == 6){
            ds = 0;
            um = um + 1;
        }
        if (um == 10){
            um = 0;
            dm = dm + 1;
        }
        if (dm == 6){
            dm = 0;
            uh = uh + 1;
        }
        if (uh == 10 ){
            uh = 0;
            dh =  dh + 1;
        }
        if (dh == 2 && uh == 4){
                uh = 0;
                dh = 0;
        }
    }
}
```

2) Total de instruções e sua sintaxe;

- Utilizamos no projeto as seguintes instruções:

- Load IM : Carrega imediato para o acumulador

- Cmp not equal: Compara o que está no acumulador com um valor imediato e ve se é diferente, manda 1 para a saída cmp da ULA

- Jmp : Jump incondicional para a instrução no imediato

- Store : Guarda valor do acumulador na memória

- Add: Soma o que está no acumulador com um valor imediato e guarda no acumulador

- Jmp Condicional: Jump com alguma condição estipulada no cmp da instrução anterior

- Cmp equal: Compara o que está no acumulador com um valor imediato e ve se é igual, mandando 1 na saída cmp da ULA.

- Load Mem : Carrega valor da memória no acumulador

- Load Basetempo : Carrega valor da base de tempo no acumulador

- Nossa palavra de controle possui 18 bits , sendo:

OPCODE	RESERVADO
10 bits	8bits

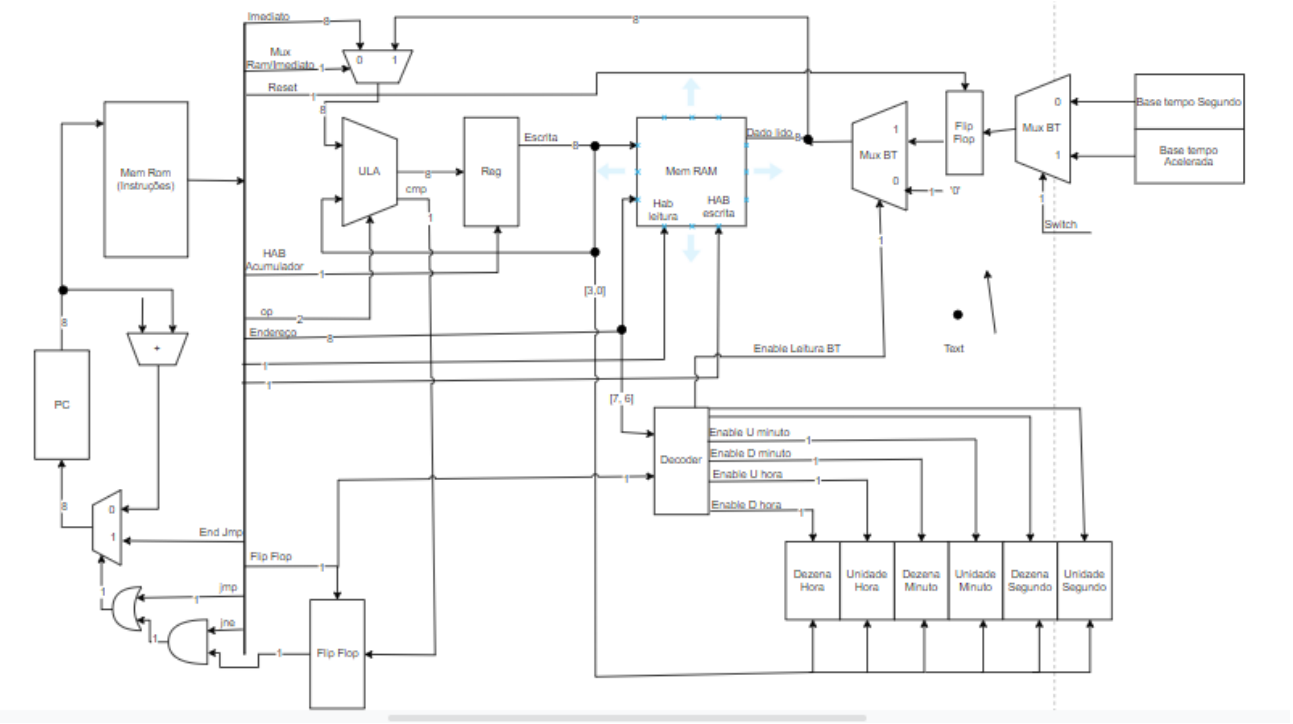
3) Arquitetura do processador;

- Para este projeto utilizamos uma arquitetura do tipo Acumulador num processador do tipo NISC.

Escolhemos esta arquitetura para o projeto do relógio pois era a que tínhamos maior familiaridade.

No entanto, ao final do projeto percebemos que essa arquitetura foi um erro, pois essa necessita de uma maior quantidade de instruções, que deixam maior margem para erros e bugs, dos quais perdemos muito tempo para resolver.

4) Diagrama de conexão do processador com os periféricos (Arquivo pdf presente no zip);



5) Fluxo de dados para o processador, com uma explicação resumida do seu funcionamento;

Explicação do fluxo de dados: O fluxo de dados é de um acumulador NISC, sendo que os pontos de controle(opcode) e imediato vem diretamente da memória ROM. Dentro da CPU temos as seguintes entidades: ULA, Acumulador, Memória ROM, Program Counter, Somador, Flipflop para guardar o resultado do compare feito na ULA e 2 mux2x1.

Fora da CPU o projeto possui 6 conversores para representar os números nos displays da FPGA, Memória RAM (com bits individuais para Wright e Read enable), Duas bases de tempo com divisores diferentes, 2 mux para selecionar a base de tempo que vai ser usada e outro para habilitar a leitura da base de tempo em questão. Além disso há um FlipFlop com função Reset, essa entidade guarda o “clock” da base tempo.

O barramentos de dados da saída e entrada da CPU possuem 8 bits, assim como a entrada da ROM possibilitando 256 instruções.

6) Listagem dos pontos de controle e sua utilização (Também presentes no pdf do fluxo de dados no zip do projeto)

- Funções da ULA:

ULA	
00	Saída = A
01	A + B
10	Cmp ! A == B
11	Cmp A == B

- Pontos de controle

		OPCODE									
	Reset [17]	Flip Flop [16]	Hab Acumula [15]	ULA 1 [14]	ULA 0 [13]	Mux RAM/Ime [12]	Jpm [11]	Jmp condiciona [10]	Hab Escrita [9]	Hab Leitura [8]	Reservado [0,7]
	Loadim	0	1	0	0	0	0	0	0	0	
	Cmp not equal	1	0	1	0	0	0	0	0	0	
	Jmp	0	0	0	0	0	1	0	0	0	
	Store	0	0	0	0	0	0	0	1	0	
	Add	0	1	0	1	0	0	0	0	0	
	Jmp condicional	0	0	0	0	0	0	1	0	0	
	Cmp equal	1	0	1	1	0	0	0	0	0	
	Loadmem	0	0	0	0	1	0	0	0	1	
	Display	0	0	0	0	0	0	0	0	0	
Reset	1	0	0	0	0	0	0	0	0	0	

- Decoder do display:

Decoder	
001	Enable U segundo
010	Enable D segundo
011	Enable U minuto
100	Enable D minuto
101	Enable U hora
110	Enable D hora

7) Considerações Finais

- O projeto funcionou corretamente na FPGA, sendo possível alterar a base de tempo com a chaves SW(0), que acelera a base tempo para verificação do funcionamento e ajuste de hora.
- Poderiamos ter projetado o decoder de uma forma melhor para evitar alguns ajustes feitos de última hora para o projeto funcionar.
- Tivemos dificuldade com alguns componentes porque apesar de funcionar na simulação da entidade individual do quartus ao colocar no hardware não funcionavam como esperado. Isso aconteceu por conta de alguns códigos VHDL que não estavam com a lógica no melhor formato, o que acabava criando alguns componentes problemáticos como Latches, que podem prejudicar o funcionamento do processador.