Relatório de Desenvolvimento
Entrega Final
Arthur Chieppe e Vinicius Eller

Objetivos de avaliação com a Entrega:

A entrega final foi realizada almejando a nota **B**, para isso os critérios a seguir foram cumpridos:

- Possui algum sistema para acertar o horário, que não seja o aceleramento da base de tempo.
- Implementação de JNE (Instrução Extra).

Como utilizar:

- Seleção de base de tempo

O relógio possui duas bases de tempo, uma que conta em segundos e outra que conta de forma muito acelerada (97656 Hz), com o intuito de demonstrar a funcionalidade do relógio como um todo. Para ativar a base de tempo comum, basta desligar o chave 8 (SW8), para o teste acelerado, basta ativar a chave 8.

- Ajuste de horário

Outra funcionalidade do relógio, é ajustar seu horário. Para isso, optamos que o processo funcione por etapas, ajustando segundos, minutos e horas, separadamente. O processo possibilita a alteração dos segundos ao apertar o botão 0 (KEY0). Ao apertá-lo, o LED0 acenderá, indicando que o usuário está no modo de configuração dos segundos. Pressionando o botão 1 (KEY1), incrementamos a quantidade de segundos. Note que enquanto ajustamos as horas, o relógio não estará contando o tempo. Para alterar os minutos, devemos pressionar o botão 0 novamente, de forma que o LED0 se apague e o LED1 acenda. Assim, estamos alterando os minutos, que podem ser incrementados com o botão 1 até o minuto desejado. Por fim, ao apertar-se o KEY0 novamente, o LED 1 se apaga e o LED2 acende, indicando que estamos ajustando as horas. Como de praxe, alteramos a hora pressionando o KEY1, e concluímos o ajuste de horário ao apertar uma última vez no KEY0.

- Reiniciar relógio

Caso deseje reiniciar o relógio, basta pressionar o botão FPGA_RESET (KEY4), enquanto o relógio estiver contando, com isso, os mostradores serão zerados. Note que não é possível reiniciar a contagem enquanto o modo de ajuste de horário estiver em curso (com qualquer um dos LEDs 0, 1 ou 2 acesos).

Total de instruções e sua sintaxe (o nome de cada instrução com o tipo de argumento que ela utiliza);

Instrução	Função	Sintaxe
NOP	Sem operação	NOP
LDA	Carrega valor da memória para registrador	LDA %REG @MEM
SOMA	Soma valor da memória com o do registrador e armazena no registrador	SOMA %REG @MEM
SUB	Subtrai valor da memória com o do registrador e armazena no registrador	SUB %REG @MEM
LDI	Carrega valor do imediato para registrador	LDI %REG $VALOR
STA	Salva o valor do registrador na memória	STA %REG @MEM
JMP	Desvio de execução para o label requerido	JMP @LABEL
JEQ	Desvio de execução (condicionado a memória e registrador serem iguais) para o label requerido	JEQ @LABEL
CEQ	Compara se o registrador e a memória possuem o mesmo valor (baseado na subtração de ambos feita pela ULA)	CEQ %REG @MEM
JSR	Chamada de subrotina	JSR @SUBROTINA
RET	Retorno de subrotina	RET
CLT	Compara se o registrador é menor que a memória (baseado na subtração de ambos feita pela ULA e uso do bit mais significativo como confirmação de se o resultado é negativo/positivo)	CLT %REG @MEM
JLT	Desvio de execução (condicionado a memória ser menor que o registrador) para o label requerido	JLT @LABEL
ADDI	Adiciona o imediato ao registrador	ADDI %REG $VALOR
JNE	Desvio de execução (condicionado a uma comparação CEQ na qual a memória e o registrador são diferentes) para o label requerido	JNE @LABEL

Formato das instruções (distribuição dos campos na palavra de instrução);

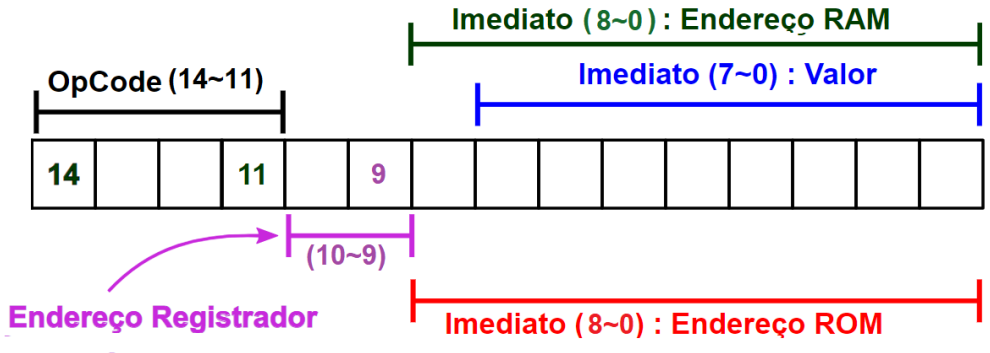

Figura - Imagem baseada na palavra de instrução da arquitetura Registrador-Memória aula 6, com as modificações necessárias para nosso projeto.

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

Para construir nosso processador com arquitetura Registrador-Memória, nos baseamos naquele construído em sala, com arquitetura acumulador.

Primeiramente, substituímos o registrador acumulador pelo modelo VHDL de Banco de Registradores, com o intuito de implementar 4 registradores de uso geral. Com mais registradores, conseguimos otimizar nosso processador para realizar uma mesma tarefa com um menor número de instruções e, por conseguinte, um menor número de *clocks.*

Outra modificação realizada no processador foi a adição de uma nova flag na ULA. Essa flag, apelidada de *flag_less*, é utilizada pela instrução CLT (*Compare Less Than) e* JLT (*Jump if Less Than)*. A flag apresenta nível lógico alto quando o inteiro no registrador escolhido é menor que o inteiro na posição de memória escolhida. Assim, quando a instrução JLT for chamada e a *flag-less* estiver em nível lógico alto, o desvio ocorre. Caso contrário, o código segue normalmente.

Essa instrução foi concebida no sentido de facilitar a comparação dos contadores com o número 10, para identificar quando deve-se chamar a subrotina de *overflow*. Um dos benefícios dela no lugar da comparação com a instrução CEQ, é que ela funcionará para qualquer número acima de 10 na comparação. Assim, ela é mais segura ao prevenir cenários em que o contador, por qualquer motivo (incluindo raios cósmicos), esteja com um valor maior que 10.

Além disso, outra instrução implementada na CPU foi a ADDI (*Add Immediate).* Com ela, é possível somar um valor imediato no registrador escolhido. Para implementá-la, partiu-se dos mesmos pontos de controle da instrução de SOMA, porém com uma mudança

no MUX que seleciona o valor da entrada B da ULA. Na instrução de soma, esse ponto de controle apresenta valor lógico baixo, o que faz com que a entrada B da ULA seja conectada à saída da RAM. No entanto, ao conectar o seletor do MUX ao nível lógico alto, pode-se somar a entrada A (registrador) com um imediato presente na instrução. Essa instrução possibilita um aumento da eficiência do projeto ao, por exemplo, inibir a necessidade de uma posição fixa de memória com valor 1 para incrementar algum registrador.

A instrução nova que implementamos na entrega final foi a introdução do **JNE *(Jump if Not Equal)*.** Utilizamos ela em conjunto com a instrução CEQ, sendo que o desvio ocorrerá se o conteúdo do registrador escolhido e da posição de memória não forem iguais. A instrução foi implementada invertendo o sinal da flag da comparação CEQ.

Listagem dos pontos de controle e sua utilização;

Pontos de Controle	Utilização
habEscritaMEM	Um único bit que habilita escrita na RAM
habLeituraMEM	Um único bit que habilita leitura na RAM
Operacao	Par de bits que indicam a ULA qual operação deve ser feita (Soma - 00, Subtração - 01, Passa - 10)
HabilitaEscritaRegistrador	Um bit que habilita escrita nos 4 registradores
MuxIntermediario	Um bit para selecionar se a entrada B da ULA vem da memória ou imediato
MuxJump	Par de bit que seleciona incremento do PC (00); Instrução (01); Endereço de Retorno (10)

Rascunho do diagrama de conexão do processador com os periféricos;

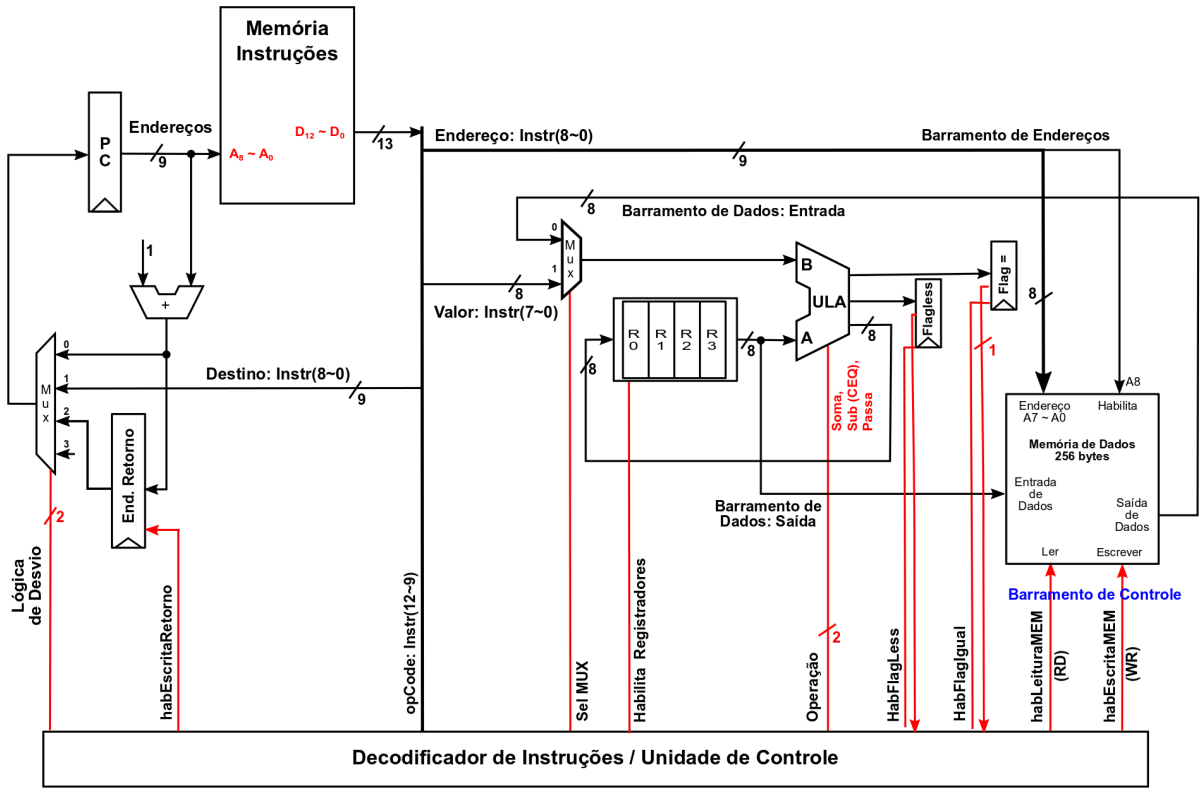

Figura - Imagem baseada no processador da aula 8. Com a adição de mais 3 registradores e um FlipFlop para armazenar a Flag para operação CLT.

Rascunho do mapa de memória;

- Rascunho do mapa de memória

Endereço em Decimal	Periférico	Largura dos Dados	Tipo de Acesso	Bloco (Página) de Memória
0 ~ 63	RAM	8 bits	Leitura/Escrita	0
64 ~ 127	Reservado	–	–	1
128 ~ 191	Reservado	–	–	2
192 ~ 255	Reservado	–	–	3
256	LEDR0 ~ LEDR7	8 bits	Escrita	4
257	LEDR8	1 bit	Escrita	4
258	LEDR9	1 bit	Escrita	4
259 ~ 287	Reservado	–	–	4

288	HEX0	4 bits	Escrita	4
289	HEX1	4 bits	Escrita	4
290	HEX2	4 bits	Escrita	4
291	HEX3	4 bits	Escrita	4
292	HEX4	4 bits	Escrita	4
293	HEX5	4 bits	Escrita	4
294 ~ 319	Reservado	–	–	4
320	SW0 ~ SW7	8 bits	Leitura	5
321	SW8	1 bit	Leitura	5
322	SW9	1 bit	Leitura	5
323 ~ 351	Reservado	–	–	5
352	KEY0 (aciona ajuste de horário)	1 bit	Leitura	5
353	KEY1 (incrementa)	1 bit	Leitura	5
354	KEY2	1 bit	Leitura	5
355	KEY3	1 bit	Leitura	5
356	FPGA_RESET (reinicia placa)	1 bit	Leitura	5
357	Base de Tempo	1 bit	Leitura	5
358 ~ 383				5
384 ~ 447	Reservado	–	–	6
448 ~ 508	Reservado	–	–	7
509	Limpa Leitura KEY1	1 bit	Escrita	7
510	Limpa Leitura KEY0	1 bit	Escrita	7
511	Limpa Base de Tempo	1 bit	Escrita	7

- Uso específico da memória RAM

Em nosso trabalho, reservamos certos endereços da memória RAM para usos específicos, a fim de padronizar e simplificar a programação em Assembly. Reservamos os endereços conforme a tabela a seguir:

End. RAM	Valor
0	Constante 0
1	Constante 1

2	Constante 2
4	Constante 4
6	Constante 6
30	Constante 10
10	Valor do Hex0
11	Valor do Hex1
12	Valor do Hex2
13	Valor do Hex3
14	Valor do Hex4
15	Valor do Hex5

Fonte do programa em assembly:

- O programa desenvolvido tem a seguinte estrutura:
    - Setup
    - Loop principal
    - Subrotinas

Para evitar erros inesperados, no setup define-se todos os valores de memória importantes (tabela acima) aos seus valores padrões. Isso também é conveniente pois possibilita a reutilização do setup ao pressionar o botão FPGA_RESET.

Outro ponto que merece a atenção é o funcionamento da subrotina de *Overflow*. Ao entrar nessa subrotina, já constatou-se que o overflow aconteceu no contador do HEX0. Assim, ela define o valor desse *display* e da memória correspondente como 0, além de incrementar o próximo *display* (HEX1). Em seguida, ele verifica se o overflow ocorreu também no HEX1. Em caso negativo, a subrotina retorna e o loop principal segue normalmente. Em casa afirmativo, a subrotina também zera o HEX1 e passa a incrementar o HEX2, seguindo essa lógica até o HEX5.

```
LDI %R1 $1              ## INICIO SETUP ##
STA %R1 @1
LDI %R1 $10
STA %R1 @30         # Posicao de memória fixa = 10
LDI %R1 $2
STA %R1 @2          # Posicao de memoria fixa = 2
LDI %R1 $4
STA %R1 @4          # Posicao de memoria fixa = 4
LDI %R1 $6
```

```
STA %R1 @6          # Posicao de memoria fixa = 6
LDI %R0 $0
STA %R0 @0          # Posicao de memoria fixa = 0
STA %R0 @10
STA %R0 @11
STA %R0 @12
STA %R0 @13
STA %R0 @14
STA %R0 @15
STA %R0 @288
STA %R0 @289
STA %R0 @290
STA %R0 @291
STA %R0 @292
STA %R0 @293
STA %R0 @510
STA %R0 @511
STA %R0 @257
STA %R0 @506
STA %R0 @505
LDI %R2 $0              # Registrador das unidades

LOOP:
LDA %R1 @356         # Verifica FPGA_RESET
CEQ %R1 @0
JEQ @0               # Se botao for pressionado vai para o inicio do
programa
LDA %R1 @352         # Verifica se usuario deseja mudar horario
(pressionando KEY0)
CEQ %R1 @0
JEQ @CHECA_INCREMENTO
JSR @ACERTA_HORARIO

CHECA_INCREMENTO:
LDA %R1 @357         # Checa se base de tempo permite incrementa
CEQ %R1 @0
JEQ @CONTINUA

INCREMENTA:
LDI %R0 $1           # Incrementa 1 ao tempo contado
STA %R0 @511
ADDI %R2 $1
STA %R2 @10

CONTINUA:
CLT %R2 @30
```

```
JLT @SALVA_HEX     # Se for menor que 10 salva no hex, caso contrario chama subrotina de overflow
JSR @OVERFLOW

SALVA_HEX:
STA %R2 @288
JMP @LOOP


OVERFLOW:            # Incrementa HEX1 e checa se é menor que 10, se for menor sai da subrotina
LDI %R2 $0
STA %R2 @10
STA %R2 @288
LDA %R0 @11
ADDI %R0 $1
CEQ %R0 @6
JEQ @HEX2
STA %R0 @289
STA %R0 @11
JMP @RETURN

HEX2:                # Incrementa HEX2 e checa se é menor que 10, se for menor sai da subrotina
LDI %R0 $0
STA %R0 @289
STA %R0 @11
LDA %R0 @12
ADDI %R0 $1
CEQ %R0 @30
JEQ @HEX3
STA %R0 @12
STA %R0 @290
JMP @RETURN

HEX3:                # Incrementa HEX3 e checa se é menor que 10, se for menor sai da subrotina
LDI %R0 $0
STA %R0 @290
STA %R0 @12
LDA %R0 @13
ADDI %R0 $1
CEQ %R0 @6
JEQ @HEX4
STA %R0 @13
STA %R0 @291
```

```
JMP @RETURN

HEX4:                # Incrementa HEX4 e checa se é menor que 10, se for
menor sai da subrotina
LDI %R0 $0
STA %R0 @291
STA %R0 @13
LDA %R0 @14
ADDI %R0 $1
CEQ %R0 @30
JEQ @HEX5
STA %R0 @14
STA %R0 @292
CEQ %R0 @4
JEQ @ATINGIU_4
JMP @RETURN

ATINGIU_4:           # Ao HEX5 chegar em 2, checa se hex4 esta abaixo
de 4 (Obejetivo, verificar as 24h)
LDA %R0 @15
CEQ %R0 @2
JNE @RETURN
JMP @LIMPA

HEX5:                # Incrementa HEX5 e checa se é menor que 10, se
for menor sai da subrotina
LDI %R0 $0
STA %R0 @292
STA %R0 @14
LDA %R0 @15
ADDI %R0 $1
STA %R0 @15
STA %R0 @293
JMP @RETURN

LIMPA:               # Zera os HEX referente a horas
LDI %R0 $0
STA %R0 @293
STA %R0 @15
STA %R0 @292
STA %R0 @14

RETURN:              # Sai da subrotina de overflow
RET

ACERTA_HORARIO:      # Subrotina de acertar horario
```

```
LDI %R0 $1
STA %R0 @510          # LIMPA KEY0
STA %R0 @256          # Acende LED0, indicando para o usuario que esta
alterando os segundos

LOOP_SEGUNDOS:
STA %R2 @288
STA %R2 @10
LDA %R0 @353
CEQ %R0 @1
JEQ @INCREMENTA_SEGUNDOS        # Se KEY1 for pressioando, incrementa os
segundos.
LDA %R0 @352
CEQ %R0 @1
JEQ @SETUP_MINUTOS              # Se KEY0 for pressionado, vai para o
ajuste de minutos.
JMP @LOOP_SEGUNDOS

INCREMENTA_SEGUNDOS:            # Logica para incrementar os segundos,
de forma a nao estourar os valores possiveis.
LDI %R0 $1
STA %R0 @509
ADDI %R2 $1
CLT %R2 @30
JLT @LOOP_SEGUNDOS              # Se valor nao exceder 9, continua o
loop.
LDI %R2 $0
LDA %R3 @11
ADDI %R3 $1

CLT %R3 @6
STA %R3 @289
STA %R3 @11
JLT @LOOP_SEGUNDOS              # Se a dezena do segundo exceder 5, zera
tanto dezena quanto unidades
LDI %R2 $0
STA %R2 @288
STA %R2 @10
STA %R2 @289
STA %R2 @11
JMP @LOOP_SEGUNDOS

SETUP_MINUTOS:                  # Desliga o LED0 e acende o LED1,
sinalizando ao usuario que esta alterando os minutos.
LDI %R0 $1
STA %R0 @510                    # LIMPA KEY0
```

```
LDI %R0 $2
STA %R0 @256
JMP @LOOP_MINUTOS

LOOP_MINUTOS:
LDA %R0 @353
CEQ %R0 @1
JEQ @INCREMENTA_MINUTOS
LDA %R0 @352
CEQ %R0 @1
JEQ @SETUP_HORAS
JMP @LOOP_MINUTOS

INCREMENTA_MINUTOS:
LDI %R0 $1
STA %R0 @509
LDA %R0 @12
ADDI %R0 $1
CLT %R0 @30
STA %R0 @290
STA %R0 @12
JLT @LOOP_MINUTOS
LDI %R0 $0
STA %R0 @290
STA %R0 @12
LDA %R3 @13
ADDI %R3 $1
CLT %R3 @6
STA %R3 @291
STA %R3 @13
JLT @LOOP_MINUTOS
LDI %R0 $0
STA %R0 @290
STA %R0 @12
STA %R0 @291
STA %R0 @13
JMP @LOOP_MINUTOS

SETUP_HORAS:                    # Liga o LED3, indicando ao usuario
que esta alterando as horas.
LDI %R0 $1
STA %R0 @510                    # LIMPA KEY0
LDI %R0 $4
STA %R0 @256
JMP @LOOP_HORAS
```

```
LOOP_HORAS:
LDA %R0 @353
CEQ %R0 @1
JEQ @INCREMENTA_HORAS
LDA %R0 @352
CEQ %R0 @1
JEQ @RETURN_HORARIO
JMP @LOOP_HORAS

INCREMENTA_HORAS:
LDI %R0 $1
STA %R0 @509

LDA %R0 @14
ADDI %R0 $1
CEQ %R0 @4
STA %R0 @292
STA %R0 @14
JEQ @ATINGIU_4_SET                  # Quando a unidade das horas atingir
4, verifica se as dezenas tem valor igual a 2. Se sim, reseta.
CLT %R0 @30
STA %R0 @292
STA %R0 @14
JLT @LOOP_HORAS
LDI %R0 $0
STA %R0 @292
STA %R0 @14
LDA %R3 @15
ADDI %R3 $1
STA %R3 @293
STA %R3 @15
JMP @LOOP_HORAS

ATINGIU_4_SET:
LDA %R1 @15                          # Ao HEX5 chegar em 2, checa se hex4
esta abaixo de 4
CEQ %R1 @2
JNE @LOOP_HORAS
LDI %R1 $0
STA %R1 @292
STA %R1 @14
STA %R1 @293
STA %R1 @15
JMP @LOOP_HORAS

RETURN_HORARIO:                      # Desliga os LEDS e limpa leitura do
```

```
botao.
LDI %R0 $1
STA %R0 @510
LDI %R0 $0
STA %R0 @256
RET
```