Processador-Relogio

Design de Computadores sexto semestre.

Relatório Intermediário

```
Pseudocódigo do relógio;

Total de instruções e sua sintaxe;

Formato das instruções;

Modos de endereçamento utilizados;

Arquitetura do processador;

Diagrama de conexão do processador com os periféricos;

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

Listagem dos pontos de controle e sua utilização;

Mapa de memória. (Nossa arquitetura não possui RAM);
```

1)Pseudocódigo está nos arquivos relogio.c e relogio.s

Ainda precisamos escrever um loop principal em Assembly que espera o valor do registrador(que vira 1 quando passa 1 segundo via hardware) e, quando o valor for 1, dar um jmp para o Assembly já feito. Ao final do Assembly, voltar para o loop principal

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
   int seg1 = 0;
   int seg2 = 0;
   int min1 = 0;
   int min2 = 0;
   int h1 = 0;
   int h2 = 0;

   while(1){
       while( h2!=2 || h1!=4 ){
           while(min2!=5 || min1!=9){
               while( seg2!=5 || seg1!=9 ){
                       if(seg1<9){
```

```
                            seg1++;
                            //printf("%d%d:%d%d:%d%d\n", h2, h1, min2,
min1, seg2, seg1);
                            //sleep(0,3);
                        }
                        else{
                            seg1 = 0;
                            if(seg2<5) seg2++;
                            //printf("%d%d:%d%d:%d%d\n", h2, h1, min2,
min1, seg2, seg1);
                            //sleep(0,3);
                        }
                }
                seg1 = 0;
                seg2 = 0;
                if(min1 < 9) min1++;
                else{
                    min1 = 0;
                    min2++;
                }
            }
            min1 = 0;
            min2 = 0;
            if(h1<9) h1++;
            else{
                h1 = 0;
                h2++;
            }
        }
        h1 = 0;
        h2 = 0;
    }
}
```

Código assembly do modelo

```
.file   "relogio.c"
        .text
        .globl  main
        .type   main, @function
main:
.LFB41:
        .cfi_startproc
        mov     $0, %edi
        mov     $0, %esi
        jmp     .L13
.L18:
        cmp     $8, %ecx
        jg      .L7
        add     $1, %ecx
```

```
        cmp     $5, %r8d
        jne     .L12
        cmp     $9, %ecx
        jne     .L12
        cmp     $8, %esi
        jg      .L10
        add     $1, %esi
.L13:
        cmp     $2, %edi
        jne     .L15
        cmp     $4, %esi
        jne     .L15
        mov     $0, %edi
        mov     $0, %esi
        jmp     .L13
.L7:
        add     $1, %r8d
        mov     $0, %ecx
.L12:
        mov     $0, %edx
        mov     $0, %eax
.L14:
        add     $1, %eax
.L5:
        cmp     $5, %edx
        jne     .L6
        cmp     $9, %eax
        je      .L18
.L6:
        cmp     $8, %eax
        jle     .L14
        mov     $0, %eax
        cmp     $4, %edx
        jg      .L5
        add     $1, %edx
        jmp     .L5
.L10:
        add     $1, %edi
        mov     $0, %esi
        jmp     .L13
.L15:
        mov     $0, %r8d
        mov     $0, %ecx
        jmp     .L12
        .cfi_endproc
.LFE41:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .section        .note.GNU-stack,"",@progbits
```

O código assembly acimea foi gerado a partir do relogio.c, dessa forma foi possível atrelar cada registrador utilizado com suas variáveis em C, chegando na seguinte relação:

- H2 : EDI
- H1 : ESI
- S2 : EDX
- S1 : EAX
- M2 : R8D
- M1 : ECX

Portanto vamos utilizar 6 registradores

2) Total de instruções e sua sintaxe;

```
| Instruções             |Binário |
| MOV   | (move)           | 0000    |
| JMP   | (jump)           | 0001    |
| CMP   |  (compare)       | 0010    |
| JG    | (if jump a>b)    | 0011    |
| ADD   | (add a+b )       | 0100    |
| JNE   | (jump  if< 0)    | 0101    |
| JE    | (jump equal a=b) | 0110    |
| SUB   | (sub a-b)        | 0111    |
| MOVLCD| (move LCD)       | 1000    |
```

- O total de instruções é 9.
- Elas possuem o seguinte formato :

OPCODE	REGISTRADOR	RESERVADO
4bits	4bits	4bits

Modos de endereçamento

- Endereçamento imediato
  - Exemplo:

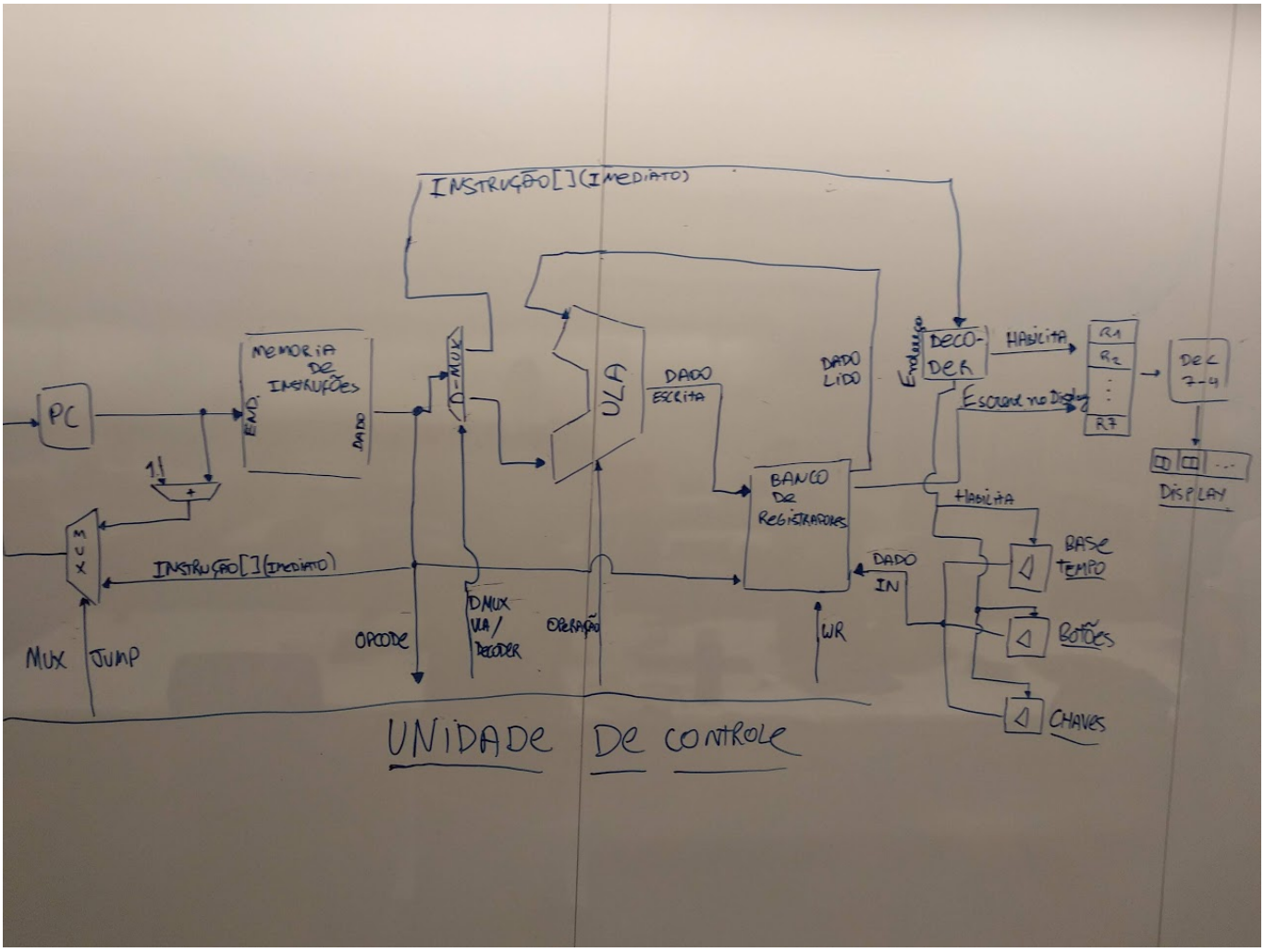
```
IMEDIATO:
mov $0, %ecx
LCD:
movlcd %edi, $1;
(Coloca o conteúdo do edi na primeiro segmento do display)
```

3) Arquitetura do processador;

- Para este projeto estaremos utilizando uma arquitetura Registrador-Memória.

O processo de decisão da arquitetura do relógio foi baseado na conversão do pseudocódigo em assembly. No momento em que realizamos a conversão, percebemos que todas as instruções necessárias para o funcionamento integral do relógio poderiam ser executadas através de operacoes entre um registrador e um imediato. Nesse sentido, usamos a instrucao mov, que é responsável por buscar o conteudo de um registrador específico do banco de registradores e realizar alguma operacao deste com um imediato.

4) Diagrama de conexão do processador com os periféricos;



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

A imagem a baixo mostra um diagrama para o nosso relógio com os periféricos

Explicação do fluxo de dados:

```
A partir de uma determinada instrução, esperamos a leitura do registrador
da base de tempo indicar que um segundo já passou para executar o loop de
instruções desencadeado pelo add de um no registrador do primeiro dígito
dos segundos.
```

Considerando essa mesma instrução add, por exemplo, a memoria de instruções transforma o conteúdo do program counter numa instrucao (Opcode + End[reg] + Imediato). Em seguida, envia-se essa instrução para a unidade de controle e a mesma gera os pontos de controle específicos para a execucao deste comando. Depois, o demux escolhe se a instrução vai para a ULA ou para o decoder de endereços, ou seja, para todos os casos exceto os movlcd, o imediato vai para a ULA.

Enquanto o imediato vai para a ULA, a instrução também vai para o banco de registradores, onde o conteúdo do endereço do registrador em questão é procurado e enviado para a entrada superior da ULA e também para os registradores do display. No caso dos registradores do display, os mesmos não estarão habilitados a menos que a instrução seja movlcd, o que impede escritas involuntárias.

Na ULA, a operação add é executada entre o imediato e o conteúdo que foi devolvido pelo banco de registradores. O resultado é armazenado no mesmo registrador envolvido na operação.

6) Listagem dos pontos de controle e sua utilização

- Dmux ULA/Decoder
- Operação
- Write/Read
- Mux Jump