INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

LEIC, LEETC

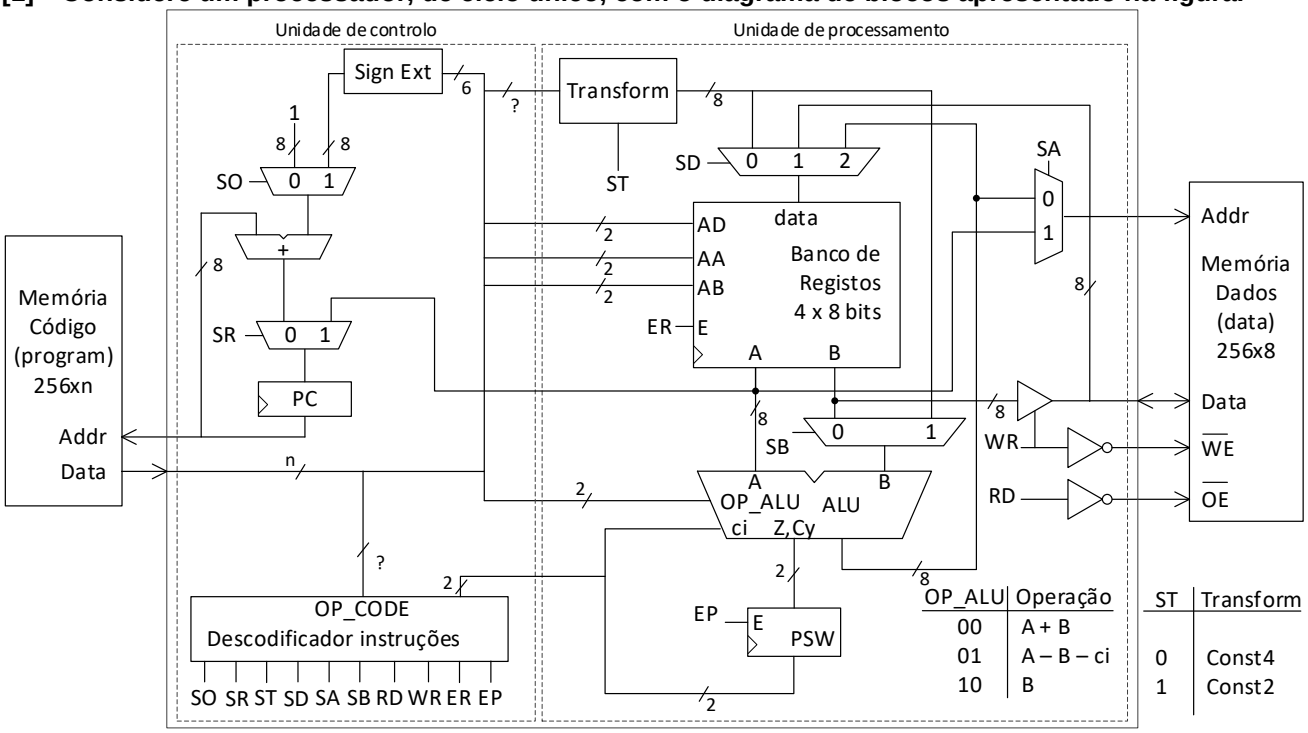
Arquitetura de Computadores

2º Teste (23/jan/2019)

Duração do Teste: 2 horas e 30 minutos

---

[1] Considere um processador, de ciclo único, com o diagrama de blocos apresentado na figura.



O processador suporta a execução do seguinte conjunto de instruções, em que as constantes `const2` e `const4` representam números naturais e a constante `offset6` representa um número relativo:

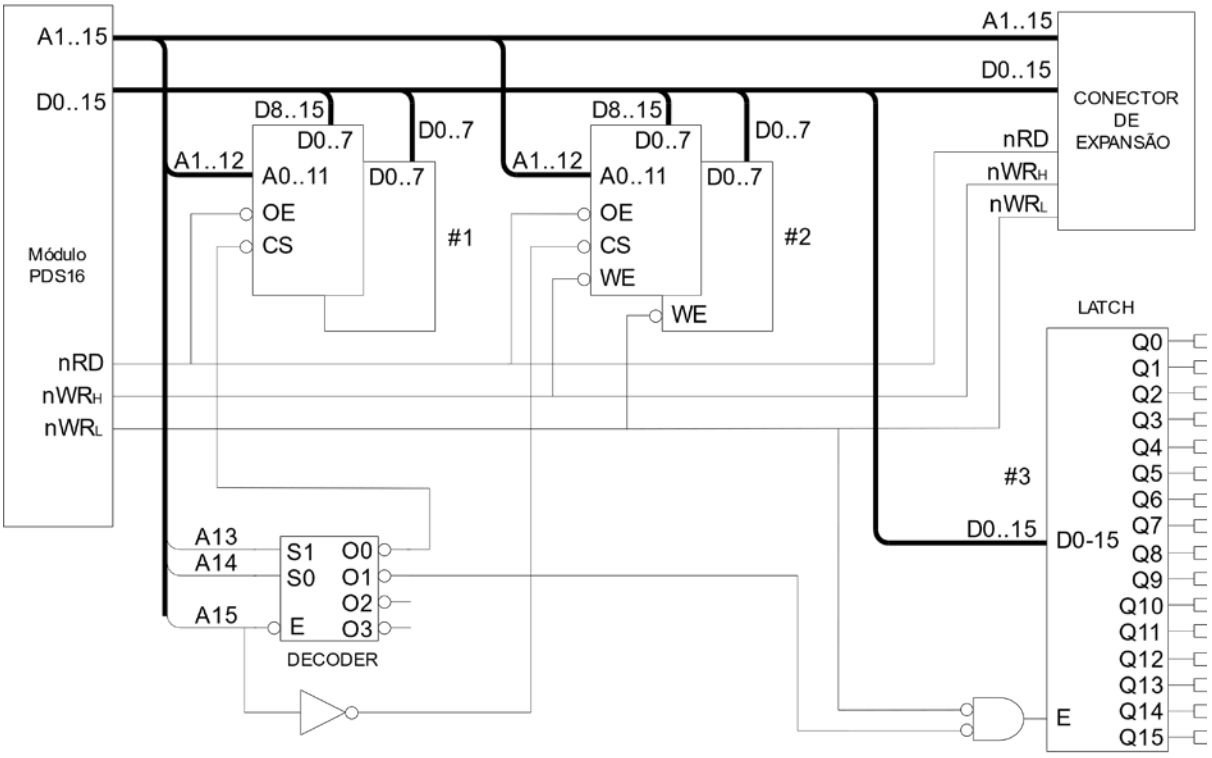
N.º	Instrução	Codificação									Descrição
		b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	`ldi rx,#const4`	0	0	0	rx_1	rx_0	c_3	c_2	c_1	c_0	`rx = const4`
2	`ld rx,[ry]`	A definir									`rx = M[ry]`
3	`st rx,[ry, #const2]`	A definir									`M[ry+const2] = rx`
4	`subb rx,ry,rz`	1	?	?	rx_1	rx_0	ry_1	ry_0	rz_1	rz_0	`rx = ry – rz - ci`
5	`mov rx,ry`	1	?	?	rx_1	rx_0	ry_1	ry_0	0	0	`rx = ry`
6	`jmp offset6`	0	0	1	O_5	O_4	O_3	O_2	O_1	O_0	`PC = PC + offset6`
7	`jnz rx`	A definir									`(Z == 0) ? PC = rx : PC = PC + 1`

a) Complete os bits de código das instruções subb e mov. Codifique as instruções `ld`, `st` e `jnz` utilizando uma codificação linear a 3 bits. Explicite os bits do código de instrução que correspondem aos sinais AA, AB, AD, OP_ALU e OPCODE. [2,5 val.]

b) Considerando que o módulo `Descodificador Instruções` é implementado usando exclusivamente uma ROM, indique a programação da mesma. [2,0 val.]

c) Proponha, justificando, um diagrama lógico para o módulo `Transform`. [0,5 val.]

[2] Considere o sistema computacional baseado no PDS16 representado na figura.



A este sistema pretende-se adicionar, <u>através do conector de expansão</u>, realizando a seleção de endereços da forma mais simplificada possível:

- RAM, com a dimensão de 16 Kbyte, suportando acesso a *byte* e a *word*, selecionada em endereços contíguos à RAM existente;
- Um porto de entrada de 8 bits;
- Um porto de saída de 16 bits, suportando acesso a *byte* e a *word*. Sugere-se que utilize o endereço coincidente com a ROM existente.

a) Desenhe o mapa de endereçamento do sistema, incluindo os dispositivos adicionais. Indique, para cada dispositivo, a referência (#...), a funcionalidade, a dimensão, os endereços de início e de fim do espaço atribuído e a eventual existência de *fold-back*. [1.5 val.]

b) Desenhe a ligação, ao conector de expansão, da nova RAM e dos circuitos para a sua seleção de endereços. Utilize circuitos integrados de RAM de 8K*8. [1.5 val.]

c) Desenhe a ligação, ao conector de expansão, dos novos portos e dos circuitos para a respetiva seleção de endereços. Utilize circuitos integrados *Latch* e *Tri-state* com dimensão de 8 bits. [1.5 val.]

d) Considere o acesso ao dispositivo #3 pela execução, em alternativa, de uma das instruções: (1) STB R0,[R1, #0]; (2) STB R0,[R1,#1]; (3) ST R0,[R1, #0]. [0.5 val.]

Admita os valores iniciais seguintes: sinais Q do dispositivo #3 = 0x5555; Registo R0 = 0x12AB; R1 = base do espaço de endereçamento atribuído ao dispositivo #3. Indique o valor que ficaria nos sinais Q para o caso de cada uma das opções indicadas.

[3] Considere a seguinte função expressa em linguagem C:

```
int8 ascCompare (uint8[] str1, uint8[] str2)
{
  uint8 idx1 = 0;
  uint8 idx2 = 0;

  while (str1[idx1] != 0 && (str1[idx1] == str2[idx2])) {
       idx1++;
       idx2++;
  }
  return str1[idx1] - str2[idx2];
}
```

a) Traduza para linguagem *assembly* do PDS16 a função `ascCompare()` que compara a *string* str1 com a *string* str2. A função ascCompare () retorna um inteiro maior que, igual a ou menor que zero, conforme a string str1 é maior que, igual a ou menor que a *string* str2. As *string* estão codificada em ASCII e terminada por 0. Defina as variáveis que entender necessário [2,5 val.]

b) Considere as definições seguintes e a função main.

```
uint8 msg1[] = "Arquitetura";
uint8 msg2[] = "Arquitectura";
uint16 comp;

void main( void ){
      comp = ascCompare( msg1, msg2 );
}
```

Traduza para linguagem *assembly* do PDS16 as definições referias e a função main [2,5 val.]

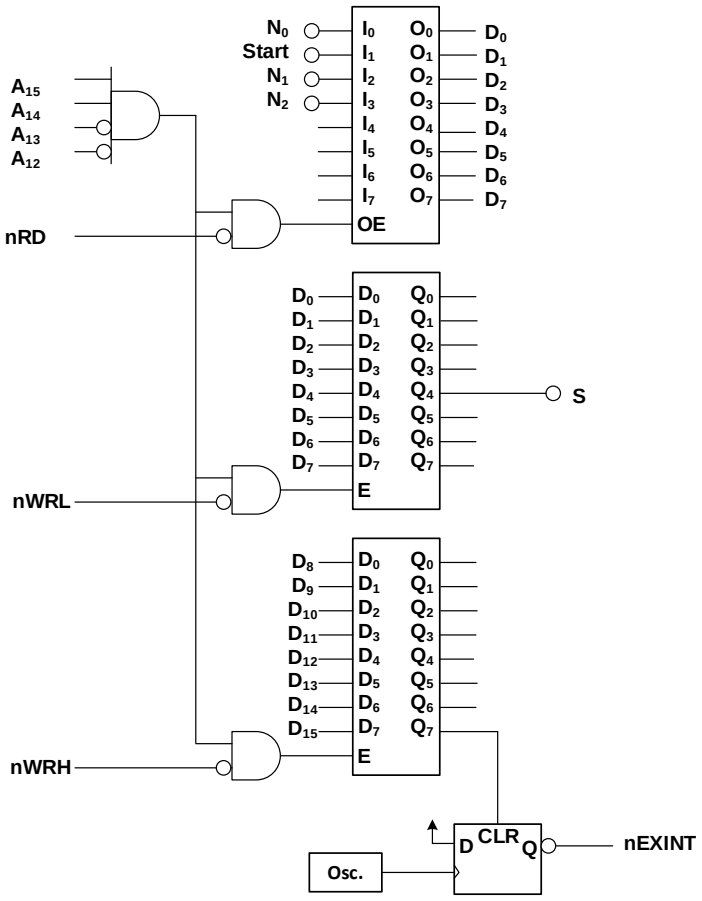
Notas:

1. Com vista ao alojamento de variáveis, assuma que a secção "`.data`" está localizada na zona de memória acessível com endereçamento direto.

2. Na programação em *assembly* deve usar as seguintes convenções: os parâmetros das funções são passados em registos, ocupando a quantidade necessária, pela ordem `r0`, `r1`, `r2` e `r3`; o valor de retorno de uma função, caso exista, é devolvido em `r0`; `int8` e `int16` significam valores inteiros com sinal representados a 8 e a 16 bit, respetivamente; `uint8` e `uint16` significam valores inteiros sem sinal representados a 8 e a 16 bit. A função preserva os registos que utiliza para além dos usados para parâmetros.

[Escreva aqui]

[4] Considere que se pretende produzir sobre a saída S sequências de impulsos. A sequência inicia-se com a transição descendente do sinal **`Start`** e tem um número de impulsos igual ao valor definido pelas entradas **N** do porto de entrada. A duração do impulso, assim como a duração do intervalo entre impulsos é igual ao período do sinal do oscilador **Osc**.



Implemente em linguagem *assembly* do PDS16 o componente de programa definido em cada uma das alíneas.

a) A função **`uint8_t get_number()`** que devolve o valor representado em binário pelas entradas **N_0**, **N_1** e **N_2**. [1 val.]

b) A função **`uint8_t get_start()`** que devolve 1 se for detetada transição de 1 para 0 no sinal **`Start`** e devolve 0 no caso contrário. A implementação desta função não deve realizar esperas. Deve detetar a transição por análise do valor de **`Start`** entre chamadas consecutivas. [1 val.]

c) A função **`void output_signal(uint8_t value)`** que afeta a saída **S** com o valor do bit de menor peso do parâmetro **`value`** e afeta com zero os restantes bits do porto de saída. [1 val.]

d) A rotina de atendimento de interrupção que se encarrega de fazer evoluir o sinal **S**. [1 val.]

e) O programa principal fazendo uso dos componentes de programa definidos nas alíneas anteriores. [1 val.]