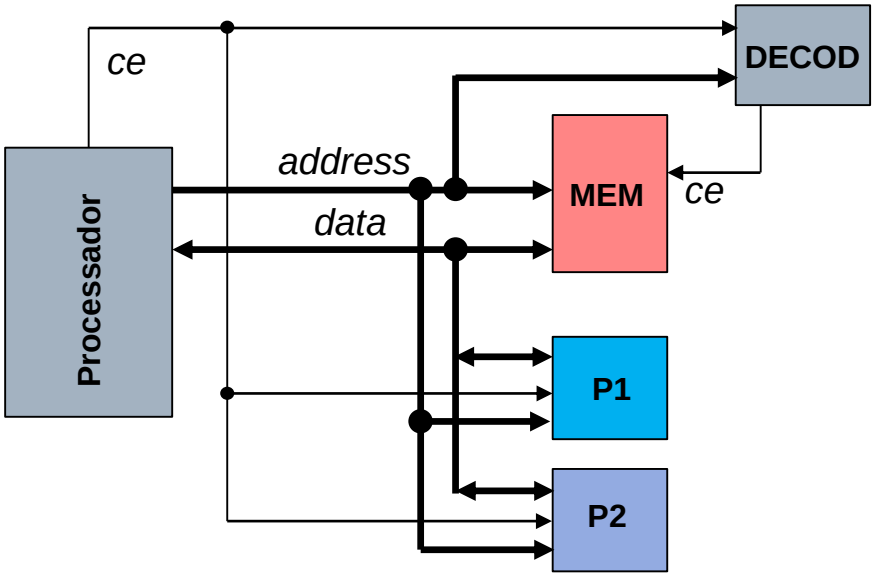
Interface entre Processador e Periféricos

Sub-sistema de entrada e saída – E/S (I/O)

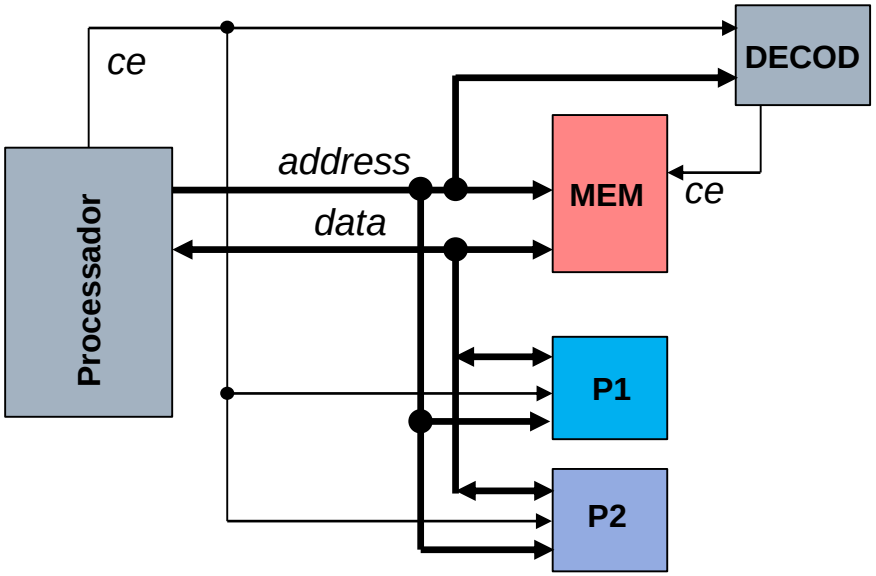
Interface entre Processador e Periféricos

- Transferência de dados
  - Tipicamente, uma operação de E/S envolve a transferência de dados entre a memória e periférico
    - Processador lê da memória em envia para periférico
    - Processador lê do periférico e armazena na memória



Interface entre Processador e Periféricos

- Transferência de dados
  - Existem basicamente três técnicas envolvidas na transferência de dados entre memória e periféricos
    - *Polling*
    - Interrupção
    - Acesso direto à memória (DMA – *Direct Memory Access*)



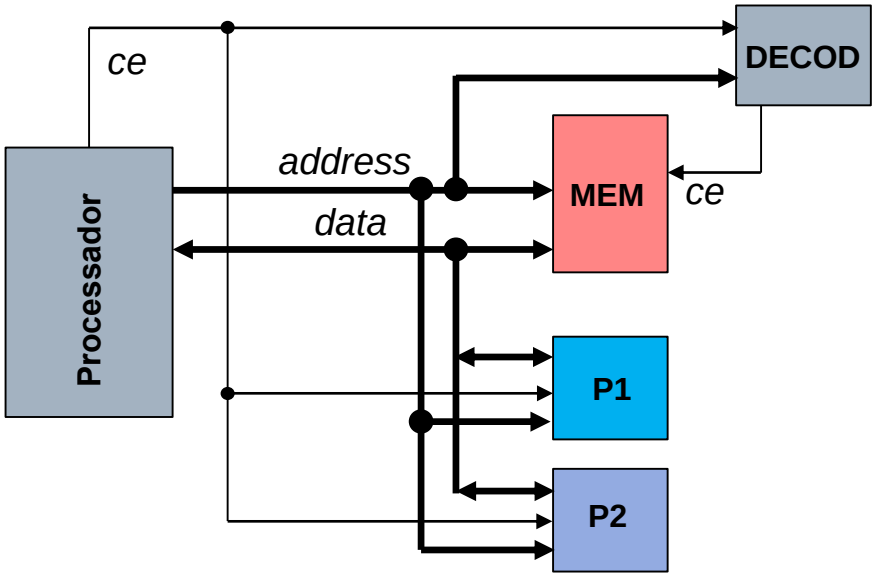
Interface entre Processador e Periféricos

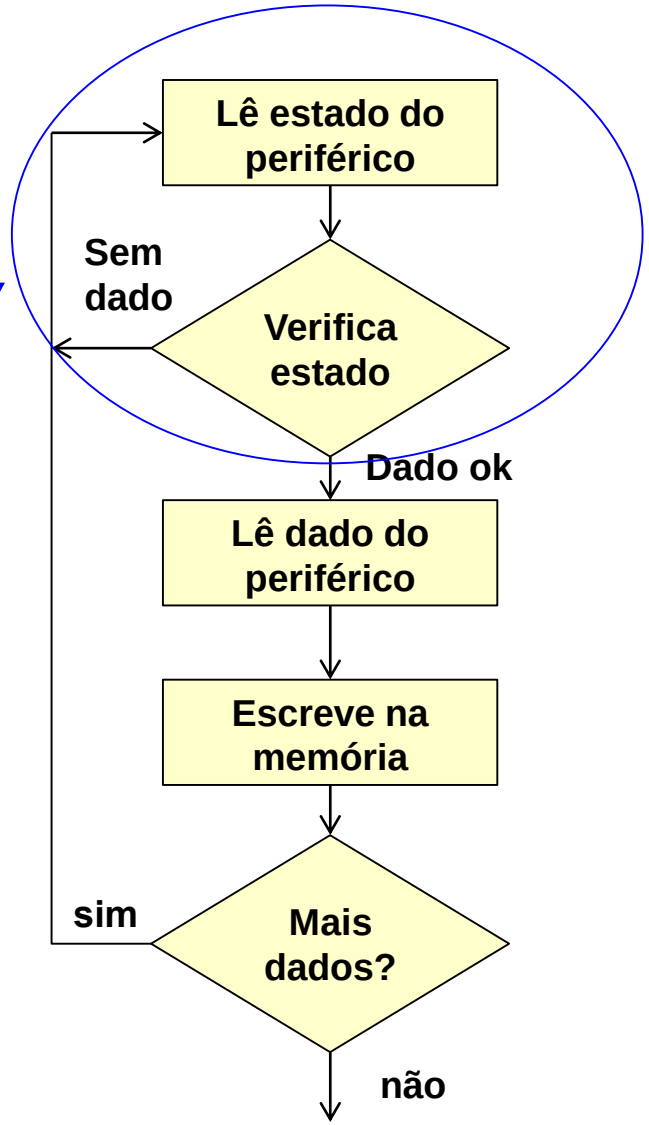
- *Polling*
  - Técnica mais simples
  - O processador controla toda a transferência de dados entre a memória e periféricos (software)
  - Antes de iniciar uma transferência, o processador verifica periodicamente o estado do periférico (e.g. registrador de *status*) para saber se
    - Existe dado disponível para leitura
    - O periférico está apto a receber dados
  - Visto que o processador é mais rápido que os periféricos, essa espera representará um desperdício de tempo de processamento

Interface entre Processador e Periféricos

- *Polling*
  - Fluxograma de uma operação de leitura de um periférico

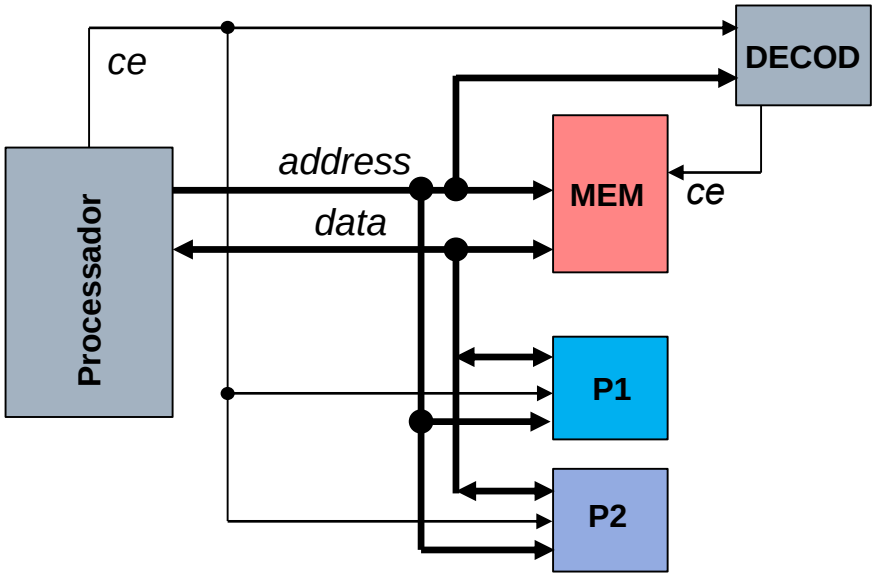
Loop que caracteriza o *polling*

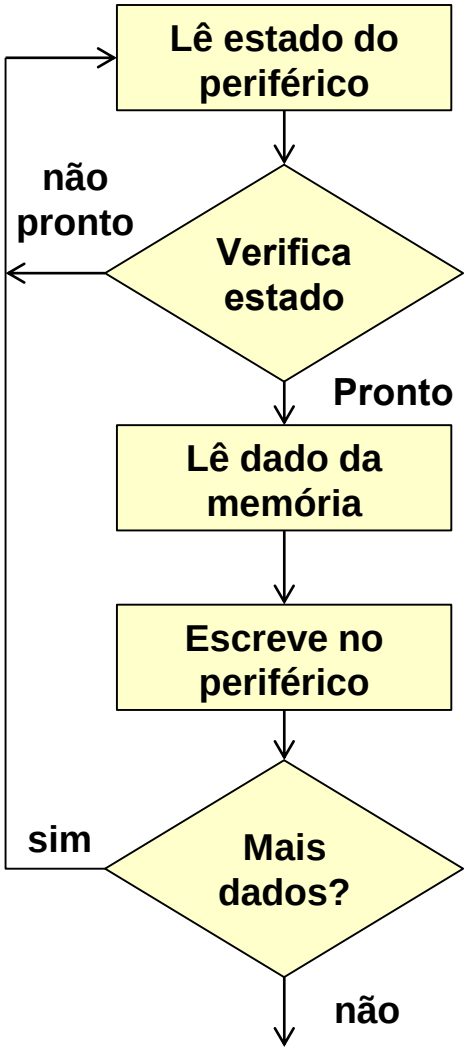




Interface entre Processador e Periféricos

- *Polling*
  - Fluxograma de uma operação de escrita em um periférico



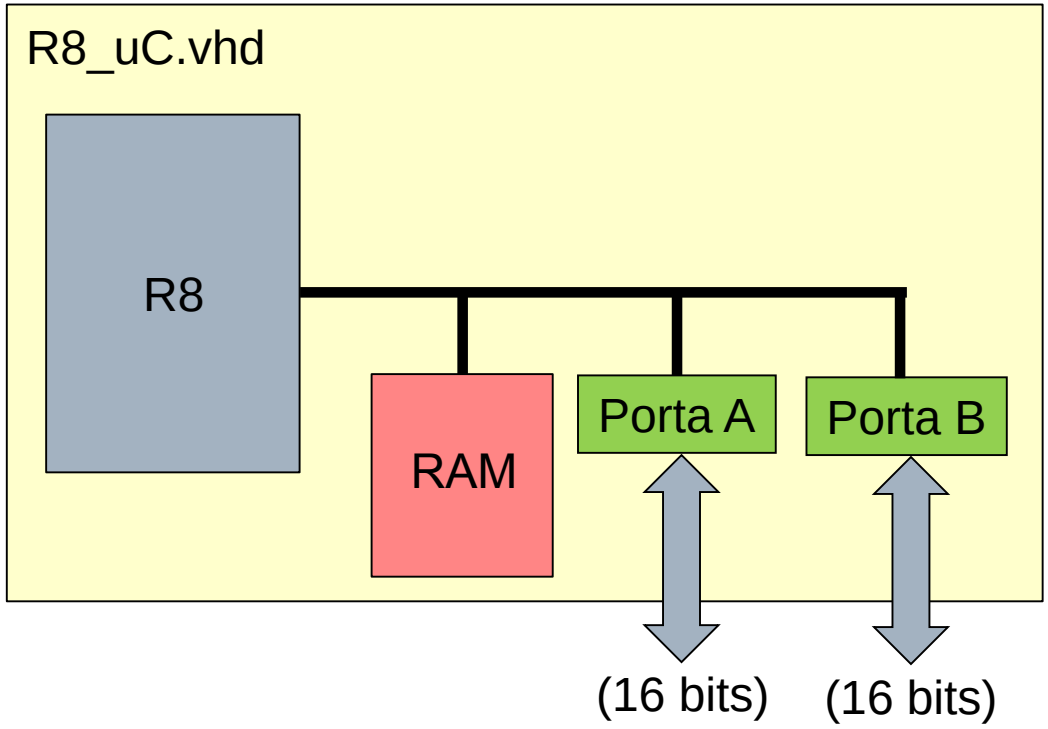


Interface entre Processador e Periféricos

- *Polling*
  - A principal vantagem da E/S com *polling é a sua* simplicidade
    - Baixo custo
    - Envolve apenas software
  - A desvantagem é que o processador perde muito tempo verificando o estado do periférico
  - Otimização
    - Verificar o estado do periférico de tempos em tempos
    - Exemplo: a cada 100ms
  - Mais informações
    - *Computer organization and architecture – William Stallings*
    - *Computer organization and design – David Patterson/John Hennessy*

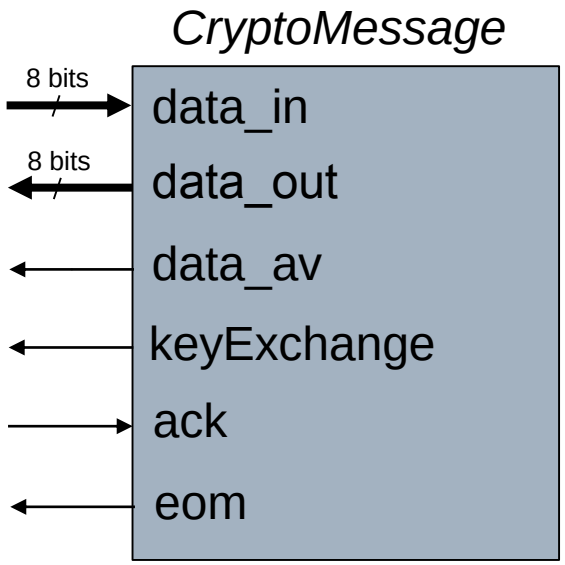
Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - O objetivo do trabalho será adicionar mais uma porta de E/S e periféricos externos ao microcontrolador



Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - *CryptoMessage*
    - Envia mensagens criptografadas
    - Descrição VHDL no *moodle*

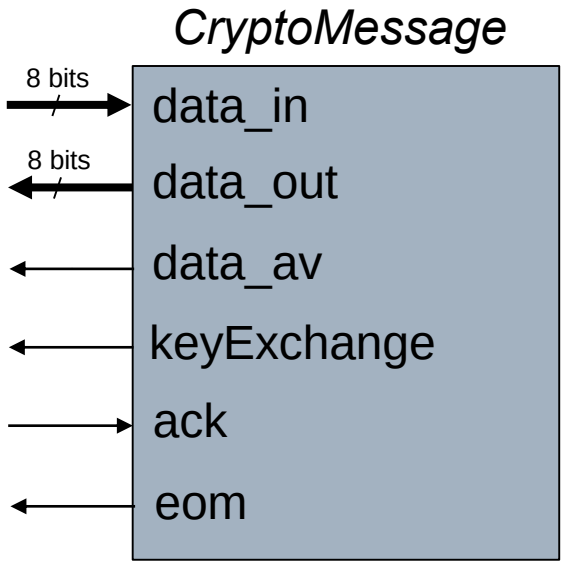


Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - *CryptoMessage*
    - Interação com o R8_uC

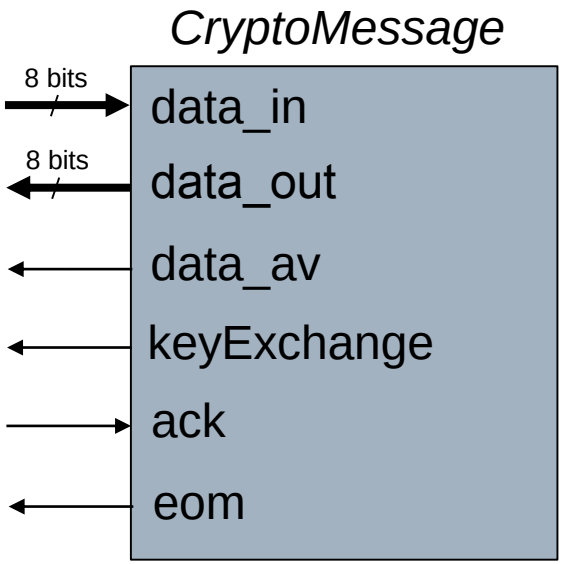
1. *CryptoMessage* ativa *keyExchange* e coloca no barramento *data_out* seu *magicNumber*
2. *R8_uC* lê o *magicNumber* e calcula o seu *magicNumber*
3. *R8_uC* coloca o seu *magicNumber* no barramento *data_in* do *CryptoMessage* e gera um pulso em *ack. Feito isso, ambos calculam a chave criptografica.*
4. *CryptoMessage* coloca um caracter da mensagem criptografado no barramento *data_out* e ativa *data_av*
5. *R8_uC* lê o caracter e gera um pulso em *ack*

Os passos 4 e 5 se repetem até o final da mensagem. Quando o *CryptoMessage* colocar o último caracter da mensagem em *data_out*, *eom* (*end-of-message*) estará ativo também, indicando o fim da mensagem.
Após a transmissão de uma mensagem completa, o processo se repete a partir do passo 1

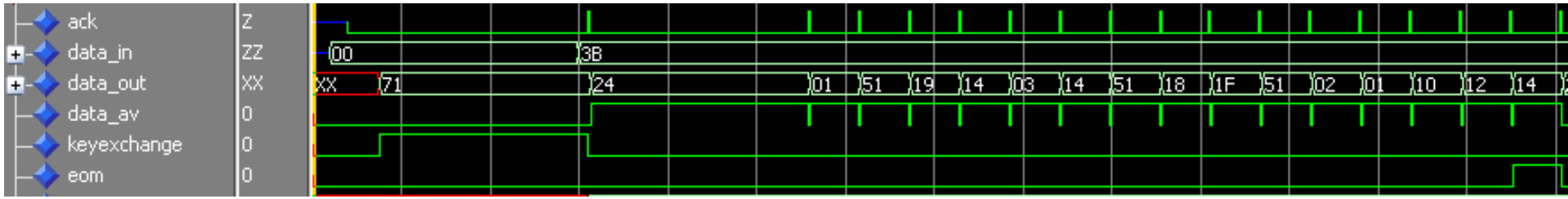


Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - *CryptoMessage*
    - Interação com o R8_uC



1. *CryptoMessage* ativa *keyExchange* e coloca no barramento *data_out* seu *magicNumber*
2. *R8_uC* lê o *magicNumber* e calcula o seu *magicNumber*
3. *R8_uC* coloca o seu *magicNumber* no barramento *data_in* e gera um pulso em *ack. Feito isso, ambos calculam a chave criptografica.*
4. *CryptoMessage* coloca um caracter da mensagem criptografado no barramento *data_out* e ativa *data_av*
5. *R8_uC* lê o caracter e gera um pulso em *ack*



Envio de uma mensagem completa

Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - *CryptoMessage*
    - Parâmetros

Tempo ocioso após o envio de uma mensagem completa

Nome do arquivo texto contendo as mensagens. Cada linha corresponde a uma mensagem.

```
entity CryptoMessage is
    generic (
        MSG_INTERVAL     : integer;     -- Clock cycles
        FILE_NAME        : string := "UNUSED"
    );
    port (
        clk          : in std_logic;
        rst          : in std_logic;
        ack          : in std_logic;
        ...
```

Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Ligar dois *CripoMessages* ao *R8_uC*
    - Da maneira que preferirem
  - Deve-se configurar adequadamente os bits das portas de E/S a fim que o *R8_uC* tenha acesso às interfaces dos periféricos
  - A verificação do estado dos periféricos (*keyExchange* e *data_av*) a fim de realizar tranferência de dados deve ser implementada em software (*polling*)

Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Encriptação/Decriptação
    - Código ASCII do caracter ***xor*** chave
    - Exemplo: caracter *a* (ASCII: 0x61)
      - Supondo chave = **0x25**
      - Encritação: 0x61 *xor* **0x25** = 0x44
      - Decriptação: 0x44 *xor* **0x25** = 0x61

Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Cálculo da chave (*Diffie-Hellman key exchange*)
    - Baseado na equação: $a^b \ mod \ q$
    - Inicialmente as partes envolvidas na comunicação (*A* e *B*) devem acordar em dois números
      - *q*: número primo
      - *a*: raiz primitiva de *q*
      - Usaremos $q = 251$ e $a = 6$
    - Para iniciar o cálculo da chave, *A* escolhe um número randômico $x < q$ e calcula seu $magicNumberA_A$
      - $magicNumber_A = a^x \ mod \ q$
    - *B* escolhe um número randômico $y < q$ calcula seu $magicNumber_B$
      - $magicNumber_B = a^y \ mod \ q$

Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Cálculo da chave (*Diffie-Hellman key exchange*)
    - Em seguida *A* e *B* trocam seus *magicNumbers* e calculam a chave da seguinte forma
      - *A*: $chave = magicNumber_B^x \bmod q$
      - *B*: $chave = magicNumber_A^y \bmod q$
    - Por incrível que pareça, *A* e *B* obtém o mesmo valor de chave!
    - O número randômico selecionado pelo R8 pode ser gerado a partir de um contador
      - Este número tem de ser menor que 251

Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Algoritmo para calcular $a^b \ mod \ q$

```
c ← 0; f ← 1
for i ← k downto 0
    do c ← 2 × c
       f ← (f × f) mod n
    if bi = 1
       then c ← c + 1
            f ← (f × a) mod n
return f
```

Note: The integer b is expressed as a binary number $b_k b_{k-1} \ldots b_0$.

Figure 9.8 Algorithm for Computing a^b mod n

Não precisamos calcular *c*
Resultado é retornado em *f*
Todas variáveis tem 1 byte
Atenção à variável *n* do algoritmo, ela é o nosso *q*

Bit *i* de *b*

Este algoritmo está implementado em VHDL no CriptoMessage (*function ExpMod*)

REF: CRYPTOGRAPHY AND NETWORK SECURITY *PRINCIPLES AND PRACTICE* - William Stallings

Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Adicionar ao processador dois registradores especiais de 16 bits: *high* e *low*. Estes registradores devem armazenar o resultado das instruções de multiplicação e divisão
  - Implementar as instruções MUL e DIV
    - MUL *reg1*, *reg2*
      - *high* = 16 bits mais significativos de *reg1 * reg2*
      - *low* = 16 bits menos significativos de *reg1 * reg2*
    - DIV *reg1*, *reg2*
      - *high* = resto da divisão inteira (*reg1 % reg2)*
      - *low* = quociente da divisão inteira (*reg1 / reg2)*
    - Em VHDL utilizar os operadores *, / e mod

Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Implementar as instruções MFH e MFL
    - MFH *reg*
      - *reg = high*
    - MFL *reg*
      - *reg = low*
  - Codificar as novas instruções de maneira a não criar conflito com as já existentes
  - Adicionar as novas instruções no arquivo r8.arq para que o montador as reconheça. No entando, elas não serão simuláveis no simulador
  - Atualizar o grafo da FSM após a adição das novas instruções e apresentar

Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Dica a fim de acelerar a simulação VHDL
    - Adicionar no arquivo Memory.vhd a *architecture* da memória utilizada na primeira parte do trabalho 1. Assim, não será necessário simular uma memória de 64 KB. Naquela *architecture* pode-se especificar o número de posições da memória independente da largura do barramento de endereço
    - Alterar o código da *architecture* a fim de tratar a interface da *entity* que é diferente da *entity* da primeira parte do trabalho 1
    - Utilizar também o R8_Simulator da primeira parte do trabalho 1 visto que ele monta o código mais rápido

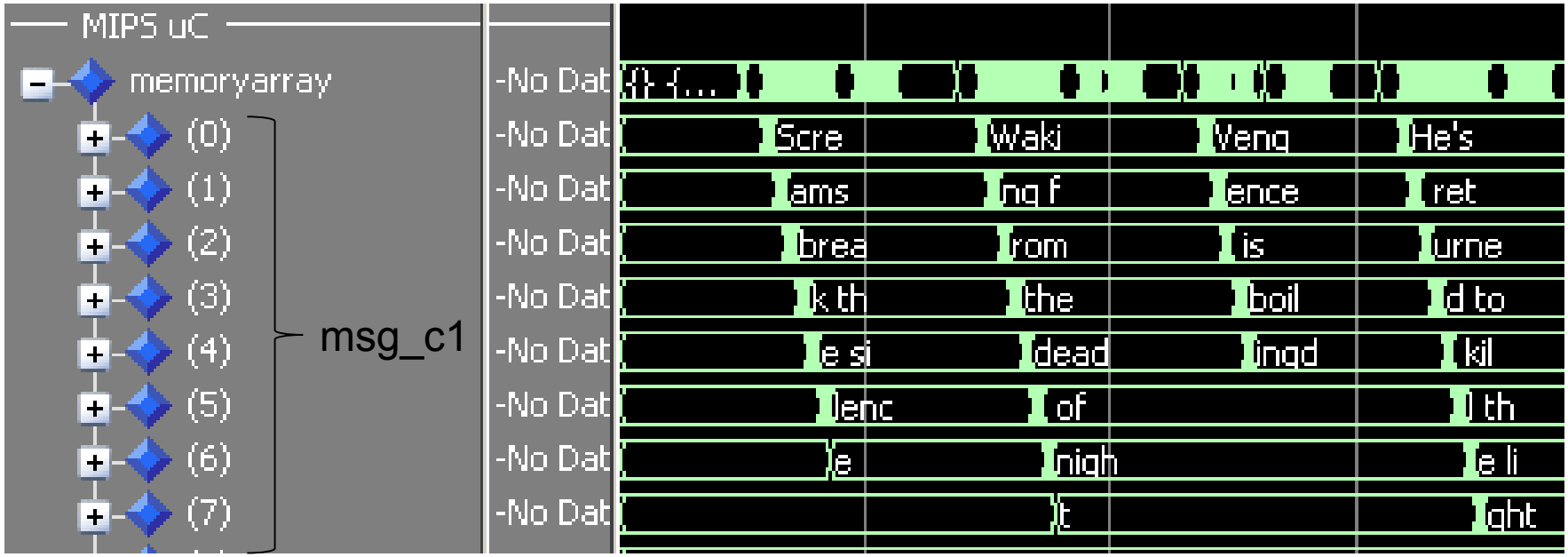
Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Aplicação
    - Criar duas variáveis para armazenar as mensagens decriptadas recebidas dos *CryptoMessages*
      - msg_c1: db #0, #0, … #0 // Colocar 80 zeros
      - msg_c2: db #0, #0, … #0 // Colocar 80 zeros
    - O programa deve ser um *loop* infinito que armazena as mensagens de cada *CryptoMessage* em uma das variáveis
    - As variáveis são sobrescritas a cada nova mensagem
    - Os arquivos contendo as as mensagens enviadas pelos *CryptoMessages* serão fornecidos junto com a descrição VHDL

Interface entre Processador e Periféricos

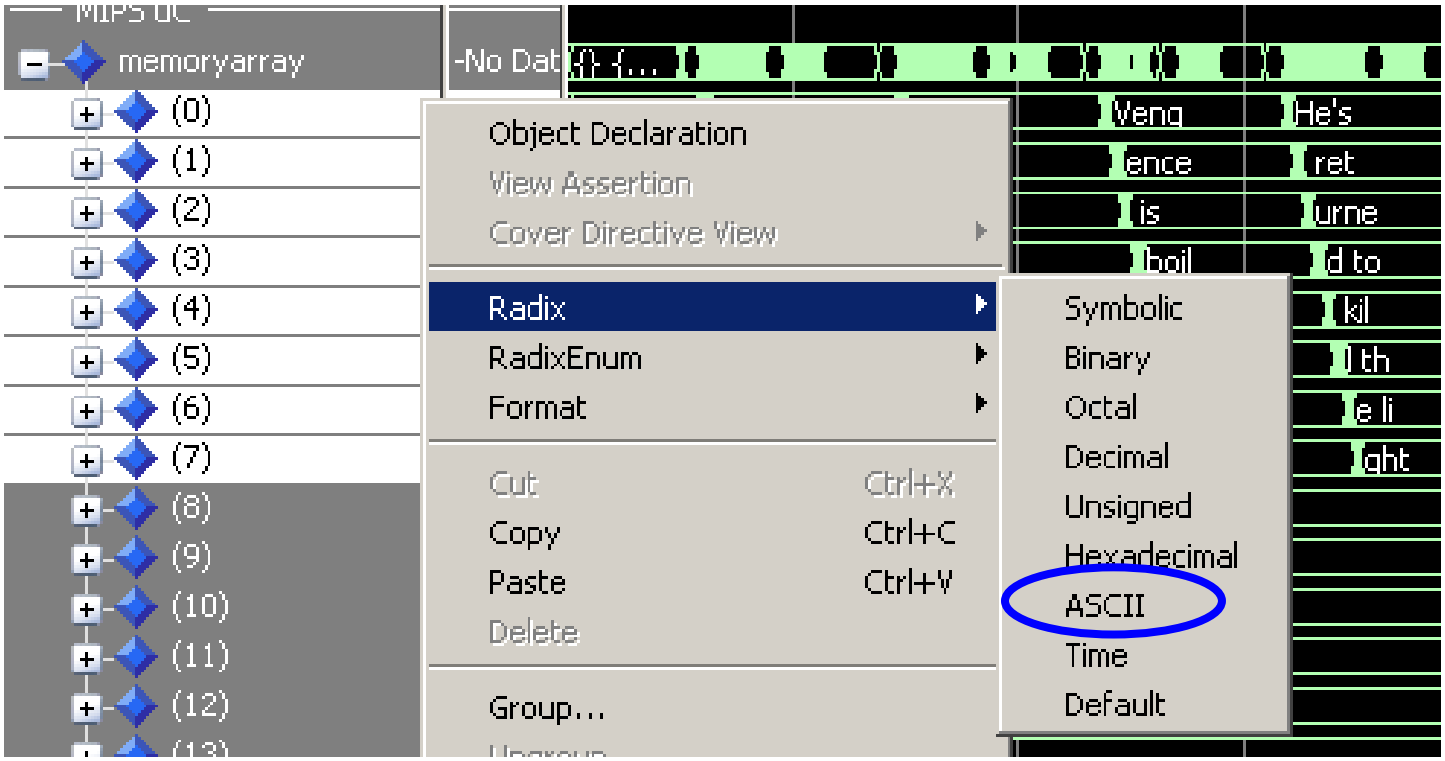
- Trabalho 2 – parte 2
  - Aplicação
    - IMPORTANTE: o armazenamento dos caracteres em memória deve ser feito de maneira que permita a leitura

Selecionar o formato ASCII na area de memória onde as mensagens são armazenadas
ModelSim: Radix→ASCII



Interface entre Processador e Periféricos

- Trabalho 2 – parte 2
  - Aplicação



Interface entre Processador e Periféricos

- Trabalho 2 – Parte 2
  - Manter mesmos grupos do trabalho 1
  - Apresentação dia 5/5
  - A nota do trabalho dará ENORME ÊNFASE à execução correta da simulação
  - A apresentação será oral, teórico-prática, frente ao computador, onde o grupo deverá explicar ao professor o projeto, a simulação e a implementação