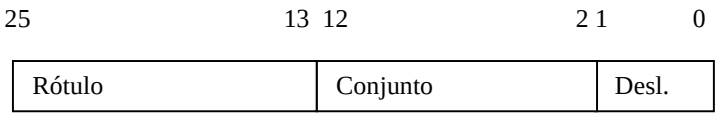
UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

TCC04.070-Organização de Computadores I – Turma :A1 –
Gabarito - Lista 2

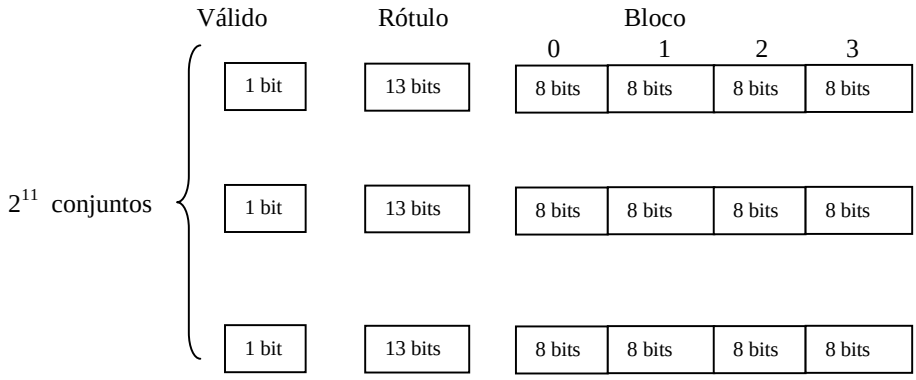
1.
   a) 256M-1=268435455
   b) $Log_2 256M = Log_2 2^{28} = 28$ bits
   c) O barramento de dados deve ser capaz de transmitir o conteúdo de um acesso à memória. Logo o barramento de dados deve ter capacidade mínima de 8 bits.
   d) Número de células × bits/célula = 256M X 8 = 2^{31}=2Gbits

2. Considerações gerais:
   Como a máquina pode endereçar 64 Mbytes e cada endereço acessa um byte, temos 64M células. Para endereçar 64M células, precisamos de 26 bits. Logo um endereço da memória principal possui 26 bits. A memória cache pode armazenar 2K blocos, logo ela possui 2K linhas. Como cada bloco que é transferido entre a memória principal e a cache possui 4 bytes, temos que os 2 bits menos significativos do endereço serão utilizados para identificar o byte que se quer dentro de um bloco, em qualquer dos mapeamentos.
   a) mapeamento direto
      Neste caso, temos uma linha por conjunto, logo teremos 2K conjuntos. Logo, em relação aos bits do endereço, precisamos de 2 bits para indicar o byte dentro do bloco, 11 bits para indicar o conjunto dentro da cache e 13 bits para indicar o rótulo (tag).

      Endereço

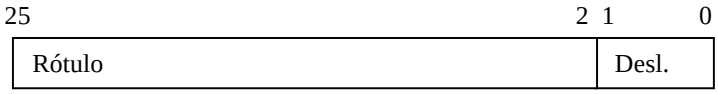


      Memória cache:

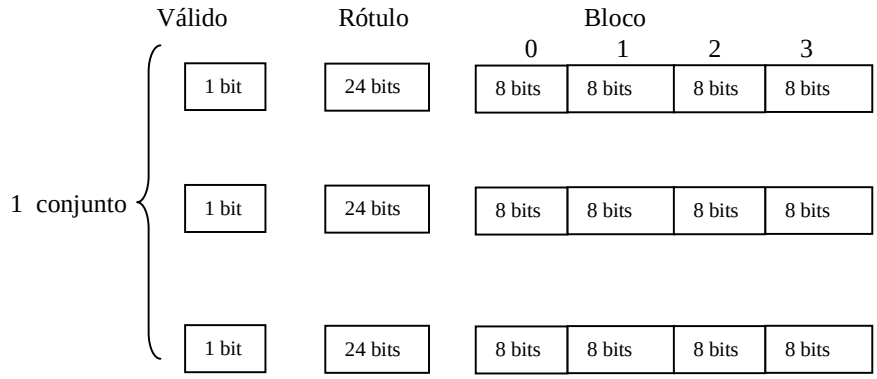


   b) mapeamento totalmente associativo
      Neste caso, temos um único conjunto. Logo, em relação aos bits do endereço, precisamos de 2 bits para indicar o byte dentro do bloco e 24 bits para indicar o rótulo (tag).

      Endereço

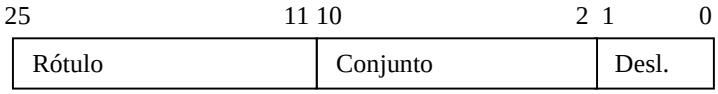


Memória cache:

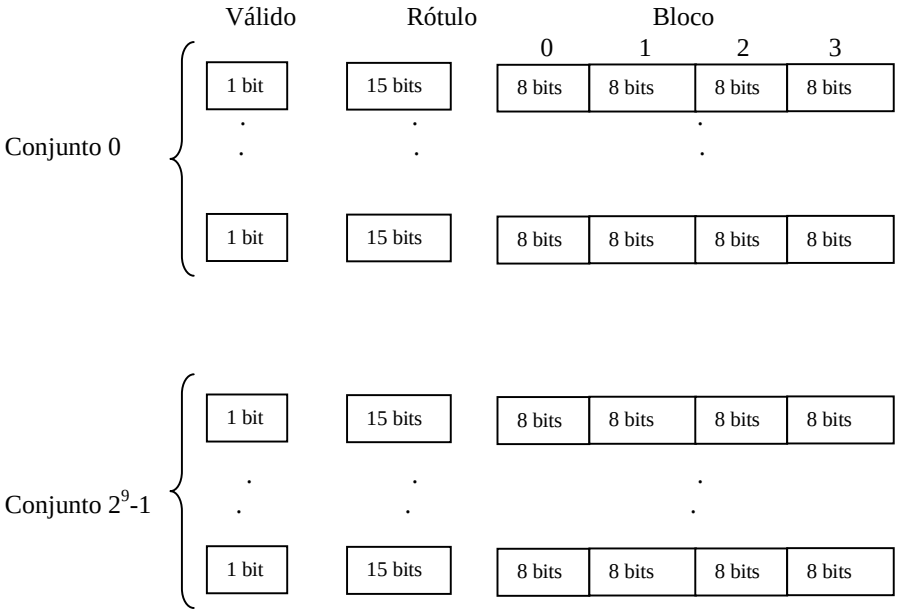


c) mapeamento associativo por conjunto com 4 linhas por conjunto.
Neste caso, temos 4 linhas por conjunto, logo teremos 2K/4= 512 conjuntos. Logo, em relação aos bits do endereço, precisamos de 2 bits para indicar o byte dentro do bloco, 9 bits para indicar o conjunto dentro da cache e 15 bits para indicar o rótulo (tag).

Endereço:



Memória cache:



3.

a) Tempo de acerto = 2ns
Penalidade por falta = 20 × 2ns = 40ns
Taxa de faltas = 0.05
Logo TMAM = 2 + 0.05 × 40ns = 4ns

b) Tempo de acerto = 1.2 × 2ns = 2.4 ns
Penalidade por falta = 20 × 2ns = 40ns
Taxa de faltas = 0.03
Logo TMAM = 2.4 + 0.03 × 40ns = 3.6ns
Então esta mudança irá melhorar o desempenho da máquina pois o TMAM diminuiu com estas mudanças.

4.

a) 32 bits pois é o tamanho do registrador de instrução
b) Como este sistema possui 16 registradores, necessita-se de 4 bits para identificar o registrador. Para endereçar 1M células, são necessários 20 bits. Logo o campo da instrução que identifica o registrador possui 4 bits, o que identifica o endereço de memória 20 bits e sobram 8 bits para o código de operação.
c) Como existem 8 bits para o código de operação, podem existir no máximo 256 operações diferentes.

5.

a)

```
        add 0 0 1          inicializa K com o valor 0
        addi 0 5 5         inicializa reg. 5 com valor 5 para controlar
                           execução do laço
LOOP    beq 1 5 EXIT       finaliza laço quando K igual a 5
        add 2 1 6          obtém endereço de A[K]
        lw 6 6 0           coloca conteúdo de A[K] no registrador 6
        add 3 1 7          obtém endereço de B[K]
        lw 7 7 0           coloca conteúdo de B[K] no registrador 7
        beq 6 7 IF         verifica se A[K] é igual a B[K]
        add 0 0 6          se A[K] é diferente de B[K], coloca 0 no
                           reg. que vai ter seu conteúdo carregado em C[K]
        beq 0 0 CARB       vai para instrução de carregamento de C[K]
IF      addi 0 6 1         se A[K] é igual a 0, coloca 1 no
                           reg. que vai ter seu conteúdo carregado em C[K]
CARB    add 4 1 7          obtém endereço de C[K]
        sw 7 6 0           coloca valor em C[K]
        addi 1 1 1         incrementa valor de K
        beq 0 0 LOOP       volta para o laço
EXIT    halt               fim do procedimento
```

6.

Em pseudo-linguagem:

```
declare A[0:3] numérico
declare TAMA numérico
declare i numérico

A[0]← -5
A[1]← -3
A[2]←  4
A[3]←  9
```

```
TAMA← 4
i ←0
Enquanto (i <TAMA) faça
   A[i]← 2×A[i]
   i ← i+1
```

Em C:

```
int main ()
{
   int A[4];
   int TAMA;
   int i;

   A[0] = -5;
   A[1] = -3;
   A[2] =  4;
   A[3] =  9;
   TAMA =  4;
   for (i=0; i<TAMA ; i++)
     A[i] = 2 × A[i];
   exit(0);
}
```

```
Registradores (conteúdo em hexadecimal):

   0-00000000
   1-00000010
   2-00000004
   3-00000010
   4-00000012
```

7.

a) Devemos transformar a representação para a base 2, para podermos identificar os campos de cada instrução:

```
End      Conteúdo
10       0000000|000|000|000|0000000000000|001
                  op  regA regB               destreg      add 0 0 1
11       0000000|001|000|010|0000000000000100
                  op  regA regB     desl.                  addi 0 2 4
12       0000000|100|001|010|0000000000000111
                  op  regA regB     desl.                  beq 1 2 7
13       0000000|000|011|001|0000000000000|100
                  op  regA regB               destreg      add 3 1 4
14       0000000|010|100|101|0000000000000000
                  op  regA regB     desl.                  lw 4 5 0
15       0000000|000|101|101|0000000000000|101
                  op  regA regB               destreg      add 5 5 5
16       0000000|000|101|101|0000000000000|101
                  op  regA regB               destreg      add 5 5 5
17       0000000|011|100|101|0000000000000000
                  op  regA regB    desl.                   sw 4 5 0
```

```
18        0000000|001|001|001|0000000000000001
                  op  regA regB     desl.                  addi 1 1 1
19        0000000|100|000|000|1111111111111000
                  op  regA regB      desl.                 beq 0 0 -8
20        0000000|110|000|000|0000000000000000
                  op  regA regB      desl.                 halt
```

b)

Registradores:

0-00000000
1-00000004
2-00000004
3-00000020
4-00000023
5-0000000C
6-AC012345
7-12345678

Memória :

End.	Conteúdo	End.	Conteúdo
10	00000001	1B	F0000000
11	00420004	1C	00000000
12	010A0007	1D	00010000
13	00190004	1E	10000000
14	00A50000	1F	7FFF0000
15	002D0005	20	00000008
16	002D0005	21	FFFFFFFC
17	00E50000	22	0000003C
18	00490001	23	0000000C
19	0100FFF8	24	00000010
1A	01800000	25	000000AF