Lista de exercícios

Bases numéricas e padrões de numeração

1. Calcule e mostre o valor decimal de cada um dos números binários de 8 bits da lista a seguir. Mostre os passos utilizados para o cálculo.

   a) 00101001 em complemento de 2
   b) 10001111 em complemento de 1
   c) 10111110 em sinal magnitude
   d) 00001101 em excesso de 127
   e) 10100000 em excesso de 127
   f) 00001111 em complemento de 1
   g) 11010100 em complemento de 2
   h) 00001001 em sinal magnitude
   i) 10101011 em sinal magnitude
   j) 10001010 sem sinal
   k) 10110001 em complemento de 2
   l) 10101001 em sinal magnitude
   m) 10011110 em excesso de 127
   n) 10110101 em complemento de 2
   o) 00101000 em complemento de 2
   p) 01101111 em excesso 127
   q) 10010101 sem sinal

2. Efetue as seguintes conversões de base:
   a) $109_{10} = ?_2$
   b) $FE_{16} = ?_{10}$
   c) $FE_{16} = ?_2$
   d) $76_8 = ?_5$
   e) $FFFF_{16} = ?_{10}$
   f) $1024_{10} = ?_2$
   g) $39A_{16} = ?_2$
   h) $1011010110_2 = ?_{16}$
   i) $354_6 = ?_{10}$
   j) $519_{10} = ?_2$
   k) $0101101101010011_2 = ?_{16}$
   l) $227_8 = ?_{10}$
   m) $343_{10} = ?_2$
   n) $110101001010110100_2 = ?_{16}$
   o) $A83_{16} = ?_{10}$
   p) $237_{10} = ?_2$

q) $10101101_2 = ?_{16}$

3. Converta os seguintes números para o formato IEEE 754 de precisão simples, mostrando os passos utilizados para a conversão e apresentando o resultado final em hexadecimal.
   a) -37.5
   b) 128.75
   c) 130.1332
   d) -312.6875
   e) -194.875
   f) 202.125
   g) 0.0625

4. Converta os seguintes números do formato IEEE 754 de precisão simples, apresentados na forma hexadecimal, para o seu valor decimal, mostrando os passos utilizados para a conversão.
   a) 40490E56
   b) 3F600000
   c) C1440A3D
   d) 45827A00
   e) 43312000
   f) B8300000
   g) 70100000

Álgebra booleana, Mapas de Karnaugh e Quine-McCluskey

5. Simplifique, se possível, as funções booleanas a seguir. Utilize álgebra booleana ou, alternativamente, mapas de Karnaugh, mostrando os passos utilizados para a simplificação:

   a) S = A + A´.B
   b) S = (A+B).(A+C)
   c) S = A´.B + A.B
   d) S = A´.B´.C´+ A´.B.C´+A´.B.C + A.B.C’
   e) S = A´.B´.C´+ A´.B.C´+A.B´.C
   f) S = A’.B’.C’ + A’.B.C + A’.B.C’ + A.B’.C’ + A.B.C’
   g) S = (A+B+C).(A´+B´+C)
   h) S = A.B.C + A.C´+ A.B´
   i) S = A´.B´+A´.B
   j) S = A´.B´.C´ + A´.B´.C + A´.B.C + A´.B.C´+ A.B.C´
   k) S = A.B´.C´ + A´.B´.C + A.B.C + A.B´.C + A´.B.C + A´.B´.C´

6. Dadas as funções abaixo, encontre a função simplificada usando o Mapa de Karnaugh, Método de Quine-McCluskey e álgebra booleana. Mostre os passos utilizados para a simplificação.

   a) f(A,B,C,D) = {0,1,11,13,14,15}
   b) f(A,B,C,D) = {0,1,2,4,8,10,11,12}
   c) f(A,B,C,D,E) = {1,2,3,5,7,11,13,17,19,23,29,31}
   d) f(A,B,C,D,E,F) = {6,9,15,18,24,27,33,36,42,45,51,54,60,63}
   e) f(A,B,C,D,E) = {0, 1, 2, 6, 9, 15, 16, 17, 23, 24, 29, 30, 31}
   f) f(A,B,C,D,E) = {2, 6, 8, 10, 13, 22, 23, 24, 31}

7. Utilizando Mapa de Karnaugh, obter a função simplificada para S = *f*(A, B, C, D) na qual S assume o valor 1 se o número binário formado pelos bits ABCD for primo e 0 caso contrário.

8. Utilizando Mapa de Karnaugh, obter as funções booleanas simplificadas para os 7 bits (A a G) utilizados para acender um display de 7 segmentos e representar o dígito hexadecimal correspondente ao valor binário de 4 bits formado pelas entradas I0, I1, I2 e I3.

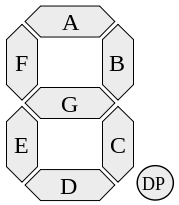

Figura 1 – Organização dos segmentos em um display de 7 segmentos

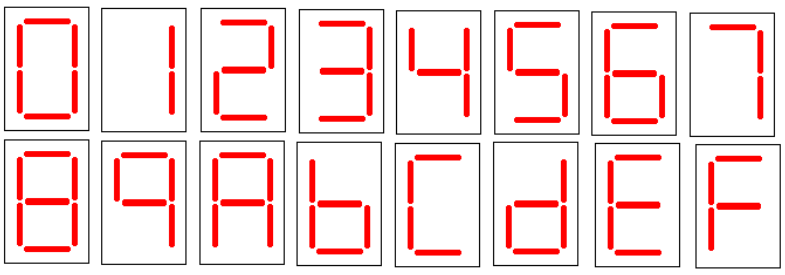

Figura 2 – Acendimento do display de 7 segmentos para cada um dos dígitos hexadecimais

Fontes: http://pt.wikipedia.org/wiki/Display_de_sete_segmentos

http://www.ebah.com.br/content/ABAAABU5gAF/microcontrolador

Circuitos sequenciais, aritméticos e ULA

9. Desenhe um circuito subtrator de 2 bits sem utilizar somadores e complemento de dois. Considere a tabela da verdade da subtração (minuendo, subtraendo e "empresta-um").

10. Desenhe um registrador (memória) de 8 bits utilizando flip-flops D e com os seguintes sinais:
    a) Entradas: 8 bits a serem armazenados, sinal de clock (borda de subida), sinal de CLR (zera os bits da saída quando em nível 0) e sinal WR (memoriza as entradas na próxima borda de subida se for 1, mantém os valores anteriores se for 0).
    b) Saídas: 8 bits atualmente armazenados.

11. Desenhe um circuito sequencial/combinacional com as seguintes características:
    a) Dois registradores (conjuntos de flip-flops) de 4 bits, denominados A e B.
    b) Um conjunto de 4 sinais de entrada, denominado I.
    c) Um sinal OP1 que habilita a execução da operação A <- I na próxima borda positiva de clock.
    d) Um sinal OP2 que habilita a execução da operação B <- I na próxima borda positiva de clock.
    e) Um sinal OP3 que habilita a execução da operação A <- (A OR B) na próxima borda positiva de clock.

12. Desenhe uma ULA de 2 bits que execute todas as seguintes operações de acordo com os sinais de controle C0 e C1:

C0	C1	Saída
0	0	A AND B
0	1	A OR B
1	0	NOT B
1	1	A + B

13. Desenhe um circuito contador de 4 bits, que incrementa o valor de contagem a cada borda de subida do clock, utilizando as seguintes abordagens:
    a) somente flip-flops (dica: utilizar a saída Q de um flip-flop D como clock para o próximo e a saída Q´ para inverter a entrada D a cada pulso de clock).
    b) utilizando uma ULA de 4 bits.

14. Desenhe um circuito sequencial / combinacional com as seguintes características:
    a) 1 registrador de 2 bits, denominado RA.
    b) 2 bits de entrada de dados (I_0 e I_1), 2 bits de entrada de controle (C_0 e C_1) e uma entrada de *clock.*
    c) O circuito efetua as seguintes operações na borda de subida do *clock*, dependendo dos valores de C_0 e C_1:

C_0	C_1	Operação
0	0	RA permanece inalterado
0	1	$RA_1 \leftarrow I_1$, bit 0 permanece inalterado
1	0	$RA_0 \leftarrow I_0$, bit 1 permanece inalterado
1	1	RA ← I

15. Desenhe um circuito sequencial / combinacional com as seguintes características:
    a) 2 registradores de 1 bit cada, denominados RA e RB.
    b) 1 bit de entrada de dados (I), 2 bits de entrada de controle (C_0 e C_1) e uma entrada de *clock.*
    c) O circuito efetua as seguintes operações na borda de subida do *clock*, dependendo dos valores de C_0 e C_1:

C_0	C_1	Operação
0	0	RA e RB permanecem inalterados
0	1	RB ← I
1	0	RA ← I
1	1	RA e RB permanecem inalterados

16. Desenhe um circuito digital com as seguintes características:
    a) Três registradores de 2 bits cada, denominados R0, R1 e R2.
    b) Cinco bits de entrada: I (2 bits), C0, C1 e *clock*.
    c) O circuito efetua as seguintes operações na borda de subida do *clock*, dependendo dos valores de C0 e C1:

C0	C1	Operação
0	0	R0 ← I
0	1	R1 ← I
1	0	R2 ← R1 subtraído de R0
1	1	Nenhuma operação

17. Desenhe um circuito sequencial que efetue a operação C <- A * B, onde A e B são registradores de 2 bits que armazenam números inteiros sem sinal e C é um registrador de 4 bits. Este circuito deve possuir um sinal EN para iniciar o cálculo e um sinal RDY indicando que o cálculo foi finalizado.
    Dica: utilizar uma ULA de 4 bits e o contador da questão 13 simplificado para 2 bits.

18. Repetir o exercício anterior, porém utilizando a mesma estratégia aprendida no ensino básico para multiplicação de números decimais: multiplicar o dígito mais à direita do multiplicador por todos os dígitos do multiplicando, depois armazenar o resultado e prosseguir com o próximo dígito, porém deslocando o resultado à esquerda, e ao final somando todos os resultados intermediários. Fazer para um multiplicando de 3 bits e um multiplicador de 2 bits.

19. Desenhe um circuito digital com as seguintes características:
    a) Três registradores de 2 bits cada, denominados R0, R1 e R2.
    b) Cinco bits de entrada: I (2 bits), C0, C1 e *clock*.
    c) O circuito efetua as seguintes operações na borda de subida do *clock*, dependendo dos valores de C0 e C1:

C0	C1	Operação
0	0	R0 ← I
0	1	R1 ← I
1	0	R2 ← R0 somado com R1
1	1	R2 ← R0 somado com R1 somado com 1

Observações:

- Os únicos componentes que podem ser utilizados no desenho são portas lógicas e flip-flops D. Caso seja necessário representar circuitos mais complexos (p.ex. uma ULA), estes devem ser desenhados com os componentes citados.
- Desprezar o "vai-um" do bit mais significativo do resultado das operações que envolvem soma.
- Não é necessário representar os bits de set e reset dos flip-flops.

20. Desenhe um circuito sequencial / combinacional com as seguintes características:
    a) (1) 3 registradores de 1 bits, denominados RA, RB e RC.
    b) (2) 1 bit de entrada de dados (I_0), 2 bits de entrada de controle (C_0 e C_1), 1 bit de entrada de habilitação (EN) e uma entrada de *clock.*
    c) (5) O circuito efetua as seguintes operações na borda de subida do *clock*, dependendo dos valores de C_0, C_1 e EN:

EN	C_0	C_1	Operação
1	0	0	RA ← I
1	0	1	RB ← I
1	1	0	RA é trocado com RB
1	1	1	RC ← 1 se RA igual a RB, 0 caso contrário
0	x	x	RA e RB permanecem inalterados.

x: não importa (*don´t care*)

Observações:

- Os únicos componentes que podem ser utilizados no desenho são portas lógicas e flip-flops D. Caso seja necessário representar circuitos mais complexos, estes devem ser desenhados com os componentes citados.

Modelos de execução e ISA

21. Pesquise uma implementação específica de máquina de fluxo de dados. Para esta implementação, descreva quais são as operações básicas definidas na ISA e escreva um pequeno programa de exemplo utilizando estas operações.

22. Para a ISA RISC ARM7TDMI, descreva 2 formas diferentes de executar a operação A <- B + C sem utilizar instruções específicas de adição da ISA (ADD).

23. Mostre os códigos binários correspondentes aos seguintes mnemônicos:
    a) mov %eax, 0x28(%esp) na ISA IA-32
    b) add r0, r1, r0 na ISA ARM7TDMI
    c) str r3,[r4] na ISA ARM7TDMI
    d) mov r1, #0x35 na ISA ARM7TDMI
    e) mov $0x1b, %eax na ISA IA-32
    f) ldr r1, =0x00103456 na ISA ARM7TDMI
    g) mov r1, r3 na ISA ARM7TDMI
    h) mov %esp, %ebp na ISA IA-32
    i) ldr r0, [r1] na ISA ARM7TDMI
    j) mov %eax, (%esp) na ISA IA-32
    k) ldr r0, [r1,#+32] na ISA ARM7TDMI
    l) mov %eax, 0x24(%esp) na ISA IA-32
    m) str r1, [r2, r5] na ISA ARM7TDMI
    n) stmia r0, {r0-r15} na ISA ARM7TDMI

24. Descrever a faixa de endereços de pelo menos um periférico mapeado em memória no manual do microcontrolador LPC214xx.

25. Quantos e quais são os registradores do 80386? Este conjunto de registradores é típico de uma ISA CISC ou RISC? Por quê?

26. Identificar pelo menos uma instrução na ISA x86 que efetue uma operação sobre dados do tipo *memory – memory.*

Fundamentos de *assembly*

27. Considere uma arquitetura ARQ qualquer, com as seguintes características:
    - oito registradores de 32 bits, denominados R0 a R7.

- conjunto de instruções:

Mnemônico assembly	Descrição
LD *reg1, [reg2], n*	Lê dado de *n bytes* do endereço de memória *reg2* e armazena em *reg1*
ST *reg1, [reg2], n*	Escreve *n* bytes do dado armazenado em *reg1* no endereço de memória *reg2*
ADD *reg1, reg2*	Soma *reg1* e *reg2* e armazena o resultado em *reg1*
SUB *reg1, reg2*	Subtrai *reg2* de *reg1* e armazena o resultado em *reg1*
MOV *reg1, reg2*	Copia conteúdo de *reg2* para *reg1*
SR *reg1, n*	Desloca o conteúdo de *reg1* em *n* bits para a direita (*n* constante), ignorando os bits que não cabem mais no registrador após o deslocamento.
SL *reg1, n*	Desloca o conteúdo de *reg1* em *n* bits para a esquerda (*n* constante), ignorando os bits que não cabem mais no registrador após o deslocamento.
AND *reg1, reg2*	Efetua a operação lógica AND bit a bit entre *reg1* e *reg2* e armazena o resultado em reg1.
OR *reg1, reg2*	Efetua a operação lógica OR bit a bit entre *reg1* e *reg2* e armazena o resultado em reg1.
MOV *reg1, n*	Copia o valor constante *n* para *reg1*
BZ *pos1*	Desvia a execução para o endereço apontado por *pos1* se o resultado da última operação lógica ou aritmética foi zero, do contrário continua a execução na próxima instrução
BMZ *pos1*	Desvia a execução para o endereço apontado por *pos1* se o resultado da última operação aritmética foi maior do que zero ou se o resultado da última operação lógica foi diferente de zero, do contrário continua a execução na próxima instrução

Com base nesta especificação, escreva os seguintes códigos em *assembly:*

a) Código que compara duas *strings,* na forma de vetores de caracteres de 1 byte cada, cujos endereços iniciais estão armazenados em R0 e R1, e armazena em R0 o valor 0 se elas forem idênticas e o valor 1 se não forem idênticas. Considerar que o limite das duas *strings* é dado pelo byte terminador de valor 0.
b) Código que calcula o resto da divisão inteira dos valores de 32 bits armazenados em R0 e R1 e armazena o resultado em R0.
c) Código que calcula a média aritmética inteira do conjunto de valores de 32 bits armazenados a partir do endereço indicado por R0. A quantidade de valores está armazenada em R1, e o resultado calculado da média aritmética deve ser armazenado em R2.
d) Código que conta a quantidade de números pares no conjunto de valores de 32 bits armazenados a partir do endereço indicado por R0. A quantidade total de

valores está armazenada em R1, e o resultado calculado da quantidade de números pares deve ser armazenado em R2.

e) Código que conta a quantidade de alunos aprovados em uma determinada disciplina (media maior ou igual a 60 e frequência maior ou igual a 75). As médias estão armazenadas em valores de 8 bits em memória a partir do endereço indicado por R0, e as frequências estão armazenadas em valores de 8 bits em memória a partir do endereço indicado por R1. A quantidade total de alunos está armazenada em R2, e o resultado calculado da quantidade de alunos aprovados deve ser armazenada em R3.

Observação: o código em *assembly* deve conter somente as instruções listadas na tabela e, eventualmente, rótulos para indicar os endereços de destino de BZ e BMZ. Os rótulos devem ser escritos no formato *rotulo:*

28. Ainda em relação à arquitetura ARQ da questão anterior, considere o seguinte programa em assembly:

```
start:    LD R4, [R0], 4
          MOV R6, 0
          MOV R7, 1
          MOV R2, 0
calc:     MOV R5, R4
AND R5, R7
          SUB R5, R6
          BZ continua
          ADD R2, R7
continua:
          SR R4, 1
          SUB R4, R6
          BMZ calc
grava: ST R2, [R1], 4
```

a) Para o seguinte estado inicial da memória e dos registradores, mostre o estado correspondente após o final da execução do programa:

Registradores:

Reg	R0	R1	R2	R3	R4	R5	R6	R7
Valor (decimal)	1000	1028	1008	1012	0	0	125	12

Memória:

End (decimal)	1000	1004	1008	1012	1016	1020	1024	1028
Valor (decimal)	25	2	7	28	14	72	0	15

29. Ainda em relação à arquitetura ARQ da questão anterior, considere o seguinte programa em assembly:

```
start:    LD R4, [R0], 4
          LD R5, [R1], 4
          MOV R6, 1
          MOV R7, 4
          MOV R1, 1
          MOV R2, 1
          SUB R5, R6
          SUB R5, R6
          ST R1, [R4], 4
          ADD R4, R7
          ST R2, [R4], 4
          ADD R4, R7
   calc:
          MOV R3, R2
          ADD R2, R1
          MOV R1, R3
          ST R2, [R4], 4
          ADD R4, R7
          SUB R5, R6
          BMZ calc
fim:
```

Para o seguinte estado inicial da memória e dos registradores, mostre o estado correspondente após o final da execução do programa:

Registradores:

Reg	R0	R1	R2	R3	R4	R5	R6	R7
Valor (decimal)	1000	1004	1008	1012	0	0	125	12

Memória:

End (decimal)	1000	1004	1008	1012	1016	1020	1024	1028
Valor (decimal)	1008	5	7	28	14	72	0	15

Implemente os seguintes programas em assembly ARM7TDMI e teste no ARMSim#, verificando os resultados.

30. Implemente um programa para multiplicar dois números inteiros contidos em R0 e R1 e armazenar o resultado em R0.

31. Implemente um programa que calcula o resto da divisão inteira dos valores de 32 bits armazenados em R0 e R1 e armazena o resultado em R0.

32. Implemente um programa para verificar se o número contido em R0 é primo. Em caso afirmativo, armazenar o valor 1 em R1, do contrário armazenar o valor 0 em R1.

33. Implemente um programa para contar o número de ocorrências, do caracter contido no primeiro byte de R0, na string cujo endereço de início é indicado por R1 e cujo tamanho em caracteres está contido em R2. Armazenar o número de ocorrências obtido em R3.

34. Implemente um programa para calcular e armazenar em R1 a quantidade de bits 1 que existem no número armazenado em R0.

35. Implementar um código para gerar o bit de paridade par para o número armazenado em R0. Ou seja, o programa deve contar o número de bits 1 contidos em R0 e armazenar em R1 o valor 1 se este número for par e o valor 0 se for ímpar.

36. Implemente um programa para ler os botões do *plugin* EmbestBoard (“Left” e “Right”) e mostrar no display se o botão da esquerda foi pressionado ou se o botão da direita foi pressionado.

37. Implemente um programa para armazenar os 30 primeiros termos da série de Fibonacci, cada um como um número de 32 bits, em uma área de memória adequada (Dica: reservar a área por meio da diretiva .space).

38. Implemente um programa para gerar uma sequência de 10 números inteiros em ordem crescente e, em seguida, os mesmos 10 números inteiros em ordem decrescente. Os números devem ser armazenados no endereço indicado por R0. O registrador R1 indica a posição de memória onde estão armazenados dois números de 32 bits cada: o valor inicial da sequência e o intervalo entre dois números adjacentes.
    Por exemplo, se o endereço indicado por R1 contiver: 12, 3.
    A sequência será: 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 39, 36, 33, 30, 27, 24, 21, 18, 15, 12.

39. Implemente um programa para encontrar os fatores primos do número armazenado em R0 e armazená-los em uma área de memória reservada onde cabem 32 valores de 32 bits cada (caso sejam encontrados mais do que 32 fatores primos, os excedentes devem ser ignorados). Ao final do programa, o registrador R1 deve receber a quantidade de fatores primos encontrados.

40. Implemente um programa para receber uma senha de 4 dígitos hexadecimais (0 a F), via teclado do *plugin* EmbestBoard, e verificar a compatibilidade com uma senha armazenada internamente. A senha deve ser fornecida de acordo com o seguinte procedimento:
    a) À medida que a senha é digitada o display LCD apresenta um asterisco para cada dígito.
    b) Ao se pressionar o botão esquerdo (“Left”), o recebimento da senha é reiniciado e o display é apagado.
    c) Ao se pressionar o botão direito (“Right”) o último valor digitado é apagado. Ou seja, este botão atua como uma forma de corrigir um valor digitado erroneamente. O asterisco correspondente deve ser apagado do display.
    d) Se a senha estiver ok o LED da esquerda deve ser aceso, caso contrário o LED da direita deve ser aceso.

41. Implemente um programa para o jogo de “Clique rápido” utilizando a interface do *plugin* EmbestBoard. Este jogo tem as seguintes características:
    a) Inicialmente o display de 8 segmentos está apagado.
    b) Após um número pré-determinado de ms, o display de 8 segmentos deve apresentar um dígito aleatório entre 0 e F (dica: executar a SWI 0x6d para obter o valor do “tick” do relógio interno do simulador e utilizar os 4 bits menos significativos como valor aleatório).
    c) Após exibir o dígito aleatório, o display de LCD deve mostrar continuamente uma contagem de tempo, atualizada a cada 10 ms, até que o usuário clique algum botão do teclado.

d) Caso o usuário clique o botão correspondente ao dígito aleatório exibido, a contagem de tempo deve ser interrompida. Esta contagem corresponde ao tempo de resposta do usuário após o dígito aleatório ter sido informado.
e) Caso o usuário clique um botão errado, o display de LCD deve mostrar "---" indicando que o usuário perdeu.

42. Implemente um programa que simule uma calculadora de números inteiros que opera no modo RPN (notação polonesa inversa, ou posfixa), utilizando a interface do *plugin* EmbestBoard. Este programa tem as seguintes características:

a) O teclado é utilizado para entrada de valores e de operações, conforme o arranjo a seguir:

1	2	3	+
4	5	6	-
7	8	9	*
	0	ENTER	/

b) Ao terminar de digitar um valor numérico, o usuário deve digitar ENTER. Quando isso ocorrer, o programa deve armazenar o valor em uma pilha de operandos.
c) Caso a pilha de operandos já esteja cheia (máx. 15), o programa não deve aceitar a entrada de valores numéricos.
d) Quando o usuário digitar uma das quatro operações básicas (+, -, *, /), o programa deve efetuar o cálculo com os dois valores no topo da pilha e empilhar o resultado.
e) Caso o usuário tente efetuar uma divisão por zero, o LED vermelho da esquerda deve acender e a operação não deve ser efetuada.
f) Caso o usuário clique sobre o botão da esquerda ("Left"), a pilha atual de operandos deve ser reiniciada (vazia).
g) O display LCD deve mostrar o estado atual da pilha, um valor por linha.

43. Implemente um programa que simule um relógio com alarme, utilizando a interface do *plugin* EmbestBoard. O programa tem as seguintes características:

a) O relógio pode funcionar em 2 modos: alarme (caracter 'A' exibido no display de 8 segmentos) e relógio (caracter 'C' exibido no display de 8 segmentos).
b) Quando no modo relógio, o programa deve mostrar no display de LCD a hora e data atuais no formato DD-MM-AAAA HH:MM:SS e atualizar este valor a cada 1 s (usar SWI 0x6d para marcação do tempo).

c) Quando no modo alarme, o programa deve mostrar no display de LCD o horário configurado para o alarme no formato HH:MM.
d) A mudança entre os modos ocorre por meio de uma das teclas do teclado (pode ser a tecla do canto inferior direito, por exemplo).
e) Em ambos os modos, um dígito qualquer pode ser ajustado teclando-se um novo valor no teclado (0 a 9). O dígito específico a ser ajustado é marcado por meio de um cursor (caracter – ou _), exibido na linha do display LCD imediatamente abaixo do valor sendo exibido. Para navegar o cursor para a esquerda ou para a direita o usuário deve utilizar os botões “Left” ou “Right” respectivamente.

Por exemplo, caso o display LCD exiba o dado a seguir, qualquer dígito teclado ajustará o valor da dezena do minuto (marcado pelo cursor logo abaixo):

```
23-05-2014 15:10:45
                 -
```

f) Quando o horário ajustado para o alarme e o horário atual coincidirem, os LEDs vermelhos deverão acender durante 15 segundos.

44. Implemente um programa em assembly ARM7TDMI que simule um controle de acesso baseado em senha utilizando a interface do *plugin* EmbestBoard no ARMSim#. Este controle de acesso possui três modos: “Memorização”, no qual o usuário configura uma nova senha; “Acesso”, no qual o sistema aguarda a senha previamente configurada para autorizar o acesso ou não; e “Bloqueado aguardando serviço”, que é atingido quando o usuário fornece uma senha incorreta em 3 tentativas consecutivas.
    a) Inicialmente, o sistema entra no modo “Memorização”. Durante este modo, o botão da esquerda (“Left”) é utilizado para iniciar o processo de entrada da senha. Ao ser pressionado, é exibida a mensagem “Digite a nova senha para memorização” no display LCD.
    b) O teclado é utilizado para entrada da senha de 8 dígitos. A cada novo dígito fornecido o display de LCD deve imprimir um novo asterisco (*). Caso o usuário queira corrigir um dígito fornecido erroneamente, deve pressionar o botão da direita (“Right”), fazendo com que o asterisco correspondente àquele dígito também seja apagado do display LCD.
    c) Após digitar a senha, o usuário deve pressionar novamente o botão da esquerda (“Left”). Caso o sistema esteja no modo “Memorização”, este deve memorizar internamente a senha fornecida, o display LCD deve mostrar a mensagem “Senha memorizada” durante 5 segundos, o display de 8 segmentos deve mostrar o dígito 0 e o LED vermelho da esquerda deve acender, indicando que o sistema entrou no modo “Acesso”.
    d) Durante o modo “Acesso”, o teclado é utilizado para a entrada da senha seguindo a mesma lógica apresentada no item b).
    e) Caso o sistema esteja no modo “Acesso” e o botão da esquerda (“Left”) seja pressionado, a senha que foi fornecida deve ser comparada com a senha memorizada. Caso sejam iguais, o display LCD deve mostrar a mensagem

"Acesso autorizado", o LED vermelho da esquerda deve ser apagado e o sistema deve retornar ao modo "Memorização".

f) Caso as senhas não sejam iguais, o display de LCD deve mostrar a mensagem "Senha incorreta". O dígito mostrado no display de 8 segmentos deve ser incrementado, indicando que uma nova tentativa de senha será realizada.

Quando o dígito mostrado no display de 8 segmentos chegar ao valor 3, o display de LCD deve mostrar a mensagem "Número de tentativas máximo atingido!" e os dois LEDs vermelhos devem acender, indicando que o sistema entrou no modo "Bloqueado aguardando serviço".

45. Implemente um programa em *assembly* ARM7TDMI que simule um jogo no qual o jogador tenta adivinhar um número de 0 a 1023 sorteado pelo sistema, utilizando a interface do *plugin* EmbestBoard no ARMSim#.
    a) Inicialmente o sistema exibe o dígito 0 no display de 8 segmentos e a mensagem "Forneça um palpite (0 a 1023)" no display de LCD. Além disso, o sistema sorteia um número aleatório entre 0 e 1023 e o armazena internamente (dica: executar a SWI 0x6d para obter o valor do "tick" do relógio interno do simulador e utilizar os 10 bits menos significativos como valor aleatório).
    b) O usuário deve fornecer um palpite por meio do teclado, cujo arranjo é mostrado abaixo. Cada novo dígito fornecido do palpite é mostrado no display de LCD.

1	2	3	
4	5	6	
7	8	9	
	0		

    c) Quando o botão da esquerda ("Left") for pressionado, o sistema deve incrementar o dígito no display de 8 segmentos, indicando o número de tentativas efetuadas, e deve comparar o palpite do usuário com o número aleatório sorteado. Caso sejam iguais, o display de LCD deve mostrar a mensagem "Parabéns! Você acertou!" e o sistema deve aguardar que o usuário pressione o botão da direita ("Right") para zerar o dígito no display de 8 segmentos e reiniciar o jogo. Além disso, o sistema deve mostrar no display de LCD o tempo decorrido desde o início do jogo (dica: executar a SWI 0x6d para obter o valor do "tick" do relógio interno do simulador, tanto no início do jogo quanto no final, e calcular a diferença de tempo).
    d) Caso o palpite não seja igual ao valor sorteado e o número de tentativas mostrado no display de 8 segmentos seja menor do que 9, o display de LCD deve mostrar a mensagem "Palpite incorreto e muito grande", caso o valor do palpite seja maior do que o valor sorteado, ou a mensagem "Palpite incorreto e muito

pequeno", caso o valor do palpite seja menor do que o valor sorteado. Em ambos os casos, a execução retorna para o item b) e um novo palpite é aguardado,

Caso o palpite não seja igual ao valor sorteado e o display de 8 segmentos esteja mostrando o valor 9, o display de LCD deve exibir a mensagem "Número de tentativas máximo atingido!" e o sistema deve aguardar que o usuário pressione o botão da direita ("Right") para zerar o dígito no display de 8 segmentos e reiniciar o jogo.

46. Implemente um programa em *assembly* ARM7TDMI que simule um editor gráfico, utilizando a interface do *plugin* EmbestBoard no ARMSim#.
    a) O programa define um cursor na forma de um traço (-) , inicialmente posicionado no canto superior esquerdo do display de LCD. A área útil do display de LCD é utilizada como área de desenho, utilizando um conjunto determinado de caracteres.
    b) O usuário pode utilizar o teclado para movimentar o cursor, conforme as teclas de seta do arranjo mostrado abaixo. As demais teclas servem para definir qual caractere será impresso no display na posição do cursor, à medida que ele é movimentado (o caractere "Espaço" imprime um espaço em branco, ou seja, apaga a posição onde se encontra o cursor).

	↑		Espaço
←		→	#
	↓		*
		O	-

    c) O botão da esquerda ("Left") é utilizado para limpar completamente a área de desenho, preenchendo-a com espaços. O botão da direita ("Right") é utilizado para inverter o desenho, de tal maneira que os caracteres diferentes de espaço sejam preenchidos com um espaço e os caracteres de espaço sejam preenchidos com '#'.
    d) O LED vermelho da esquerda deve ser aceso sempre que um caractere diferente de espaço estiver selecionado para ser preenchido na posição do cursor. O LED vermelho da direita, por sua vez, deve ser acesso sempre que o caractere de espaço estiver selecionado.

    O display de 8 segmentos deve mostrar o percentual da área de desenho preenchida com caracteres diferentes de espaço, em múltiplos de 10%. Por exemplo, se 62% das posições da área de desenho estiverem preenchidas, o display de 8 segmentos deve

mostrar o dígito 6. Se 100% estiverem preenchidos o display de 8 segmentos deve mostrar a letra ‘A’.

Pipelines

47. Descreva pelo menos mais 2 exemplos (fora os exemplos de lavanderia e linha de montagem de veículos, apresentados em sala de aula) que se beneficiam ou se beneficiariam do uso de um *pipeline.* Os exemplos não precisam ser correlatos à área de computação, porém deve ser detalhado como o *pipeline* funcionaria nestes casos.

48. Pesquisar alguns *pipelines* (pelo menos 2) de arquiteturas comerciais. Tentar responder as seguintes perguntas:
    a) Quantos e quais são os estágios?
    b) Outros detalhes técnicos
    c) Sugestões de arquiteturas: ARM (ARM7TDMI, Cortex-M, etc.), x86, IA-32, Intel 64, PowerPC.

49. Considere o seguinte programa em assembly ARM7TDMI (obtém tamanho de string):

```
.text
      b start
str:  .asciz "ABC"
       .equ   nul, 0
       .align
start: ldr   r0, =str          @ r0 = &str
        mov   r1, #0
loop:  ldrb  r2, [r0], #1      @ r2 = *(r0++)
        add   r1, r1, #1        @ r1 += 1
        cmp   r2, #nul          @ if (r1 != nul)
        bne   loop              @    goto loop
        sub   r1, r1, #1        @ r1 -= 1
stop:
```

Este programa será executado em uma arquitetura fictícia ARQ, com ISA compatível com a ISA ARM7TDMI. O *pipeline* desta arquitetura tem as seguintes características:

- 5 estágios: IF, ID, EX, MEM e WB.
- Qualquer operação de acesso à memória (estágios IF e MEM) utiliza 2 ciclos de *clock.* Todos os demais estágios executam em 1 ciclo de *clock*
- Estágios IF e MEM podem operar simultaneamente (acessam espaços de memória diferentes).
- Todas as atualizações de registradores ocorrem no final do estágio WB.

- Implementa adiantamento de registradores a partir do final do estágio EX para o início do estágio ID da próxima instrução. Instruções que leem registradores da memória são capazes de adiantá-los ao final do estágio MEM.
- Endereços e/ou valores de condição de *branches* condicionais só estão disponíveis ao final do estágio EX.
- Instruções que não fazem acesso à memória ocupam o estágio MEM durante 1 ciclo de *clock.*
- O endereço de destino de um *branch* está disponível somente ao final do estágio EX deste *branch.*

Com base nesta especificação, desenhe o cenário de execução das instruções no *pipeline*, ciclo por ciclo, para o programa completo.

- Usar como modelo as tabelas apresentadas nos exemplos de conflitos da aula teórica - ciclo de *clock* na dimensão horizontal (ou vertical, se preferir), instruções na dimensão vertical (ou horizontal, se preferir) e o estágio ocupado na célula correspondente da tabela.

50. O seguinte programa em *assembly* é executado na arquitetura ARQ apresentada no exercício 27 da lista*:*

```
start:
            MOV R2, 0
            MOV R3, 1
calc:
            ADD R2, R0
            SUB R1, R3
            BMZ calc
            MOV R0, R2
```

Considerando que o *pipeline* de ARQ tem as seguintes características:

- 5 estágios: IF (busca de instrução), ID (decodificação de instrução), EX (execução de instrução), MEM (leitura/escrita de memória) e WB (atualização de registradores).
- Na ausência de conflito, todos os estágios operam em 1 ciclo de *clock.*
- Em instruções que não acessam a memória para escrita ou leitura de dados, a execução passa diretamente do estágio EX para o estágio WB.
- Estágios IF e MEM possuem conflito estrutural.
- Todas as atualizações de registradores ocorrem no final do estágio WB.
- Implementa *register forwarding* a partir do final do estágio EX e também do final do estágio MEM (caso o registrador tenha sido lido da memória) para o início do estágio ID da próxima instrução.
- A condição de um *branch* condicional só pode ser avaliada, durante o estágio EX da instrução de *branch* condicional, após o final do estágio EX da última operação aritmética executada (ADD ou SUB).

- Endereços de destino de *branches* condicionais só estão disponíveis ao final do estágio EX.

Elabore uma tabela com o cenário de execução das instruções no *pipeline* (ou seja, qual instrução está ocupando cada estágio), ciclo por ciclo, para a execução **completa** do programa em *assembly* apresentado, considerando que o valor inicial armazenado em R0 é 4 e o valor inicial armazenado em R1 é 2. Utilize a tabela abaixo como modelo:

	Ciclo						
Instrução	1	2	3	4	5	6	...
Instr1	IF	ID	EX	...	...	...	...
Instr2		IF	ID	EX	...	...	...
...	...	...	...	...	...	...	...

onde *Instr1, Instr2* e subsequentes correspondem à sequência de execução das instruções do programa.

51. Refazer o exercício 50 da lista considerando as seguintes alterações:
    - todas as instruções utilizam o estágio MEM
    - se acessa a memória efetivamente -> 2 ciclos para MEM, com conflito com IF
    - se não acessa -> 1 ciclo para MEM, sem conflito com IF

    Condições iniciais:

    R2 -> valor 1000, no endereço 1000 da memória está contido o número 3
    R3 -> valor 1004, no endereço 1004 da memória está contido o número 2

    Código assembly proposto:

```
      LD R0, [R2], 4
      LD R1, [R3], 4
      MOV R5, R0
      SUB R5, R1
      BZ processa
      BMZ processa
      SUB R5, R5
      BZ fim
processa:
      SUB R0, R1
      MOV R5, R0
      SUB R5, R1
      BZ resultado
      BMZ processa
      SUB R5, R5
      BZ fim
resultado:
      MOV R0, 0
fim:
```

52. Considerando que o *pipeline* de ARQ do exercício 50 tem as seguintes características:

- 5 estágios: IF (busca de instrução), ID (decodificação de instrução), EX (execução de instrução), MEM (leitura/escrita de memória) e WB (atualização de registradores).
- Na ausência de conflito, todos os estágios operam em 1 ciclo de clock.
- O estágio MEM sempre ocupa 1 ciclo de *clock,* independente da instrução sendo executada acessar ou não a memória para dados.
- Estágios IF e MEM possuem conflito estrutural (MEM tem prioridade), a menos que o estágio MEM não esteja acessando efetivamente a memória.
- Todas as atualizações de registradores ocorrem no final do estágio WB.
- Implementa adiantamento de registrador a partir do final do estágio EX da instrução atual para o início do estágio ID da próxima instrução.
- Todos os registradores que uma instrução utiliza devem estar disponíveis no início do estágio ID desta instrução.
- A condição de um *branch* condicional só pode ser avaliada, durante o estágio EX da instrução de *branch* condicional, após o final do estágio EX da última operação aritmética executada (ADD ou SUB).
- Endereços de destino de *branches* condicionais só estão disponíveis ao final do estágio EX do respectivo *branch*.

Considerando que o seguinte programa em assembly será executado:

```
start:
          LD R4, [R0], 4
          LD R5, [R1], 4
          MOV R6, 0
          MOV R7, 1
calc:
          ADD R6, R7
          SUB R4, R5
          BMZ calc
          BZ grava
          ADD R4, R5
          SUB R6, R7
grava: ST R6, [R2], 4
          ST R4, [R3], 4
```

a) Para o seguinte estado inicial da memória e dos registradores, desenhe o cenário de execução das instruções no *pipeline*, ciclo por ciclo:

Registradores:

Reg	R0	R1	R2	R3	R4	R5	R6	R7
Valor (decimal)	1000	1004	1008	1012	0	0	125	12

Memória:

End (decimal)	1000	1004	1008	1012	1016	1020	1024	1028
Valor	9	4	7	28	14	72	0	15

b) Quantas bolhas são causadas por conflitos estruturais, quantas bolhas são causadas por conflitos de dados e quantas bolhas são causadas por conflitos de desvio?

c) Que alterações poderiam ser efetuadas no programa em *assembly* para evitar a ocorrência dos conflitos de dados?

Interrupções e exceções

53. Pesquisar e descrever como funciona o mecanismo de interrupção / exceção de um microprocessador comercial, baseado em qualquer arquitetura RISC ou CISC à escolha. Responder as seguintes perguntas:
    a) Quantas interrupções ele suporta?
    b) Quantas exceções ele suporta?
    c) As interrupções são vetorizadas? Se sim, uma explicação básica da sequência de operações, desde a detecção da IRQ, passando pela identificação da IRQ (como?) e pela leitura do vetor de interrupções (de que endereço?). Se não, explicar como o mecanismo funciona.

Barramentos e Dispositivos de Entrada e Saída

54. Pesquisar e descrever em detalhes como funciona um dos seguintes dispositivos de E/S:
    a) monitor de LED
    b) scanner

c) touch screen

55. (Tanenbaum, 2013) Usando certa fonte, uma impressora monocromática a laser pode imprimir 50 linhas de 80 caracteres por página. O caractere médio ocupa um quadrado de 2 mm x 2 mm, dos quais cerca de 25% é *toner.* O resto é branco. A camada de *toner* tem 25 micra de espessura. O cartucho de *toner* da impressora mede 25 x 8 x 2 cm. Quantas páginas podem ser impressas com um cartucho de *toner*?

56. Considerando que uma impressora a jato de tinta pinga gotas com volume igual a 2 picolitros, que a densidade de pontos em uma página é de 200 pontos por polegada e que se pretende preencher páginas inteiras de 8 polegadas de largura por 10 polegadas de altura com tinta, calcule quantas páginas se pode preencher com um cartucho de 12,8 mililitros. Mostrar os passos utilizados para o cálculo.

    Dados para conversão:

    1 picolitro = 1×10^{-12} litros

    1 mililitro = 1×10^{-3} litros

    N pontos por polegada = quadrado de N x N pontos em uma superfície de 1 polegada quadrada = 1 x 1 polegada

57. Considere que um CD possui uma capacidade de 700 MB de dados e que armazena arquivos de áudio digital (formato WAV) com as seguintes características:

    - Número de amostras digitalizadas por segundo: 44100 (44.1 kHz)
    - Tamanho de cada amostra: 16 bits, 2 canais (estéreo)

    Calcule quantos minutos de áudio podem ser armazenados neste CD.

58. Pesquisar e descrever em detalhes como funciona um barramento/interface de comunicação padronizado. Escolher uma opção a partir da lista a seguir:
    a) SPI
    b) USB
    c) PCI Express
    d) SATA
    e) RS-485

    Responder as seguintes perguntas:

    a) Características técnicas (transporta bits em série ou paralelo? É bidirecional? Funciona no modo “mestre-escravo”? e outras).
    b) Para que tipos de comunicação / interfaceamento é mais comumente utilizada?
    c) Qual(is) a(s) velocidade(s) de comunicação?

Hierarquias de memória

59. Considerando um sistema de memória com endereçamento de 32 bits, o qual contém um nível de cache associativo de 4 vias, modo de escrita *write-through,* composto por 32 blocos de cache de 256 bytes cada e política de substituição LRU, e no qual cada acesso da CPU à memória é feito de 256 em 256 bytes, responda as seguintes perguntas:
    a) Qual seria o estado da cache e da memória principal após as seguintes operações:
        - leitura de 2048 bytes do endereço 0x00010180
        - leitura de 4500 bytes do endereço 0x00010500
        - escrita de 480 bytes no endereço 0x00010136
        - leitura de 2736 bytes do endereço 0x000105B0
        - escrita de 280 bytes no endereço 0x00010300

        Mencionar:
    - índice de cada bloco de cache ocupado (válido) e o rótulo do bloco de memória nele armazenado.
    - quais blocos de cache estão *dirty*.
    - número de *hits* e *misses* para a execução das operações citadas.
    - quais bytes da memória principal foram efetivamente alterados após estas operações.

Observação: para acessos desalinhados à memória (ou seja, que não iniciam em endereço múltiplo de 256), considerar que a primeira parte deste acesso é efetuada até o próximo endereço múltiplo de 256, e que os acessos subsequentes são alinhados.

Por exemplo:

leitura de 480 bytes a partir de 0x10136 -> próximo múltiplo de 256 é 0x10200, portanto o primeiro acesso é até o endereço anterior a este -> 0x10200 – 0x10136 = 0xCA bytes (202 bytes) é o tamanho deste acesso. Os demais acessos são de 256 em 256 bytes ou o tamanho que sobrar, se for menor do que 256.

60. Considerando um sistema de memória com endereçamento de 16 bits, o qual contém um nível de cache associativo de 2 vias, modo de escrita *write-back,* composto por 8 blocos de cache de 8 bytes cada e política de substituição LRU, e no qual cada acesso da CPU à memória é feito de 4 em 4 bytes, responda as seguintes perguntas:
    a) Qual seria o estado da cache e da memória principal após as seguintes operações:
        - leitura de 20 bytes a partir do endereço 0x0010
        - leitura de 48 bytes a partir do endereço 0x0054

- escrita de 14 bytes a partir do endereço 0x0014
- leitura de 20 bytes a partir do endereço 0x0058
- escrita de 28 bytes a partir do endereço 0x0056

Mencionar:

- índice de cada bloco de cache ocupado (válido) e o rótulo do bloco de memória nele armazenado.
- quais blocos de cache estão *dirty*.
- número de *hits* e *misses* para a execução das operações citadas.
- quais bytes da memória principal foram efetivamente alterados após estas operações.

Observação: para acessos desalinhados à memória (ou seja, que não iniciam em endereço múltiplo de 4), considerar que este acesso é efetuado até o próximo endereço múltiplo de 4, e que os acessos subsequentes são alinhados

61. Considerando um sistema de memória com endereçamento de 16 bits, no qual cada acesso da CPU à memória é feito de 4 em 4 bytes e contendo um nível de memória cache de dados com as seguintes características:
    - 8 blocos de *cache* de 8 bytes cada;
    - associatividade em conjuntos de 4 vias;
    - modo de escrita *write-through;*

    a) Mostre qual seria o estado da *cache* e da memória principal após as seguintes operações:
        - leitura de 24 bytes a partir do endereço 0x0030
        - escrita de 8 bytes a partir do endereço 0x0024
        - leitura de 48 bytes a partir do endereço 0x0048
        - escrita de 12 bytes a partir do endereço 0x0050
        - leitura de 8 bytes a partir do endereço 0x0038

        Para cada bloco de *cache* deve ser apresentado: o índice do conjunto, um identificador de qual bloco de memória principal está ali armazenado e o valor do rótulo deste bloco (se for válido), um bit indicando se o conteúdo daquele bloco de *cache* é válido ou não e os bits de "sujo" e de "idade" (se aplicáveis)

    b) Quantos acessos resultarão em *misses* e quantos em *hits*?

    c) Quantos bits do endereço de 16 bits são utilizados para rótulo, quantos para deslocamento do endereço dentro do bloco de *cache* e quantos para índice do conjunto?

    d) Quais bytes da memória principal foram alterados após estas operações?

62. Repetir o exercício 61 considerando as seguintes alterações no sistema de cache:

- associativa em conjuntos de 2 vias;
- modo de escrita *write-back.*

63. Considerando o seguinte programa em *assembly* executando na arquitetura ARQ do exercício 27:

```
start:
        LD R4, [R0], 4
        LD R5, [R1], 4
        MOV R6, 0
        MOV R7, 1
calc:
        ADD R6, R7
        SUB R4, R5
        BMZ calc
        BZ grava
        ADD R4, R5
        SUB R6, R7
grava: ST R6, [R2], 4
        ST R4, [R3], 4
```

Considere o seguinte estado inicial da memória e dos registradores:

Registradores:

Reg	R0	R1	R2	R3	R4	R5	R6	R7
Valor (decimal)	1000	1004	1008	1012	0	0	125	12

Memória:

End (decimal)	1000	1004	1008	1012	1016	1020	1024	1028
Valor	6	2	7	28	14	72	0	15

Considere que o sistema de memória de ARQ possui um nível de memória *cache* de **instruções** com as seguintes características:

- 4 blocos de *cache* de 8 bytes cada;
- associatividade em conjuntos de 2 vias;

e) Para a execução do código apresentado, considerando os estados iniciais de memória e registradores também apresentados, considerando que os endereços de memória são todos de 16 bits e considerando que o endereço inicial do código em memória é 0x348 (hexadecimal), quantos acessos para busca de instrução resultarão em *misses* e quantos em *hits*?

f) Desenhe o estado final da *cache* após a execução do código da questão 2. Para cada bloco de *cache* deve ser apresentado o índice, o valor do rótulo do bloco de memória principal ali armazenado (se for válido) e um bit indicando se o conteúdo daquele bloco de *cache* é válido ou não.

g) Quantos bits do endereço de 16 bits são utilizados para rótulo, quantos para deslocamento do endereço dentro do bloco de *cache* e quantos para índice do bloco?

Obs: considerar que a busca de cada instrução da memória (32 bits) é uma operação separada, que gera um *hit* ou *miss.* Ainda, caso ocorra um *miss* para qualquer instrução, o bloco inteiro ao qual ela pertence é trazido para a *cache* antes da próxima instrução ser buscada.

64. Considerando um sistema de memória com endereçamento de 16 bits, o qual contém um nível de cache com as seguintes características:
    - 4 blocos de 8 bytes cada
    - Associativa em conjunto de 2 vias
    - Write-back
    - Acessos são feitos 4 bytes por vez

Dados os seguintes acessos, mostrar o estado final da cache e da RAM:

a) leitura de 12 bytes do endereço 0x0100
b) escrita de 8 bytes do endereço 0x0108
c) leitura de 8 bytes do endereço 0x010C
d) escrita de 4 bytes no endereço 0x0100

65. Considerando o seguinte programa em *assembly* executando na arquitetura ARQ do exercício 27:

```
      LD R0, [R2], 4
      LD R1, [R3], 4
      MOV R5, R0
      SUB R5, R1
      BZ processa
      BMZ processa
      SUB R5, R5
      BZ fim
processa:
      SUB R0, R1
      MOV R5, R0
      SUB R5, R1
      BZ resultado
      BMZ processa
      SUB R5, R5
      BZ fim
resultado:
      MOV R0, 0
fim:
```

Condições iniciais:

R2 -> valor 1000, no endereço 1000 da memória está contido o número 3
R3 -> valor 1004, no endereço 1004 da memória está contido o número 2

Considere que o sistema de memória de ARQ possui um nível de memória *cache* de **instruções** com as seguintes características:

- 4 blocos de *cache* de 8 bytes cada;
- associatividade em conjuntos de 2 vias;

h) Para a execução do código apresentado, considerando os estados iniciais de memória e registradores também apresentados, considerando que os endereços de memória são todos de 16 bits e considerando que o endereço inicial do código em memória é **0x0500**, quantos acessos para busca de instrução resultarão em *misses* e quantos em *hits*?
i) Desenhe o estado final da *cache* após a execução do código da questão 2. Para cada bloco de *cache* deve ser apresentado o índice, o valor do rótulo do bloco de

memória principal ali armazenado (se for válido) e um bit indicando se o conteúdo daquele bloco de *cache* é válido ou não.

j) Quantos bits do endereço de 16 bits são utilizados para rótulo, quantos para deslocamento do endereço dentro do bloco de *cache* e quantos para índice do bloco?

Obs: considerar que a busca de cada instrução da memória (32 bits) é uma operação separada, que gera um *hit* ou *miss.* Ainda, caso ocorra um *miss* para qualquer instrução, o bloco inteiro ao qual ela pertence é trazido para a *cache* antes da próxima instrução ser buscada.

Arquiteturas paralelas

66. Descreva uma arquitetura de multiprocessador disponível atualmente no mercado. Responda as seguintes perguntas:
    a) Quantos núcleos reais e/ou virtuais são definidos em um processador que implementa esta arquitetura?
    b) A arquitetura implementa algum esquema de *multithreading*? Se sim, como é o esquema e como funciona o compartilhamento de recursos?
    c) Como é a topologia de memória entre os diversos núcleos de execução?
    d) Como é o *pipeline* desta arquitetura (se esta informação estiver disponível)?

67. (Tanenbaum, 2013) Em uma determinada CPU, uma instrução que encontra uma ausência de *cache* de nível 1, mas uma presença na *cache* de nível 2, leva k ciclos no total. Se for usado *multithreading* para mascarar ausências da *cache* de nível 1, quantas *threads* precisam ser executados ao mesmo tempo usando *multithreading* de granulação fina para evitar ciclos ociosos? Justifique a sua resposta.

68. Em uma determinada arquitetura de computação paralela se deseja executar uma aplicação na qual 20% das instruções executadas possuem uma dependência sequencial. Considerando que se deseja expandir o número *N* de núcleos de execução para se diminuir o tempo total de execução em pelo menos 75%, calcule qual seria o valor mínimo de N.

69. Em uma determinada arquitetura de computação paralela se deseja executar uma aplicação na qual 10% das instruções executadas possuem uma dependência sequencial. Considerando que este programa será executado em uma arquitetura paralela de 8 núcleos, calcule qual será o percentual de redução do tempo de execução se comparada com uma arquitetura com 1 único núcleo.