GABARITO- AD2 - Organização de Computadores 2009.2
Data de entrega: 14/11/2009

1. (1,0) Crie 3 conjuntos de instruções de um, dois, e três operandos, definidas em Linguagem Assembly, necessárias para a realização de operações aritméticas e elabore programas para o cálculo das equações abaixo (no total de 3 programas para cada item abaixo, sendo o 1º programa utilizando o conjunto de 1 operando, o 2º utilizando o conjunto de 2 operandos e finalmente o 3º utilizando o conjunto de 3 operandos).

```
I_ CONJUNTO DE INSTRUÇÕES PARA 1 OPERANDO:
   LOAD M      =>    ACC   <-  M
   STORE M     =>    M     <-  ACC
   ADD M       =>    ACC   <-  ACC + M
   SUB M       =>    ACC   <-  ACC - M
   MUL M       =>    ACC   <-  ACC * M
   DIV M       =>    ACC   <-  ACC / M

II_ CONJUNTO DE INSTRUÇÕES PARA 2 OPERANDOS:
   ADD X,Y   =>   X <- X + Y
   SUB X,Y   =>   X <- X - Y
   MUL X,Y   =>   X <- X * Y
   DIV X,Y   =>   X <- X / Y
   MOV X,Y   =>   X <- Y

III_ CONJUNTO DE INSTRUÇÕES PARA 3 OPERANDOS:
   ADD Y , Z, X      =>    X <- Y + Z
   SUB Y , Z, X      =>    X <- Y - Z
   MUL Y , Z, X      =>    X <- Y * Z
   DIV Y , Z, X      =>    X <- Y / Z
```

1.1. Y = (A – D/C) - (B + (D/E - B) * (C + A * B) - E)

PARA 1 OPERANDO:

```
      LOAD D      =>    ACC <-D
      DIV C       =>    ACC <-ACC / C
      STORE T1    =>    T1  <-ACC
      LOAD A      =>    ACC <-A
      SUB T1      =>    ACC <-ACC – T1
      STORE T1    =>    T1  <-ACC
      LOAD D      =>    ACC <-D
      DIV E       =>    ACC <-ACC / E
      SUB B       =>    ACC <-ACC – B
      STORE T2    =>    T2  <-ACC
      LOAD A      =>    ACC <-A
      MUL B       =>    ACC <-ACC * B
      ADD C       =>    ACC <-ACC + C
      MUL T2      =>    ACC <-ACC * T2
      ADD B       =>    ACC <-ACC + B
      SUB E       =>    ACC <-ACC – E
      STORE T2    =>    T2  <-ACC
      LOAD T1     =>    ACC <-T1
      SUB T2      =>    ACC <-ACC – T2
      STORE Y     =>    Y  <-ACC
```

PARA 2 OPERANDOS:

```
      MOV T1,D   =>   T1 <- D
      DIV T1,C   =>   T1 <- T1 / C
      MOV T2,A   =>   T2 <- A
      SUB T2,T1  =>   T2 <- T2 – T1
      MOV T1,D   =>   T1 <- D
      DIV T1,E   =>   T1 <- T1 / E
      SUB T1,B   =>   T1 <- T1 – B
      MOV T3,A   =>   T3 <- A
```

```
MUL T3,B    =>    T3 <- T3 * B
ADD T3,C    =>    T3 <- T3 + C
MUL T1,T3   =>    T1 <- T1 * T3
ADD T1,B    =>    T1 <- T1 + B
SUB T1,E    =>    T1 <- T1 – E
SUB T2,T1   =>    T2 <- T2 – T1
MOV Y,T2    =>    Y <- T2
```

PARA 3 OPERANDOS:

```
DIV D, C, T1      =>    T1 <- D / C
SUB A, T1, T1     =>    T1 <- A – T1
DIV D, E, T2      =>    T2 <- D / E
SUB T2, B, T2     =>    T2 <- T2 – B
MUL A , B, T3     =>    T3 <- A * B
ADD C , T3, T3    =>    T3 <- C + T3
MUL T2 , T3, T2   =>    T2 <- T2 * T3
ADD B , T2, T2    =>    T2 <- B + T2
SUB T2, E, T2     =>    T2 <- T2 – E
SUB T1, T2, Y     =>    Y <- T1 – T2
```

1.2. Y = (A + D * (C + B/(E * (B/F - C))) - E)

PARA 1 OPERANDO:

```
LOAD B      =>    ACC <-B
DIV F       =>    ACC <-ACC / F
SUB C       =>    ACC <-ACC – C
MUL E       =>    ACC <-ACC * E
STORE T1    =>    T1  <-ACC
LOAD B      =>    ACC <-B
DIV T1      =>    ACC <-ACC / T1
ADD C       =>    ACC <-ACC + C
MUL D       =>    ACC <-ACC * D
ADD A       =>    ACC <-ACC + A
SUB E       =>    ACC <-ACC – E
STORE Y     =>    Y   <-ACC
```

PARA 2 OPERANDOS:

```
MOV T1,B    =>    T1 <- B
DIV T1,F    =>    T1 <- T1 / F
SUB T1,C    =>    T1 <- T1 – C
MUL T1,E    =>    T1 <- T1 * E
MOV T2,B    =>    T2 <- B
DIV T2,T1   =>    T2 <- T2 / T1
ADD T2,C    =>    T2 <- T2 + C
MUL T2,D    =>    T2 <- T2 * D
ADD T2,A    =>    T2 <- T2 + A
SUB T2,E    =>    T2 <- T2 - E
MOV Y,T2    =>    Y  <- T2
```

PARA 3 OPERANDOS:

```
DIV B, F, T1      =>    T1 <- B / F
SUB T1, C, T1     =>    T1 <- T1 – C
MUL T1, E, T1     =>    T1 <- T1 * E
DIV B, T1, T1     =>    T1 <- B / T1
ADD C, T1, T1     =>    T1 <- C + T1
MUL D, T1, T1     =>    T1 <- D * T1
ADD A, T1, T1     =>    T1 <- A + T1
SUB T1, E, Y      =>    Y <- T1 – E
```

2. (1,0) Considere o programa assembly abaixo (baseado no assembly visto no capítulo 9 do livro texto) e o respectivo código de máquina :

LDA X 127 *1A*

	JZ FIM	523	*1B*
	LDA Z	124	*1C*
	ADD Y	325	*1D*
ORG	SUB T	426	*1E*
	STR X	227	*1F*
	JZ FIM	523	*20*
	PRT X	B27	*21*
	JMP ORG	81E	*22*
FIM	HLT	000	*23*

Considere que as variáveis Y, Z, T e X estão armazenadas na memória nos seguintes endereços e com os conteúdos apresentados abaixo:

Variável	Endereço (hexadecimal)	Valor (hexadecimal)
Z	24	00B
Y	25	001
T	26	004
X	27	01A

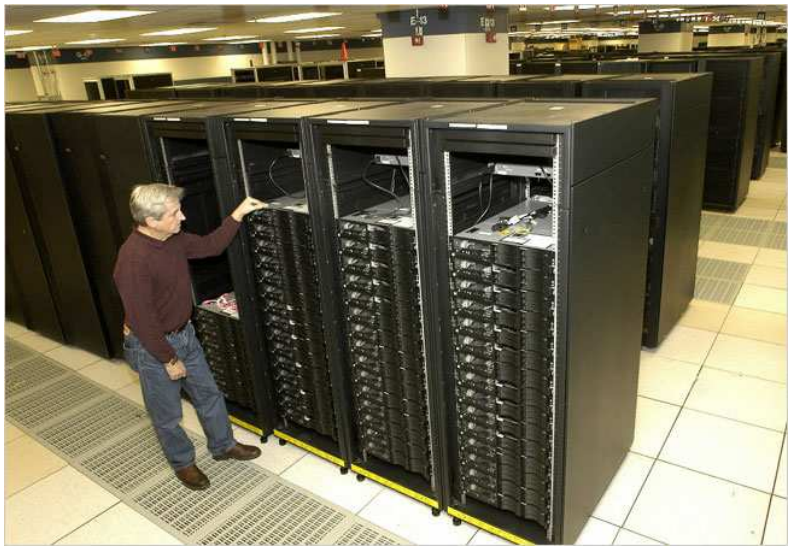
Considere que o programa esteja armazenado na MP a partir do endereço 1A (hexadecimal), e o contador de instruções tem o valor 1A. Para a execução de cada instrução, mostre os valores do CI, RI , acumulador e das variáveis Y, Z, T e X

CI	*Conteúdo*	*Descrição*	*RI*	*ACC*	*Z*	*Y*	*T*	*X*
1A			*000*	*000*	*00B*	*001*	*004*	*01A*
1B	*127*	*LDA X (ACC <- X)*	*127*	*01A*	*00B*	*001*	*004*	*01A*
1C	*523*	*JZ 23 (Se ACC=0,PC <- 23)*	*523*	*01A*	*00B*	*001*	*004*	*01A*
1D	*124*	*LDA Z (ACC <- Z)*	*124*	*00B*	*00B*	*001*	*004*	*01A*
1E	*325*	*ADD Y (ACC <- ACC + Y)*	*325*	*00C*	*00B*	*001*	*004*	*01A*
1f	*426*	*SUB T (ACC <- ACC – T)*	*426*	*008*	*00B*	*001*	*004*	*01A*
20	*227*	*STR X (X <- ACC)*	*227*	*008*	*00B*	*001*	*004*	*008*
21	*523*	*JZ 23 (Se ACC=0,PC <- 23)*	*523*	*008*	*00B*	*001*	*004*	*008*
22	*B27*	*PRT X, (exibe valor de X)*	*B27*	*008*	*00B*	*001*	*004*	*008*
1E	*81E*	*JMP 1E (PC = 1E)*	*81E*	*008*	*00B*	*001*	*004*	*008*
1F	*426*	*SUB T (ACC <- ACC – T)*	*426*	*004*	*00B*	*001*	*004*	*008*
20	*227*	*STR X (X <- ACC)*	*227*	*004*	*00B*	*001*	*004*	*004*
21	*523*	*JZ 23 (Se ACC=0,PC <- 23)*	*523*	*004*	*00B*	*001*	*004*	*004*
22	*B27*	*PRT X, (exibe valor de X)*	*B27*	*004*	*00B*	*001*	*004*	*004*
1E	*81E*	*JMP 1E (PC = 1E)*	*81E*	*004*	*00B*	*001*	*004*	*004*
1F	*426*	*SUB T (ACC <- ACC – T)*	*426*	*000*	*00B*	*001*	*004*	*004*
20	*227*	*STR X (X <- ACC)*	*227*	*000*	*00B*	*001*	*004*	*000*
23	*523*	*JZ 23 (Se ACC=0,PC <- 23)*	*523*	*000*	*00B*	*001*	*004*	*000*
***	*000*	*HLT (END)*	*000*	*000*	*00B*	*001*	*004*	*000*

3. (1,0) Faça uma busca na lista dos 500 sistemas de computadores com melhor desempenho do mundo em http://www.top500.org e descreva os três primeiros colocados (pesquise neste mesmo site e na internet).

1º. Lugar: Roadrunner (Papa-léguas)
Especificações:

Família: Cluster IBM
Local: DOE/NNSA/LANL
Modelo: BladeCenter QS22Cluster
Computador: BladeCenter QS22/LS21 Cluster, PowerCell 8i 3.2Ghs / Opteron DC 1.8 GHz, Voltarie Infiniband
Vendedor: IBM
Ano de instalação: 2008
Sistema Operacional: Linux
Interconexão: Infiniband
Processador: PowerXCell 8i 3200MHz (12,8 GFlops)
total: 129.600 cores / 1.105.000 GFlops



RoadRunner - DOE/NNSA/LANL
Fonte: www.zedomax.com

2º. Lugar: Jaguar
Especificações:

Família: Cray XT
Local: Oak Ridge National Laboratory
Modelo: Cray XT5 QuadCore
Computador: Cray XT5 QC 2.3 GHz
Vendedor: Cray Inc.
Ano de instalação: 2008
Sistema operacional: CNL
Interconexão: XT4 Internal Interconnect
Processador: AMD x86_64 Opteron Quad Core 2300MHz (9,2 GFlops)
Total: 150.152 cores / 1.059.000 GFlops

JAGUAR - Oak Ridge National Laboratory
Fonte: http://www.symmetrymagazine.org

3º. Lugar: Jugene
Especificações:

Família: IBM BlueGene
Local: Forschungszentrum Juelich (FZJ)
Modelo: BlueGene/P
Computador: Blue Gene/p Solution

Vendedor: IBM
Ano de instalação: 2009
Sistema operacional: CNK/SLES 9
Interconexão: Proprietário
Processador: PowerPC 450 850 MHz (3.4 GFlops)
Total: 294.912 cores / 825.500 GFlops

Jugene - Forschungszentrum Juelich (FZJ)
Fonte: http://www.ttecx.de

4. (1,0) Responda:
4.1. Dados os valores de memória abaixo e uma máquina de 1 endereço com um acumulador:
palavra 20 contém 40
palavra 30 contém 20
palavra 40 contém 60
palavra 50 contém 40
Quais valores as seguintes instruções carregam no acumulador?
-Load imediato 30
-Load direto 30
-Load indireto 30

- LOAD IMEDIATO 30
Nesta instrução o valor a ser colocado no acumulador corresponde ao valor fornecido como operador, portanto: ACC <- 30 (o valor a ser colocado no acumulador é 30)

- LOAD DIRETO 30
Nesta instrução o valor a ser colocado no acumulador corresponde ao valor contido no endereço de memória fornecido como operador, portanto:
ACC <- (30) (o valor a ser colocado no acumulador é 20)

- LOAD INDIRETO 30
Nesta instrução o valor a ser colocado no acumulador corresponde ao valor contido no endereço que consta como valor no endereço de memória fornecido como operador, portanto
ACC <- ((30)) (o valor a ser colocado no acumulador é 40)

4.2. Analise os modos de endereçamento direto, indireto e imediato, estabelecendo diferenças de desempenho, vantagens e desvantagens de cada um. Utilizando o conjunto de instruções proposto na questão 1, para dois e três operandos, projete instruções com endereçamento imediato, direto e indireto.

MODO DE ENDEREÇA-	*DEFINIÇÃO*	*VANTAGENS*	*DESVANTAGENS*	*DESEMPENHO*

MENTO				
Imediato	*O campo operando contém o dado*	*Rapidez na execução da instrução*	*Limitação do tamanho do dado. Inadequado para o uso com dados de valor variável*	*Não requer acesso a memória principal. Mais rápido que o modo direto*
Direto	*O campo operando contém o endereço do dado*	*Flexibilidade no acesso a variáveis de valor diferente em cada execução do programa*	*Perda de tempo, se o dado é uma constante*	*Requer apenas um acesso a memória principal. Mais rápido que o modo indireto*
Indireto	*O campo operando corresponde ao endereço que contém a posição onde está o conteúdo desejado,*	*Manuseio de vetores (quando o modo indexado não está disponível). Usar como "ponteiro"*	*Muitos acessos à MP para execução*	*Requer 2 acessos a memória principal*

Sugestão de projeto de instruções

OBS: Em geral para diferenciar as instruções é acrescido ou modificado caracter(es) no rótulo da instrução base. Nos casos abaixo, foram acrescidos as letras: i, d, n ; para indicar os modos Imediato, direto e indireto, respectivamente

	Imediato	**Direto**	**Indireto**

2 operandos	`ADDi X,Y => (X)<-(X)+Y`
	`SUBi X,Y => (X)<-(X)-Y`
	`MULi X,Y => (X)<-(X)*Y`
	`DIVi X,Y => (X)<-(X)/Y`
	`MOVi X,Y => (X)<-Y`
	
	Neste modo, Y é um valor imediato
		`ADDd X,Y => (X)<-(X)+(Y)`
		`SUBd X,Y => (X)<-(X)-(Y)`
		`MULd X,Y => (X)<-(X)*(Y)`
		`DIVd X,Y => (X)<-(X)/(Y)`
		`MOVd X,Y => (X)<-(Y)`
		
		Neste modo, Y é um registrador ou endereço de memória
			`ADDn X,Y => (X)<-(X)+((Y))`
			`SUBn X,Y => (X)<-(X)-((Y))`
			`MULn X,Y => (X)<-(X)*((Y))`
			`DIVn X,Y => (X)<-(X)/((Y))`
			`MOVn X,Y => (X)<-((Y))`
			
			Neste modo, Y é um registrador ou endereço de memória que contenha a posição do valor desejado

OBS: Da mesma forma, para diferenciar instruções de 2 operandos para 3 operandos, acrescenta-se ou modifica-se caracter(es) no rótulo da instrução base. Neste exemplo foi acrescentado a letra T

	Imediato	**Direto**	**Indireto**

3 operandos	`ADDTi Y,Z,X=>(X)<-Y+(Z)`
	`SUBTi Y,Z,X=>(X)<-Y-(Z)`
	`MULTi Y,Z,X=>(X)<-Y*(Z)`
	`DIVTi Y,Z,X=>(X)<-Y/(Z)`
	
	Neste modo, Y é um valor imediato
		`ADDTd Y,Z,X=>(X)<-(Y)+(Z)`
		`SUBTd Y,Z,X=>(X)<-(Y)-(Z)`
		`MULTd Y,Z,X=>(X)<-(Y)*(Z)`
		`DIVTd Y,Z,X=>(X)<-(Y)/(Z)`
		
		Neste modo, Y é um registrador ou endereço de memória
			`ADDTn Y,Z,X=>(X)<-((Y))+(Z)`
			`SUBTn Y,Z,X=>(X)<-((Y))-(Z)`
			`MULTn Y,Z,X=>(X)<-((Y))*(Z)`
			`DIVTn Y,Z,X=>(X)<-((Y))/(Z)`
			
			Neste modo, Y é um registrador ou endereço de memória que contenha a posição do valor desejado

4.3. Qual é o objetivo do emprego do modo de endereçamento base mais deslocamento? Qual é a diferença de implementação e utilização entre esse modo e o modo indexado? Forneça 1 exemplo de instrução para modo indexado e 1 para base mais indexamento

O base mais deslocamento tem como seu principal objetivo permitir a modificação de endereço de programas ou módulos destes (que é a relocação de programa), bastando para isso uma única alteração no registrador base.

O base mais deslocamento tem como característica o endereço ser obtido da soma do deslocamento com o registrador base, diferindo do modo indexado onde o do registrador base é fixo e variar no deslocamento, ao contrário deste onde o deslocamento é fixo e com a alteração do registrador base permite-se a mudança do endereço.

Exemplos de instruções modo indexado:

LDX Ri, Op ==> ACC <--- ((Op) + (Ri))

ADX Ri, Op ==> ACC <--- ACC + ((Op) + (Ri))

Exemplo: instrução base mais deslocamento:

LDB Rb, Op ==> (ACC) <--- ((Op) + (Rb))

ADB Rb, Op ==> ACC <--- ACC + ((Op) + (Rb))

Sendo,

Op = Operando
Ri = Registrador de índice
Rb = Registrador base

5. (1,0) Explique, comparando:

5.1. Compilação e Interpretação (Dê exemplos de linguagens que se utilizem de compiladores e de linguagens que se utilizem de interpretadores).

A compilação consiste na análise de um programa escrito em linguagem de alto nível (programa fonte) e sua tradução em um programa em linguagem de máquina (programa objeto).

Na interpretação cada comando do código fonte é lido pelo interpretador, convertido em código executável e imediatamente executado antes do próximo comando.

A interpretação tem como vantagem sobre a compilação a capacidade de identificação e indicação de um erro no programa-fonte (incluindo erro da lógica do algoritmo) durante o processo de conversão do fonte para o executável.

A interpretação tem como desvantagem o consumo de memória devido ao fato de o interpretador permanecer na memória durante todo o processo de execução do programa. Na compilação o compilador somente é mantido na memória no processo de compilação e não utilizado durante a execução. Outra desvantagem da interpretação está na necessidade de tradução de partes que sejam executadas diversas vezes, como os loops que são traduzidos em cada passagem. No processo de compilação isto só ocorre uma única vez. Da mesma forma pode ocorrer para o programa inteiro, em caso de diversas execuções, ou seja, a cada execução uma nova interpretação.

Exemplos de linguagem interpretadas: ASP, BASIC, Java, PHP, Python, Lisp entre outras.

Exemplos de linguagem compilada: C, Pascal, Delphi, Visual Basic, entre outras.

5.2. Sistemas SMP e Sistemas NUMA (Forneça exemplos atuais de sistemas SMP e Sistemas NUMA).

Sistemas SMP (ou UMA) têm como característica o acesso a todas as partes da memória principal com tempo de acesso uniforme. Em sistemas NUMA, todos os processadores possuem também acesso a todas as partes da memória principal podendo diferir o tempo de acesso em relação às posições da memória e processador.

Nos sistemas SMP o aumento no número de processadores tem como conseqüência problemas de tráfego no barramento comum degradando o desempenho. Uma solução para isto é a utilização de clusters, que tem, usualmente, como conseqüência alterações significativas na aplicação (software). Nos sistemas NUMA podem-se ter vários nós multiprocessadores, cada qual com seu próprio barramento, resultando em pequenas alterações na aplicação (software).

Exemplo de NUMA: Cray T3D

Exemplo de SMP: Maioria dos servidores da HP, Compac, IBM.

5.3. Arquiteturas RISC e Arquiteturas CISC . Faça uma pesquisa em sites confiáveis e descreva como a Intel classifica o Pentium e a justificativa para tal classificação. Cite a fonte.

RISC: Reduced Instruction Set Computer – Computador com um conjunto reduzido de instruções
CISC - Complex Instruction Set Computer: Computador com um conjunto complexo de instruções

CISC: Principais características:

Possui microprogramação para aumento da quantidade de instruções incluindo novos modos de endereçamento, de forma a diminuir a complexidade dos compiladores e em conseqüência permitir linguagens de alto nível com comandos poderosos para facilitar a vida dos programadores. Em contrapartida, muitas instruções significam muitos bits em cada código de operação, instrução com maior comprimento e maior tempo de interpretação.

RISC: Principais características:

Menor quantidade de instruções e tamanho fixo. Não há microprogramação. Permite uma execução otimizada, mesmo considerando que uma menor quantidade de instruções vá conduzir a programas mais longos. Uma maior quantidade de registradores e suas utilizações para passagem de parâmetros e recuperação dos dados, permitindo uma execução mais otimizada de chamada de funções. Menor quantidade de modos de endereçamento com o objetivo da reduzir de ciclos de relógio para

execução das instruções. Instruções de formatos simples e únicos tiram maior proveito de execução com pipeline cujos estágios consomem o mesmo tempo.

Segundo a Intel, o processador Pentium é classificado basicamente como CISC, mas contém implementações RISC de forma a executar instruções simples de forma mais rápida com pouca influência da microprograma.

Veja o texto a seguir retirado do site www.guiadohardware.com.br sobre as 2 tecnologias: RISC e CISC

"...ao invés da vitória de uma das duas tecnologias, atualmente vemos processadores híbridos, que são essencialmente processadores CISC, mas incorporam muitos recursos encontrados nos processadores RISC (ou vice-versa).

Apesar de por questões de Marketing, muitos fabricantes ainda venderem seus chips, como sendo "Processadores RISC", não existe praticamente nenhum processador atualmente que siga estritamente uma das duas filosofias. Tanto processadores da família x86, como o Pentium II, Pentium III e AMD Athlon, quanto processadores supostamente RISC, como o MIPS R10000 e o HP PA-8000, ou mesmo o G4, utilizado nos Macintoshs misturam características das duas arquiteturas, por simples questão de performance. Por que ficar de um lado ou de outro, se é possível juntar o melhor dos dois mundos? A última coisa que os fabricantes de processadores são é teimosos, sempre que aparece uma solução melhor, a antiga e abandonada...."

6. (1,5) Um sistema de computação possui somente 7 bits para representar dados. Considere o seguinte conjunto de bits:

A=1111111
B=0000001
C=1000001
D=0101001

6.1. (0,3) Indique o valor decimal para A, B, C e D considerando que eles representam:

6.1.1. inteiros sem signal

A=1111111 = $2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 127$
B=0000001 = $2^0 = 1$
C=1000001 = $2^6 + 2^0 = 65$
D=0101001 = $2^5 + 2^3 + 2^0 = 41$

6.1.2. inteiros em sinal e magnitude

A=1111111 = $-(2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) = -63$
B=0000001 = $+(2^0) = +1$
C=1000001 = $-(2^0) = -1$
D=0101001 = $+(2^5 + 2^3 + 2^0) = +41$

6.1.3. inteiros em complemento a 2

A=1111111 = $-2^6 + (2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) = -1$
B=0000001 = $+2^0 = +1$
C=1000001 = $-2^6 + (2^0) = -63$
D=0101001 = $+2^5 + 2^3 + 2^0 = +41$

As seguintes operações foram realizadas neste sistema:

E=A+B
F=A+C
G=C+D
H=B+D

6.2. (1,2) Indique o valor decimal para E, F, G e H considerando que A, B, C, D, E, F, G e H representam:

6.2.1. inteiros sem sinal

```
E=A+B  =  1111111 + 0000001 =   0000000  =   ( 0 ) 10   =>  HOUVE OVERFLOW
F=A+C  =  1111111 + 1000001 =   1000000  =   ( 64 )10   =>  HOUVE OVERFLOW
G=C+D  =  1000001 + 0101001 =   1101010  =   ( 106 )10  =>  NÃO HOUVE OVERFLOW
H=B+D  =  0000001 + 0101001 =   0101010  =   ( 42 )10   =>  NÃO HOUVE OVERFLOW
```

6.2.2. inteiros em sinal e magnitude

```
E=A+B  =  1111111 + 0000001 =   1111110  =   ( -62 ) 10   =>  NÂO HOUVE OVERFLOW
F=A+C  =  1111111 + 1000001 =   1000000  =   (  -0 ) 10   =>  HOUVE OVERFLOW
```

```
G=C+D  =  1000001 + 0101001 =      0101000  =   ( 40 )10  =>  NÃO HOUVE OVERFLOW
H=B+D  =  0000001 + 0101001 =      0101010  =   ( 42 )10   =>  NÃO HOUVE OVERFLOW
```

6.2.3. inteiros em complemento a 2

```
E=A+B  =  1111111 + 0000001 =      0000000  =   ( 0 )10    =>  NÃO HOUVE OVERFLOW
F=A+C  =  1111111 + 1000001 =      1000000  =   ( -64 )10  =>  NÃO HOUVE OVERFLOW
G=C+D  =  1000001 + 0101001 =      1101010  =   ( -22 )10  =>  NÃO HOUVE OVERFLOW
H=B+D  =  0000001 + 0101001 =      0101010  =   ( 42 ) 10  =>  NÃO HOUVE OVERFLOW
```

Explique, para cada uma das 4 somas acima, se a operação irá causar estouro (overflow) ou não.

6.2.1-Ocorrerá overflow para E, porque o valor correto da soma é 128 que não é possível de ser representado com 7 bits utilizando-se representação inteiro sem sinal.
Ocorrerá overflow para F, porque seu valor correto é 192 que não é possível de ser representado com 7 bits utilizando-se a representação inteiro sem sinal.
Não ocorrerá overflow para G e H porque seu valores corretos são 106 e 42 que podem ser representados com 7 bits utilizando-se representação inteiro sem sinal.

6.2.2-Ocorrerá overflow para F porque seu valor correto é -64 que não é possível de ser representado com 7 bits utilizando-se a representação sinal e magnitude. Não ocorrerá overflow para E, G e H porque seus valores corretos são -62, +40 e +42 que podem ser representados com 7 bits utilizando-se a representação sinal e magnitude.

6.2.3-Não ocorrerá overflow para E, F, G e H porque seus valores corretos são 0, -64,-22 e +42 que podem ser representados com 7 bits utilizando-se representação em complemento a 2.

7. (1,5) Considere um computador, cuja representação para ponto fixo e para ponto flutuante utilize 18 bits. Na representação em ponto flutuante, números normalizados estão no formato +/-(1, b_{-1} b_{-2} b_{-3} b_{-4} b_{-5} b_{-6} b_{-7} b_{-8} b_{-9} b_{-10} b_{-11} b_{-12} b_{-13}× $2^{Expoente}$), onde o bit mais à esquerda representa o sinal (0 para números positivos e 1 para números negativos), os próximos 4 bits representam o expoente em complemento a 2 e os 13 bits seguintes representam os bits b_{-1} a b_{-13}, como mostrado na figura a seguir:

7.1. (0,5) Para o conjunto de bits $(2DDC0)_{16}$, indique o valor que está sendo representado em decimal neste computador, considerando-se que o conjunto representa:

$(2DDC0)_{16} = (10\ 1101\ 1101\ 1100\ 0000)_2$

7.1.1. um inteiro sem sinal

$$2^{17} + 2^{15} + 2^{14} + 2^{12} + 2^{11} + 2^{10} + 2^8 + 2^7 + 2^6 = 187840$$

7.1.2. um inteiro em sinal magnitude

$$-(2^{15} + 2^{14} + 2^{12} + 2^{11} + 2^{10} + 2^8 + 2^7 + 2^6) = -56768$$

7.1.3. um inteiro em complemento a 2

$$-2^{17} + (2^{15} + 2^{14} + 2^{12} + 2^{11} + 2^{10} + 2^8 + 2^7 + 2^6) = -74304$$

7.1.4. um número em ponto flutuante

Sinal => 1 = negativo
Expoente => 0110 = 6
Mantissa => 1110111000000 =
Resultado = -(1, 1110111000000) x 2^{+6} = $-(1111011,1)_2$ = -123,5

7.2. (0,2) Qual será a representação em ponto flutuante dos seguintes valores decimais neste computador:
7.2.1. +32,0

$+32,0 = 100000,0 = 1,00000 \times 2^5$

Sinal = 0 = positivo
Expoente = 5 = 0101 (complemento a 2)
Mantissa = 0000000000000
Resposta: 0 0101 0000000000000

7.2.2. –0,2

-0,2 = 0,0011001100110011... = 1,10011001100110011... x 2^{-3}
Sinal = 1 = negativo
Expoente = -3 = 1101 (complemento a 2)
Mantissa = 10011001100110011...
Resposta: 1 1101 1001100110011

7.3. (0,4) Os conjuntos de bits que representam o menor e maior valor para o expoente não são utilizados para representar números normalizados. Indique o menor e o maior valor positivo e o menor e maior valor negativo de números normalizados que podem ser representados na representação em ponto flutuante para este computador. Mostre os valores em decimal.

Maior número posiivo: 0 0110 1111111111111 = +1,1111111111111 ×2^{+6} = + 127,9921875
Menor número positivo: 0 1001 0000000000000 = +1,0 ×2^{-7} = +0,0078125
Menor número negativo 1 0110 1111111111111 = -127,9921875
Maior número negativo: 0 1001 0000000000000 = -0,0078125

7.4. (0,4) Caso se utilize a representação em excesso para representar o expoente, indique o excesso a ser utilizado e como será a representação dos números dos itens 2.2.1 e 2.2.2.

Excesso = $2^{e-1} – 1$, sendo e = 4, excesso=7

7.2.1. +32,0

+32,0 = 100000,0 = 1,00000 x 2^5
Sinal = 0 = positivo
Expoente = 5 + 7 = 12 = 1100 (por excesso)
Mantissa = 0000000000000
Resposta: 0 1100 0000000000000

7.2.2. –0,2

-0,2 = 0,0011001100110011... = 1,10011001100110011... x 2^{-3}
Sinal = 1 = negativo
Expoente = -3 +7 = 4 = 0100 (por excesso)
Mantissa = 10011001100110011...
Resposta: 1 0100 1001100110011

8. (1,0) Explique como funcionam os três mecanismos utilizados para transferir dados entre um dispositivo de E/S e a memória de um sistema de computação: por programa (polling), por interrupção e por acesso direto à memória.

Por programa: A UCP indica à interface de entrada e saída que deseja realizar uma operação de transferência de dados e fica interrogando a interface para saber se ela está pronta para realizar a transferência de dados. Quando a UCP recebe uma resposta positiva da interface, ela realiza a transferência de dados. Para ler dados da interface e colocar os dados na memória, ela realiza operações de leitura de dados da interface e escrita na memória. Para escrever dados na interface, ela realiza operações de leitura da memória e escrita na interface.

Por interrupção: A UCP indica à interface de entrada e saída que deseja realizar uma operação de transferência de dados e realiza outras instruções que não se referenciam a esta operação, ou seja, a UCP não fica interrogando a interface para identificar quando ela está pronta. Quando a interface está pronta para realizar a transferência, ela gera um sinal de interrupção que é recebido pela UCP. A UCP ao receber este sinal, termina de realizar a instrução que estava sendo realizada, salva o

contexto onde esta instrução estava sendo realizada, e executa as instruções para realizar a transferência de dados com a interface.

Por acesso direto à memória (DMA) : Um controlador de DMA realiza diretamente a transferência de dados entre a interface e a memória sem envolver a UCP nesta transferência. A UCP necessita enviar alguns parâmetros para o controlador de DMA: o endereço da interface, o tipo de transferência (escrita ou leitura de dados), o endereço de memória para ler ou escrever os dados e o número de bytes a serem transferidos. O controlador de DMA realiza toda a transferência de dados entre a interface e a memória e a UCP não necessita executar nenhuma instrução para realizar esta transferência. Quando a transferência acaba, o controlador de DMA gera um sinal de interrupção para a UCP indicando que a transferência foi realizada.

9. (1,0) Explique o mecanismo de funcionamento de um monitor de vídeo LCD (indique a referência bibliográfica que você usou).

Texto base e imagens retirados de: http://www.img.lx.it.pt (acessado em 10/11/09) e do http://www.clubedohardware.com (acessado em 10/11/09)

Os monitores do tipo LCD são baseados na tecnologia de cristal líquido, cujas moléculas possuem a característica de serem alinhadas sob a influência de campos elétricos, permitindo ou não a passagem de luz mediante o alinhamento dessas moléculas.

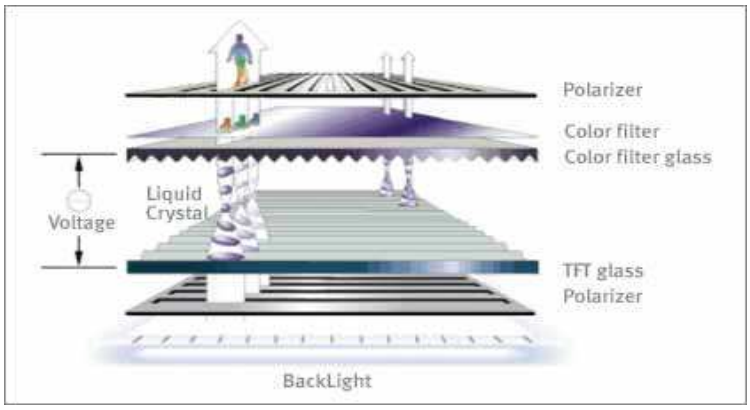
De uma forma geral, esta tecnologia baseia-se em produzir imagens sobre uma superfície plana composta por cristal líquido e filtros coloridos. Duas superfícies com filtros polarizados, que podem ser encarados como um conjunto de fios muito finos paralelos, controlam os raios de luz que passam através das moléculas de cristal líquido. As linhas de um dos filtros são dispostas perpendicularmente às linhas do outro filtro, e as moléculas entre as duas superfícies são forçadas a um estado torcido, direcionando os raios de luz da mesma forma.

Assim, quando não há nenhum campo elétrico aplicado às moléculas, a direcção do raio de luz vai-se alterando à medida que passa pelo cristal até encontrar a segunda superfície, cuja direção das ranhuras coincidirá com a do raio de luz.

Se um campo for aplicado ao cristal, as moléculas dispõem-se verticalmente, fazendo com que os raios de luz percorram o intervalo sem alterar a sua direcção, até encontrarem a segunda superfície que bloqueará os raios.

De um modo simplificado, a ausência de um campo aplicado é sinónimo de passagem de luz. Por sua vez, quando aplicamos uma tensão, esta luz será bloqueada.

Uma fonte de luz fluorescente, identificada geralmente pelo termo backlight, é responsável pela emissão dos raios que são alinhados pelos filtros polarizados. A luz direccionada passa, então, pela camada contendo milhares de bolhas de cristal líquido arranjadas em pequenas células que, por sua vez, estão dispostas em linhas na tela. Uma ou mais células formam um pixel no monitor.



Esquema de funcionamento do LCD.

Fonte: *http://www.img.lx.it.pt*

Em ecrãs LCD policromáticas, cada pixel é formado por três células de cristal líquido. Cada uma delas, filtrada por filtros vermelho, verde ou azul. Ao passar por essas células filtradas, a luz produz as cores que são vistas nas telas LCD.

A área à volta do pixel, já colorido, é colocada a preto, aumentando assim o contraste. Por sua vez, a luz passa ainda por um polarizador, para aumentar a nitidez e intensidade do pixel.

Existem dois principais tipos de telas de cristais líquidos: matriz passiva e matriz activa (TFT - Thin-Film Transistor).

Na tecnologia de matriz passiva, a tela consiste numa matriz de fios horizontais e verticais. A interseção dos fios define um pixel, e a corrente que controla os pixels é enviada através desses fios.

Para determinar o nível de brilho de cada pixel, aplica-se uma carga elétrica para que o cristal se realinhe e altere a direção do raio de luz. O processo é repetido sequencialmente por linha, desde a parte superior da tela até à inferior. Para cada linha da matriz de pixels, a corrente apropriada flui pelas colunas até à linha selecionada, para que o cristal seja alinhado na direção desejada.

O cristal líquido usado nesta solução apresenta baixo tempo de resposta, ou seja, depois de as moléculas serem orientadas pela carga elétrica, demoram algum tempo para retornar ao seu estado anterior, desalinhado. A resposta lenta do cristal faz com que cenas dinâmicas não sejam muito nítidas, fato que desaconselha o uso da matriz passiva, por exemplo, em jogos e filmes.

Por outro lado, o endereçamento usado pelos monitores de matriz passiva também é responsável por um outro efeito indesejável: quando um pixel é activado, pode haver alguma influência sobre os pixels vizinhos na mesma linha e coluna. Esse efeito é conhecido por crosstalk, um distúrbio causado por campos elétricos que afetam circuitos ou sinais adjacentes, afetando diretamente a aparência dos pixels próximos. Uma solução encontrada pelos fabricantes para reduzir esse problema foi dividir a tela numa metade inferior e outra superior, de modo a fazer o varredura em cada uma delas independentemente. Este tipo de tela é chamada de DSTN (Dual Scan Twisted Nematic). Alguns modelos ainda contam com um recurso extra, que faz um endereçamento simultâneo de duas linhas.

No caso do TFT, este faz uso de transístores em cada pixel que permitem que estes sejam controlados individualmente, podendo ser ativados ou desativados de forma independente. Deste modo, é possível visualizar o movimento nas imagens de forma mais rápida, pois é garantido que a imagem de um pixel não irá afetar a imagem do pixel vizinho.

Principais características encontradas em monitores LCD:

- *Tempo de resposta (ou desempenho): este parâmetro mede o tempo que a tela leva para mudar um pixel de "desligado" (preto) para "ligado" (branco). Este tempo é medido em milissegundos e quanto menor, melhor.*
- *Brilho: este parâmetro indica o quão bem você poderá ver imagens na sua tela em ambientes muito claros.*
- *Taxa de contraste: mede a diferença de brilho entre a quantidade de branco máxima e a quantidade de preto máxima que o monitor consegue gerar. Quanto maior este fator, melhor a taxa de contraste dinâmica (DC):*
- *Ângulo de visão: O ângulo de visão indica o ângulo máximo que o usuário pode estar em relação ao monitor e ainda ver o conteúdo da tela.*
- *Conexões: os monitores LCD podem usar dois tipos de conexões, VGA (usando um plugue chamado D-Sub) ou DVI-D.*