AD2 - Organização de Computadores 2015.1
Data de entrega:19/05/2015

"Atenção: Como a avaliação a distância é individual, caso seja constatado que provas de alunos distintos são cópias umas das outras, independentemente de qualquer motivo, a todas será atribuída a nota ZERO. As soluções para as questões podem ser buscadas por grupos de alunos, mas a redação final de cada prova tem que ser individual."

1. (1,0) Um sistema de computação utiliza 6 bits para representar dados. Considere o seguinte conjunto de bits que representam números inteiros em complemento a 2:

A=100001
B=111111
C=001001
D=011111

As seguintes operações foram realizadas:

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

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

As seguintes operações foram realizadas:
E=A+B => E =100001 + 111111 = 1 100000 => soma de 2 números negativos, resultando em um número negativo => não houve overflow

F=C+D => F = 001001 + 011111 = 0 101000 => soma de 2 números positivos, resultando em um número negativo, => houve overflow

G=B+C => G = 111111 + 001001 = 1 101000 => soma de 1 número negativo com 1 positivo, resulta em positivo ou negativo, seguindo o sinal de quem for maior em módulo, mesmo com bit de carry=> não houve overflow

H=A+D => H = 100001 + 011111 = 1 000000 => soma de 1 número negativo com 1 positivo, resulta em Positivo ou negativo, seguindo quem for maior em módulo=> não houve overflow

2. (2,0) Considere um computador, cuja representação para ponto fixo e para ponto flutuante utilize 20 bits. Na representação em ponto flutuante, as combinações possíveis de bits representam números normalizados do tipo +/-(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} b_{-14} × 2^{Expoente}), onde o bit mais à esquerda representa o sinal (0 para números positivos e 1 para números negativos), os próximos 5 bits representam o expoente em sinal e magnitude e os 14 bits seguintes representam os bits b_{-1} a b_{-14}, como mostrado na figura a seguir:

a) (0,2) Mostre a representação do número -2343,125 na representação em ponto flutuante neste computador.

Convertendo para binário = 100100100111,001 => colocando na notação científica 1,00100100111001 x 2^{+11}
Temos então:
Sinal = 1 (negativo)
Expoente = +11 = 01011 (sinal magnitude)
Mantissa = , 100100100111001
Resultado: 1 01011 00100100111001 ou 1010 11001001 00111001

b) (0,6) Para conjunto de bits obtido acima, indique o valor que está sendo representado em decimal, considerando-se que o conjunto representa:

(1010 11001001 00111001) $_2$

(b.1) (0,2) um inteiro sem sinal
Temos: $2^{19} + 2^{17} + 2^{15} + 2^{14} + 2^{11} + 2^8 + 2^5 + 2^4 + 2^3 + 2^0 = +706.873$

(b.2) (0,2) um inteiro em sinal magnitude
Temos: $-(2^{17} + 2^{15} + 2^{14} + 2^{11} + 2^8 + 2^5 + 2^4 + 2^3 + 2^0) = -182.585$

(b.3) (0,2) um inteiro em complemento a 2
Temos: $-2^{19} + (2^{17} + 2^{15} + 2^{14} + 2^{11} + 2^8 + 2^5 + 2^4 + 2^3 + 2^0) = -341.703$

c) (0,4) Qual será a representação em ponto flutuante dos seguintes valores decimais neste computador:

c.1) +27,0
Convertendo para binário = 11011,0 => colocando na notação científica 1,10110 x 2^{+4}
Temos então:
Sinal = 0 (positivo)
Expoente = 4 = 00100
Mantissa = , 10110
Resultado: 0 00100 10110000000000 ou 0001 00101100 00000000

c.2) –2,625
Convertendo para binário = 10,101 => colocando na notação científica 1,0101 x 2^{+1}
Temos então:
Sinal = 1 (negativo)
Expoente = 1 = 00001
Mantissa = ,0101
Resultado: 1 00001 0101000000000 ou 1000 01010100 00000000

d) (0,4) Considere que todas as combinações possíveis de bits representam números normalizados quando se utiliza a representação para ponto flutuante. 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.

Menor valor positivo: 0 11110 00000000000000 => 1,00000000000000 x 2^{-14} *= + 0,0000610351*
Maior valor positivo: 0 01111 1111111111111 => 1,1111111111111 x 2^{+15} *= + 65535*

Maior valor negativo: 1 11110 00000000000000 => 1,00000000000000 x 2^{-14} *= - 0,0000610351*
Menor valor negativo: 1 01111 1111111111111 => 1,1111111111111 x 2^{+15} *= - 65535*

e) (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 c.1 e c.2.

Excesso = (5 bits) = $2^{5-1} - 1 = 15$

e.1) +27,0
Convertendo para binário = 11011,0 => colocando na notação científica 1,10110 x 2^{+4}
Temos então:
Sinal = 0 (positivo)
Expoente = 4_{10} *=> 4 + 15 = 19 =>* 10011_2 *(por excesso)*
Mantissa = , 10110
Resultado: 0 10011 10110000000000 ou 0100 11101100 00000000

e.2) –2,625

Convertendo para binário = 10,101 => colocando na notação científica 1,0101 x 2^{+1}

Temos então:

Sinal = 1 (negativo)

Expoente = 1_{10} => 1 + 15 = 16 => 10000 (por excesso)

Mantissa = ,0101

Resultado: 1 10000 010100000000 ou 1100 00010100 00000000

3. (1,0) Explique como é o mecanismo de funcionamento de um scanner (indique a referência bibliográfica que você usou).

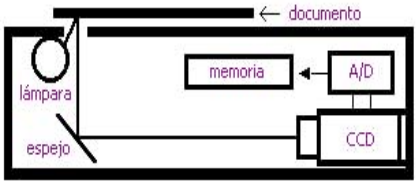
(fonte e imagem : www.clubedohardware.com.br e *www.criaweb.com.br)*

O funcionamento de um scanner consiste em transformar as imagens visuais lidas em sinais digitais que podem ser entendidos pelo computador. Para isto o scanner utiliza suas células fotosensíveis de forma a transformar a imagem em um conjunto de pontos, cada um com uma cor, ou seja: num bitmap.

Como em toda conversão analógico-digital, neste processo perdemos sempre um pouco da qualidade da imagem original. O quanto vamos perder em termos de qualidade, depende justamente da resolução suportada e do número de cores reconhecidas pelo scanner.

Um scanner é formado pelos seguintes elementos:

- *Uma fonte de luz fluorescente ou incandescente para iluminar o objeto que se deseja digitalizar.*
- *Um sistema óptico, geralmente formado por espelhos, que recolhe a luz refletida pelo objeto e a dirige ao fotosensor.*
- *Um fotosensor que recolhe a luz refletida pelo objeto e a transforma em um sinal elétrico analógico, normalmente um chip CCD.*
- *Um conversor analógico/digital (ACD ou A/D), que converte o sinal elétrico que produz o fotosensor em impulsos digitais em formato binário (zeros e uns), entendíveis por uma máquina informática.*
- *Um dispositivo que se encarrega de armazenar essa imagem ou de transpassá-la a um computador para que seja armazenada ali.*



Estrutura básica de um scanner (imagem obtida de www.criaweb.com)

Ademais, os scanners necessitam de um software específico para poder tratar as imagens que através dele se obtém. Existem dois tipos básicos de software de tratamento de imagens:

- *Programas para a digitalização de objetos em imagens. Produzem arquivos digitais gráficos formados por imagens de mapa de bits.*
- *Programas para a digitalização de documentos como textos, denominados OCR (Optical Character Recognition) ou ICR (Intelligent Character Recognition). Produzem documentos digitais formados por caracteres ASCII que se podem editar e armazenar e por elementos gráficos de mapas de bits.*

Para digitalizar um objeto com um scanner se deve situar o mesmo (um documento, uma fotografia, uma ilustração, um slide, um desenho, etc.) sobre a tela do scanner, onde é banhado por raios de luz procedentes da fonte de luz do scanner. A luz refletida pelo objeto passa ao sistema óptico, que centra a luz no fotosensor, geralmente do tipo CCD, que converte a intensidade da luz que recebem em uma série de sinais elétricos analógicos equivalentes.

O CDC (Charge Coupled Device) pode conter milhares de células fotoelétricas densamente empacotadas que agem convertendo o sinal luminoso de intensidade variável que recebem em um sinal elétrico de voltagem proporcional. O CCD pode ser linear ou matricial, O primeiro se utiliza nos scanners planos e de mão, e os segundos em scanners de transparências, câmeras digitais e câmeras de vídeo.

Os sinais elétricos analógicos procedentes do fotosensor são enviados ao ADC, que os converte em sinais digitais codificados aptos para ser lidos pelo software apropriado..

Os scanners realizam o processo de leitura do original em um passo, onde o mecanismo do scanner

captura em uma só passada a imagem com todos seus atributos de cor, separando-se logo os componentes de cor vermelha, azul e verde da luz refletida mediante um prisma ou um filtro, sendo enviados então a uma faixa de CCD's responsáveis de cada cor particular.

4. (1,0) Explique como é realizada a transferência de dados e a arbitração em um barramento PCI (sugestões de fonte de consulta: livro do Stallings e o site http://computer.howstuffworks.com. Na sua resposta indique as suas fontes de consulta).

Texto retirados do Livro texto da disciplina do Willian Stallings e das Notas de aula Disciplina Arquitectura e Organização Interna de Computadores – Universidade do Minho – PT

"A interface do PCI é composta pelo seguinte conjunto de 62 pinos na especificação de 32 bits e por 94 pinos na versão de 64 bits, porém muitos destes sinais não precisam ser tratados por todos os dispositivos, assim uma implementação mínima de um dispositivo PCI é composta de apenas 47 pinos caso o dispositivo nunca se torne um master do barramento ou 49 pinos caso este dispositivo seja projetado para se tornar o mesre do barramento. Esta pinagem reduzida é uma das grandes vantagens do PCI. Os dispositivos conectados ao barramento e assumirão a posição de mestre, conhecido como iniciador (initiator) ou escravo, conhecido como alvo (target). Abaixo segue uma descrição das vias de um barramento PCI.

AD0-AD31 *– Nestas vias do barramento os dados e os endereços são multiplexados, assim no início de uma transferência, este barramento indica o endereço, e na fase seguinte, os dados.*

C/BE0-C/BE3 *– Estes sinais indicam qual o comando que será executado (leitura, escrita, etc.), essa informação é enviada durante a fase de endereço. Já durante a fase de dados cada bit desses quatro sinais indica quais dos quatro bytes do barramento estão ativos, assim é possível acessar os bytes de dados de forma individual.*

FRAME *– Este sinal é ativado pelo Mestre do Barramento para dar início a um ciclo de transferência.*

IRDY *- Initiator Ready. Indica que o Mestre está pronto para ler ou enviar dados. Quando este sinal não é ativado, o escravo irá esperar tantos wait states quanto forem necessários.*

TRDY *- Target Ready. Indica que o Escravo está pronto para receber dados (escrita) ou que o dado lido já está disponível (leitura). Quando este sinal não é ativado, o Mestre irá gerar tantos wait states quando forem necessários.*

DEVSEL *- Ativado pelo Escravo quando reconhece o seu endereço. Desta forma o Mestre pode saber se o dispositivo Escravo está ativo ou presente no barramento.*

REQ *- Requisição enviada ao Bus Arbitrer, para que o dispositivo se torne Mestre do barramentos. Cada dispositivo tem seu próprio sinal REQ.*

GNT *- Grant. Através deste sinal o Bus Arbitrer indica ao dispositivo solicitante que o barramento está liberado, permitindo assim que se torne Bus Master. Cada dispositivo tem seu próprio sinal GNT. INTA, INTB, INTC, INTD - São linhas de interrupção a serem usadas pelos dispositivos PCI. Cada dispositivo e cada slot é ligado a um desses sinais, que podem ser compartilhados, ou seja, uma mesma linha INT pode ser usada por mais de um slot. O padrão PCI prevê o compartilhamento de interrupções.*

AD32-AD63 *- Continuação do barramento de dados e endereços nos slots PCI de 64 bits.*

C/BE4-C/BE7 *- Continuação do barramento de comando e habilitação de bytes nos slots PCI de 64 bits.*

REQ64 *- Requisição de transferência de 64 bits.*

ACK64 *- Indica que o Escravo está apto a realizar transferência de 64.*

CLK *- Clock este é o trem de pulso para todas as transferências do PCI.*

RST *– Reset é o sinal que instrui o dispositivo a assumir um estado inicial, reiniciando todos os registradores ou qualquer componente de estado para o estado padrão, este sinal normalmente é enviado para o PCI quando o computador inicia.*

PERR *- Parity Error é utilizado para reportar erros de paridade na transmissão dos dados.*

SERR *- System Error indica erros do sistema, como erro de paridade no endereço ou qualquer outr tipo de erro fatal.*

Transferência de Dados

Um transferência de dados consiste em uma fase de endereço seguido de uma ou mais fases de dados, uma operação de I/O padrão tem somente uma fase de endereço e uma fase de dados, já transferências para memória, quando se transfere blocos de dados a operação consiste em uma fase de endereço seguida de multiplas fases de dados escrevendo/lendo blocos contiguos de uma única vez. Tanto o alvo como o iniciador podem finalizar uma sequência de transferência a qualquer momento, o dispositivo alvo faz isso desabilitando o sinal FRAME após enviar o último dados solicitado, já o inciador ou master pode ativar o sinal STOP que sinaliza para o alvo o final da transferência.

Todos os eventos são sincronizados pela transição descendente do relógio que ocorre no meio de cada ciclo de relógio.

Texto abaixo retirado da fonte complementar já citada

Os eventos mais significativos

- *Logo que um mestre no barramento assume o controle, pode iniciar a transação impondo FRAME. Esta linha mantém a imposição até que o iniciador esteja pronto para completar a última fase de dados. O iniciador coloca também o endereço inicial no barramento de endereços e o comando de leitura nas linhas C/BE.*
- *No início do segundo ciclo o dispositivo alvo irá reconhecer o seu endereço nas linhas AD.*
- *O iniciador cessa de alimentar o barramento de endereços. É necessário em todas as linhas de sinal que possam ser alimentadas por mais do que um dispositivo, de forma a que o esgotamento dos sinais de endereço prepare o barramento para ser usado pelo dispositivo alvo. O iniciador muda a informação nas linhas de C/BE para designar quais são as linhas de AD a serem usadas na transferência para os dados correntemente endereçados (de 1 a 4 octetos). O iniciador, também, impõe IRDY para indicar que está pronto para o primeiro item de dados.*
- *O dispositivo selecionado impõe DEVSEL para indicar que reconheceu o seu endereço e vai responder. Coloca os dados solicitado nas linhas de AD e impõe TRDY para indicar que estão presentes dados válidos no barramento.*
- *O iniciador lê o dado no início do ciclo 4 e muda, conforme o necessário, as linhas de habilitação de octetos, em preparação para a leitura seguinte.*
- *Neste exemplo, o alvo necessita de algum tempo para preparar o segundo bloco de dados para transmissão. Consequentemente, baixa o sinal TRDY para sinalizar o iniciador que não irá haver dados no ciclo seguinte. Em conformidade, o iniciador não lê as linhas de dados no início do quinto ciclo de relógio e não muda a habilitação de octeto durante aquele ciclo. O bloco de dados é lido no início do ciclo 6.*
- *Durante o ciclo 6, o alvo coloca o terceiro item de dados no barramento. Contudo, no exemplo, o iniciador não está ainda pronto para ler o item de dados (e.g. há uma condição de tampão temporário cheio). Por isso baixa IRDY. Isto fazer com que o alvo possa manter o terceiro item no barramento durante um ciclo extra de relógio.*
- *O iniciador sabe que a terceira transferência de dados é a última. Impõe IRDY para sinalizar que está pronto para completar a transferência.*
- *O iniciador baixa IRDY, regressando o barramento ao estado de espera e o alvo baixa TRDY e DEVSEL.*

Árbitragem do Barramento: *(Texto base retirado de fonte complementar já citada)*

O PCI recorre a um esquema de arbitragem centralizada e síncrona no qual cada mestre tem um sinal único para requerimento e para concessão. Estas linhas de sinais estão presas a um árbitro central e um simples esquema requerimento-concessão é usado para obter o acesso ao barramento.

A especificação PCI não impõe um algoritmo particular de arbitragem. O árbitro pode usar uma abordagem primeiro a chegar primeiro a ser servido, uma abordagem andar à volta ou qualquer outra espécie de esquema de prioridade.

Considere 2 dispositivos (A e B) que requerem a posse do barramento.

1. *Em algum ponto anterior ao início do ciclo 1, A impõe o sinal REQ. O árbitro faz a amostragem do sinal no início do ciclo 1 de relógio.*
2. *Durante o ciclo 1, B requisita o uso do barramento impondo o seu sinal REQ.*
3. *Ao mesmo tempo, o árbitro impõe GNT-A para garantir a A o acesso ao barramento.*
4. *O mestre A faz a amostragem de GNT-A no início de ciclo 2 e toma conhecimento que lhe foi concedido o acesso ao barramento. Descobre, também, IRDY e TRDY em baixo, indicando que o barramento está livre. Em concordância, impõe FRAME e coloca a informação de endereço no barramento de endereços e o comando no barramento C/BE (não visível). Mantém REQ-A, porque tem uma segunda transação a efectuar a seguir a esta.*
5. *O árbitro de barramento faz a amostragem de todas as linhas GNT no início do ciclo 3 e toma uma decisão de conceder o barramento a B na próxima transacção. Impõe GNT-B e baixa GNT-A. B não irá poder usar o barramento até que este retorne ao estado de ócio.*
6. *A FRAME baixa indica que a última e (única) transferência de dados está em progresso. Coloca os dados no barramento e assinala o alvo com IRDY. O alvo lê os dados no início do próximo ciclo de relógio.*
7. *No início do ciclo 5, B descobre IRDY e FRAME em baixo e consequentemente pode tomar o controlo do barramento impondo FRAME. Também põe em baixo a linha de REQ porque apenas pretende efetuar uma transação.*

5. (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). No conjunto de instruções proposto

para dois e três operandos, projete instruções com endereçamento imediato, direto e indireto.

I_ CONJUNTO DE INSTRUÇÕES PARA 1 OPERANDO:

```
ADD X       => (ACC)<-(ACC)+(X)
SUB X       => (ACC)<-(ACC)-(X)
MUL X       => (ACC)<-(ACC)*(X)
DIV X       => (ACC)<-(ACC)/(X)
LOAD X      => (ACC)<-  (X)
STORE X     => (X)  <-(ACC)
```

II_ CONJUNTO DE INSTRUÇÕES PARA 2 OPERANDOS:

OBS: Em geral, para diferenciar as instruções conforme o modo de endereçamento, é acrescido ou modificado o mneumônico da instrução base. Nos casos abaixo, foram acrescidos as letras: i, d, n ; para indicar os modos Imediato, direto e indireto, respectivamente E caso estas mesmas instruções sejam utilizadas em um processador que possua também instruções de 1 e 3 operandos, uma nova identificação poderá ser acrescida ao rótulo da instrução para diferenciar das demais com 1 ou 3 operandos.

Imediato	Direto	Indireto

`ADDiD X,Y => (X)<-(X)+Y`
`SUBiD X,Y => (X)<-(X)-Y`
`MULiD X,Y => (X)<-(X)*Y`
`DIViD X,Y => (X)<-(X)/Y`
`MOViD X,Y => (X)<-Y`
 	
Neste modo, Y é um valor imediato
	`ADDdD X,Y => (X)<-(X)+(Y)`
	`SUBdD X,Y => (X)<-(X)-(Y)`
	`MULdD X,Y => (X)<-(X)*(Y)`
	`DIVdD X,Y => (X)<-(X)/(Y)`
	`MOVdD X,Y => (X)<-(Y)`
	
	Neste modo, Y é um registrador ou endereço de memória
		`ADDnD X,Y => (X)<-(X)+((Y))`
		`SUBnD X,Y => (X)<-(X)-((Y))`
		`MULnD X,Y => (X)<-(X)*((Y))`
		`DIVnD X,Y => (X)<-(X)/((Y))`
		`MOVnD X,Y => (X)<-((Y))`
		
		Neste modo, Y é um registrador ou endereço de memória que contenha a posição do valor desejado

III_ CONJUNTO DE INSTRUÇÕES PARA 3 OPERANDOS:

OBS: Em geral para diferenciar as instruções é acrescido ou modificado o mneumônico da instrução base. Nos casos abaixo, foram acrescidos as letras: i, d, n ; para indicar os modos Imediato, direto e indireto, respectivamente. E caso estas mesmas instruções sejam utilizadas em um processador que possua também instruções de 1 e 2 operandos, uma nova identificação poderá ser acrescida ao rótulo da instrução para diferenciar das demais com 1 ou 2 operandos.

Imediato	Direto	Indireto

`ADDiT X,Y,Z=>(X)<-(Y)+ Z`
`SUBiT X,Y,Z=>(X)<-(Y)- Z`
`MULiT X,Y,Z=>(X)<-(Y)+ Z`
`DIViT X,Y,Z=>(X)<-(Y)/ Z`
 	
Neste modo, Z é um valor imediato
	`ADDdT X,Y,Z=>(X)<-(Y)+(Z)`
	`SUBdT X,Y,Z=>(X)<-(Y)-(Z)`
	`MULdT X,Y,Z=>(X)<-(Y)*(Z)`
	`DIVdT X,Y,Z=>(X)<-(Y)/(Z)`
	
	Neste modo, Z é um registrador ou endereço de memória
		`ADDnT X,Y,Z=>(X)<-(Y)+((Z))`
		`SUBnT X,Y,Z=>(X)<-(Y)-((Z))`
		`MULnT X,Y,Z=>(X)<-(Y)*((Z))`
		`DIVnT X,Y,Z=>(X)<-(Y)/((Z))`
		
		Neste modo, Z é um registrador ou endereço de memória que contenha a posição do valor desejado

X= (A/D - C)+(B*(D - E/B)* (C + A) + E)

<u>PARA 1 OPERANDO:</u>

LOAD A => (ACC) <- (A)
DIV D => (ACC) <- (ACC) / (D)
SUB C => (ACC) <- (ACC) – (C)
STORE T1 => (T1) <- (ACC)
LOAD E => (ACC) <- (E)
DIV B => (ACC) <- (ACC) / (B)
STORE T2 => (T2) <- (ACC)
LOAD D => (ACC) <- (D)
SUB T2 => (ACC) <- (ACC) – (T2)
LOAD C => (ACC) <- (C)
ADD A => (ACC) <- (ACC) + (A)

```
STORE T3        => (T3)  <- (ACC)
LOAD B => (ACC) <- (B)
MUL T2          =>       (ACC) <- (ACC) * (T2)
MUL T3          =>       (ACC) <- (ACC) * (T3)
ADD E           =>       (ACC) <- (ACC) + (E)
ADD T1          =>       (ACC) <- (ACC) + (T1)
STORE X         => (X)  <- (ACC)
```

PARA 2 OPERANDOS:

```
MOVdD T1,A    =>    (T1) <- (A)
DIVdD T1,D    =>    (T1) <- (T1) / (D)
SUBdD T1,C    =>    (T1) <- (T1) - (C)
MOVdD T2,E    =>    (T2) <- (E)
DIVdD T2,B    =>    (T2) <- (T2) / (B)
MOVdD T3,D    =>    (T3) <- (D)
SUBdD T3,T2   =>    (T3) <- (T3) - (T2)
MOVdD T4,C    =>    (T4) <- (C)
ADDdD T4,A    =>    (T4) <- (T4) + (A)
MOVdD T2,B    =>    (T2) <- (B)
MULdD T2,T3   =>    (T2) <- (T2) * (T3)
MULdD T2,T4   =>    (T2) <- (T2) * (T4)
ADDdD T2,E    =>    (T2) <- (T2) + (E)
ADDdD T1,T2   =>    (T1) <- (T1) + (T2)
MOVdD X,T1    =>    (X)  <- (T1)
```

PARA 3 OPERANDOS:

```
DIVdT T1, A, D    =>   (T1) <- (A)   / (D)
SUBdT T1,T1, C    =>   (T1) <- (T1)  - (C)
DIVdT T2, E, B    =>   (T2) <- (E)   / (B)
SUBdT T2, D,T2    =>   (T2) <- (D)   - (T2)
ADDdT T3, C, A     =>  (T3) <- (C)   + (A)
MULdT T2, B,T2     =>  (T2) <- (B)   * (T2)
MULdT T2,T2,T3     =>  (T2) <- (T2)  * (T3)
ADDdT T2,T2, E     =>  (T2) <- (T2)  + (E)
ADDdT  X,T1,T2     =>  (X) <- (T1)   + (T2)
```

6. (1,0) Faça uma busca na lista dos 500 sistemas de computadores com melhor desempenho do mundo em http://www.top500.org e descreva o primeiro colocado (pesquise neste mesmo site e na internet).

Tianhe-2 – Fonte http://tecnoblog.net

1o. colocado: Tianhe-2 Localizado no Centro Naciona

l de Supercomputadores em Guangzhou – China. Com um desempenho de cerca de 33.862,7 Tflops/s. Possui cerca de 3.120.000 cores, provenientes de cerca de 32 mil processadores Intel Xeon E5-269v2 com 12 cores de 2.2GHz e ainda 48 mil coprocessadores Intel Xeon Phi, chips de alto desempenho com mais de 50 núcleos cada, conectados via porta PCI Express. Possui também cerca de 1 petabyte de RAM. Sistema operacional baseado em Kylin Linux. Rede proprietária de interconexão: TH

7. (1,0) Responda as questões abaixo:

a) Analise os modos de endereçamento direto, indireto e imediato, estabelecendo diferenças de desempenho, vantagens e desvantagens de cada um

MODO DE ENDEREÇAMENTO	*DEFINIÇÃO*	*VANTAGENS*	*DESVANTAGENS*	*DESEMPENHO*
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*

b) 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?

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

c) 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.

d) Analise os modos de endereçamento direto, indireto e imediato, estabelecendo diferenças de desempenho, vantagens e desvantagens de cada um. 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.

8. (1,0) Descreva as instruções SIMD do processador Intel core I7 (extensões SSE), detalhando seu funcionamento e aplicabilidade. Pesquise na wikipedia e nos sites da Intel.

Fonte: Intel® 64 and IA-32 Architectures Software Developer's Manual (Manual para desenvolvedores de software para arquiteturas Intel-64 e IA-32), encontrado em http://www.intel.com/Assets/PDF/manual/253665.pdf

O processador I7 utiliza o mesmo conjunto de instruções de seu antecessor (Core 2 duo) acrescido do conjunto complementar de instruções: SSE 4.2
O conjunto de instruções do I7 é composto de:

- *Instruções de uso geral*
- *Instruções x87 FPU (instruções que operam com operandos em ponto flutuante.*
- *Extensões SIMD*
    - *MMX*
    - *SSE*
    - *SSE2*
    - *SSE3 (incluíndo o subgrupo SSSE3)*
    - *SSE4 (incluíndo os subgrupos SSE4.1 e SSE4.2), sendo o I7 o primeiro modelo de processador a utilizar o subgrupo SSE4.2*

Extensões SIMD

Quatro extensões foram introduzidas na arquitetura IA-32 permitindo que processadores executem instruções que trabalhem com multiplos dados (SIMD). Estas extensões incluem a tecnologia MMX e extensões SSE, SSE2 e SSE3.

***Instruções MMX**:*
Instruções MMX operam com operandos inteiros do formato byte, doubleword, ou quadword contidos na memória, nos registradores MMX e / ou nos registradores de uso geral.
As. instruções MMX são divididas nos seguintes subgrupos: transferência de dados, conversão aritmética, comparação, operações lógicas, deslocamento e rotação, e instruções de gerenciamento de estado.

Extensão SSE:
Introduzidas inicialmente no Pentium III, foram adicionados oito novos registradores de 128 bits, conhecido como XMM0 a XMM7. As extensões AMD64 da AMD (originalmente chamado de x86-64 e depois repetido pela Intel) adicionaram mais oito, de XMM8 até XMM15.
Instruções SSE são divididos em quatro subgrupos

• Instruções SIMD de ponto flutuante que operam com os registradores XMM
• Instruções de gerenciamento do estado MXSCR
• Instruções SIMD de inteiro de 64bits que operam sobre os registradores XMM
• Controle de cacheabilidade, prefetch e instrução de ordenação de instruções

Extensão SSE2:
Introduziada com o Pentium IV, operam com operandos de ponto flutuante com dupla precisão e operandos inteiros de formato: byte, doubleword e quadword armazenados nos registradores XMM.
Estas instruções são divididas em quatro subgrupos:
• Instruções de ponto-flutuante com dupla precisão
• Instruções de conversão para ponto flutuante de simples precisão
• Instruções SIMD com formato interiro de 128bits
• Controle de cacheabilidade, prefetch e instrução de ordenação de instruções

Extensão SSE3:
A extensão SSE3 oferece 13 instruções que aceleram o desempenho da capacidade matemática do SSE, SSE2 e x87 FPU. Estas instruções podem ser agrupadas nas seguintes categorias:
• Uma instrução x87FPU usado na conversão de formato inteiro
• Uma instrução SIMD de inteiro que endereça carga de dados não alinhados
• Duas instruções SIMD de ADD / SUB de ponto flutuante
• Quatro instruções SIMD de ADD / SUB horizontal de ponto flutuante
• Três instruções SIMD de LOAD / MOVE / DUPLICATE de ponto flutuante
• Duas instruções para sincronização de threads

O Subgrupos SSSE3 fornece 32 instruções (representadas por 14 mnemônicos) para acelerar os cálculos com números inteiros. Estes incluem:

- *Doze instruções que realizam operações de adição ou subtração horizontal.*
- *Seis instruções que avaliam valores absolutos.*
- *Duas instruções que realizam operações de multiplicação e adição de forma a acelerar o avaliação de produtos de ponto flutuante.*
- *Duas instruções que aceleram operações de multiplicação no formato inteiro e produzir valores inteiros com escamação.*
- *Duas instruções que executam um byte-wise, shuffle no local indicado pelo operando de controle de shuffle.*
- *Seis instruções que negativam inteiros no operando destino se o sinal do elemento correspondente ao operando de origem é menor que zero.*
- *Duas instruções que alinham dados de uma composição de dois operandos.*

Extensão SSE4:
Intel ® Streaming SIMD Extensions 4 (SSE4) apresenta 54 novas instruções classificadas nos seguintes subgrupos:
- *SSE 4.1: 47 instruções*
- *SSE 4.2 7 instruções (inicialmente lançadas para os I7).*

O SSE4.1 é voltado para melhorar o desempenho dos meios de comunicação, imagem e cargas de trabalho 3D. SSE4.1 acrescenta instruções que melhoram significativamente a compilação e vetorização. Aumenta o auxilio no processamento do formato dword.

As 47 instruções do SSE4.1 incluem:
• Duas instruções que realizam multiplicação no formato dword.
• Duas instruções de ponto flutuante do tipo dot products com seleção de entrada / saída
• Uma instrução que realiza cargamento com streaming hint.
• Seis instruções simples de conversão de formatos.
• Oito instruções de expansão de suporte para formatos inteiros MIN / MAX.
• Quatro instruções com suporte ao arrendondamento de ponto flutuante com o modo de arredondamento selecionável..
• Sete instruções que otimizam a inserção e extração de dados de registos XMM
• Doze instruções para conversão de formatos de inteiro (sinal e extensão zero).
• Uma instrução de otimização SAD (soma de diferença absoluta) para geração de blocos de pequeno tamanhos
• Uma instrução auxiliar em operações de busca horizontal.

• Uma instrução otimizada para comparações mascaradas.
• Uma instrução adiciona comparação de igualdade para formato qword .
• Uma instrução adiciona satuação sem sinal para formato dword;

Os sete instruções SSE4.2 incluem:

- *Cinco das sete instruções SSE4.2 podem usar um registrador XMM como origem ou destino. Nestas estão inclusas 4 instruções para processamento de texto/string e uma instrução SIMD para comparações no formato quadword.*
- *As 2 instruções restantes utilizam registradores de uso geral para melhorar o desempennho no processamento de funções em áreas específicas.*